The VLDB Journal (2020) 29:1171–1189
https://doi.org/10.1007/s00778-020-00610-z

REGULAR PAPER


# Diversified spatial keyword search on RDF data

**Zhi Cai[1] · Georgios Kalamatianos[2] · Georgios J. Fakas[2] · Nikos Mamoulis[3] · Dimitris Papadias[4]**

Received: 18 April 2019 / Revised: 22 November 2019 / Accepted: 10 February 2020 / Published online: 12 March 2020


**Abstract**

The abundance and ubiquity of RDF data (such as DBpedia and YAGO2) necessitate their effective and efficient retrieval. For this purpose, keyword search paradigms liberate users from understanding the RDF schema and the SPARQL query language. Popular RDF knowledge bases (e.g., YAGO2) also include spatial semantics that enable location-based search. In an earlier location-based keyword search paradigm, the user inputs a set of keywords, a query location, and a number of RDF spatial entities to be retrieved. The output entities should be geographically close to the query location and relevant to the query keywords. However, the results can be similar to each other, compromising query effectiveness. In view of this limitation, we integrate textual and spatial diversification into RDF spatial keyword search, facilitating the retrieval of entities with diverse characteristics and directions with respect to the query location. Since finding the optimal set of query results is NP-hard, we propose two approximate algorithms with guaranteed quality. Extensive empirical studies on two real datasets show that the algorithms only add insignificant overhead compared to non-diversified search, while returning results of high quality in practice (which is verified by a user evaluation study we conducted).

**Keywords** Diversity · Ptolemy's spatial diversity · Keyword search · Spatial RDF data · Ranking

## 1 Introduction

With the proliferation of knowledge-sharing communities, such as Wikipedia, and the advances in automated information extraction from the Web, large knowledge bases, including DBpedia [14] and YAGO [54], are made available to the public. Such knowledge bases typically adopt the resource description framework (RDF) data model. A knowledge base in RDF is a table of ⟨subject, predicate, object⟩ triplets, where subjects correspond to entities and objects can be other entities or literals (i.e., constants) associated with the subjects via the predicates. For example, the triplet ⟨Beethoven, born_in, Bonn⟩ captures the fact that the entity Beethoven was born in the city of Bonn. The English version of DBpedia currently describes 4.5M entities, including about 1.4M persons, 883K places, 411K creative works, 241K organizations, 251K species, etc. YAGO contains more than 10M entities (e.g., persons, organizations, cities) and 120M facts about these entities. Data.gov [13] is the largest open-government, data-sharing website that has more than a thousand datasets in RDF format with a total of 6.4 billion triplets, covering information about business, finance, health, education, local government, etc.

Recently, RDF has been enriched with *spatial semantics*. For example, YAGO2 [34] is an extension of YAGO that includes spatial and temporal data. Such knowledge bases enable location-based retrieval. Indicatively, a key research direction of BBC News Lab is: *How might we use geolocation and linked data to increase relevance and expose the coverage of BBC News?* [6]. To fully utilize spatially enriched

✉ Georgios J. Fakas
georgios.fakas@it.uu.se

Zhi Cai
caiz@bjut.edu.cn

Georgios Kalamatianos
georgios.kalamatianos@it.uu.se

Nikos Mamoulis
nikos@cs.uoi.gr

Dimitris Papadias
dimitris@cs.ust.hk

1 College of Computer Science, Beijing University of Technology, Beijing, China

2 Department of Information Technology, Uppsala University, Uppsala, Sweden

3 Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece

4 Department of Computer Science and Engineering, HKUST, Clear Water Bay, Hong Kong

RDF data, the GeoSPARQL standard [5], defined by the Open Geospatial Consortium (OGC), extends RDF and SPARQL to represent geographic information. RDF stores such as Virtuoso [53], Parliament [43], and Strabon [39] are developed to support GeoSPARQL features. However, retrieval on such systems requires that query issuers fully understand the query language (e.g., SPARQL or GeoSPARQL) and the data domain, which is restrictive and discouraging for common users.

In view of this limitation, keyword search paradigms facilitate retrieval using only keywords [16–23,26,40,46,50]. Given a query that consists of a set of keywords, an answer is a subgraph of the RDF graph. The vertices of the subgraph should collectively cover all the input keywords. The sum of the lengths of the paths connecting the keywords defines a *looseness* score for the subgraph [27,40,50]. Compact results, i.e., subgraphs of low looseness, are more relevant. This is analogous to finding the smallest (tuple) subgraphs in relational keyword search [35] and general keyword search on graphs [32].

RDF keyword search has been enhanced to be location aware. Shi et al. [45] propose a model for searching *spatial entities*, i.e., entities associated with locations. For example, Bonn is a spatial entity since it has a fixed location, whereas Beethoven is not. A spatial keyword search query takes as input a location, a set of query keywords, and an integer $k$. The result is the set of top-$k$ spatial entities according to a ranking function that considers both the spatial distance between each candidate entity and the query location, and the graph-based proximity of the keywords to the entity. More precisely, a *qualified place* $p$ is a spatial entity for which there is a compact tree rooted at $p$ that collectively covers all query keywords. To effectively capture the textual semantics of each entity, in a preprocessing phase, the original RDF graph is reduced to a graph for which the keywords on all emitting edges from an entity are absorbed by the entity. Hence, a *document* (i.e., a set of keywords) is generated for each entity and the edges carry no keyword information. In addition, each entity document absorbs all literals (i.e., constants) associated with it.

Figure 1a–d shows the preprocessed graph representation of several triplets extracted from DBpedia. Each node is an entity associated with a document (denoted by the set of keywords in curly brackets), predicates, and literals [40]. Squares correspond to places, for which the locations have been extracted and are shown in Fig. 1e. Circles are non-spatial entities of the RDF graph. The edges model the relationships between entities. Assume a top-3 query issued by a tourist at location $q$ in Fig. 1e with keywords {*ancient*, *roman*, *catholic*, *history*}. According to [45], the result would consist of places $p_1$ (rooted at subgraph $\{p_1, v_1, v_2, v_3\}$, Fig. 1a), $p_2$ (Fig. 1b) and $p_3$. This is a good result in terms of semantic relevance and spatial distance, as the places (1) are rooted at compact subgraphs covering all query keywords [32,40] and (2) are geographically close to the query location $q$. However, results based entirely on relevance may have similar content [15,38,47] and location. For instance, the top-3 places share nodes $v_1$ and $v_3$, implying similar semantics (they all represent *communes*). In addition, they are all located in the same direction with respect to the query.

Indeed, several studies reveal that users strongly prefer spatially [49] and textually [56] diversified query results over un-diversified ones. Thus, in this paper, we introduce diversified spatial keyword search on RDF data. Our framework enables a trade-off between relevance and diversity. Namely, the output places, in addition to being relevant to the query, should minimize the number of common nodes in their subgraphs and should have diverse locations w.r.t. direction. For instance, a diversified query result for Fig. 1 could include $p_1$, $p_4$ (a river confluence) and $p_5$ (a church). These places are close to $q$, their subgraphs are compact, and they contain all keywords. Moreover, they are diverse because they are located around $q$ and their subgraphs have no common nodes. For this purpose, we propose a new spatial diversity metric (*Ptolemy's diversity*) which also considers the query location and has several attractive properties, e.g., it is naturally normalized to range [0, 1], satisfies triangle inequality, etc. These properties render Ptolemy's diversity superior to the existing metrics for spatial diversity that consider either only the distance [37] or the angle [51] between a pair of locations.

We show that diversified spatial keyword query evaluation on RDF data is NP-hard, by a reduction from the maximum clique problem. Thus, we propose two efficient branch-and-bound algorithms. The first, referred to as IAdU, generates the results by adding and updating the scores of candidate entities. The second algorithm, ABP, incrementally builds results by adding the best pair at each iteration. IAdU is faster than ABP, but has an approximation bound of 4, whereas ABP returns a 2-approximation of the optimal solution. This trade-off renders the investigation of both algorithms interesting. Concretely, our contributions can be summarized as follows:

- We define the problem of top-$k$ diversified spatial keyword search and show that it is NP-hard.
- We introduce Ptolemy's spatial diversity, a novel spatial diversity metric.
- We propose two efficient algorithms for retrieval of diverse results.
- We provide a theoretical analysis with approximation bounds of our algorithms.
- We conduct a thorough experimental evaluation on real datasets, demonstrating the efficiency of our algorithms, the effectiveness, and user preference (we conducted a user evaluation) of our methodology.

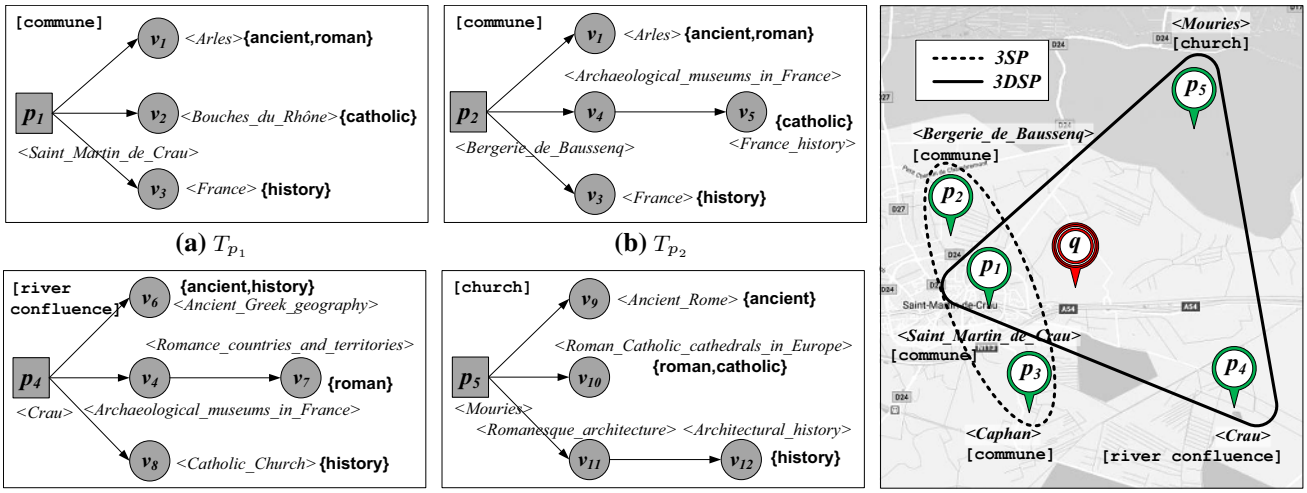



**Fig. 1** Example of spatial keyword query and results

The rest of the paper is organized as follows: Sect. 2 presents related work. Section 3 contains the necessary background on spatial RDF keyword search. Section 4 formalizes the top-$k$ diversified spatial keyword search problem and introduces the general framework. Sections 5 and 6 present the IAdU and ABP algorithms. Section 7 provides a theoretical analysis of their approximation bounds. Section 8 contains our experimental evaluation. Finally, Sect. 9 concludes the paper with directions for the future work.

## 2 Related work

To the best of our knowledge, there is not any previous work on diversified spatial keyword search over RDF graphs. Hereby, we briefly discuss work related to keyword search on RDF data and (spatial) diversification and how it relates to our work.

**Keyword search on RDF data** A keyword-based retrieval model over RDF graphs, such as [18,40,48,50], identifies a set of maximal subgraphs whose vertices contain the query keywords. They follow the definition as proposed in earlier work of keyword search on graphs, [7,8,32,35,36] (which is also analogous to the definition we use in this work). Diversified keyword search on RDF graphs [9] is limited only to the diversification of results by considering the content and the structure of the results.

**Diversification** Diversification of query results has attracted a lot of attention recently as a method for improving the quality of results by balancing similarity (relevance) to a query $q$ and dissimilarity among results [12,24,25,30,52]. Diversification has also been considered in keyword search over graphs and databases, where the result is usually a subgraph that contains the set of query keywords. In conventional (non-diversified) keyword search methods, a set of results usually consists of many duplicated answers that contain the same set of nodes (i.e., nodes containing a query keyword). Thus, users are overwhelmed with many similar answers with minor differences [38]. Two recent works, PerK [47] and DivQ [15], address this problem by using Jaccard distance on the set of nodes of the results, namely by considering the common nodes. In [38], the problem of finding duplication-free answers is addressed. Liu et al. [42] developed a feature selection algorithm in order to highlight the differences among structural XML data.

**Spatial diversification** Several works consider spatial diversification, which finds results such that objects are well spread in the region of interest. In [29,37], diversity is defined as a function of the distances between pairs of objects in $R$. However, considering only the distance between a pair and disregarding their orientation could be inappropriate. In view of this, van Kreveld et al. [51] incorporate the notion of angular diversity, wherein a maximum objective function controls the size of the angle made by an object in $R$, the query location $q$, and an unselected object.

There is no previous work on spatial diversification over RDF data. Our work extends the only existing spatial RDF keyword search framework [45] to support both spatial and textual diversity. In the next section, we describe [45] in detail.

## 3 Background

An RDF knowledge base can be modeled as a directed graph, where each vertex $v$ is an entity associated with a *document*

$\psi$ containing the entity's URI, its emitting edges (i.e., predicates), and literals. An entity $p$ is called a *place vertex* or *place*, if it is associated with a spatial location. Each RDF triplet corresponds to a directed edge from an entity (subject) to another entity (object). A top-$k$ semantic place ($k$SP) query $q$ consists of three arguments: (i) the query location $q \cdot \lambda$, (ii) the query keywords $q \cdot \psi$, and (iii) the number of requested semantic places $k$.

**Definition 1 Qualifying Tree** Given a $k$SP query $q$ and an RDF graph $G = \langle V, E \rangle$, a qualifying tree $T = \langle V', E' \rangle$ is a subgraph of $G$, i.e., $V' \subseteq V$, $E' \subseteq E$, such that $T$ is rooted at a place vertex and $\cup_{v \in V'} v \cdot \psi \supseteq q \cdot \psi$.

Simply speaking, the documents of the vertices in a qualifying tree collectively cover all the query keywords. Given a $k$SP query, there may exist multiple qualifying trees with the same root $p$, but different sets of vertices. Following the existing work on keyword search over graphs [32,40], the looseness of a qualifying tree is defined as follows:

**Definition 2 Looseness** Given a qualifying tree $T = \langle V', E' \rangle$, let $d_g(p, t_i) = \min_{v \in V' \wedge t_i \in v.\psi} d(p, v)$ be the length of the shortest path from root $p$ to keyword $t_i \in q \cdot \psi$, where $d(p, v)$ is the shortest path from $p$ to $v$. The looseness of $T$ is defined as $L(T) = 1 + \sum_{t_i \in q \cdot \psi} d_g(p, t_i)$.

Looseness aggregates the proximity of the query keywords to the root of the tree. 1 is added to the sum of the paths for normalization purposes. The lower the looseness, the more relevant the root of the tree is to the vertices that cover the query keywords. Given a place vertex $p$, the *tightmost qualifying tree* (TQT) $T_p$ for the given query keywords is the qualifying tree rooted at $p$ with the minimum looseness.[1] For instance, all trees in Fig. 1 are TQTs. A $k$SP query $q$ aims at finding the $k$ places that minimize $f(L(T_p), S(p)) = \alpha \cdot L(T_p) + (1 - \alpha) \cdot S(p)$, where $T_p$ is the TQT of $p$ and $S(p)$ is the Euclidean distance between the query location and $p$. Parameter $\alpha$ is used to control the relative importance of textual relevance and spatial proximity.

Shi et al. [45] propose the basic semantic place (BSP) and semantic place retrieval with pruning (SPP) algorithms for $k$SP query processing. BSP retrieves the place vertices in the RDF graph in ascending order of their spatial distances to the query location using an R-tree [4,33]. For each retrieved place $p$, BSP computes the corresponding TQT $T_p$. TQT computation is performed by breadth-first search from $p$ until the query keywords are covered. SPP is an extension of BSP that applies two pruning techniques. The first discards *unqualified* places for which there does not exist a tree rooted at them covering all query keywords. This is achieved by a reachability index (i.e., TFlabel [11]) and a pruning rule that disregards places whose TQT cannot be constructed. The second one eliminates places by aborting their TQT computation, based on dynamically derived bounds on their looseness. The original algorithms compute and return the top-$k$ places in a batch; in our implementation, we modify them to incrementally retrieve the next place at each iteration according to its relevance score.

[1] If multiple trees rooted at $p$ have the same minimum looseness, we can: (1) select one of them at random or (2) keep all trees. If we use option (2), each place would be characterized by multiple node sets with respect to the query keywords. The proposed methods are applicable for both options. For the ease of presentation, we adopt option (1) in the rest of the paper.

## 4 $k$DSP problem definition

A top-$k$ diversified semantic place ($k$DSP) query generalizes a $k$SP query by combining a *relevance function* to the query and a *diversity function* on the set of query results that considers their relative location and content. In accordance with [45,55], we represent the RDF data in their *native graph form* (i.e., using adjacency lists) in memory. Disk-based graph representations for RDF data (e.g., [57]) can also be used for larger-scale data. At a preprocessing phase, we also perform the following. (1) We extract the document descriptions of all vertices and index them by an inverted file, which facilitates the fast search of vertices containing a given keyword. (2) For each vertex, we store in a table the document description and the spatial location (in the case of a place entity), which enables direct access to the keywords and location of a vertex during graph browsing. (3) We use an R-tree [28] to spatially index all place entities, which facilitates incremental nearest place retrieval. Section 4.1 presents the relevance function by building upon the $k$SP model of [45]. Section 4.2 introduces the diversity function, and Sect. 4.3 defines the $k$DSP problem. Table 1 contains the symbols used throughout the paper.

### 4.1 Relevance function

Consider a $k$DSP query, with location $q \cdot \lambda$ and keywords $q \cdot \psi$. Recall that for any place entity $p$, TQT $T_p$ denotes the tightmost tree rooted at $p$ that covers all query keywords $q \cdot \psi$. In the context of $k$DSP queries, we define the *looseness-based relevance* of a place $p$ as follows:

$$fL(p) = 1 - \frac{\min(L(T_p), L_{\max})}{L_{\max}}, \tag{1}$$

where $L(T_p)$ is defined according to Definition 2 and $L_{\max}$ is the maximum looseness that we can tolerate (the concept of $L_{\max}$ has been used often in earlier work, e.g., [36]). For instance, considering the example of Fig. 1, for $T_{p_1}$ we have

**Table 1** Notations

| Notation | Definition |
|---|---|
| $q$ | Query with location $q \cdot \lambda$, a set of keywords $q \cdot \psi$ and the number $k$ of requested place entities |
| $p$ | A place vertex on RDF |
| $R$ | Result of a $k$DSP query, a set of $\|R\| = k$ places (Def. 3) |
| $T_p$ | The tightmost qualifying tree (TQT) rooted at place vertex $p$ |
| $L(T_p)$ | Looseness of TQT $T_p$ rooted at $p$ (Def. 2) |
| $S(p)$ | Spatial distance between $q \cdot \lambda$ and $p$ |
| $fL(p)$ | (Normalized) Looseness-based relevance of $p$ w.r.t. query $q$ (Eq. 1) |
| $fS(p)$ | (Normalized) Spatial distance score of $p$ w.r.t. $q \cdot \lambda$ (Eq. 2) |
| $f(p)$ | Relevance score of place $p$ w.r.t. $q$ (Eq. 3) |
| $Df(p)$ | Diversity score of $p$ w.r.t. other $p'$s in $R$ (Eq. 7) |
| $HDf(p)$ | Holistic diversity and relevance function of $p$ (Eq. 8) |
| $Df(p, p')$ | Diversity between $p$ and $p'$ (Eq. 6) |
| $dL(p, p')$ | Contextual Jaccard diversity between $p$ and $p'$ |
| $dS(p, p')$ | Ptolemy's spatial diversity between $p$ and $p'$ (Eq. 5) |
| $HDf(p, p')$ | Holistic diversity between $p$ and $p'$ (Eq. 9) |
| $HDf(R)$ | Holistic diversity and relevance score of $R$ (Eq. 10) |
| $f(R)$ | Weighted summation of $f(p)$ for all $p$ in $R$ |
| $Df(R)$ | Weighted summation of $Df(p)$ for all $p$ in $R$ |
| $dL(p)$ | Summation of $dL(p, p')$ for all $p'$s in $R$, excluding $p$ |
| $dS(p)$ | Summation of $dS(p, p')$ for all $p'$s in $R$, excluding $p$ |
| $\alpha$ | Trade-off between $L(.)$ and $S(.)$ in $f(L(T_p), S(p))$ |
| $\beta$ | Trade-off between $fL(.)$ and $fS(.)$ in $f(p)$ (Eq. 3) |
| $\gamma$ | Trade-off between $dL(.)$ and $dS(.)$ in $Df(T_p, T_{p'})$ (Eq. 6) |
| $\lambda$ | Trade-off between relevance and diversity in $HDf(p)$ (and $HDf(p, p')$) (Eq. 8 (and 9)) |
| $cHDf(p)$ | The contribution of $p$ if added to $R$ (used by IAdU heuristic) |

$L(T_{p_1}) = 5$ and assuming $L_{\max} = 15$, then $fL(T_{p_1}) = 0.67$. We also define the *spatial distance score* $fS(p)$ of a place $p$ as:

$$fS(p) = 1 - \frac{\min(S(p), S_{\max})}{S_{\max}}, \tag{2}$$

where $S(p)$ is the Euclidean distance between $p$ and $q$ and $S_{\max}$ is the maximum distance that can be tolerated (e.g., the largest distance among all pairs of places in the map of a city; the concept of $S_{\max}$ has also been used in earlier work, e.g., [2]). Considering the same example for $p_1$ with $S(p_1) = 1.93$ km and $S_{\max} = 5$ km, then $fS(p_1) = 0.61$. Both relevance and distance scores range in [0, 1], which is helpful when comparing diversification scores (to be discussed shortly). The *holistic relevance* $f(p)$ of a place $p$ is:

$$f(p) = \beta \cdot fL(p) + (1 - \beta) \cdot fS(p), \tag{3}$$

where $\beta$ controls the contribution of the two relevance components ($\beta = 0$ considers only $fS(p)$ and $\beta = 1$ only $fL(p)$).

## 4.2 Diversity function

Let $T_p$ and $T_{p'}$ be the TQTs of places $p$ and $p'$. The Jaccard distance between the vertex sets of $T_p$ and $T_{p'}$ provides a simple and effective way to measure diversity of keyword search results [15,47]. Specifically, if we overload $T_p$ to denote the set of nodes in the TQT $T_p$, we can define:

$$dL(T_p, T_{p'}) = \frac{|T_p \cup T_{p'}| - |T_p \cap T_{p'}|}{|T_p \cup T_{p'}|}. \tag{4}$$

The Jaccard distance ranges in [0, 1] and satisfies triangle inequality [41] (as we discuss later, this property enables approximation bounds on the proposed algorithms). For instance, in our example of Fig. 1, the two trees $T_{p_1}$ and

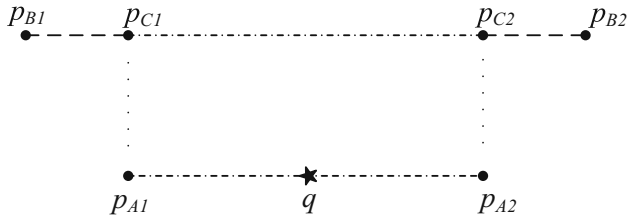



**Fig. 2** Ptolemy's Spatial Diversity $dS(p_{A1}, p_{A2}) > dS(p_{B1}, p_{B2}) > dS(p_{C1}, p_{C2})$

$T_{p_2}$ (with two common nodes) will give us $dL(T_{p_1}, T_{p_2}) = (7-2)/7 = 5/7$.

To measure geographic variety of two places $p$ and $p'$ with respect to query location $q \cdot \lambda$, we introduce *Ptolemy's spatial diversity* $dS(p, p')$ as follows:[2]

$$dS(p, p') = \frac{||p, p'||}{||p, q \cdot \lambda|| + ||p', q \cdot \lambda||}, \tag{5}$$

where $||p, p'||$ is the Euclidean distance between $p$ and $p'$. Similar to Jaccard distance, $dS(p, p')$ is naturally normalized to range [0, 1], since $||q, p|| + ||q, p'|| \geq ||p, p'||$ (triangle inequality). We illustrate other attractive properties of our spatial scattering function with the help of Fig. 2. Two places $p$ and $p'$ receive a maximum diversity score $dS(p, p') = 1$, if they are diametrically opposite to each other w.r.t. to $q \cdot \lambda$, e.g., points $p_{A1}$ and $p_{A2}$. Pair of places $(p_{C1}, p_{C2})$ have the same distance as pair $(p_{A1}, p_{A2})$, but $dS(p_{C1}, p_{C2}) < dS(p_{A1}, p_{A2})$, because $p_{C1}$ and $p_{C2}$ are in the same direction w.r.t $q$ (i.e., north of $q$). Pair $(p_{B1}, p_{B2})$ are further from each other compared to the places in pair $(p_{C1}, p_{C2})$ and consequently have a higher diversity score. (This can be shown using Pythagorean theorem.) In addition, when a place $p$ is far from $q$, the diversity score of any place pair $(p, p')$ that includes $p$ is heavily penalized, because $||p, q \cdot \lambda||$ and $||p, p'||$ become similar and dominate over $||p', q \cdot \lambda||$.

Finally, as we show in Sect. 7, this measure also satisfies the triangle inequality and helps us derive tight approximation ratios for our greedy algorithms.

Given $dL(p, p')$ and $dS(p, p')$, $Df(p, p')$ measures the total diversity between places $p$ and $p'$:

$$Df(p, p') = \gamma \cdot dL(p, p') + (1-\gamma) \cdot dS(p, p'), \tag{6}$$

where $\gamma$ controls the contribution of the two diversification components. The weighting parameters $\beta, \gamma$ can be unified to a single parameter which captures the relative importance of content and location in the computation of relevance and diversity.

The *diversity score* $Df(p)$ of $p$ in the query result $R$, containing $k$ places, is computed as:

$$Df(p) = \sum_{p' \in R, p' \neq p} Df(p, p'). \tag{7}$$

Equation 8 shows the *holistic score* $HDf(p)$ of place $p$ that combines relevance and diversity, where $\lambda$ adjusts their trade-off. A linear function and the respective trade-off $\lambda$ have been used extensively in earlier work in diversity, e.g., [52].[3] We multiply $f(p)$ by $k-1$ in order to normalize both components in the same range (since $Df(p)$ compares $p$ against the other $k-1$ elements in the result set $R$). The relevance $f(p)$ of $p$ is computed by Eq. 3.

$$HDf(p) = (1-\lambda) \cdot (k-1) \cdot f(p) + \lambda \cdot Df(p). \tag{8}$$

To simplify the presentation, we introduce the holistic diversity function of a pair of places, where we re-define our objective as:

$$HDf(p, p') = (1-\lambda) \cdot (f(p) + f(p')) + 2\lambda \cdot Df(p, p'). \tag{9}$$

$Df(p, p')$ is scaled up by a factor of 2 to balance the two values of $f(p)$ and $f(p')$. Note that computing the holistic diversity function, denoted as $HDf(R)$, of all places of a set $R$ using either Eq. 8 or Eq. 9 gives the same result:

$$HDf(R) = \sum_{p \in R} HDf(p) = \sum_{p, p' \in R, p \neq p'} HDf(p, p'). \tag{10}$$

In addition, we introduce notations $f(R)$ and $Df(R)$ for the weighted and normalized summation of $f(p)$ and $Df(p)$ scores, respectively, of all $p \in R$. Namely, $HDf(R) = f(R) + Df(R)$, where $f(R) = (1-\lambda) \cdot (k-1) \cdot \sum_{p \in R} f(p)$ and $Df(R) = \lambda \cdot \sum_{p \in R} Df(p)$. We also denote $dL(T_p)$ (as $dL(T_p) = \sum_{p' \in R, p' \neq p} dL(T_p, T_{p'})$ and analogously, $dS(p) = \sum_{p' \in R, p' \neq p} dS(p, p')$. We can easily see that $Df(p) = \gamma \cdot dL(T_p) + (1-\gamma) \cdot dS(p)$.

### 4.3 Problem definition

Finally, we can define the diverse $k$SP place ($k$DSP) retrieval problem as follows.

[2] We name this metric after the Greco-Roman mathematician Ptolemy because we later use Ptolemy's inequality to prove that it satisfies triangle inequality.

[3] We observed, by experimentation, that the default setting (i.e., $\lambda = \beta = \gamma = 0.5$) produces effective results; hence, in practice, these tuning parameters can be dropped, rendering our framework fairly simple to apply.

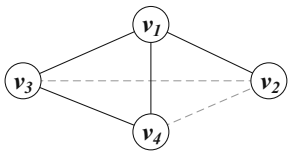
**Definition 3** $k$DSP Problem Definition. Given a query $q$ with location $q \cdot \lambda$, set of keywords $q \cdot \psi$, and an integer $k$, the $k$DSP query returns a set $R$ of $k$ place entities that have the highest $HDf(R)$ score.

Since the objective function $HDf(p)$ of a place $p$ necessitates the comparison with the other $k-1$ places of a candidate $R$ set, we have to consider all $O(n^k)$ candidate $R$ sets. This problem as proven by Theorem 1 is NP-hard. In view of this limitation, in the next sections, we propose efficient greedy algorithms with approximation guarantees. Note that the above definition is equivalent to the max-sum problem [52].

**Theorem 1** *The kDSP problem is NP-hard.*

***Proof*** In order to prove the hardness of $k$DSP, we construct a reduction from the clique problem: given an undirected graph $G(V, E)$ and a positive integer $k$, $(k \leq |V|)$, the decision problem is to answer if $G$ contains a clique of size $k$. We start the reduction, by creating the complementary graph $G'(V, E')$ of $G$, where $E'$ contains the edges not present in $E$, i.e., for each pair of vertices $v_i, v_j$, edge $(v_i, v_j) \in E'$ iff $(v_i, v_j) \notin E$. Then, we generate an instance of $k$DSP as follows. Each vertex $v_i$ in $V$ corresponds to a place $p_i$ that has a TQT $T_{p_i}$, rooted at node $p_i$ of the RDF graph. For every edge $(v_i, v_j)$ in $E'$, we add node $v_{i,j}$ as a child of roots $p_i$ and $p_j$ in the TQT $T_{p_i}$ and $T_{p_j}$, respectively. This reduction takes polynomial time, since the cost is $O(1)$ per edge, and the number of edges is $O(|V|^2)$. After generating the TQTs, we set $\lambda = 1$ (i.e., we disregard relevance) and $\gamma = 1$ (i.e., we disregard Ptolemy's diversity) and construct a $k$DSP query, such that, based on the query location and keywords, (i) the places retrieved are those corresponding to the vertices of $G$ and (ii) the TQTs of the places are exactly those defined above. Then, the original graph $G$ contains a clique of size $k$, iff there is a $k$DSP result $R$ with holistic diversity function $HDf(R) = k \cdot (k-1)$.

To explain this, assume that there is a clique of $k$ vertices $v_1, \ldots v_k$ in $G$. Consider a $k$DSP result that contains the $k$ corresponding places $R = \{p_1, \ldots p_k\}$. Since there is no edge connecting vertices $(v_i, v_j)$ in $G'$, each pair $(T_{p_i}, T_{p_j})$ of TQTs have zero overlap, and their contextual diversity (Eq. 4) is $dL(T_{p_i}, T_{p_j}) = 1$. For $\gamma = 1$, the total diversity between two places equals their contextual diversity, i.e., $Df(T_{p_i}, T_{p_j}) = dL(T_{p_i}, T_{p_j}) = 1$. Based on Eq. 9, for $\lambda = 1$, the holistic diversity of a pair $(p_i, p_j)$ of places becomes $HDf(p_i, p_j) = 2 \cdot Df(T_{p_i}, T_{p_j}) = 2$. Finally, according to Eq. 10, the holistic diversity of the result $R$ is the total diversity for the $k \cdot (k-1)/2$ distinct pairs of places in $R$, i.e., $HDf(R) = k \cdot (k-1)$. Conversely, if there is no clique of size $k$ in $G$, any result $R$ of $k$ places in $k$DSP has holistic diversity $HDf(R) < k \cdot (k-1)$. This is because there is at least a pair of places $(p_i, p_j)$ in $R$, whose corresponding vertices $(v_i, v_j)$ are connected in $G'$. Subsequently, there is a common node $v_{i,j}$ in trees $T_{p_i}$ and $T_{p_j}$. Thus, based on Eq. 4, their contextual diversity is $dL(T_{p_i}, T_{p_j}) < 1$, and the holistic diversity of all the pairs in $R$ cannot reach $k \cdot (k-1)$. This completes the proof. □




**(a)** Example Graph $G$

$T_{p_1} = \{p_1\}$
$T_{p_2} = \{p_2, v_{2,3}, v_{2,4}\}$
$T_{p_3} = \{p_3, v_{2,3}\}$
$T_{p_4} = \{p_4, v_{2,4}\}$

**(b)** Corresponding $T_p$ Trees

**Fig. 3** Example of reduction

Figure 3b shows (as sets of nodes) the resulting TQTs for the input graph of Fig. 3a. The gray dashed lines represent edges of $G'$. Each of these edges (e.g., $(v_2, v_4)$) adds the same node (e.g., $v_{2,4}$) under the roots of the corresponding trees (e.g., $T_{p_2}$ and $T_{p_4}$). The holistic diversity of the 4DSP query containing all four places is: $2 \cdot (Df(T_{p_1}, T_{p_2}) + Df(T_{p_1}, T_{p_3}) + Df(T_{p_1}, T_{p_4}) + Df(T_{p_2}, T_{p_3}) + Df(T_{p_2}, T_{p_4}) + Df(T_{p_3}, T_{p_4})) = 2(1 + 1 + 1 + 0.75 + 0.75 + 1) = 11 < 4 \cdot 3$. Thus, we can state that there is no clique of size 4 in $G$. On the other hand, the result $R = \{p_1, p_3, p_4\}$ of the 3DSP query has holistic diversity: $2 \cdot (Df(T_{p_1}, T_{p_3}) + Df(T_{p_1}, T_{p_4}) + Df(T_{p_3}, T_{p_4})) = 2 \cdot (1 + 1 + 1) = 3 \cdot 2$. Consequently, there is a clique involving $\{v_1, v_3, v_4\}$ in $G$. In general, any result of $k$ places with score $k \cdot (k-1)$ corresponds to a clique of size $k$.

As a final note, we can easily construct the TQTs, shown in the example, as follows. We consider as many query keywords as the maximum degree of a vertex in $G'$ (i.e., 2 keywords $w_1$ and $w_2$ in this example). Then, we assume that each node added to the trees contains one of the keywords (e.g., $v_{2,3}$ contains $w_1$ and $v_{2,4}$ contains $w_2$). For every vertex with the highest degree in $G'$, the root node (e.g., $p_2$) of the corresponding TQT does not contain any of the query keywords. For each of the other vertices, the root node contains the keywords that are not covered by the non-root nodes (e.g., $p_3$ contains $w_2$ and $p_1$ contains both keywords). Again, the construction can be done in PTIME.

## 5 Incremental addition and update (IAdU) algorithm

We apply a greedy heuristic in a combination with a branch-and-bound approach that can be injected to any $k$SP algorithm (e.g., BSP, SPP) [45]. The heuristic iteratively constructs the result set $R$ by selecting a new place entity $p$ that maximizes the contribution it can make toward the overall score $HDf(R)$. The contribution $cHDf(p)$ of a

**Algorithm 1** The IAdU Algorithm

***IAdU*** $(q)$
1: $R = \emptyset$; MaxHeap $H = \emptyset$, ordered by $cHDf(\cdot)$
2: $\theta = \infty$
3: **repeat**
4: **if** $(max(H) \geq \theta)$ **then**
5: $curP = H.deHeap()$
6: add $curP$ to $R$
7: **for** each $p$ in $H$ **do**
8: $cHDf(p)$ += $HDf(p, curP)$
9: $\theta = (1 - \lambda) \cdot (\sum_{p' \in R} f(p') + |R| \cdot f_{min}) + 2\lambda \cdot |R|$
10: **else**
11: Get next $curP$ using a $kSP$ algorithms (e.g., BSP, SPP or SP); $f_{min} = f(curP)$
12: **if** $(R == \emptyset)$ **then**
13: $cHDf(curP) = f(curP)$
14: addOnR($curP$)
15: **else**
16: **for** each $p$ in $R$ **do**
17: $cHDf(curP) += HDf(p, curP)$
18: $H.add(curP, cHDf(curP))$
19: $\theta = (1 - \lambda) \cdot (\sum_{p' \in R} f(p') + |R| \cdot f_{min}) + 2\lambda \cdot |R|$
20: **until** $|R| = k$
21: **return** $R$

$p$ to be added to the current result set $R$ is defined as follows:

$$cHDf(p) = \begin{cases} f(p), & \text{if } R = \emptyset, \\ \sum_{p' \in R} HDf(p, p'), & \text{otherwise.} \end{cases} \quad (11)$$

$cHDf(p)$ considers the $f(p)$ score and also the diversity of $p$ against the existing elements in $R$. In the first iteration, $R$ is empty; thus, the available contribution of a place can only be the corresponding $f(p)$ score. The contributions of all other places are then updated to reflect the new entry in $R$. Then, the algorithm iteratively selects the place that maximizes $cHDf(p)$ to $R$, adds it to $R$, and updates the score of the unselected places. The population of all valid places can be prohibitively large and expensive to calculate. Thus, we employ a branch-and-bound paradigm that incrementally generates and processes places in combination with a threshold. More precisely, we reuse $kSP$ algorithms to incrementally retrieve places in descending order of their $f(\cdot)$ scores. Note that the $kSP$ algorithms of [45] do not produce results incrementally but return the top-$k$ results as a batch; still, we can easily modify them to generate results incrementally. (More precisely, we can use a revised threshold that facilitates the output of the current largest result.) In summary, we have to update $cHDf(p)$ scores in two cases: (1) when a place is added to $R$, where we need to update the score of all seen elements and (2) when a new place is emerged from our $kSP$ algorithms, where we need to calculate its diversity score against all elements in $R$. Finally, IAdU algorithm uses a threshold, $\theta$, that facilitates the pruning of unseen places if they cannot qualify in $R$.

| $curP$ | $H$ | $\theta$ | $R$ |
|---|---|---|---|
| $p_6$, 0.9 | | | $p_6$ |
| $p_3$, 0.8 | $(p_3, 3.31)$ | 3.7 | |
| $p_2$, 0.6 | $(p_3, 3.31)$, $(p_2, 2.8)$ | 3.5 | |
| $p_1$, 0.5 | $(p_3, 3.31)$, $(p_1, 3.19)$, $(p_2, 2.8)$ | 3.4 | |
| $p_4$, 0.4 | ~~$(p_3, 3.31)$~~, $(p_4, 3.22)$, $(p_1, 3.19)$, $(p_2, 2.8)$ | 3.3 | $p_6$, $p_3$ |
| $p_5$, 0.3 | ~~$(p_1, 6.4)$~~, $(p_5, 6.31)$ $(p_4, 6.02)$, $(p_2, 5.5)$ | 6.3 | $p_6$, $p_3$, $p_1$ |
| $p_7$, 0.1 | ~~$(p_2, 8.71)$~~, $(p_5, 7.31)$ $(p_4, 7.02)$, $(p_7, 6.5)$ | 6.5 | $p_6$, $p_3$, $p_1$, $p_2$ |

**Fig. 4** Example of the IAdU algorithm

Algorithm 1 illustrates the pseudo-code of the IAdU Algorithm. A max heap $H$ maintains seen places according to their $cHDf(\cdot)$ values and is initially set to null (line 1). In addition, a threshold of $cHDf(\cdot)$ score, $\theta$, of all unseen places is also maintained and is initially set to $\infty$ (line 2). The algorithm starts by adding the place $p$ with the largest $f(p)$ score on $R$ (obtained by $kSP$ algorithms; lines 12–14). During the next iterations, new places are added to the heap $H$ according to their $cHDf(\cdot)$ score (lines 15–19) which is calculated against places already in $R$ (lines 16, 17). The threshold $\theta$ is updated accordingly (line 19). More precisely, $\theta$ considers the minimum value $f(\cdot)$ of a seen place and the maximum diversity (i.e., 1) against all elements in $R$. If the top place on the heap $H$, which has the largest score, has a score greater than the current threshold, then this place will have the next largest score. Thus, we de-heap it and add it on $R$ (lines 4–6). Due to the new addition on $R$, we need to update accordingly the $cHDf(\cdot)$ score of all places still in $H$ (lines 7 and 8) and the threshold of unseen places (line 9). The algorithm terminates when the size of $R$ becomes $k$ (line 20).

**Example** We demonstrate the IAdU algorithm with the example in Fig. 4 for $k = 4$. In Fig. 4, we show the current value of (i) $curP$ (with its respective $f(p)$), (ii) heap $H$, (iii) $\theta$ and (iv) $R$. At first, the place with the largest $f(.)$ score is added on $R$ (i.e., $p_6$). Then, iteratively, we retrieve and compute the scores of the next places that are retrieved by the $k$SP algorithm and add them to the heap (according to their $HDf(,)$ values against $p_6$, which is currently the only element in $R$) and also update the threshold. After processing $p_4$, $max(H)$ (corresponding to $p_3$) becomes larger than the threshold; thus, $p_3$ is de-heaped and added on $R$. Then, all elements in $H$ and $\theta$ are updated accordingly. We repeat this until we obtain the 4 results.

**Complexity**. The running time of the algorithm is dominated by the retrieval of the necessary places, in increasing order of their relevance, using the $k$SP algorithm, until the result

**Algorithm 2** The ABP Algorithm

***ABP*** $(q)$

1: $R = \emptyset$
2: $\theta = \infty$
3: $F_{List} = \emptyset$
4: **repeat**
5: **if** $(|F_{List}| < 2) \wedge (max(maxPairH) \geq \theta)$ **then**
6: $(p_i, p_j) = maxPairH.deHeap()$
7: add $p_i$ and $p_j$ to $R$
8: **if** $(|R| < \lfloor k/2 \rfloor)$ **then**
9: delete $PairsH[p_i]$, $PairsH[p_j]$
10: delete $p_i$ and $p_j$ from $F_{List}$
11: $\theta = (1-\lambda) \cdot (max(F_{List}) + last(F_{List})) + 2 \cdot \lambda$
12: **Update(***maxPairH***)**
13: **else**
14: Get next $p$ using a $kSP$ algorithm (e.g., BSP, SPP)
15: add $p$ to $F_{List}$
16: **for** each $p'$ in $F_{List}$ **do**
17: $HDf(p, p') = (1-\lambda) \cdot (f(p) + f(p')) + 2\lambda \cdot Df(p, p')$
18: add pair $(p, p')$ to $PairsH[p]$
19: deheap and add top pair of $PairsH[p]$ on $maxPairH$
20: $\theta = (1-\lambda) \cdot (max(F_{List}) + last(F_{List})) + 2 \cdot \lambda$
21: **until** $|R| \geq \lfloor k/2 \rfloor$
22: **if** $k$ is odd **then**
23: add an arbitrary place from $F_{List}$ to $R$
24: **return** $R$

***Update(maxPairH)***

1: **while** $(max(maxPairH) \cap R \neq \emptyset)$ **do**
2: $(p_i, p_j) = maxPairH.deHeap()$
3: **if** $(p_i \cap R == \emptyset)$ **then**
4: **repeat**
5: $(p_i, p_l) = PairsH[p_i].deHeap()$
6: **if** ($p_l$ not in $R$) **then**
7: add $(p_i, p_l)$ on $maxPairH$
8: **until** $(p_l \cap R == \emptyset) \vee (|PairsH[p_i]| == 0)$

is finalized. Let $K$ be the number of these places; the cost of their retrieval is $O(K \cdot kSPT)$, where $kSPT$ is the time required to generate a result incrementally using a given $kSP$ algorithm. An additional cost is due to the updates to the contributions of the retrieved places and heap updates due to emergence of new results. This cost is bounded by the $K \cdot k$ heap updates, i.e., it is $O(K \cdot k \cdot logK)$. Finally, we have to add the cost for updating the contributions which is $O(K^2)$ because for all pairs $(p, p')$ of retrieved places we have to compute $HDf(p, p')$. Thus, the complexity of the algorithm is $O(K \cdot (kSPT + k \cdot logK + K))$.

## 6 Add best pairs (ABP) algorithm

The add best pairs (ABP) algorithm greedily constructs the result set $R$ by iteratively selecting the pair of places $(p, p')$ with the best $HDf(p, p')$ score. As opposed to IAdU, which selects the next place by considering its diversity to the places already selected, ABP selects the next pair $(p, p')$ based on only its $HDf(p, p')$ value, independently of $p$ or $p'$'s diversity to places already in $R$. Once a pair is selected, both its constituent elements and any pairs they make are removed from further consideration by the algorithm (in a *lazy fashion*). Since a single pair is selected in each iteration, $\lfloor k/2 \rfloor$ iterations apply when the value of $k$ is even. When $k$ is odd, an arbitrary place is chosen to be inserted in the result set $R$ as its last entity.

In a nutshell, ABP efficiently implements this heuristic by iteratively processing the places and by managing (1) a ranked list, $F_{List}$, of places in descending order of their $f(p)$ scores, (2) one max heap $PairsH(p)$ *for each* $p$ in $F_{List}$, containing pairs $(p, p')$ with previously seen places $p'$ (organized by their $HDf(p, p')$ score), (3) a max heap, $maxPairH$, which organizes all top elements of the $PairsH(p)$'s. Finally, we maintain (4) a threshold $\theta$, which helps us to check whether the top pair on $maxPairH$ is guaranteed to be the one with the highest $HDf(\cdot, \cdot)$ among pairs which have not been added to the result $R$ yet.

Algorithm 2 illustrates the pseudo-code of the algorithm. We start with an empty result set $R$, an empty list $F_{List}$ of retrieved places, and a threshold $\theta = \infty$. Initially, lines 14–20 retrieve the two most relevant places, say $p_1$ and $p_2$, to $q$ using the $k$SP algorithm and add them to $F_{List}$. $PairsH[p_1]$ is an empty heap because there is no place in $F_{List}$ before $p_1$. After obtaining the second place $p = p_2$, then we add on $PairsH[p]$, the first pair $(p = p_2, p_i = p_1)$, which becomes the top pair in all $PairsH$'s, so it is then de-heaped and added to $maxPairH$.

Then, the $\theta$ threshold takes as value the best possible case for a place pair which consists of the next place (not retrieved yet) and one of the places in $F_{List}$ (line 20). This case happens, when the place in $F_{List}$ with the maximum relevance (denoted by $max(F_{List})$) is combined with a place from those not seen yet having the maximum possible relevance. This place should have the same relevance (at most) as the relevance of the last accessed place by the $k$SP algorithm (denoted by $last(F_{List})$). At the same time, these two places should have the maximum possible diversity score (i.e., 2).

As soon as there are at least two places in $F_{List}$, the algorithm can check at line 5 whether the top pair in $maxPairH$ has a score greater than $\theta$. If so, this pair $(p_i, p_j)$ is de-heaped from $maxPairH$ and added to $R$. Then, the heap $PairsH[p_i]$, which keeps track of the pairs that include $p_i$ and other places in $F_{List}$ is deleted, since it will not be of any further use. The same happens with $PairsH[p_j]$. $p_i$ and $p_j$ are also removed from $F_{List}$. Then, the threshold $\theta$ is updated to reflect these changes. Finally, $maxPairH$ is updated in a lazy fashion in order for the currently best pair on the top of this heap to be a valid pair (function ***Update()***).

Specifically, in the *Update()* function, while the top pair of $maxPairH$ includes a place in $R$ (this place is already selected and cannot be selected again), then we de-heap this

pair $(p_i, p_j)$. If the place in this pair which is not yet in $R$ is $p_i$, we then have to replace it with a new pair from $PairsH[p_i]$, so that the top of this heap is not a pair $(p_i, p_l)$ for which $p_l \in R$. Hence, while for the top pair $(p_i, p_l)$, we have $p_l \in R$, we keep de-heaping $(p_i, p_l)$. If, in the end, for the top $(p_i, p_l)$ of $PairsH[p_i]$, $p_l$ is not in $R$, we add $(p_i, p_l)$ on $maxPairH$ and stop the updates. If $PairsH[p_i]$ becomes empty, then a replacement from it is not achievable. Then, we iteratively repeat this process for the new top of $maxPairH$ (line 1); namely, we check if the top pair intersects with $R$ and if so repeat the process until the current top pair does not.

In the main algorithm, if the top of $maxPairH$ does not have a score better than $\theta$ (line 5), then ABP retrieves the next place $p$ using the $k$SP algorithm and creates the corresponding $PairsH[p]$ by comparing $p$ with all places in $F_{List}$ (lines 14–18). $maxPairH$ is also updated to include the top of $PairsH[p]$, i.e., the best pair that includes $p$ (line 19). The threshold $\theta$ is then decreased (line 20) because the last relevance score $last(F_{List})$ is updated to the relevance score of $p$ (which is smaller compared to the places retrieved before $p$). Hence, potentially at the next iteration, the condition of line 5 becomes true due to (i) the decrease in $\theta$, (ii) the new addition to $maxPairH$, which may increase the score of its top element.

ABP terminates as soon as $\lfloor k/2 \rfloor$ pairs are added to $R$. If $k$ is even, then we are done; if $k$ is odd, ABP adds one arbitrary place from $F_{List}$ to $R$, e.g., the one with the maximum relevance.

Note that each heap, $PairsH[p]$, includes all pairs of $p$ having as elements previously seen places in $F_{List}$. Thus, each pair can appear only once in the list of these heaps, i.e., on the heap of the constituent element with the smallest $f(.)$ score. During the execution of the algorithm, if a pair is added on $R$, then the two respective heaps are deleted for further consideration. However, still other heaps may contain pairs that contain the two newly added to $R$ places. We apply a lazy approach in managing these pairs. More precisely, we only delete a pair that includes an added place, if it is de-heaped from the heap (as it is on the top of the heap). Therefore, a heap may still include pairs with many elements that have already been added on $R$. Similarly, on $maxPairH$, we apply a *lazy approach* to manage this heap by ensuring only that the top pair includes elements that are not in $R$. Namely, the heap may still include pairs (other than the top) containing elements that have already been added on $R$. We only de-heap as to disregard a pair, when it is on the top of the heap. In such cases, we replace it with the next pair from the respective $PairsH[.]$ heap.

**Example** The example of Fig. 5 illustrates ABP for $k = 4$ (i.e., 2 pairs). We iteratively retrieve places and their respective $f(p)$ scores and add them on $F_{List}$. In Fig. 5a, the pair

| $F_{List}$ | $PairsH$ | $\theta$ | $maxPairH$ | $R$ |
|---|---|---|---|---|
| $p_6$, 0.9 | | | ~~$(p_3,p_6,$ 3.31)~~ | $p_3$,$p_6$ |
| $p_3$, 0.8 | ~~$(p_3,p_6,$ 3.31)~~ | 3.7 | $(p_4,p_6,$ 3.22) | |
| $p_2$, 0.6 | ~~$(p_2,p_6,$ 2.8)~~, $(p_2,p_3,$ 2.7) | 3.5 | $(p_1,p_3,$ 3.21) | |
| $p_1$, 0.5 | ~~$(p_1,p_3,$ 3.21)~~, $(p_1,p_2,$ 3.21), $(p_1,p_6,$ 3.19) | 3.4 | $(p_2,p_6,$ 2.8) | |
| $p_4$, 0.4 | ~~$(p_4,p_6,$ 3.22)~~, $(p_4,p_3,$ 2.8), $(p_4,p_2,$ 1.2), $(p_4,p_1,$ 1) | **3.3** | | |

**(a)**

| $F_{List}$ | $PairsH$ | $\theta$ | $maxPairH$ | $R$ |
|---|---|---|---|---|
| $p_2$, 0.6 | $(p_2,p_3,$ 2.7) | | ~~$(p_1,p_3,$ 3.21)~~ ~~$(p_1,p_2,$ 3.21)~~ | $p_3$, $p_6$ |
| $p_1$, 0.5 | ~~$(p_1,p_2,$ 3.21)~~, $(p_1,p_6,$ 3.19) | | $(p_2,p_6,$ 2.8) | $p_3$, $p_6$, $p_1$, $p_2$ |
| $p_4$, 0.4 | ~~$(p_4,p_3,$ 2.8)~~, ~~$(p_4,p_2,$ 1.2)~~, $(p_4,p_1,$ 1) | | $(p_4,p_2,$ 1.2) | |
| $p_5$, 0.3 | ~~$(p_5,p_4,$ 1.1)~~, $(p_5,p_1,$ 1), $(p_5,p_2,$ 1) | **2.9** | $(p_5,p_4,$ 1.1) | |

**(b)**

**Fig. 5** Example of the ABP algorithm

$(p_3, p_6)$ is first de-heaped and added on $R$, since it qualifies the threshold. However, note that $p_6$ and $p_3$ may still appear in other pairs of the heap, i.e., in pairs $(p_4, p_6)$, $(p_1, p_3)$, and $(p_2, p_6)$. ABP deletes such pairs in a lazy fashion when needed. Hence, the next top pair $(p_4, p_6)$ to $(p_3, p_6)$ is disregarded (indicated with strike-through format). Then, ABP de-heaps the next pair from the respective $PairsH$ heap ($PairsH[p_4]$), which is $(p_4, p_3)$. We disregard this pair as well, since $p_3$ is already in $R$. Then, we de-heap pair $(p_4, p_2)$ and add on $maxPairH$ (Fig. 5b). Now, $(p_1, p_3)$ becomes the top of the heap which also needs to be replaced with $(p_1, p_2)$; $(p_1, p_2)$ becomes the top of the heap. $\theta$ in Fig. 5a is defined based on $p_6$ and $p_4$ as they represent the first and last elements in the $F_{List}$. In Fig. 5b, $\theta$ will be based on $p_2$ and $p_5$, as they represent the current first and last elements of the list.

**Complexity** The running time of the algorithm is dominated by the retrieval cost of the places by the $k$SP module and the management of the list of heaps $PairH[]$ and $maxPairH$. The former (similar to the IAdU Algorithm) is $O(K \cdot kSPT)$ time where $kSPT$ is the cost of retrieving a place and $K$ is the number of places that have to be retrieved until ABP's termination. Each $PairH[p]$ heap has a maximum size $K$. Since each of the $K^2$ pairs can appear only once in all heaps, the total time managing all these heaps is bounded $O(K^2 \cdot \log(K^2))$. The $maxPairH$ heap also has maximum size of

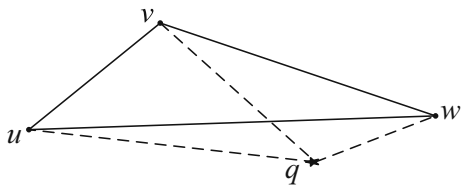



**Fig. 6** Ptolemy's spatial diversity (Lemma 1)

$K^2$, since we may have to en-heap all pairs from $PairH[p]$ heaps, and the management of $maxPairH$ also costs $O(K^2 \cdot \log(K^2))$. Hence, the worst case complexity of the algorithm is $O(K \cdot (kSPT + K \cdot \log(K^2)))$. Hence, the complexity of this algorithm is higher than that of IAdU. In practice, both algorithms perform well, because $K$ is relatively small. Their costs are dominated by the $kSPT$ factor, which is typically larger that the other ones.

## 7 Theoretical analysis

In this section, we analyze the approximation bounds of our algorithms (IAdU and ABP). We first deduce some preliminary results from our problem formulation, which enable us to achieve tight bounds for the algorithms. Our proof is based on two key observations: (i) for a result set $R$, the summation of the $HDf(u, v)$ values of all pairs $(u, v) \in R$ equals $HDf(R)$ (Eq. 10) and (ii) $HDf(u, v)$ satisfies the triangle inequality (proved below).

### 7.1 Preliminary results

**Lemma 1** *Given $u, v, w \in V$, the Ptolemy's spatial diversity $dS(u, v)$ satisfies triangle inequality, as given by*

$$dS(u, v) + dS(v, w) \geq dS(u, w).$$

***Proof*** We use Fig. 6 to support our construction. By definition of $dS(u, v)$, the inequality in Lemma 1 can be rewritten as

$$\frac{\|u, v\|}{\|q, u\| + \|q, v\|} + \frac{\|v, w\|}{\|q, v\| + \|q, w\|} \geq \frac{\|u, w\|}{\|q, u\| + \|q, w\|}.$$

By multiplying both sides of the inequality by $(\|q, u\| + \|q, v\|)(\|q, u\| + \|q, w\|)(\|q, v\| + \|q, w\|)$, we get

$$\begin{aligned}&(\|q, u\| + \|q, w\|)(\|q, u\| \cdot \|v, w\| \\ &\quad + \|u, v\| \cdot \|q, v\| + \|v, w\| \cdot \|q, v\| + \|u, v\| \cdot \|q, w\|) \\ &\quad - \|u, w\|\,(\|q, u\| + \|q, v\|)(\|q, v\| + \|q, w\|) \geq 0.\end{aligned} \tag{12}$$

For the rest of the proof, we assume that $\|u, w\| \neq 0$, since when $\|u, w\| = 0$, the inequality in question becomes obvious. To prove Eq. 12, we make use of Ptolemy's inequality [3], which gives a relationship between the side lengths and the diagonals of a quadrilateral with vertices $q$,$u$,$v$, and $w$ as

$$\|u, v\| \cdot \|q, w\| + \|q, u\| \cdot \|v, w\| \geq \|u, w\| \cdot \|q,\ v\|. \tag{13}$$

We now consider Eq. 13 for two cases:

(i) $\|q, v\| = 0$, or
(ii) $\|q, v\| = \frac{\|u,v\| \cdot \|q,w\| + \|q,u\| \cdot \|v,w\|)}{\|u,w\|}$.

Under condition (i), Eq. 12 reduces to:

$$\begin{aligned}&\|q, u\|^2 \cdot \|v, w\| + \|q, u\| \cdot \|q, w\| \\ &\quad \times (\|u, v\| + \|v, w\| - \|u, w\|) \\ &\quad + \|u, v\| \cdot \|u, w\|^2 \geq 0.\end{aligned} \tag{14}$$

Under condition (ii), Eq. 12 is simplified to:

$$\begin{aligned}&\|q, u\| \cdot \|q, k\|\,((\|u, v\| + \|v, w\|)^2 - \|u, w\|^2) \\ &\quad + \|u, v\| \cdot \|v, w\|\,(\|q, u\| - \|q, w\|)^2 \geq 0.\end{aligned} \tag{15}$$

Both Eqs. 14 and 15 hold as a consequence of the triangle inequality on $u$, $v$, $w$, i.e., $\|u, v\| + \|v, w\| - \|u, w\| \geq 0$. This completes the proof. □

**Lemma 2** *Given $u, v, w \in V$, the diversification function $Df(u, v)$ (where $Df(u, v) = dL(u, v) + dS(u, v)$) satisfies triangle inequality, as given by*

$$Df(u, v) + Df(v, w) \geq Df(u, w).$$

***Proof*** By definition of $Df(u, v)$, the inequality can be rewritten as: $(dL(u, v) + dS(u, v)) + (dL(v, w) + dS(v, w)) \geq (dL(u, w) + dS(u, w))$. From Lemma 1, we know that $dS(u, v)$ satisfies the inequality. Additionally, Levandowsky and Winter [41] have shown that Jaccard distance is also a metric, and hence, it satisfies the triangle inequality, which means that $dL(v, w)$ (defined as the Jaccard distance between two node sets) also satisfies the inequality. Thus, the two inequalities satisfied are:

$$\begin{aligned}&dL(u, v) + dL(v, w) \geq dL(u, w) \\ &dS(u, v) + dS(v, w) \geq dS(u, w).\end{aligned} \tag{16}$$

The addition of these equations completes the proof. □

In general, the diversity function $Df(u, v)$ maintains its triangle inequality properties as long as the constituent components follow triangle inequality.

**Theorem 2** *Given $u, v, w \in V$, $HDf(u, v)$ (Eq. 9) satisfies triangle inequality, i.e.,*

$$HDf(u, v) + HDf(v, w) \geq HDf(u, w).$$

***Proof*** By expanding $HDf(u, v)$ according to its definition, we get: $(f(u) + f(v) + 2 \cdot Df(u, v)) + (f(v) + f(w) + 2 \cdot Df(v, w)) \geq (f(u) + f(w) + 2 \cdot Df(u, w)) \implies 2 \cdot f(v) + 2 \cdot Df(u, v) + 2 \cdot Df(v, w) \geq 2 \cdot Df(u, w) \implies$

$$f(v) + (Df(u, v) + Df(v, w) - Df(u, w)) \geq 0. \quad (17)$$

From Lemma 2, we know that $Df(u, v) + Df(v, w) \geq Df(u, w)$, and hence, Eq. 17 holds. □

Note that the triangle inequality property of $HDf(u, v)$ is independent to $f(.)$. Namely, $f(.)$ does not need to satisfy the triangle inequality and can have an arbitrary value. It is easy to see that the triangle inequality property of $HDf(u, v)$ is independent to the value of the tuning parameters (i.e., $\beta$, $\gamma$, and $\lambda$).

## 7.2 Approximation bounds

**Theorem 3** *IAdU algorithm achieves an approximation ratio of 4.*

***Proof*** Consider a complete undirected graph, where each node $u$ corresponds to a place entity and an edge (e.g., $e(u, v)$) corresponds to the distance between the corresponding pair of places. More precisely, each edge has its $HDf(u, v)$ value as edge weight. IAdU selects, at every iteration, the place $u$ that has the maximum available contribution $cHDf(u)$. The heuristic used in IAdU is similar to the one proposed by Ravi et al. [44]. The difference lies in the selection of its first edge, i.e., the first step. In particular, they first (i) select the pair of nodes with the maximum pairwise distance in the entire graph and (ii) complete the remaining top-$k$ result set by successively selecting the next element that maximizes the distance to the set of already selected elements. They prove by mathematical induction that their greedy heuristic achieves an approximation of 4.

To achieve our desired analysis, we show a different deduction for the base case of the inductive step. For the base case of $k = 1$ IAdU adds the node with the highest $f(u)$ score in $V$. We can easily see that this constitutes the optimal result for $k = 1$. With the second addition, IAdU adds a new node $v$ that participates in an edge with $u$ and has the maximum $HDf(u, v)$ score. We prove below that the value of $HDf(u, v)$ is at least half the maximum value of $HDf(u, v)$ in the optimal solution for $k = 2$. □

**Lemma 3** *The first two nodes added by IAdU form an edge with a score at least half of the maximum score of all edges in the complete graph.*

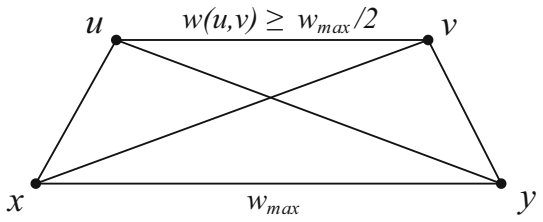



**Fig. 7** Case 3 of the inductive step of $k = 2$ (IAdU algorithm)

***Proof*** Let us assume that the optimal solution for $k = 2$ (i.e., the best edge) is the edge $e = (x, y)$ and has a weight of $w_{\max}$. Now consider the greedy selection of IAdU of edge $(u, v)$ in the first two iterations, where node $u$ is selected before node $v$. Three cases arise:

**Case 1** $|\{u, v\} \cap e| = 2$ (namely, the selected edge is the optimal edge $e$). Trivial. The optimal edge is selected.

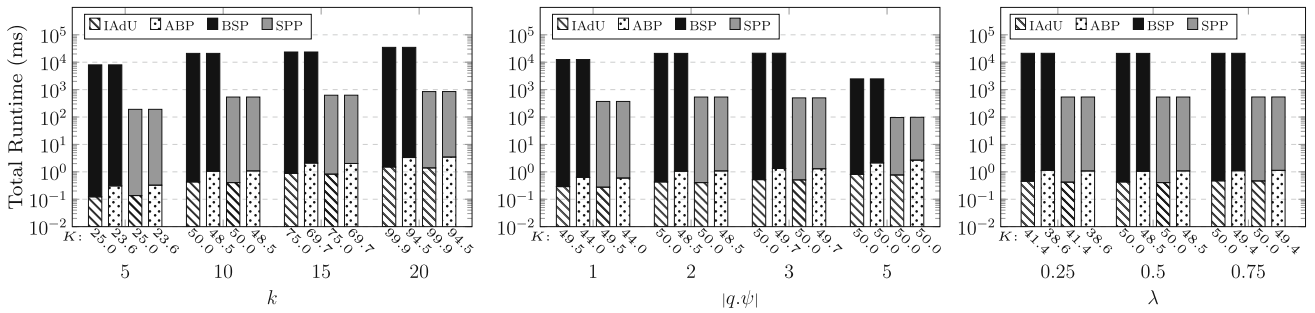
**Case 2** $|\{u, v\} \cap e| = 1$ (i.e., only one node of the selected edge belongs to the optimal edge). W.l.o.g. assume that $u = x$. For the subcase (a), node $u$ is selected before $v$; then, the optimal edge will be selected by the greedy heuristic of IAdU. For the subcase (b), where $v$ is selected first, it is easy to see that between edges $(v, x)$ and $(v, y)$, the one with the larger weight is selected. This edge's weight is at least $w_{\max}/2$ according to the triangle inequality between $v$, $x$, and $y$.

**Case 3** $|\{u, v\} \cap e| = 0$ (i.e., none of the selected edge's nodes belongs to the optimal edge). We use Fig. 7 to illustrate this case. W.l.o.g. assume that node $u$ was selected first by IAdU. By the triangle inequality, we have $w(u, x) + w(u, y) \geq w(x, y) = w_{\max}$. And due to the greedy selection of IAdU, $w(u, v) > w(u, x), w(u, y)$. Hence, $w(u, v) \geq w_{\max}/2$.

This completes the inductive step for $k = 2$. The rest of the proof (i.e., the inductive case for $k > 2$) follows directly from [44]. □

**Theorem 4** *ABP algorithm achieves an approximation ratio of 2.*

***Proof*** We make the same mapping as above, namely we have a complete undirected graph, where each node corresponds to a TQT and caries an $f(u)$ score, and each edge $(u, v)$ carries its $HDf(u, v)$ score as edge weight. The greedy heuristic in [31] achieves the approximation ratio of 2 in the case where edge weights satisfy the triangle inequality. This heuristic chooses in every iteration a new pair of elements that has the maximum pairwise distance. In ABP, we efficiently implement the same heuristic under the suggested mapping, wherein at each iteration we add to $R$ the edge $(u, v)$ with the current maximum $HDf(u, v)$ score in $G$ and then delete this edge and all its adjacent edges from $G$. We continue until we get the $k/2$ edges. Since $HDf(u, v)$ satis-




**(a)** Varying $k$ ($|q.\psi| = 2$, $\lambda = \beta = \gamma = 0.5$) **(b)** Varying $|q.\psi|$ ($k = 10$, $\lambda = \beta = \gamma = 0.5$) **(c)** Varying $\lambda$ ($|q.\psi|$=2,$k$=10,$\beta$=$\gamma$=0.5)

**Fig. 8** Efficiency on DBpedia

fies the triangle inequality, it follows that ABP also achieves an approximation of 2. □

## 8 Experiments

We evaluate the efficiency of the proposed greedy algorithms and the effectiveness (and approximation quality) of the proposed $k$DSP framework against the $k$SP framework of [45]. Finally, we present a user evaluation comparing $k$SP and $k$DSP results.

### 8.1 Setup

**Datasets** We used the datasets and settings that have been used in [45], namely DBpedia and YAGO (version 2.5). The DBpedia RDF graph has 8,099,955 vertices and 72,193,833 edges. Among all vertices, 883,665 are places with coordinates. It has 2,927,026 unique words; a word appears on average in the documents of 56.46 vertices. YAGO has 8,091,179 vertices and 50,415,307 edges. Among these vertices, 4,774,796 are places. It has 3,778,457 distinct words, and each word appears on average in 7.83 nodes. For both datasets, we use an inverted index to organize the documents of nodes.

**Queries** We defined the query points to be in metropolitan areas (e.g., New York, London, Beijing, Tokyo), which contain plethora of places. For each of these cities, we extracted queries with locations and keywords that can return many results. More precisely, we used queries which, according to the $kSP$ framework, can produce at least five times more results than our largest $k$. Specifically, as we discuss below in our experimental settings, the largest $k$ is 20; thus, we generated queries (80 in total) for which the non-diversified version (i.e., $kSP$) can retrieve at least 100 results.

**Experimental settings** Our methodology is evaluated by varying the (i) number $k$ of requested places, (ii) the number of query keywords $|q \cdot \psi|$ and the diversification parameters, (iii) $\lambda$, (iv) $\beta$, and (v) $\gamma$. By default, $k = 10$, $|q \cdot \psi| = 2$, $\lambda = \beta = \gamma = 0.5$ (i.e., the *default setting*). In each experiment, we vary one parameter while fixing the remaining ones to their default values. Specifically, we report the result when parameter $k$ varies in {5, **10**, 15, 20}, $|q \cdot \psi|$ varies in {1, **2**, 3, 5} and $\lambda$, $\beta$, $\gamma$ vary in {0.25, **0.50**, 0.75}.

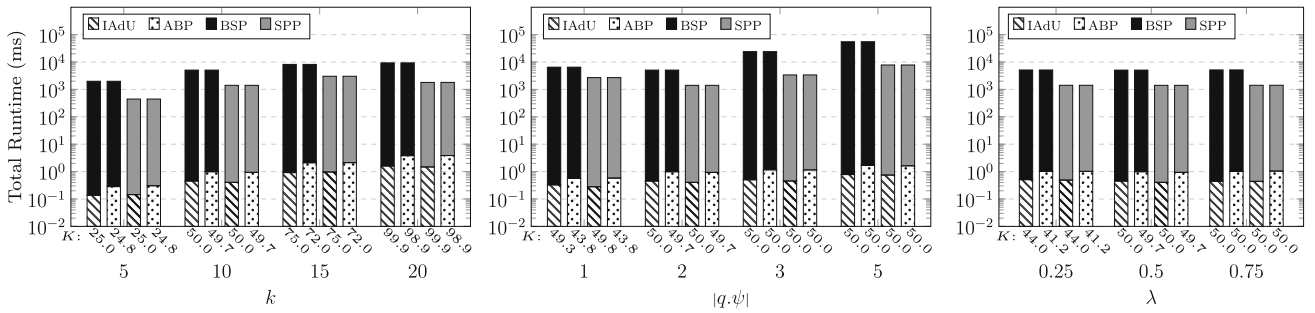
We set $L_{\max}$ as $5 * |q \cdot \psi|$ (the concept of $L_{\max}$ has been used often in earlier work, e.g., [36]). We set $S_{\max}$ as the largest distance in the map of the city. (this concept has also been used in earlier work [2])

**Platform** All methods were implemented in Java and evaluated on a 2.7 GHz dual-core, quad-thread machine, with 16 GBytes of memory, running Windows 10.

**Indexes and preprocessing costs** The time and space costs of constructing the indexes and the data structures are moderate (see [45] for more details). For instance, the construction of the R-tree of the DBpedia and YAGO datasets requires about 3 min and 31 min and occupies 50 MB and 273 MB, respectively. The inverted index construction requires about 4 min and 1 min and occupies 1307 MB and 231 MB respectively. The DBpedia data is richer in terms of text, and therefore, it needs more time to build the corresponding inverted indexes. The reachability index (TFlabel) [11] construction requires 22 min and 6 min, respectively. The R-tree, TFlabel index, and RDF graph are all memory resident; on the other hand, the inverted index is disk resident.

### 8.2 Efficiency evaluation

In the first set of experiments (Figs. 8, 9), we measured the average run-time costs of the tested algorithms on the 80 queries for the various parameter values on DBpedia and YAGO. We show (the average of) the combined costs of our algorithms (IAdU and ABP) with the $k$SP algorithms (BSP and SPP). Recall that our methods (IAdU and ABP) use as a module a $k$SP algorithm to incrementally retrieve the places in order of relevance to the query $q$. Hence, there are four combinations: BSP+IAdU, BSP+ABP, SPP+IAdU,




**(a)** Varying $k$ ($|q.\psi| = 2$, $\lambda = \beta = \gamma = 0.5$) **(b)** Varying $|q.\psi|$ ($k = 10$, $\lambda = \beta = \gamma = 0.5$) **(c)** Varying $\lambda$($|q.\psi|$=2, $k$=10, $\beta$=$\gamma$=0.5)

**Fig. 9** Efficiency on YAGO2

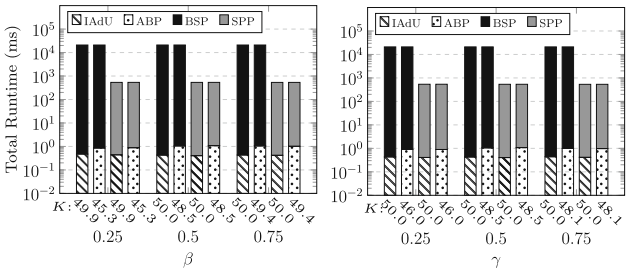
and SPP+ABP. Each bar adds up the total cost of the corresponding combination; the top part is the cost of applying the corresponding $k$SP algorithm without diversification to retrieve the same number of places as the complete $k$DSP algorithm. We also show, at the bottom at each bar, the average number $K$ of places (and the corresponding TQTs) retrieved by the diversification algorithms.

The results show that $k$SP retrieval dominates the combined cost. Naturally, SPP outperforms BSP because of the two pruning rules employed by SPP [45]. The diversification algorithms, IAdU and ABP, require insignificant time (in comparison with $k$SP algorithms). In all cases, this cost is 1 ms or less. Hence, although ABP costs up to twice the time required by IAdU, this extra time is negligible when combined to the place retrieval costs by BSP and SPP. Overall, the additional overhead required to achieve diversification is negligible in comparison with the $k$SPs cost; however, the number of places $K$ that have to be retrieved in order to answer a $k$DSP query is roughly $K = 5 * k$. Still, their retrieval is necessary in order to ensure that the results qualify the diversification requirements.

**Varying** $k$. In Figs. 8a and 9a, we depict the effect of $k$. As expected, $K$ and all respective costs increase with $k$. Observe that the $k$SP retrieval costs continue to dominate the overall cost of $k$DSP queries.

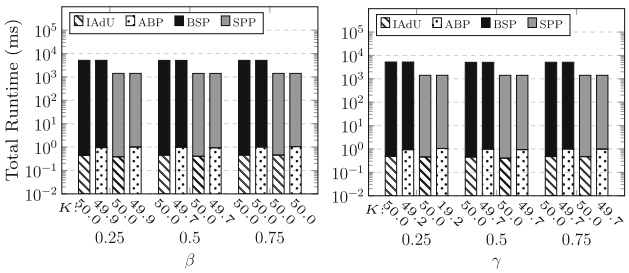
**Varying** $|q \cdot \psi|$. Figures 8b and 9b show how the number of keywords affects the running time. As $|q \cdot \psi|$ increases, both $k$SP and $k$DSP costs slowly increase. $k$SP algorithms need to explore more RDF vertices in order to discover the TQTs covering all keywords. This also results in an increase in the size of the TQTs; in turn, $k$DSP algorithms spend more time on computing Jaccard distances. On the other hand, $|q \cdot \psi|$ has almost negligible impact on $K$.

Note that for the DBpedia dataset, the cost when $|q \cdot \psi| = 5$ drops for the following two reasons. In these experiments, we use different sets of queries for each value of $|q \cdot \psi|$ (e.g., $\{ancient\}$ for $|q \cdot \psi| = 1$, $\{ancient, roman\}$ for $|q \cdot \psi| = 2$, etc.). This is in contrast to the experiments comparing $k$ and $\lambda$ (e.g., Fig. 8a, c), where we have the same query for all experiments, and we just vary the values of $k$ and $\lambda$. In addition, for larger values of $|q \cdot \psi|$, we could run fewer queries because there are not many places that include all keywords in their trees. Indicatively, in Fig. 8b, the number of queries that we ran for of $|q \cdot \psi| = 3$ is 80, while the number of queries that we ran for $|q \cdot \psi| = 5$ is 63. Naturally, these queries included mostly keywords which are frequent and their graph distance to the places that include them is not large. Hence, the retrieval of these places may be easier (i.e., their looseness scores may be easier to compute) compared to places retrieved for queries with smaller $|q \cdot \psi|$ values.




**(a)** Varying $\beta$ **(b)** Varying $\gamma$

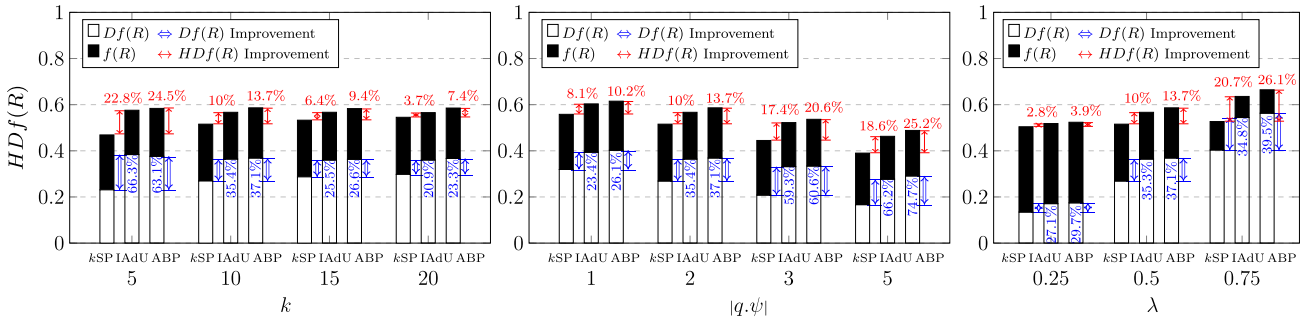
**Fig. 10** Efficiency on DBpedia




**(a)** Varying $\beta$ **(b)** Varying $\gamma$

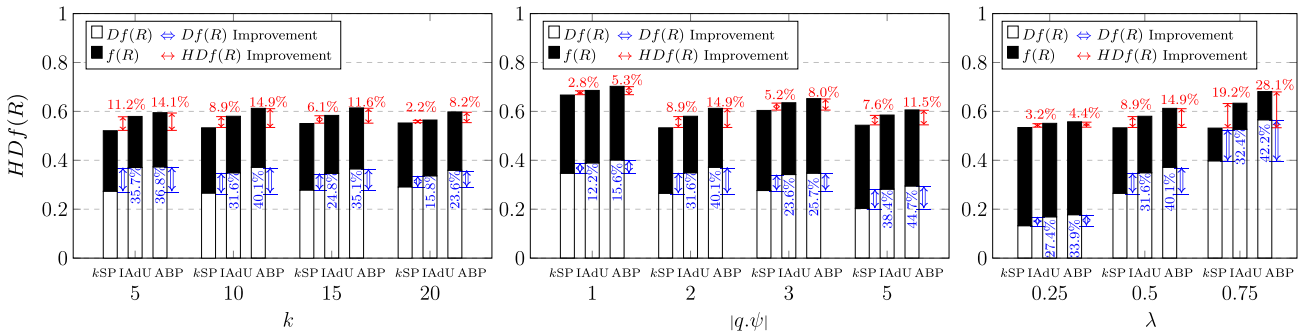
**Fig. 11** Efficiency on YAGO2

**Varying** $\lambda$, $\beta$, $\gamma$. Figures 8c and 9c show the effect of $\lambda$ on the running time on the two datasets. Figures 10 and 11 show the effect of $\beta$ and $\gamma$. The results show that the various values




**(a)** Varying $k$ ($|q.\psi| = 2$, $\lambda = \beta = \gamma = 0.5$) **(b)** Varying $|q.\psi|$ ($\lambda = \beta = \gamma = 0.5$, $k = 10$) **(c)** Varying $\lambda$($|q.\psi|$=2,$k$=10,$\beta$=$\gamma$=0.5)

**Fig. 12** $HDF(R)$ scores on DBpedia




**(a)** Varying $k$ ($|q.\psi| = 2$, $\lambda = \beta = \gamma = 0.5$) **(b)** Varying $|q.\psi|$ ($\lambda = \beta = \gamma = 0.5$, $k = 10$) **(c)** Varying $\lambda$ ($|q.\psi|$=2,$k$=10,$\beta$=$\gamma$=0.5)

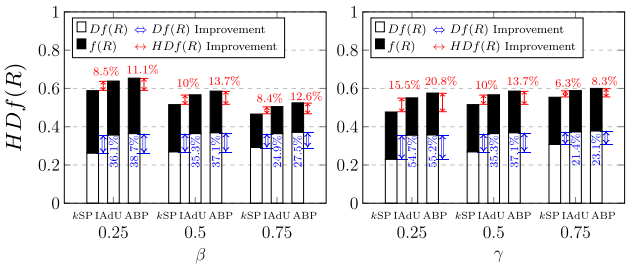
**Fig. 13** $HDF(R)$ scores on YAGO2

of these parameters have no significant impact on the total time.

## 8.3 Effectiveness

We assess the effectiveness of the two frameworks ($k$SP against $k$DSP) by comparing their respective $HDf(R)$ and $Df(R)$ scores. (Note that the use of either BSP or SPP makes no difference, since both these algorithms return the same sets of places in the same order.) Figures 12, 13, 14, and 15 show the average of the $HDf(R)$ scores of the $k$SP approach and those of the IAdU and ABP algorithms for the two datasets (recall that we cannot obtain the optimal $HDf(R)$ scores due to the high computational cost required). The top (bottom) of each bar shows the $f(R)$ ($Df(R)$) score of each method. We also show the (average) percentage of the $HDf(R)$ improvement of IAdU and ABP against $k$SP by using the single lined arrows (placed on the top side of the respective bar). In addition, we also show the (average) percentage of the $Df(R)$ improvement of IAdU and ABP against $k$SP by using the double lined arrows (placed in the middle of the respective bar).

The improvement of IAdU and ABP against $k$SP in terms of $HDf(R)$ can be significant (up to 10% and 13.75%, respectively, for DBpedia and up to 8.9% and 14.9% for

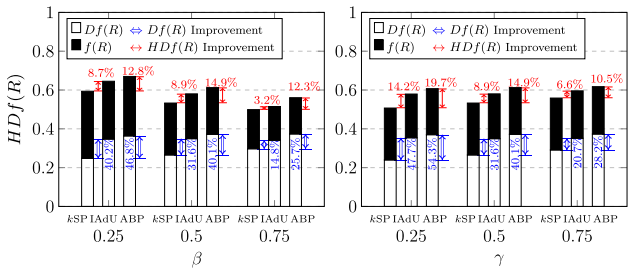



**(a)** Varying $\beta$ **(b)** Varying $\gamma$

**Fig. 14** $HDF(R)$ scores on DBpedia

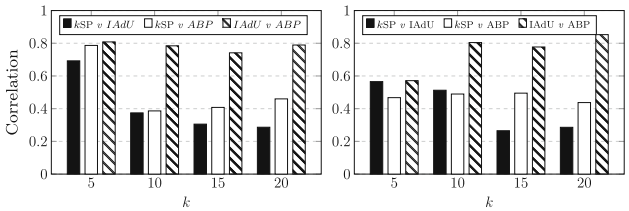
YAGO2 for the default settings). (Note that the $HDF(R)$ improvement after diversification is capped by $\lambda = 0.5$.) The improvement of diversity alone, $Df(R)$, is significantly larger than that of the respective holistic $HDf(R)$ score (up to 33.5% and 37.1% for DBpedia and 31.6% and 40.1% for YAGO). ABP always achieves (marginally) better improvement than IAdU with respect to both $HDf(R)$ and $Df(R)$ scores. The gap reflects their (comparative) approximation quality. ABP performs 3.8% (i.e., 13.7–9.9%) and 6% (i.e., 14.9–8.9%) better than IAdU on the two datasets, respectively.

In order to assess the approximation quality of IAdU and ABP, we conducted an experiment where we compared their




**(a)** Varying $\beta$ **(b)** Varying $\gamma$

**Fig. 15** $HDF(R)$ scores on YAGO2




**(a)** DBpedia **(b)** YAGO2

**Fig. 16** Ranking correlation

results to those of a brute force (BF) baseline algorithm. BF finds the combination $R$ of $k$ places that maximizes $HDf(R)$. Since BF is very slow, we could only run it for values of $k$ up to 7. The $HDf(R)$ score of the solution computed by BF is at most 1.5% (at least 0.74%) and 3.5% (at least 0.92%) compared to the $HDf(R)$ scores of the solutions returned by ABP and IAdU, respectively. On the other hand, as expected, BF is up to several orders of magnitude slower and does not scale well with $k$. This confirms that the proposed algorithms indeed yield high-quality solutions, efficiently.

**Varying** $k$. Figures 12a and 13a show the effect of $k$ on $HDf(R)$ and $Df(R)$ scores. As the value of $k$ increases, the improvement of the diversification algorithms against $k$SP reduces, because it becomes harder to find more diverse results (especially in the two-dimensional space of locations).

**Varying** $|q \cdot \psi|$. Figures 12b and 13b show that as the number of keywords $|q \cdot \psi|$ increases, the improvement of $HDf(R)$ and $Df(R)$ also increases. This is because the size of TQTs increases and it becomes easier to find more diverse ones. Namely, more nodes (and more new paths) are included in the TQTs; this results in a larger expected Jaccard distance between two TQTs.

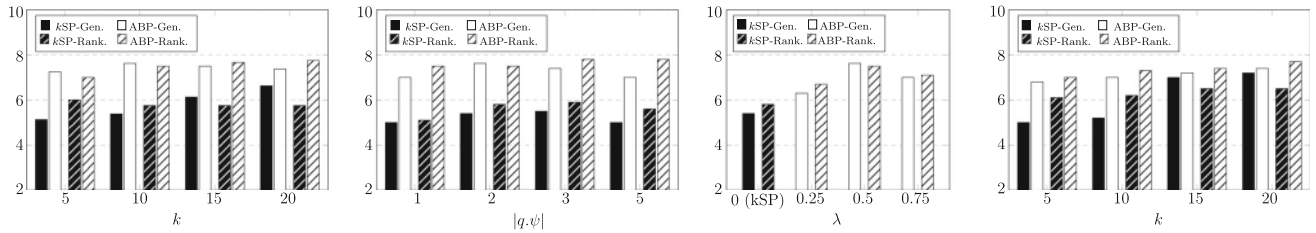
**Varying** $\lambda, \beta, \gamma$. Figures 12c and 13c show the effect of $\lambda$ on $HDf(R)$ and $Df(R)$ scores. We observe that as the value of $\lambda$ increases, so does the improvement of both $HDf(R)$ and $Df(R)$. This is expected, as bigger $\lambda$ values favor diversity and consequently magnify the difference between the approaches. Figures 14 and 15 show the effect of $\beta$, $\gamma$ on $HDf(R)$ and $Df(R)$ scores. An interesting observation here is that as $\gamma$ increases, the improvement gap drops (recall that $\gamma$ is a trade-off between spatial and content diversity). Larger $\gamma$ values give higher weight to content diversity and since it is easier to find diverse results in terms of content than in the 2-dimensional space of locations, the results of $k$SP and IAdU have higher chances to find more diverse place sets.

### 8.3.1 Re-ranking

Diversification achieves an effective re-ranking of search results by combining both relevance and diversity. This facilitates a *bird's eye view* of results, which is preferable by users [1,10] (also verified by our own user evaluation, Sect. 8.4). Figure 16a, b depicts the *average correlation* of the place rankings by $k$SP with that of each $k$DSP algorithm (i.e., IAdU and ABP) for all tested queries (for the default settings). As $k$ increases, the correlation of the $k$SP ranking against the $k$DSP ranking drops. Namely, the $k$SP ranking of places differs significantly from the corresponding $k$DSP ranking. Our user evaluation (Sect. 8.4) revealed that the top $k$SP results are very similar whereas the top $k$DSP results are very diverse, offering a bird's eye view of results which was favored by our evaluators. Hence, our approach is useful even for large values of $k$, for which the $HDF(R)$ improvement is not large as we discussed before (see Figs. 12, 13). The figure also shows that the rankings by IAdU and APB are highly correlated (i.e., their results have high overlap).

## 8.4 User evaluation

We also conducted a user evaluation on $k$DSP and $k$SP queries, which confirms the preference of users to diversified results and the effectiveness of re-ranking. We asked help from ten evaluators, who are professors and researchers from our universities. (None of them was involved in this paper.) First, we familiarized them with the query concepts and relevance metrics (distance and looseness). In addition, we explained to them the concepts of (1) diversity and (2) ranking facilitating a bird's eye view; to avoid any bias, we avoided to discuss their advantages or disadvantages. Then, we presented to the evaluators ten random queries and their top-$k$ $k$SP and $k$DSP results and ask them to evaluate: (1) the general content of the results and (2) their ranking. For each set of top-$k$ results, we showed a map with the places and the TQT of each place. We presented the output of each method in a random order (to avoid any bias) and asked the evaluators to give a preference score in a scale of one to ten, considering how representative and informative the overall top-$k$ results were.




**(a)** Varying $k$ (DBpedia) **(b)** Varying $|q.\psi|$ (DBpedia) **(c)** Varying $\lambda$ (DBpedia) **(d)** Varying $k$ (YAGO2)

**Fig. 17** User evaluation

Figure 17a, d averages the evaluators' preference scores of the two methodologies (i.e., $k$SP and $k$DSP), for the two criteria (i.e., general content and ranking), for various values of $k$, for the two datasets (using the default settings). For $k$DSP, we used ABP (since IAdU and ABP give similar results). For the first criterion (general content), we observe that the users prefer diversified results ($k$DSP) for small values of $k$ (i.e., for $k = 5$ and $k = 10$). For larger values of $k$, we observe that the gap between preference of $k$DSP and $k$SP results is reduced. This is because the results of these two methodologies have higher overlap for larger $k$. On the other hand, for the second criterion (ranking), users prefer $k$DSP over $k$SP for all values of $k$. The study revealed that for small values of $k$ (i.e., $k = 5$ and $k = 10$), the results of the $k$SP approach included many similar places (e.g., for $k = 3$ for the example of Fig. 1, all three places were communes located in the same area); on the other hand, the results of the $k$DSP approach included almost completely diverse places (e.g., for $k = 3$, only one place was a commune). This finding justifies the preference for $k$DSP ranking for large values of $k$. The top places are typically diverse to each other, whereas only some bottom results had some similarity to previous ones; this bird's eye view is preferable by users. For example, for $k = 20$, the top 10 places are all of different types (e.g., a commune appears only once), and some of these types also appear in the bottom 10 places (e.g., additional communes). In general, the user evaluation findings are in accordance with the effectiveness findings (discussed in Sect. 8.3).

Figure 17b averages the evaluators' preference scores of the two methodologies for various values of $|q \cdot \psi|$ (for $k = 10$). We observe that users prefer for all values of $|q \cdot \psi|$ diversified results. More precisely, we observe that the value of $|q \cdot \psi|$ does not affect significantly the difference between the preference of $k$SP over $k$DSP results. Namely, the difference between the user preferences for the two criteria remains approximately the same for different values of $|q \cdot \psi|$. Figure 17c averages the evaluators' preference scores of the two methodologies for various values of $\lambda$ ($k = 10$). Note that when $\lambda = 0$, $k$DSP gives the same results as $k$SP. We observe that users prefer diversified results; more precisely, users prefer results produced with $\lambda = 0.5$ as they facilitate more effective diversification.

### 8.5 Discussion

In conclusion, the combination of SPP and ABP appears to be the best choice for diversified spatial keyword search on RDF data. SPP is very fast (compared to BSP) for $k$SP incremental search, while ABP is negligibly more expensive than IAdU and achieves better approximation and effectiveness scores. For example, search on DBpedia using SPP+ABP never requires more than a second; hence, real-time results can be obtained. Although the increase in $k$ reduces the effectiveness improvement, the achieved re-ranking based on relevance and diversification remains useful for all values of $k$. Finally, our user evaluation confirms the users' preference for $k$DSP over $k$SP results and ranking.

## 9 Conclusions

In this work, we enrich spatial keyword search on RDF data with the ability to diversify query results. Our framework combines relevance and diversification, w.r.t. both content and location. We propose two greedy algorithms (IAdU and ABP) and provide theoretical guarantees for their quality. Our experiments on real data verify the effectiveness, approximation quality, and efficiency of our algorithms (where ABP is shown to be superior to IAdU) and confirm that our framework is preferred by human evaluators. In our future work, we will study alternative scoring functions for the spatial and content-based search components (e.g., road network distance in place of Euclidean distance).

**Acknowledgements** Open access funding provided by Uppsala University. Zhi Cai was supported by National Key R&D Program of China (No. 2017YFC0803300) and the Beijing Natural Science Foundation (No. 4172004). Nikos Mamoulis was partially funded by the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 657347. Dimitris Papadias was supported by GRF Grant 16231216 from Hong Kong RGC.

## References


1. Adomavicius, G., Kwon, Y.: Improving aggregate recommendation diversity using ranking-based techniques. IEEE Trans. Knowl. Data Eng. **24**(5), 896–911 (2012)
2. Ahuja, R., Armenatzoglou, N., Papadias, D., Fakas, G.J.: Geo-social keyword search. In: SSTD, pp. 431–450 (2015)
3. Alsina, C., Nelsen, R.B.: Charming Proofs: A Journey into Elegant Mathematics. Mathematical Association of America, Washington (2010)
4. Athitsos, V., Potamias, M., Papapetrou, P., Kollios, G.: Nearest neighbor retrieval using distance-based hashing. In: ICDE, pp. 327–336. IEEE Computer Society (2008)
5. Battle, R., Kolas, D.: Enabling the geospatial semantic web with parliament and geosparql. Semant. Web **3**(4), 355–370 (2012)
6. BBC-Lab-Post: Linked data (2013). http://www.bbc.co.uk/blogs/internet/entries/63841314-c3c6-33d2-a7b8-f58ca040a65b
7. Bergamaschi, S., Domnori, E., Guerra, F., Lado, R.T., Velegrakis, Y.: Keyword search over relational databases: a metadata approach. In: SIGMOD Conference, pp. 565–576. ACM (2011)
8. Bergamaschi, S., Guerra, F., Interlandi, M., Lado, R.T., Velegrakis, Y.: Combining user and database perspective for solving keyword queries over relational databases. Inf. Syst. **55**, 1–19 (2016)
9. Bikakis, N., Giannopoulos, G., Liagouris, J., Skoutas, D., Dalamagas, T., Sellis, T.: Rdivf: diversifying keyword search on RDF graphs. In: TPDL, pp. 413–416 (2013)
10. Carbonell, J.G., Goldstein, J.: The use of MMR, diversity-based reranking for reordering documents and producing summaries. In: SIGIR, pp. 335–336. ACM (1998)
11. Cheng, J., Huang, S., Wu, H., Fu, A.W.: TF-label: a topological-folding labeling scheme for reachability querying in a large graph. In: SIGMOD, pp. 193–204 (2013)
12. Cheng, S., Chrobak, M., Hristidis, V.: Slowing the firehose: Multi-dimensional diversity on social post streams. In: EDBT, pp. 17–28. OpenProceedings.org (2016)
13. Data.gov: U.s. government's open data. http://www.data.gov/
14. DBpedia: http://wiki.dbpedia.org
15. Demidova, E., Fankhauser, P., Zhou, X., Nejdl, W.: Divq: diversification for keyword search over structured databases. In: SIGIR, pp. 331–338 (2010)
16. Dimitriou, A., Theodoratos, D.: Efficient keyword search on large tree structured datasets. In: KEYS, pp. 63–74. ACM (2012)
17. Dimitriou, A., Theodoratos, D., Sellis, T.K.: Top-k-size keyword search on tree structured data. Inf. Syst. **47**, 178–193 (2015)
18. Elbassuoni, S., Blanco, R.: Keyword search over RDF graphs. In: CIKM, pp. 237–242 (2011)
19. Fakas, G.J.: Automated generation of object summaries from relational databases: a novel keyword searching paradigm. In: DBRank, ICDE, pp. 564–567 (2008)
20. Fakas, G.J.: A novel keyword search paradigm in relational databases: object summaries. DKE **70**(2), 208–229 (2011)
21. Fakas, G.J., Cai, Y., Cai, Z., Mamoulis, N.: Thematic ranking of object summaries for keyword search. Data Knowl. Eng. **113**, 1–17 (2018)
22. Fakas, G.J., Cai, Z., Mamoulis, N.: Size-*l* object summaries for relational keyword search. PVLDB **5**(3), 229–240 (2011)
23. Fakas, G.J., Cai, Z., Mamoulis, N.: Versatile size-*l* object summaries for relational keyword search. IEEE Trans. Knowl. Data Eng. **26**(4), 1026–1038 (2014)
24. Fakas, G.J., Cai, Z., Mamoulis, N.: Diverse and proportional size-*l* object summaries for keyword search. In: SIGMOD, pp. 363–375 (2015)
25. Fakas, G.J., Cai, Z., Mamoulis, N.: Diverse and proportional size-*l* object summaries using pairwise relevance. VLDBJ **25**(6), 791–816 (2016)
26. Fakas, G.J., Cawley, B., Cai, Z.: Automated generation of personal data reports from relational databases. JIKM **10**(2), 193–208 (2011)
27. Fu, H., Anyanwu, K.: Effectively interpreting keyword queries on RDF databases with a rear view. In: ISWC, pp. 193–208 (2011)
28. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
29. Haritsa, J.R.: The KNDN problem: a quest for unity in diversity. IEEE Data Eng. Bull. **32**(4), 15–22 (2009)
30. Hasan, M., Kashyap, A., Hristidis, V., Tsotras, V.J.: User effort minimization through adaptive diversification. In: KDD, pp. 203–212. ACM (2014)
31. Hassin, R., Rubinstein, S., Tamir, A.: Approximation algorithms for maximum dispersion. Oper. Res. Lett. **21**(3), 133–137 (1997)
32. He, H., Wang, H., Yang, J., Yu, P.S.: BLINKS: ranked keyword searches on graphs. In: SIGMOD, pp. 305–316 (2007)
33. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Trans. Database Syst. **24**(2), 265–318 (1999)
34. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: a spatially and temporally enhanced knowledge base from wikipedia. Artif. Intell. **194**, 28–61 (2013)
35. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient IR-style keyword search over relational databases. In: VLDB, pp. 850–861 (2003)
36. Hristidis, V., Papakonstantinou, Y.: Discover: keyword search in relational databases. In: VLDB, pp. 670–681 (2002)
37. Jain, A., Sarda, P., Haritsa, J.R.: Providing diversity in k-nearest neighbor query results. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 404–413. Springer, Berlin (2004)
38. Kargar, M., An, A., Yu, X.: Efficient duplication free and minimal keyword search in graphs. TKDE **26**(7), 1657–1669 (2014)
39. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: a semantic geospatial DBMS. In: ISWC, pp. 295–311 (2012)
40. Le, W., Li, F., Kementsietsidis, A., Duan, S.: Scalable keyword search on large RDF data. TKDE **26**(11), 2774–2788 (2014)
41. Levandowsky, M., Winter, D.: Distance between sets. Nature **234**(5323), 34–35 (1971)
42. Liu, Z., Sun, P., Chen, Y.: Structured search result differentiation. PVLDB **2**(1), 313–324 (2009)
43. Parliament: A high-performance triple store, sparql endpoint, and reasoner. http://parliament.semwebcentral.org
44. Ravi, S.S., Rosenkrantz, D.J., Tayi, G.K.: Facility Dispersion Problems: Heuristics and Special Cases, pp. 431–450. Springer, Berlin (1991)
45. Shi, J., Wu, D., Mamoulis, N.: Top-k relevant semantic place retrieval on spatial RDF data. In: SIGMOD, pp. 1977–1990 (2016)
46. Sinha, S.B., Lu, X., Theodoratos, D.: Personalized keyword search on large RDF graphs based on pattern graph similarity. In: IDEAS, pp. 12–21. ACM (2018)
47. Stefanidis, K., Drosou, M., Pitoura, E.: Perk: personalized keyword search in relational databases through preferences. In: EDBT, pp. 585–596 (2010)

48. Stefanidis, K., Fundulaki, I., Stefanidis, K., Fundulaki, I.: Keyword search on RDF graphs: it is more than just searching for keywords. In: Gandon, F., Guéret, C., Villata, S., Breslin, J., Faron-Zucker, C., Zimmermann, A. (eds.) ESWC (Satellite Events). Lecture Notes in Computer Science, vol. 9341, pp. 144–148. Springer, Cham (2015)
49. Tang, J., Sanderson, M.: Evaluation and user preference study on spatial diversity. In: Gurrin, C., et al. (eds.) ECIR. Lecture Notes in Computer Science, vol. 5993, pp. 179–190. Springer, Berlin (2010)
50. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In: ICDE, pp. 405–416 (2009)
51. Van Kreveld, M., Reinbacher, I., Arampatzis, A., Van Zwol, R.: Multi-dimensional scattered ranking methods for geographic information retrieval. GeoInformatica **9**(1), 61–84 (2005)
52. Vieira, M.R., Razente, H.L., Barioni, M.C.N., Hadjieleftheriou, M., Srivastava, D., Traina, C., Tsotras, V.J.: On query result diversification. In: ICDE, pp. 1163–1174 (2011)
53. Virtuoso: http://virtuoso.openlinksw.com
54. Yago: http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/
55. Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z.: A distributed graph engine for web scale RDF data. PVLDB **6**(4), 265–276 (2013)
56. Zhang, M., Hurley, N.: Avoiding monotony: improving the diversity of recommendation lists. In: RecSys (2008)
57. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.: gstore: answering SPARQL queries via subgraph matching. PVLDB **4**(8), 482–493 (2011)