Textually Relevant Spatial Skylines

Jieming Shi Dingming Wu Nikos Mamoulis

Abstract—We study the modeling and evaluation of a *spatio-textual skyline* (STS) query, in which the skyline points are selected not only based on their distances to a set of query locations, but also based on their relevance to a set of query keywords. STS is especially relevant to modern applications, where points of interest are typically augmented with textual descriptions. We investigate three models for integrating textual relevance into the spatial skyline. Among them, model STD, which combines spatial distance with textual relevance in a derived dimensional space, is found to be the most effective one. STD computes a skyline which not only satisfies the intent of STS, but also has a small and easy-to-interpret size. We propose an efficient algorithm for computing STD-based skylines, which operates on an IR-tree that indexes the data. The effectiveness of our STD model and the efficiency of the proposed algorithm are evaluated on real data sets.

Index Terms—**TOBEADDED**

1 Introduction

In modern applications (e.g., Google Maps), points of interest are typically augmented with textual descriptions. For example, in TripAdvisor and Foursquare, people share tips and add tags for the places that they have visited; on Flickr, people share photos associated with GPS coordinates and textual descriptions. Retrieving points of interest based on both spatial proximity and textual relevance, i.e., *searching with local intent*, is witnessed everywhere and has attracted a lot of research interest [1], [2], [3], [4], [5]. In this paper, we propose a new data analysis operator for such data, the *spatio-textual skyline* (STS) query, which allows users to find places that are (i) near a group Q of spatial query points and (ii) relevant to a set of keywords. STS integrates the textual relevance of objects into the spatial skyline, so that the result is interesting in terms of both Euclidean distance and classic keyword search.

1.1 Background and Motivation

The general skyline operation [6] selects a set of interesting objects from a large set of multidimensional objects. An object is interesting if it is not *dominated by* any other object, i.e., it is not worse than any other object in all dimensions. For example, Figure 1(a) shows a set of two dimensional points $p_1, \ldots, p_5$; each point models a hotel, such that the x-value is its distance to the beach, while the y-value is its price (thus, in each dimension, smaller values are better). The skyline is $\{p_1, p_3\}$; for example, p_2 is dominated by p_3, because the x-value of p_3 is smaller and the y-value of p_3 is not larger than that of p_2.

The spatial skyline query (SSQ) [7] finds the *spatial skyline* of a set $\mathcal{P}$ of spatial locations (e.g., points of interest), based on a set of input query locations Q. In brief, each object $p \in \mathcal{P}$ is associated with $|Q|$ *derived dimensions*; the i-th derived value for p is the Euclidean distance $d(q_i, p)$ between p and query location q_i in Q. Then, the spatial skyline is the set of points that are not dominated by any other point in the space of derived dimensions. For example, consider the locations $\mathcal{P} = \{p_1, \ldots, p_6\}$ of a set of restaurants, as shown in Figure 1(b) and the positions $Q = \{q_1, q_2\}$ of two persons, who want to find candidate restaurants to have dinner together. Figure 1(c) shows in the derived 2-dimensional space ($|Q| = 2$) the distances of all restaurants to q_1 and q_2. SSQ finds the spatial skyline $\{p_1, p_2, p_3\}$, which is the set of interesting restaurants to q_1 and q_2, because for the remaining ones both q_1 and q_2 would have to travel more in order to meet.

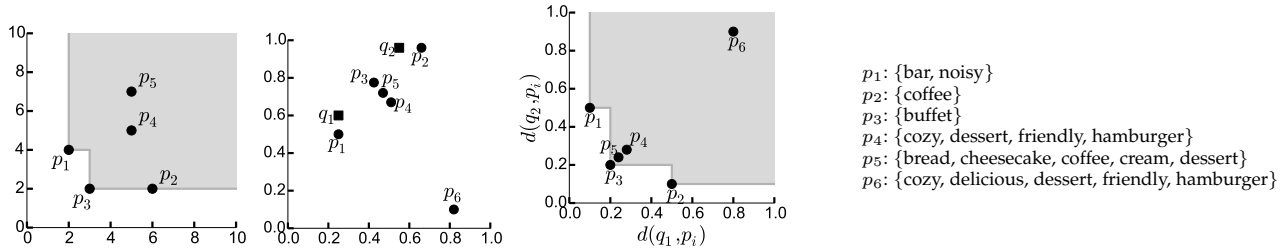
SSQ computes skyline objects that are interesting with respect to only their Euclidean distances to Q; thus, SSQ does not allow users to specify any non-spatial preferences expressed by textual search. Assume that the two persons in our previous example prefer restaurants that offer "hamburger" and "dessert" in a "cozy" environment and have "friendly" service, and let the textual descriptions of the six restaurants be as shown in Figure 1(d). The SSQ result (i.e., $\{p_1, p_2, p_3\}$) fails to include the interesting restaurants p_4 and p_6 w.r.t. the textual preferences of the users.

1.2 Contributions and Outline

The STS operator that we propose in this paper aims at addressing the aforementioned weakness of SSQ in disregarding textual relevance as a search criterion. STS generalizes SSQ and finds application in modern retrieval tasks. For instance, consider a group of people who visit a city (e.g., London) and stay in different hotels. These people may want to visit an attraction of the city, which is not far from their locations and at the same time it is relevant to their preferences (e.g., "museum", "art", "gallery", "culture"). As another example, consider a couple who want to visit several attractions in London, such as London eye, Buckingham palace, British museum and St Paul's Cathedral. The couple needs a list of candidate restaurants that are not far from the

- *J. Shi and N. Mamoulis are with the Department of Computer Science, University of Hong Kong, Hong Kong. E-mail: {jmshi, nikos}@cs.hku.hk*
- *D. Wu is with the College of Computer Science & Software Engineering, Shenzhen University, China and the Department of Computer Science, University of Hong Kong, Hong Kong. E-mail: dmwu@cs.hku.hk*

Corresponding author: D. Wu



(a) General Skyline (b) Locations (c) Derived Space (SSQ) (d) Restaurant Descriptions

Fig. 1. Skyline Examples

attractions and satisfy their preferences (e.g., "vegetarian", "healthy", "dining on a budget"), in order for them to enjoy meals during the trip.

In Section 2, we propose three models for integrating textual relevance into spatial skylines. Model DDA (Derived Dimension Augmentation) simply adds textual relevance to the set of derived dimensions. Model KBFF (Keyword Boolean Filtering First) applies a two-step process that first selects candidate objects whose descriptions contain at least one of the query keywords and then computes the spatial skyline of the candidates. Model STD (Spatio-Textual Dominance) replaces the spatial distance measure of the derived dimensions by a combined *spatio-textual* distance; thus, the skyline computed by STD is both spatially and textually relevant to the query set Q.

We evaluate the three models and compare them with SSQ visually and quantitatively in Section 3, using real data sets. The results clearly show that STD is the most appropriate model for STS. An additional advantage of STD is that it computes a skyline orders of magnitude smaller than that of SSQ, DDA, and KBFF on the same data, that makes it easier to manage and analyze.

Besides studying the effectiveness of STD, we propose efficient algorithms for computing spatio-textual skylines based on STD in Section 4. Given a large volume of objects, the challenge is that the skyline should be computed in a derived (i.e., dynamic) spatio-textual distance space, which cannot be pre-indexed. We present a basic algorithm BSTD which operates on an IR-tree [3] that indexes the objects. BSTD employs pruning rules to discover skyline objects efficiently. In order to further improve the performance of BSTD, we propose an advanced algorithm ASTD that uses a memory-based R-tree to index the skyline objects found so far and reduce the dominance tests and I/O cost significantly. The performance of the proposed algorithms is evaluated on real data under various parameter settings, in Section 5. Section 6 reviews work related to the STS operator. Finally, we conclude the paper in Section 7.

2 MODELS

In this section, we first provide some basic definitions and then describe in detail three spatio-textual skyline (STS) models. Let $\mathcal{P}$ be the set of objects for which we want to compute the STS. Each object $p \in \mathcal{P}$ is a point location $p = \langle x, y \rangle$ associated with a textual description that is represented as a *term vector* [8] $p.\psi$. Each dimension of the vector corresponds to a term and its value (weight) is calculated using the tf-idf weighing scheme [9]. The STS operator takes in a query which comprises a set of query point locations $Q = \{q_1, q_2, \dots\}$ and a set of query keywords $Q.\psi = \{t_1, t_2, \dots\}$. As an example, consider the textual descriptions of the six objects in Figure 1(d); their term vectors are shown as columns in Table 1. Given a set of query keywords $Q.\psi$, the textual relevance of each object p can be modeled as a ranking function $w(Q.\psi, p.\psi)$ (e.g., language models [10]). Typically, larger values of the ranking function model higher textual relevance.

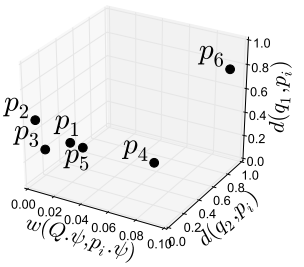
TABLE 1
Term Vectors

	p_1	p_2	p_3	p_4	p_5	p_6
bar	0.389	0	0	0	0	0
bread	0	0	0	0	0.156	0
buffet	0	0	0.778	0	0	0
cheesecake	0	0	0	0	0.156	0
coffee	0	0.477	0	0	0.0954	0
cozy	0	0	0	0.119	0	0.0954
cream	0	0	0	0	0.156	0
delicious	0	0	0	0	0	0.156
dessert	0	0	0	0.0753	0.0602	0.0602
friendly	0	0	0	0.119	0	0.0954
hamburger	0	0	0	0.119	0	0.0954
noisy	0.389	0	0	0	0	0

2.1 Derived Dimension Augmentation (DDA)

In SSQ, each object p has $|Q|$ derived spatial dimensions. The value of each dimension is the Euclidean distance $d(q_i, p)$ between p and a query location $q_i \in Q$. Model DDA adds one more derived dimension $w(Q.\psi, p.\psi)$ to each object p, modeling the textual relevance of p to the query keywords $Q.\psi$. STS is then computed as the skyline in the derived $(|Q| + 1)$-dimensional space. This model is similar to SSQ with static non-spatial attributes, proposed in [7]; the difference is that the textual relevance is dynamically calculated and depends on the query keywords.

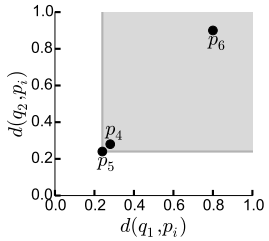
Example 1 Consider the six objects $p_1, \dots, p_6$ and query $Q = \{q_1, q_2\}$ with query keywords $Q.\psi$ ={*cozy, delicious, dessert, friendly, hamburger*} in Figure 1(b). Based on DDA, the STS result includes all the six objects, as shown in Figures 2(a) and 2(b) (the numbers in bold indicate that an object is not dominated, due to the combination of these values). In this example, the textual relevance $w(Q.\psi, p.\psi)$ is computed using language models [10]: if $p.\psi$ contains no term in $Q.\psi$, then $w(Q.\psi, p.\psi) = 0$; otherwise, $w(Q.\psi, p.\psi) = \prod_{t_i \in Q.\psi} \hat{w}(t_i, p.\psi)$, where $\hat{w}(t_i, p.\psi)$ is defined as follows. If term t_i appears in $p.\psi$, $\hat{w}(t_i, p.\psi)$ is the weight of t_i in $p.\psi$; otherwise, $\hat{w}(t_i, p.\psi)$ is a smoothing factor (e.g., 0.02). Finally, we scale the textual relevance by raising $w(Q.\psi, p.\psi)$ to the power of $1/|Q.\psi|$, in order to not



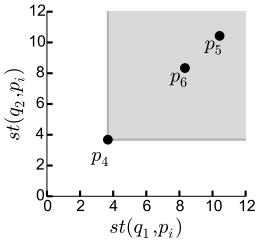
(a) Derived Space (DDA)

	$d(q_1,p_i)$	$d(q_2,p_i)$	$w(Q.\psi,p_i.\psi)$
p_1	**0.1**	0.5	0
p_2	0.5	**0.1**	0
p_3	**0.2**	**0.2**	0
p_4	**0.28**	**0.28**	**0.076**
p_5	**0.26**	**0.26**	**0.0249**
p_6	0.8	0.9	**0.096**

(b) Derived Values (DDA)



(c) KBFF Example



(d) STD Example

Fig. 2. Models

get extremely small values. For instance, $w(Q.\psi, p_4.\psi) = (0.119 \times 0.02 \times 0.0753 \times 0.119 \times 0.119)^{1/5} = 0.076$.

2.2 Keyword Boolean Filtering First (KBFF)

Model KBFF adapts SSQ to consider textual relevance, following two steps. It first selects from $\mathcal{P}$ only the *candidate* objects whose textual descriptions $p.\psi$ contain at least one term from the query keywords $Q.\psi$ (partial matches). Then, SSQ is applied on the candidates set $\mathcal{C} \subseteq \mathcal{P}$ selected in the first step. We can also have a tighter variant of KBFF that selects, as candidates, objects for which $Q.\psi \subseteq p.\psi$ (full matches). This version of KBFF is similar to the query studied in [11], which applies boolean keyword filtering first and then computes a general skyline.

Example 2 When applied on the data of Example 1, KBFF first obtains the candidate objects $\mathcal{C} = \{p_4, p_5, p_6\}$, because their textual descriptions overlap with $Q.\psi$. Then, by applying SSQ on $\mathcal{C}$ only, the resulting STS is $\{p_5\}$, as shown in Figure 2(c).

2.3 Spatio-Textual Dominance (STD)

Model STD is based on *integrating*, for each query location $q_i \in Q$, Euclidean distance with textual relevance to the candidate STS objects $p_j \in \mathcal{P}$, defining a unified *spatio-textual* distance function $st(q_i, p_j)$. In this paper, we adopt the *weighted distance* [4] $st(q_i, p_j) = d(q_i, p_j)/w(Q.\psi, p_j.\psi)$. Compared to other possible aggregate functions, e.g., the weighted sum $\alpha \cdot d(q_i, p_j) + (1-\alpha)(1 - w(Q.\psi, p_j.\psi))$ [5], the weighted distance is parameter-free and has no normalization problems. Note that $w(Q.\psi, p_j.\psi) \in [0, 1]$, therefore $st(q_i, p_j) \geq d(q_i, p_j)$.

Using function $st()$, for a given query Q, we can map each object $p_i \in \mathcal{P}$ to a space of $|Q|$ derived *spatio-textual dimensions*. Object p_j is *spatio-textually dominated* (Definition 1) by p_i if and only if p_j is not better than p_i in all derived dimensions and p_j is worse than p_i in at least one derived dimension. Based on spatio-textual dominance, we can directly define the spatio-textual skyline (STS) (Definition 2). *Spatio-textual dominance* and *spatio-textual skyline* are abbreviated to *dominance* and *skyline* or *STS*, respectively, when the context is clear. Note that any object $p_i \in \mathcal{P}$ for which $w(Q.\psi, p_i.\psi) = 0$ has infinite $st(q_j, p_i)$ to any $q_j \in Q$; these objects can be immediately pruned from the skyline, since they are dominated by any object in $\mathcal{P}$ with non-zero textual relevance.

Definition 1. (Spatio-Textual Dominance) $p_i \ dom_Q \ p_j \iff \forall q_k \in Q : st(q_k, p_i) \leq st(q_k, p_j)$ and $\exists q_k \in Q : st(q_k, p_i) < st(q_k, p_j)$.

Definition 2. (Spatio-Textual Skyline) Given query Q, the spatio-textual skyline (STS) of $\mathcal{P}$ includes the objects in $\mathcal{P}$ that are not spatio-textually dominated by any other object in $\mathcal{P}$.

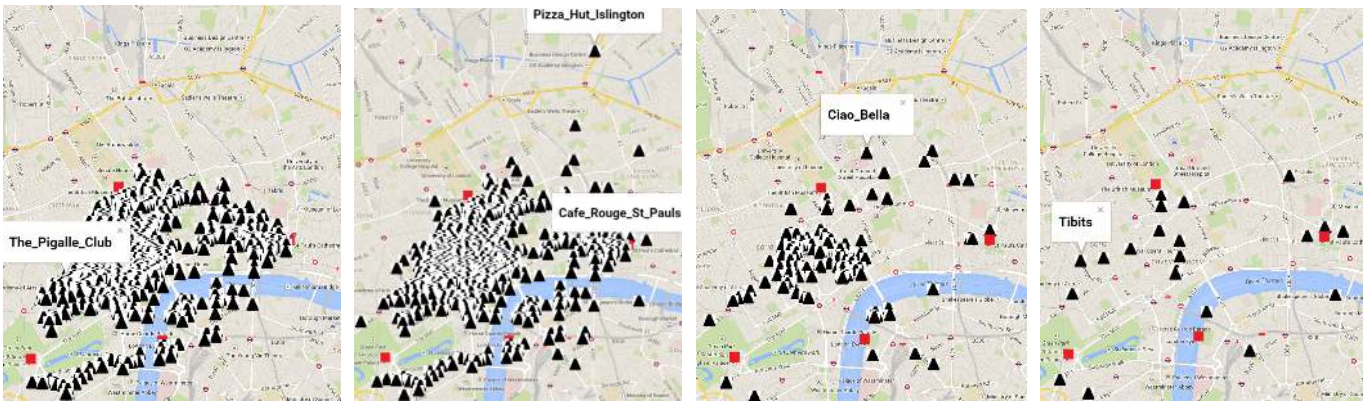
Example 3 Using the data of our running example, similar to model KBFF, STD discards the textually irrelevant objects w.r.t. $Q.\psi$, as these objects have infinite derived spatio-textual distances to every $q_i \in Q$. Figure 2(d) shows the derived spatio-textual distances of the remaining objects. The STS is $\{p_4\}$.

2.4 Discussion

In general, a good STS operator should compute a skyline that is interesting in terms of both the Euclidean distance and the textual relevance. The objects with large Euclidean distance and low textual relevance (or even irrelevant) should be penalized. Model DDA considers the spatial dimensions and the textual dimension separately, so it may include in the skyline objects far from the query locations (e.g., p_6) or with low textual relevance (e.g., p_5). It may even include irrelevant objects w.r.t. the query keywords (e.g., p_1, p_2, and p_3). In general, the skyline computed by DDA is expected to be large. Model KBFF is better than DDA, since it immediately prunes textually irrelevant objects. However, the remaining objects are only evaluated based on their spatial dimensions, even when they differ significantly w.r.t. their textual relevance. Hence, the result of KBFF may include spatially close objects of low textual relevance (e.g., p_5). Model STD is expected to be better than DDA and KBFF, since STD discards irrelevant objects and penalizes objects with large Euclidean distance and low textual relevance. We note that DDA can be seen as an instance of a class of models that augment to SSQ additional non-spatial derived dimensions, whereas KBFF and STD belong to a class of models, which embed textual relevance in the derived spatial dimensions (in the case of KBFF, this is done with a step function).

3 Model Evaluation

In this section, we evaluate the three models DDA, KBFF, and STD and compare them with SSQ based on a case study and quantitative measurements. We use two datasets for the evaluation. TripAdvisor-LDN (used in Section 3.1) contains 9,346 restaurants with a dictionary of 169 terms from London and its neighboring area. Each restaurant has a textual description extracted from its tags, title, and reviews. The average length of the textual description per restaurant is 2.56. Flickr-LDN (used in Section 3.2) contains 406,151



(a) SSQ (b) DDA (c) KBFF (d) STD

Fig. 3. London Tour Case Study

pictures in London crawled from Flickr with a dictionary of 222,613 terms. Each picture is associated with a GPS coordinate and a set of terms extracted from its tags and title. The average number of terms per picture is 9.56.

3.1 Case-based Analysis

Consider one of the application scenarios discussed in the introduction. A tourist is planning to have a family trip to London. He plans to visit London eye, Buckingham palace, British museum, and St Paul's Cathedral (query locations) and wants to find a list of restaurants (skyline objects) that are not far from the venues to be visited and provide services that meet his textual preferences *chicken wings*, *dessert*, *healthy*, and *dining on a budget*.

3.1.1 Visualization-based Analysis

Figure 3 shows the skyline results of the four methods SSQ, DDA, KBFF, and STD, using the set of landmarks {London eye, Buckingham palace, British museum, and St Paul's Cathedral} as Q and the textual preferences of the tourist as $Q.\psi$. The red squares on the maps are the landmarks; the black triangles are the skyline objects found by each model. We now analyze the result and discuss the differences of the four methods.

SSQ vs. STD. Recall that SSQ disregards $Q.\psi$ and only computes the spatial skyline. This results in the pruning of some spatially and textually interesting objects by SSQ. For instance, object "Tibits" in Figure 3(d) with textual description *vegan, international, healthy, dining on a budget, families with children, sandwiches, breakfast/brunch, lunch, dinner, reservations, late night, dessert, takeout, fusion, vegetarian, large groups, buffet, organic* is a skyline object in STD because it is spatially and textually relevant to the query. On the other hand, SSQ prunes this object as it is spatially dominated by other 14 objects on its east. For instance, object "The Pigalle Club" in Figure 3(a) with textual description *british* spatially dominates "Tibits" since it is just a little bit closer to all the query locations compared to "Tibits". However, "The Pigalle Club" has low textual relevance to $Q.\psi$ and it is not interesting. In general, SSQ returns numerous objects that have low or zero relevance to the query keywords.

DDA and KBFF vs. STD. Unlike SSQ, models DDA and KBFF take textual relevance into account; however, they may include in the skyline numerous objects that are not interesting. DDA reports some distant skyline objects with high textual relevance. For instance, object "Pizza Hut Islington" with textual description *large groups, families with children, dining on a budget, lunch, pizza & pasta, buffet, dinner, dessert, takeout*, close to the right-upper corner of Figure 3(b), is regarded as a skyline object due to its high textual relevance. However, it is not included in the skylines of other models because of its large Euclidean distance to the query locations. Same as SSQ, DDA retrieves some spatially close but textually irrelevant objects, e.g., "Cafe Rouge St Pauls" in Figure 3(b). In fact, the result of DDA is a superset of the result of SSQ if each object has distinct coordinate, since the derived space of SSQ is a subspace of the derived space of DDA and each place has a distinct location.[1] KBFF includes many objects with low textual relevance. For instance, object "Ciao Bella" with textual description *dinner, reservations, late night, dessert, italian, breakfast/brunch, delivery, lunch* in Figure 3(c) is reported as a skyline object because of its small Euclidean distance to one of the query locations, although it has low textual relevance (it contains only keyword *dessert*). By visually comparing the four models in Figure 3, we can conclude that STD is the most effective model, as it computes a small skyline with spatially close and textually interesting objects w.r.t. the query.

3.1.2 Statistics-based Analysis

Table 2 reports some statistics about the results of the four methods. In terms of computational cost, DDA is the most expensive model (as it operates in a space of higher dimensionality), while the costs of the other model are similar. We will explain the reasons behind the efficiency of STD in Section 5. We proceed to analyze the statistics of the skylines computed by the four methods.

TABLE 2
Statistics of the Skylines by Different Models

	Time (ms)	Result Size	Spatial Coverage (%)
SSQ	225	1158	0.0118
DDA	322	1195	0.0497
KBFF	210	111	0.0147
STD	188	21	0.0131

1. For two SSQ skyline objects with identical locations, one of them can dominate the other in DDA. However, such cases are very rare.

Result Size Analysis. According to Table 2 and Figure 3, the skyline computed by STD is significantly smaller compared to the skylines found by SSQ and DDA and also much smaller compared to the skyline by KBFF. Smaller skylines are better, in general, as in practice the user can spend more time in analyzing the objects in them. In our example, the tourist would be overwhelmed by the results of SSQ, DDA, and KBFF, while we expect her to have no problem to interpret and use the result of STD. The result of SSQ is large because (i) all the objects inside the convex hull of the query locations are skyline objects [7] (no matter how textually relevant they are) and (ii) the objects outside the convex hull but near it also have high probability to be included in the spatial skyline. The result of DDA is even larger, as it is a superset of the spatial skyline by SSQ if all objects have distinct locations. Although KBFF has a keyword filtering step, the number of objects containing at least one query keyword may still be quite large; thus, the spatial skyline over a large candidate set gives a large result. On the other hand, STD computes a skyline of smaller size, because it penalizes the objects far from the query locations and the objects with low textual relevance. Hence, only a few objects have good scores based on the ranking function $st()$. In summary, compared to DDA and KBFF, STD is more powerful in distinguishing important objects with respect to both spatial distance and textual relevance, resulting in a small and useful skyline which does not overwhelm the user. At the same time, the spatial skyline alone (computed by SSQ) is too large to be useful unless the convex hull of the query locations is very small.

Spatial Coverage Analysis. We also compare the models in terms of their *spatial coverage*, i.e., ratio of the area of their skyline's MBR to the area of the map that contains all places (i.e., the area of London and its suburbs for the TripAdvisor-LDN dataset). In Table 2, we can see that the skylines of SSQ, STD, and KBFF all have small spatial coverages, since the spatial proximity plays an important role in the skyline computation. On the other hand, the spatial coverage of the skyline produced by DDA is much larger, since separately considering the spatial and textual dimensions may report distant objects with high textual relevance. Since SSQ considers only the spatial dimensions when computing the skyline, we consider the spatial coverage of the skyline computed by SSQ as a benchmark. Note that the spatial coverage of the skyline of STD is only slightly larger than that of the benchmark. Hence, we conclude that the skyline result of STD is very good in terms of spatial proximity to the set of query objects Q.

Result Quality Analysis. The objective of the STS operator is to retrieve skyline objects that are close to the query locations and relevant to the query keywords. Hence, we evaluate the quality of the skylines by the different models in terms of (a) the average textual relevance of the skyline objects, (b) the percentage of the skyline objects that are textually relevant, (c) the average SUMD (the sum of the Euclidean distances between a skyline object and the query locations), (d) the average MAXD (the maximum of the Euclidean distances between a skyline object and the query locations), and (e) the average MIND (the minimum of the Euclidean distances between a skyline object and the query locations). Table 3 displays the results of the four methods according to the above five measurements. Regarding the average textual relevance, STD is the best model and KBFF comes closely the second. The results of SSQ and DDA have very low average textual relevance. The reason is that the objects returned by STD and KBFF are all relevant to the query keywords (having non-zero textual relevance), while the percentage of the relevant objects (having non-zero textual relevance) in SSQ and DDA are only 7.51% and 10.6%, respectively. Note that, when computing the average textual relevance of the skyline objects for SSQ and DDA, we assign a low textual relevance (e.g., 0.001) to the skyline objects with zero textual relevance. With respect to the spatial proximity (SUMD, MAXD, and MIND) measures, the results of the four methods are comparable. This shows that model STD succeeds in its goal to find spatially close objects while taking the textual relevance into account.

TABLE 3
Result Quality Measurements

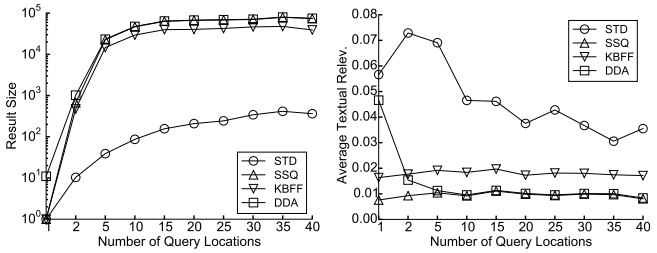
(Avg.)	T. Relev.	Relev. Obj. (%)	SUMD	MAXD	MIND
SSQ	0.00254	7.51	0.0711	0.0310	0.00774
DDA	0.00331	10.6	0.0720	0.0313	0.00783
KBFF	0.0220	100	0.0728	0.0315	0.00776
STD	0.0312	100	0.0810	0.0349	0.00595

3.2 Quantitative Evaluation

In this section, we use Flickr-LDN data to evaluate the four methods in terms of the average result size and the average textual relevance under various parameter settings. In each experiment, we evaluate the computed skylines by varying one of the following query parameters: (i) the number of query locations (default $|Q|$=10), (ii) the number of query keywords (default $|Q.\psi|$=10), and (iii) the spatial coverage of the query locations (default 0.4% of the map's area), while keeping the other parameters to their default values. For each parameter setting, we run 100 STS queries and average the statistics of the computed skylines. The queries are generated through the following steps. (i) We randomly select an object p from the dataset as center point. (ii) We randomly generate the other $|Q|-1$ query locations within a square centered at p with area no larger than the query spatial coverage constraint. (iii) The first $|Q.\psi|$ keywords of p serve as the query keywords; if p has less than $|Q.\psi|$ keywords, we randomly select other objects in the square to obtain more keywords from them until $|Q.\psi|$ keywords are collected. The test queries in Section 5 are also generated using the same strategy above. We also evaluated how the spatial coverage and other measures (such as the average SUMD) of the skyline are affected by changing the query parameters. The evaluation results (omitted due to space constraints) show that the differences between the four models with respect to these measures are insensitive to the parameters and they are as shown in Section 3.1.2.

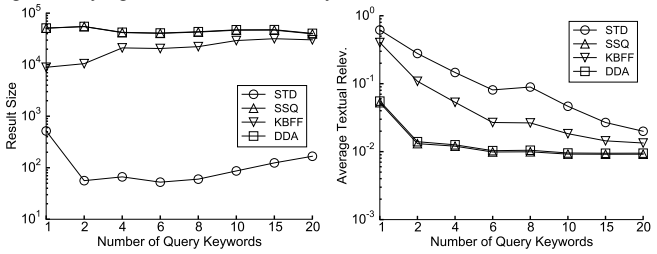
3.2.1 Varying the Number of Query Locations

Figure 4(a) displays the average size of the skyline produced by the four models when varying the number of query locations. STD produces skylines orders of magnitude smaller than those computed by the other models. As the number



(a) Result Size (b) Average Textual Relev.

Fig. 4. Varying the Number of Query Locations



(a) Result Size (b) Average Textual Relev.

Fig. 5. Varying the Number of Query Keywords

of query locations increases, the skyline sizes of all models increase. The result size of STD increases at a slower pace compared to the other three methods. In fact, we expect that typical STS queries will have less than 10 query locations; for such numbers the result of STD contains less than 100 places. The result size of KBFF is very large, which indicates that the Boolean filtering approach employed by it is not effective. The skyline result of DDA is even larger than that of KBFF (and always larger than that of SSQ, as discussed). Figure 4(b) displays the average textual relevance of the skyline objects for all methods. STD produces results of consistently higher quality compared to competitor models. In general, the textual relevance of the STD skyline decreases with the number of query locations, since more objects with low textual relevance may enter the skyline. The average textual relevance of DDA is large when there is only one query location, since DDA considers textual relevance as an additional independent dimension and the influence of this dimension is high when the number of query locations is small. The average textual relevance of the results by SSQ and DDA is low because they put none or little focus on this aspect. KBFF produces slightly better results, because it prunes totally irrelevant places; still, the included places may have low textual relevance.

3.2.2 Varying the Number of Query Keywords

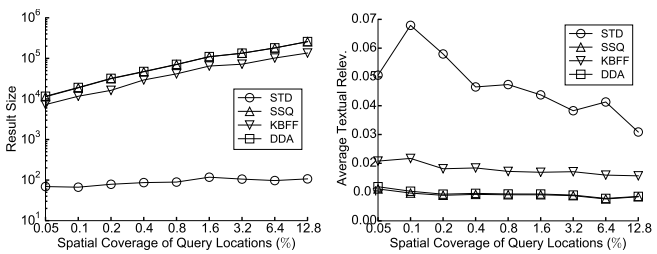
The next experiment assesses the effect of the number of query keywords $|Q.\psi|$ on the quality of the produced skylines by all models. Figure 5(a) displays the result size of the four methods. Again, the result size of STD is significantly smaller than the competitors for the reasons explained before. As Figure 5(b) shows, STD has significantly higher textual relevance compared to its competitors. The gap decreases as the number of query keywords increases, because it becomes more difficult to find results of high relevance to all the keywords. Still, we expect that the number of keywords in a typical STS query is less than 10, where STD is notably more superior than competitor models.

3.2.3 Varying the Spatial Coverage of Q

In the next experiment, we generated query workloads of various spatial coverages of the query locations Q, ranging from 0.05% to 12.8%, while keeping $|Q|$ = 10 and $|Q.\psi|$ = 10. The London area is $1572km^2$. Spatial coverage varying from 0.05% to 12.8% equals to varying from $0.786km^2$ (e.g., a square with edge length $887m$) to $201.216km^2$ (e.g., a square with edge length $14.185km$). As Figure 6(a) shows, the result size of STD is significantly smaller than those of the competitors and close to around 100, even when the query locations are very far from each other. STD puts strong requirements on both spatial and textual aspects of the query, therefore its result size is not very sensitive to the spatial coverage of Q. On the other hand, since SSQ and DDA include all the objects in the convex hull of the query locations in the skyline (while KBFF prunes only some of them), their skyline size increases in proportion to the query spatial coverage. The average textual relevance of STD is significantly larger than that of the competitors (Figure 6(b)) and the relevances of the computed skylines decrease as the spatial coverage of the query locations increases. The reason is that as the query spatial coverage increases, the spatial distances to the query locations increase and become a more important factor than textual relevance in the derived distances $st()$.

3.2.4 Tuning KBFF

In the filtering step of model KBFF, we select the objects whose textual descriptions contain at least one of the query keywords. We now experiment on varying the required number of matched keywords in selecting candidates for KBFF and testing the effect on the quality of the skyline. We set the number of query locations to 10, the number of query keywords to 10, and the spatial coverage of the query locations to 0.4%. The number of required matched keywords varies from 1 to 10. Table 4 displays the results of this experiment. The number of queries having empty skyline increases as the number of matched keywords increases. This is because there are only few objects whose textual descriptions contain all the query keywords. Hence, the skyline result size decreases dramatically. When computing the average textual relevance, for queries with empty skyline, the textual relevance is set to 0. The average textual relevance goes up and then down mainly due to the increasing number of queries with empty skyline. Although the average textual relevance remains high, a big disadvantage of KBFF is the difficulty in determining the required number of matched keywords. Small values of this threshold make KBFF less effective than STD (as shown



(a) Result Size (b) Average Textual Relev.

Fig. 6. Varying the Spatial Coverage of the Query Locations

TABLE 4
Varying the Number of Matched Keywords in KBFF

# Matched Keywords	Result Size	Textual Relevance	Queries with Empty Skyline (%)
1	29381.25	0.0184	0
2	5280.77	0.0289	0
3	1028.34	0.0427	0
4	406.36	0.0713	0
5	152.24	0.0982	3
6	41.35	0.121	14
7	7.3	0.121	30
8	3.16	0.109	49
9	0.95	0.0939	63
10	0.36	0.0804	71

in the previous experiments), while large values result in having more queries with empty skyline.

4 Algorithms for STD-based STS

In this section, we focus on designing efficient algorithms for STD, which was found in Section 3 to be the most effective STS model. A naive algorithm for computing STD is a sequential scan based algorithm (SS), based on the paradigm of [12]. Given a query, SS first materializes the values of all derived dimensions for each object. Then, all the objects are sorted in ascending order of the sum of the values of their derived dimensions. Algorithm SS scans the sorted list and checks whether each object is in the skyline by comparing it with the previously found skyline objects. The disadvantage of SS lies in the computation, materialization and sorting the values of all derived dimensions for all objects.

For the sake of efficiency, we now propose a basic and an advanced algorithm for STD. We firstly briefly review the IR-tree index, which is used by our algorithms, in Section 4.1. The basic algorithm for STD is presented in Section 4.2. The advanced and more efficient algorithm for STD, which utilizes a data structure for bookkeeping the discovered skyline objects, is described in Section 4.3.

4.1 Preliminary: the IR-Tree

Our algorithms can be applied on any hierarchical *spatio-textual* index, which facilitates efficient search based on both spatial distance and textual relevance to a reference object q (e.g., see [1], [5]). The IR-tree [3] is the state-of-the-art index in this class. Figure 7 illustrates the basic structure of an exemplary IR-tree. In our context, the objects from $\mathcal{P}$ are grouped into the leaf nodes of the index; each leaf node also has a pointer to an inverted file, which indexes the text descriptions of all the objects in it. An inverted file has two main components: (i) a *vocabulary* of all distinct terms appearing in the descriptions of objects and (ii) a *posting list* for each term t, i.e., a sequence of pairs (id, w), where id is the identifier of the object in the leaf node whose description contains t and w is the weight of t in the description. Each non-leaf node contains the MBRs of its children nodes and also a pointer to an inverted file indexing the *pseudo-descriptions* of its children nodes (i.e., for each node, the union of all text descriptions of the objects indexed under it). Given a query q, comprising of a spatial location and a textual description $q.\psi$, the MBR of an IR-tree entry can be used to compute a lower bound of the Euclidean distance between q and any object indexed under the entry. The pseudo-description of the entry can be used to compute an upper bound on the textual relevance between $q.\psi$ and any objects indexed under the entry.

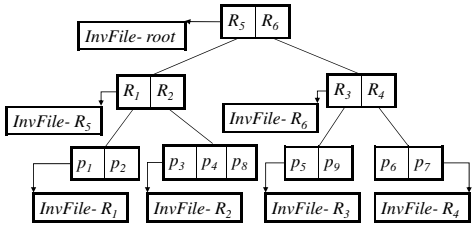


Fig. 7. IR-Tree

4.2 Basic Algorithm for STD

We first introduce definitions and lemmas that can facilitate pruning when using the STD model in Section 4.2.1. The basic algorithm is presented in Section 4.2.2.

4.2.1 Definitions and Lemmas

In order to compute the skyline, we must apply a *dominance test* for each object. The dominance test compares the target object with other objects in all dimensions. The objects that pass this test (i.e., they are not dominated by other objects) are reported as the skyline objects. The objects that fail the test are non-skyline objects and they are pruned. Since dominance tests can be computationally expensive, we identify *dominance regions* for objects, based on Q, and simplify the test for an object to a simple inclusion test in the dominance regions of other objects.

Specifically, for any object $p \in \mathcal{P}$ and query location $q_k \in Q$, let $C^p_{q_k}$ be the circle centered at q_k with radius $st(q_k, p)$. Let $\mathcal{U}$ be the MBR of the whole data set $\mathcal{P}$. Then:

Definition 3. The **uncertainty region** of an object p is $R_u(p) = \bigcup_{q_k \in Q} C^p_{q_k}$. The **dominance region** of an object p is $R_d(p) = \mathcal{U} - R_u(p)$.

Lemma 1. Given a query Q, the objects inside the dominance region $R_d(p)$ of p are dominated by p, i.e., $\forall p' \in R_d(p), p \ dom_Q \ p'$.

Proof: For any $p' \in R_d(p)$, we have $\forall q_k \in Q, d(q_k, p') > st(q_k, p)$. Since $w(Q.\psi, p'.\psi) \in [0, 1]$, $st(q_k, p') = d(q_k, p')/w(Q.\psi, p'.\psi) \geq st(q_k, p)$. Therefore, $\forall p' \in R_d(p), p \ dom_Q \ p'$. □

On the other hand, for any $p' \in R_u(p)$, we cannot confirm whether p dominates p'.

Example 4 In Figure 8(a), given query Q with three query locations q_1, q_2, and q_3, the three circles are $C^{p_1}_{q_1}, C^{p_1}_{q_2}$, and $C^{p_1}_{q_3}$, respectively. The white region is the uncertainty region $R_u(p_1)$ of p_1 and the gray region is the dominance region $R_d(p_1)$ of p_1. Since object $p' \in R_d(p_1)$, p' is dominated by p_1. Similarly, p'' is dominated by p_1 according to Lemma 1.

The shapes of the uncertainty and dominance regions of an object are irregular. For the sake of easy implementation and efficiency, we approximate the uncertainty region $R_u(p)$

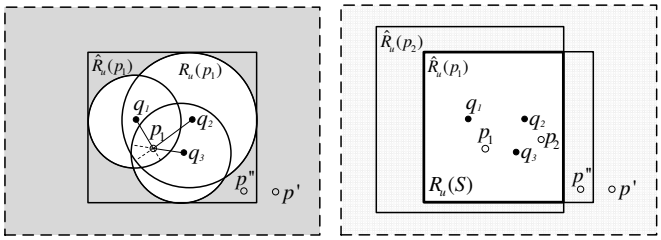
of a point p by its MBR $\hat{R}_u(p)$, as illustrated by the solid rectangle in Figure 8(a). This way, the dominance region can be approximated by $\hat{R}_d(p) = \mathcal{U} - \hat{R}_u(p)$. Obviously, Lemma 1 holds for $\hat{R}_d(p)$, since $\hat{R}_d(p) \subseteq R_d(p)$. In Figure 8(a), since $p' \in \hat{R}_d(p_1)$, p' is dominated by p_1. On the other hand, $p'' \in \hat{R}_u(p_1)$, which means that we cannot determine whether p'' is dominated or not solely by applying Lemma 1 using $\hat{R}_d(p_1)$.

Lemma 1 can be used to prune a non-skyline object p' based on the dominance region of a single object p. We now introduce Lemma 2 that prunes non-skyline objects using the dominance region of a *set* of objects. The dominance region of a set is larger than that of a single object, therefore its pruning effectiveness is higher.

Lemma 2. Given a query Q, consider a set of objects $S = \{p_1, p_2, \dots\}$. The uncertainty region of S is $R_u(S) = \bigcap_{p_i \in S} \hat{R}_u(p_i)$. The dominance region of S is $R_d(S) = \mathcal{U} - R_u(S)$. The objects inside $R_d(S)$ are not part of the skyline.

Proof: Suppose $\exists p' \in R_d(S)$, such that p' is not dominated by any object in S. This means that $\forall p_i \in S, p' \in R_u(p_i) \implies p' \in \bigcap_{p_i \in S} R_u(p_i) \subset \bigcap_{p_i \in S} \hat{R}_u(p_i) = R_u(S)$. This contradicts the assumption that $p' \in R_d(S) = \mathcal{U} - R_u(S)$. □

Example 5 Figure 8(b) shows a set of objects $S = \{p_1, p_2\}$ and a query Q with three locations q_1, q_2, and q_3. The approximate uncertainty regions of p_1 and p_2 are the two solid rectangles $\hat{R}_u(p_1)$ and $\hat{R}_u(p_2)$. The uncertainty region $R_u(S)$ of S is the white region, i.e., $\hat{R}_u(p_1) \cap \hat{R}_u(p_2)$. The dominance region $R_d(S) = \mathcal{U} - R_u(S)$ of S is the gray region. Obviously, $\forall p_i \in S, R_d(p_i) \subseteq R_d(S)$. Any object p' located in $R_d(S)$ is not a skyline object. The same holds for p'', while, as we have seen before, p'' cannot be pruned solely by $\hat{R}_d(p_1)$.



(a) Lemma 1 (b) Lemma 2

Fig. 8. Lemma Examples

4.2.2 Algorithm BSTD

We propose a basic algorithm BSTD for computing the STD-based spatio-textual skyline. It adopts the IR-tree and utilizes the lemmas introduced in the previous section. Algorithm 1 shows a pseudo code of BSTD. Initially, the skyline result S is empty. B is the uncertainty region of the set of skyline objects found so far, initialized to be the entire space $\mathcal{U}$, i.e., to contain the whole data set $\mathcal{P}$. B is used to prune non-skyline objects, according to Lemmas 1 and 2. Region B is updated whenever a new skyline object is found (lines 10 and 19). The algorithm processes the objects in ascending order of $\sum_{q_i \in Q} st(q_i, p)$ with the help of a minimum heap H.

The elements in H can be leaf entries (i.e., objects) or non-leaf entries of IR-tree. Let $e = \langle \Omega, \psi \rangle$ be a non-leaf entry in the IR-tree where Ω is the MBR bounding the objects in the subtree pointed to by e and ψ is the pseudo text description of e. According to the properties of the IR-tree, given query Q, the minimum Euclidean distance $d(q_i, \Omega)$ between q_i and Ω is a lower bound on the Euclidean distance between q_i and the objects bounded in Ω, while $w(Q.\psi, \psi)$ is a upper bound on the textual relevance of the objects inside. Thus, we can associate each non-leaf entry e in the IR-tree with $|Q|$ derived dimensions. The value of each dimension is $d(q_i, \Omega)/w(Q.\psi, \psi)$ (i.e., $st(q_i, e)$). Obviously, the derived dimensions of the non-leaf entry stand for the best case of the objects in the subtree. Then e can be ordered in H based on $\sum_{q_i \in Q} d(q_i, \Omega)/w(Q.\psi, \psi)$.

The first object reported from the heap is guaranteed to be a skyline object [12] (lines 8–10). In the beginning, the root of the IR-tree is added to heap H. Algorithm BSTD then repeats the following procedure until the heap is empty. The first element e is removed from the heap and evaluated. It may refer to an object or a non-leaf entry that points to a subtree of objects. If e does not intersect B, meaning that the object(s) referred by e are dominated by the current found skyline objects and can be pruned (line 6). Otherwise, if e refers to an object, it is checked against each of the found skyline objects. If none of the skyline objects dominate it, the object is added to the skyline. Otherwise, it is pruned (lines 11–19). If e is a non-leaf entry, the node N pointed to by e is loaded. For each entry e' in N, if e' does not intersect B, it is pruned. Otherwise, it is added to the heap (lines 20–24).

Correctness. The objects are evaluated in ascending order of $\sum_{q_i \in Q} st(q_i, p)$. Hence, the objects accessed later cannot dominate the objects found earlier (see [12], [13]). Thus, for the currently accessed object from the heap, if it is not dominated by the currently found skyline objects, it is guaranteed to be in the skyline. The algorithm discovers skyline objects progressively and in the end returns the correct result.

Analysis. Although the algorithm can prune objects using the uncertainty region B, for the objects that cannot be pruned, an expensive dominance test has to be conducted, comparing with all the derived dimensions of all the current found skyline points. For example, assume that the algorithm has found $m = 1000$ skyline objects so far, each of which has $n = 10$ derived dimensions. If there are $l = 10000$ objects that cannot be pruned, the algorithm will conduct $m \times n \times l = 10^8$ comparisons.

4.3 Advanced Algorithm for STD

We now propose an advanced algorithm that utilizes an in-memory data structure, the *skyRtree*, and three pruning rules, so that the I/O and CPU cost during the verification of candidate skyline points is reduced.

4.3.1 skyRtree

The skyRtree is an in-memory R-tree that indexes all the discovered skyline objects in the space of $|Q|$ derived dimensions (e.g., see Figure 2(d)). When a new skyline object is identified, it is inserted into the skyRtree.

Algorithm 1 BSTD(Query Q, Dataset $\mathcal{P}$, Index $IRTree$)

```
1: S = ∅; B = U
2: MinHeap H = ∅
3: Add root of IRTree to H
4: while H is not empty do
5:     e = deHeap(H)
6:     if e.MBR ∩ B ≠ ∅ then                          ▷ Lemmas 1 and 2
7:         if e is an object then
8:             if S = ∅ then
9:                 Add e to S
10:                B = B ∩ R_u(e)
11:            else
12:                isSkyline = True
13:                for each p ∈ S do
14:                    if p dom_Q e then
15:                        isSkyline = False
16:                        break
17:                if isSkyline then
18:                    Add e to S
19:                    B = B ∩ R_u(e)
20:        else                                       ▷ e is a non-leaf entry
21:            N = readNode(e)
22:            for each entry e' in N do
23:                if e'.MBR ∩ B ≠ ∅ then             ▷ Lemmas 1 and 2
24:                    Add (e', Σ_{q_i∈Q} st(q_i, e')) to H
25: return S
```

4.3.2 Pruning using the skyRtree

The MBRs of the skyRtree nodes serve as summaries for the skyline objects stored in the subtrees defined by them. Given query Q, let $S = \{p_1, p_2, \dots\}$ be a group of already discovered skyline objects indexed by a skyRtree node. Let Λ be the MBR of S in the $|Q|$-dimensional derived space (i.e., the MBR of a skyRtree node). Let p^u_Λ be the upper-most corner of Λ, defined by the maximum values in all $|Q|$ dimensions of the skyline objects. Formally, the kth value of p^u_Λ is equal to $\max_{\forall p_i \in S} st(q_k, p_i)$. Proposition 1 utilizes p^u_Λ to prune non-skyline objects.

Proposition 1. If object p is dominated by p^u_Λ, p is not a skyline object.

Proof: Point p^u_Λ stands for the worst case of the skyline objects inside MBR Λ. If an object is dominated by p^u_Λ, it must be dominated by all the skyline objects inside the MBR. □

The pruning power of an MBR at a lower level, using Proposition 1, is stronger than that of an MBR at a higher level. The proposition allows us to compare an object p with the upper-most corner of an MBR Λ to prune p, rather than performing comparisons between p and every skyline object inside Λ.

Let p^l_Λ be the lower-most corner of MBR Λ bounding S. Formally, the k-th value of p^l_Λ is equal to $\min_{\forall p_i \in S} st(q_k, p_i)$. Proposition 2 uses p^l_Λ of the *root MBR* of the skyRtree to confirm whether an object is in the skyline.

Proposition 2. Let p^l_r be the lower-most corner of the root MBR. If p^l_r cannot dominate object p, p is a skyline object.

Proof: Point p^l_r stands for the best case of all the discovered skyline objects. If an object cannot be dominated by p^l_r, none of the discovered skyline objects can dominate it. Hence, it is a skyline object. □

Proposition 3 uses p^l_Λ of the MBRs of non-leaf skyRtree entries to avoid comparing the skyRtree points indexed under them with candidate skyline objects.

Proposition 3. Let p^l_Λ be the lower-most corner of the MBR bounding S. If p^l_Λ does not dominate a candidate skyline object p, then p is not dominated by any skyline object in S and needs not be compared with them.

The proof is trivial and omitted. The common goal of Propositions 1, 2, and 3 is to reduce the comparisons (dominance tests) required for any candidate skyline object p. Proposition 1 can also be applied to prune a *set* of objects, whose MBR in the derived space is dominated by p^u_Λ. We now proceed to introduce the derived dimensions of the non-leaf entries in the IR-tree. Using this transformation, an IR-subtree containing a set of objects may be pruned using Proposition 1.

As stated in Section 4.2.2, every non-leaf entry e in the IR-tree can be associated with $|Q|$ derived dimensions that stand for the best case of the objects in the subtree. Proposition 1 can then be easily applied on the derived dimensions of non-leaf entries to prune IR subtrees.

4.3.3 Algorithm ASTD

We now present an advanced algorithm (ASTD), which computes the skyline based on the STD model. ASTD follows the framework of algorithm BSTD, but it utilizes the skyRtree to save computational cost. Algorithm 2 is a pseudocode of ASTD.

Algorithm 2 ASTD(Query Q, Dataset $\mathcal{P}$, Index $IRTree$)

```
1: S = ∅; B = U
2: MinHeap H = ∅
3: skyRtree = ∅
4: Add root of IRTree to H
5: while H is not empty do
6:     e = deHeap(H)
7:     if e.MBR ∩ B ≠ ∅ then                          ▷ Lemmas 1 and 2
8:         if e is an object then
9:             if S = ∅ or ¬ SKYRTREEPRUNES(e) then
10:                Add e to S
11:                B = B ∩ R_u(e)
12:                Insert e into skyRtree
13:        else                                       ▷ e is a non-leaf entry
14:            Compute the derived dimensions of e
15:            if ¬ SKYRTREEPRUNES(e) then
16:                N = readNode(e)
17:                for each entry e' in N do
18:                    if e'.MBR ∩ B ≠ ∅ then         ▷ Lemmas 1 and 2
19:                        Add (e', Σ_{q_i∈Q} st(q_i, e')) to H
20: return S
```

Each time a new skyline object is discovered, it is inserted into the skyRtree (line 12). Function SKYRTREEPRUNES (Algorithm 3) is used to check whether a single object or a subtree can be pruned (lines 9 and 15). It utilizes the skyRtree and the pruning techniques of Section 4.3.2 to decide whether the input parameter p (a single object or a set of objects in a subtree) can be dominated or not. The skyRtree indexes all the skyline objects known so far. Function SKYRTREEPRUNES returns true if the input p is

dominated. The skyRtree is traversed in a breadth first manner. In the beginning, the input parameter p is compared with the corners p_r^l and p_r^u of the root MBR. If p_r^l does not dominate p, then none of the skyline objects in the skyRtree can dominate p and false is returned according to Proposition 2. If p_r^u dominates p, then all the skyline objects in the skyRtree can dominate p and true is returned according to Proposition 1. If neither of the above cases holds, a $Queue$ is created containing the root of the skyRtree.

SKYRTREEPRUNES repeats the following procedure until $Queue$ is empty. The first element N in $Queue$ is removed for evaluation. If N is a non-leaf node, for each entry e in N, if p_e^u of e dominates p, then p must be dominated by the skyline points in the subtree rooted at e according to Proposition 1. Therefore, the function returns true. Otherwise, if p_e^l of e dominates p, which means there may exist skyline objects in the subtree rooted at e that dominate p, e is added to the end of $Queue$ for further exploration. If p_e^l does not dominate p, the subtree rooted at e can be discarded without any further processing according to Proposition 3. If N is a leaf node, all the entries in N are skyline objects and we check whether p is dominated by these skyline objects. In this case, the function returns true. When $Queue$ becomes empty, this means that p is not dominated by any skyline object and false is returned.

Algorithm 3 SKYRTREEPRUNES(p)

```
1: Get p_r^l and p_r^u of the root MBR in the skyRtree
2: if p_r^l not dom_Q p then return False                ▷ Proposition 2
3: if p_r^u dom_Q p then return True                     ▷ Proposition 1
4: Add the root of skyRtree to Queue
5: while Queue is not empty do
6:     Remove the first element N from Queue
7:     if N is a non-leaf node of skyRtree then
8:         for each entry e in node N do
9:             Get p_e^l and p_e^u of e
10:            if p_e^u dom_Q p then return True          ▷ Proposition 1
11:            if p_e^l dom_Q p then
12:                Add e to Queue                         ▷ Proposition 3
13:    else
14:        for each entry e in node N do
15:            if e dom_Q p then return True
16: return False
```

5 EFFICIENCY ANALYSIS

In this section, we evaluate the efficiency of BSTD and ASTD under various parameter settings, i.e., varying the number of query locations, the number of query keywords, the spatial coverage of query locations, and the database size.

5.1 Methods

BSTD and ASTD can be applied on any hierarchical hybrid index which facilitates search based on Euclidean distance and textual relevance. We adopt the state-of-the-art IR-tree [3]; we assert that the improvement of ASTD over BSTD is similar if alternative indexes are used instead. We denote the version of BSTD that applies on the original IR-tree by BSTD-IR. However, by experimentation, we observed that BSTD and ASTD on the IR-tree incur too many I/O accesses. The reason is that for each IR-tree node, there is a separate inverted file. This means that, during STS computation, for each accessed node N and for each term t in $Q.\psi$ we need (i) a random access to locate the posting list of t in the file of N and (ii) another random access to access the posting list. Since these posting lists are quite short, the incurred I/O cost is not compensated by the information processed. In order to alleviate this problem, we designed and implemented a variant of the IR-tree, the IR$^+$-tree, which includes only one inverted file. For each term t, there is a single posting list indexing all (non-leaf and leaf) entries, for which the (pseudo-)descriptions contain t. Given the query keywords $Q.\psi$, before traversing the IR$^+$-tree structure, we locate the $|Q.\psi|$ posting lists and load them into memory. Each posting list is implemented as a hash table whose keys are the ids of entries and the values are the corresponding term weights. Thus, when a node N is accessed, we can easily get the term weights for N's entries, without incurring any I/O accesses. We denote the implementations of the proposed algorithms that apply on the IR$^+$-index simply by BSTD and ASTD.

5.2 Datasets and Settings

For the evaluation we used two datasets, Flickr-LDN (described in Section 3) and Flickr-NY, which contains 1,505,243 pictures in New York crawled from Flickr with a dictionary of 726,958 terms. Each picture is associated with a GPS coordinate and a set of terms extracted from its tags and title. The average number of terms per object is 10.51. All tested methods were implemented in Java and the experiments were conducted on a 3.4 GHz quad-core machine running Ubuntu 12.04 with 16 GBytes memory. The page size of the IR-tree is 8KB that is translated to a fanout of 184. An LRU buffer that caches at most 500 pages is used. In order to remove the caching effects by the operating system, we drop the system cache before executing each query. We generated queries by varying the number of query locations (default $|Q|$ = 10), the number of query keywords (default $|Q.\psi|$ = 10), and the spatial coverage of the query locations (default 0.4%). We measure the average runtime, I/Os, and number of dominance tests over generated workloads of 100 queries for each parameter setting.

5.3 Experiments

5.3.1 Varying the Number of Query Locations

In the first experiment, we vary the number $|Q|$ of query locations from 1 to 40, while keeping the other parameters to their default values. As Figure 9 shows, ASTD is 2 to 4 times faster than BSTD, while BSTD outperforms BSTD-IR by an order of magnitude. The I/O costs of the three methods are proportional to the runtime. ASTD conducts 2 to 10 times fewer dominance tests compared to BSTD. This is due to use of skyRtree by ASTD and its pruning power. BSTD-IR performs the same number of dominance tests as BSTD, since they are based on the same algorithm. BSTD is faster than BSTD-IR, because of the compact storage of the posting lists in the inverted file and the efficient strategy adopted for locating the required posting lists, as described in Section 5.1. The runtime, I/O cost, and the number of dominance tests of all methods increase as the number of

query locations increases since the dimensionality of the problem increases and so does the result size (shown in Figure 4(a)).

5.3.2 Varying the Number of Query Keywords

We now test the effect of varying the number $|Q.\psi|$ of query keywords. As Figure 10(a) shows, the runtime of all methods in general increases with the number of query keywords. BSTD is faster than BSTD-IR by a factor of around 10. ASTD is 3 to 4 times faster than BSTD, which is consistent with the improvement in the I/O cost and the number of dominance tests shown in Figures 10(b) and 10(c), respectively. ASTD outperforms BSTD by a factor of (i) 3 to 6 in terms of I/O cost and (ii) 3 to 20 in terms of dominance tests. This is because the skyRtree prunes objects and groups of objects, avoiding unnecessary dominance tests and I/O accesses.

The cost of ASTD slightly decreases when the keywords increase from 1 to 2 and then increases. When $|Q.\psi|$=1, many objects have high textual relevance, which leads to relatively high I/O cost (Figure 10(b)), large number of dominance tests (Figure 10(c)), and large result size (Figure 5(a)). Consequently, the computational and I/O cost of ASTD is relatively high. Given two query keywords, much fewer objects have high textual relevance and these dominate numerous other objects, decreasing the cost of ASTD and the size of the skyline. Interestingly, for $|Q.\psi|$ >2, the discriminative power of the textual relevance becomes weaker, which conversely leads to a larger number of skyline candidates, higher I/O cost and more dominance tests. Therefore, the runtimes of BSTD-IR and BSTD increase monotonically with $|Q.\psi|$ because these methods have high I/O cost, which dominates their savings in dominance tests from $|Q.\psi| = 1$ to $|Q.\psi| = 2$.

5.3.3 Varying the Spatial Coverage of Q

We vary the spatial coverage of the query locations from 0.05% to 12.8%, while keeping $|Q| = 10$ and $|Q.\psi| = 10$. In Figure 11(a), ASTD is 3 to 4 times faster than BSTD, which is more than 10 times faster than BSTD-IR. The runtime slightly increases when the spatial coverage increases, which is consistent with I/O performance (Figure 11(b)) and the number of dominance tests (Figure 11(c)). In Figure 11(b), the I/O difference between BSTD-IR and BSTD is due to the use of the IR$^+$-tree by BSTD, while the 5 times I/O reduction from BSTD to ASTD is due to the pruning power of the skyRtree. In Figure 11(c), the number of dominance tests by ASTD is 2–4 times lower than that of BSTD. The number of dominance tests decreases when spatial coverage varies from 1.6 to 12.8. The reason is that as the spatial coverage of the query locations increases, most skyline objects are near the query locations; thus, a lot of distant objects are dominated and, consequently, the number of dominance tests is reduced. On the other hand, the increase of the spatial coverage causes higher I/O cost (Figure 11(b)), since the query locations cover a larger area and more index nodes need to be loaded to evaluate the objects around them.

5.3.4 Scalability

We randomly sample 100K, 500K, and 1M objects from Flickr-NY data set to form data sets of various sizes for scalability evaluation purposes. The other parameters are fixed ($|Q| = 10$, $|Q.\psi| = 10$, and 0.4% spatial coverage of Q). Figure 12(a) compares the runtimes of the three methods for various data sizes. The runtime increases linearly as the database size increases and the algorithms maintain their relative performance difference. BSTD needs 3 to 4 times more I/Os and 2 to 7 times more dominance tests than ASTD does on large data (as shown in Figures 12(b) and 12(c)). This indicates that the skyRtree is powerful for pruning objects. Table 5 shows the result size and the spatial coverage of the skylines of the four datasets. Note that the result size and the spatial coverage of the skyline increases at a much lower pace compared to the database size, which is consistent with our observation that the STD model produces skylines of manageable size. Due to this effect, the memory overhead of the skyRtree is negligible (at most 10KB in all settings).

TABLE 5
Skyline Size and Spatial Coverage

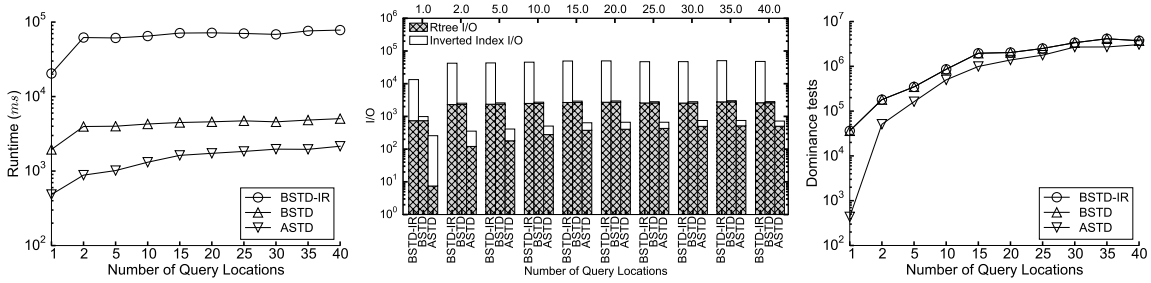
Data Set Size	100K	500K	1000K	1500K
Skyline Size	84.77	200.6	334.8	429.1
Spatial Coverage (%)	0.114	0.130	0.127	0.139

6 Related Work

Previous work can be classified into static and dynamic skyline computation. A static skyline is computed from a set of objects with static dimensional values, which do not depend on a query input. In contrary, a dynamic skyline is computed based on *derived* dimensional values for the objects w.r.t. some query input.

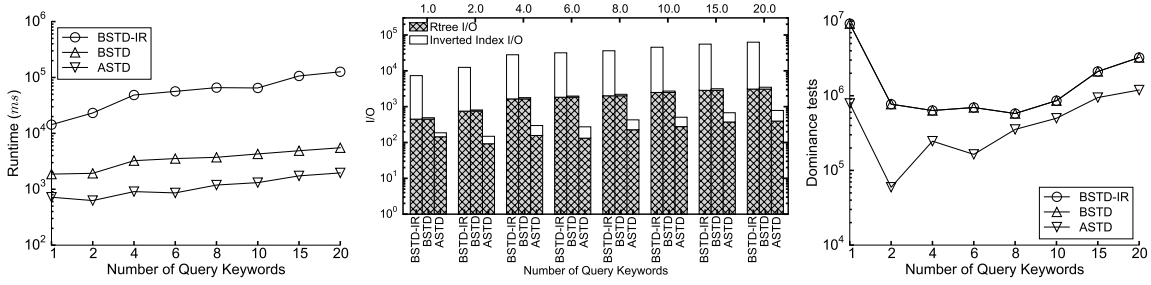
Static Skyline. The skyline query was first studied by the database community in [6], where a *divide and conquer* (D&C) approach was compared to a *block nested loop* (BNL) method for non-indexed data. In order to reduce the cost of skyline evaluation over large datasets, *sort filter skyline* (SFS) [12] sorts all the objects before applying BNL; the sort order guarantees an object can only dominate other objects that follow it in the order. SaLSa [14] and LESS [13] are optimized versions of SFS. The *object-based space partitioning* (OPS) algorithm [15] reduces the dominance checks during skyline computation by organizing the skyline found so far in a *left-child/right-sibling* tree. The skyRtree used by ASTD in this paper shares the same motivation, but uses a different indexing technique and pruning rules compared to OPS.

Static skyline techniques have also been designed for indexed data, e.g., [6], [16], [17], [18], [19]. Besides proposing BNL, Borzsonyi et al. [6] also discuss how to computing a static skyline from data indexed by a B-tree or an R-tree. The *branch and bound skyline* (BBS) algorithm [16] operates on an R-tree and identifies skyline objects progressively by accessing nodes and objects according to their distances to the origin of the axes. BBS is shown to be I/O optimal and superior to previous approaches. BSTD and ASTD follow the intuition of BBS but are for dynamic spatio-textual skyline computation that is beyond static skyline. ZBtree [18] indexes objects based on the Z-order curve. This order helps to avoid redundant dominance checks. The keyword-matched skyline query [11] computes a static skyline using



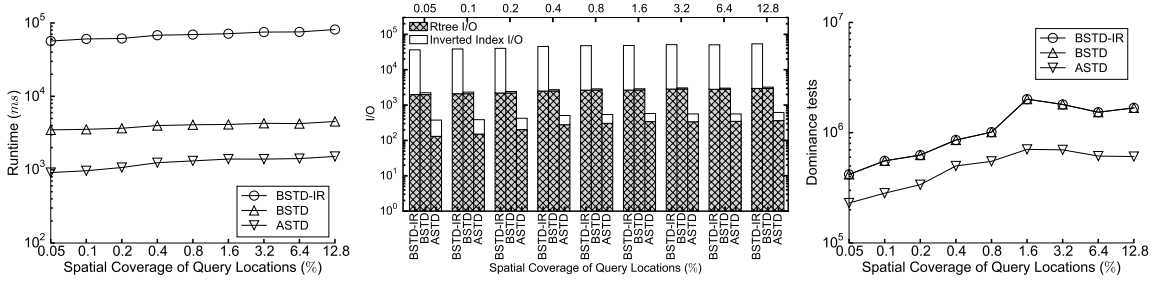
(a) Runtime (b) I/O (c) Dominance Tests

Fig. 9. Varying the Number of Query Locations



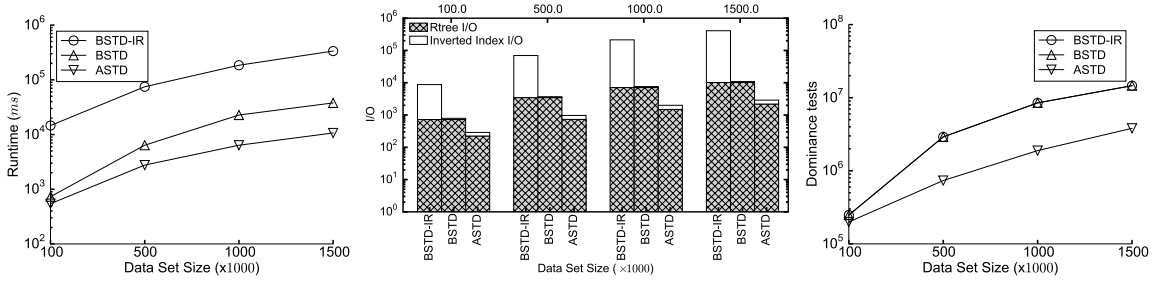
(a) Runtime (b) I/O (c) Dominance Tests

Fig. 10. Varying the Number of Query Keywords



(a) Runtime (b) I/O (c) Dominance Tests

Fig. 11. Varying the Spatial Coverage of Query Locations



(a) Runtime (b) I/O (c) Dominance Tests

Fig. 12. Varying the Database Size

only the objects that qualify a boolean keyword search. This is similar to our proposed KBFF model, which has been proved less effective than STD.

Dynamic Skyline. Dynamic skyline computation finds important applications in business planning, trip advising, and recommender systems. It has been studied in various spaces, e.g., Euclidean space [7], metric space [20], [21], transportation/road networks [22], [23], and large graphs [24]. The most relevant work to ours is the spatial skyline query (SSQ) [7]. In our work, we extend the SSQ model to integrate dynamically computed distances to the query locations with dynamically computed textual relevance to a set of query keywords. In order to compute SSQ efficiency, precomputed indexes based on Voronoi Diagrams have been used in [7], [25]. However, the properties of Voronoi Diagrams do not hold in our integrated spatio-textual derived space. Our experimental analysis (Section 3), besides showing that the spatial only skyline computed by SSQ does not consider textual relevance, demonstrates that our STD model computes a skyline, which is small in size and thus more useful.

Preference queries on multi-cost transportation networks can be modeled as a dynamic skyline problem, by considering the multiple costs from a query point to the nodes of the network [22]. A specialized method for computing the spatial skyline in a grid network, based on Manhattan distances is proposed in [26]. Given multiple query locations in a road network, a spatial skyline where Euclidean distance is replaced by shortest path distance can be computed [23]. A filter-and-refine framework with pruning

rules and a carefully-designed index structure [24] was proposed for such dynamic skylines in large graphs based on shortest path distance. Dynamic skyline queries in metric spaces where the triangular inequality holds were studied in [20]; efficient pruning techniques using Metric indexes (e.g., M-tree [21]) were proposed and used. Two optimization techniques, called dynamic indexing and k-dispersion extensions were proposed in [27] to further improve the efficiency of dynamic skyline computation in metric spaces. The distance measurement $st()$ in our problem does not obey the triangular inequality. Therefore, these techniques are not applicable to our problem.

The collaborative expansion based algorithms used in [22], [23] consider each query location in a round-robin manner and expand the search space around them until the first skyline object is discovered. These algorithms guarantee that all the skyline objects are included in the explored space when the first skyline object is determined. Then, in a shrinking stage, the candidates in the explored space are refined to obtain the final skyline results. Collaborative expansion is not efficient for our problem, since it requires retrieving all the possible candidates when the first skyline is discovered, which means that a large percentage of the search space has to be accessed (i.e., high I/O cost), without any pruning. Besides, an expensive verification phase is required.

The spatial skyline operator has also been extended for objects with augmented type information [28]. Specifically, given a set of objects belonging to m types, for each object o, the value of its i-th derived dimension is the minimum Euclidean distance between o and all objects belonging to type i. The skyline result is evaluated on the m-dimensional space. The Location-based Textual Skyline (LTS) query [29] has a similar intuition to our spatio-textual skyline; it is based on the preferences on the spatial distance to a user's location and relevance to the a set of keywords specified by the user. LTS is similar to the DDA model studied in this paper, which considers spatial and textual dimensions separately. An important difference is that LTS only considers a single query location. Another work relevant to our DDA model is the spatial skyline with *static* non-spatial attributes [7]. The difference is that textual relevance in our problem is dynamically calculated based on query keywords, making it impossible to apply a one-time general skyline computation independently to the query on the static non-spatial attributes to get a subset of the result as [7] did. Direction-based spatial skyline (DSS) queries [30] find the result not only by comparing the distances to the query point but also considering the directions of the query point. DSS queries also take only one point as a query. A recommendation algorithm [31] suggests items, such as restaurants, to a mobile user based on skyline queries, taking into account user's current location and preferences. It also takes only one point as query. The above techniques differ from our proposal in both the application scenario and problem definition. On the other hand, they support our argument that integrating non-spatial attributes into the spatial skyline finds many important applications in place recommendation, trip planning, and advertisement.

A related query to dynamic skylines, finding application in multi-decision making, is the Group Nearest Group (GNG) query [32]. Given a data set D, a query point set Q, and an integer k, GNG finds a subset of ω ($|\omega| \leq k$) of points in D, such that the total distance from all points in Q to the nearest point in ω is not greater than any other subset ω' ($|\omega'| \leq k$) of points in D. The Reverse Skyline Query [33] returns the objects whose dynamic skyline contains a given query object. Finally, a cache-aware algorithm [34] uses the results of past dynamic skyline queries to help reducing the computation cost of future queries.

Spatial keyword Search. Spatial keyword search extends classic keyword search to retrieve objects (e.g., documents) considering relevance to a set of input keywords as well as proximity to the location of the query issuer. This problem has been extensively studied during the past decade. A comprehensive comparison of indexing techniques for spatial keyword search appears in [2]. The IR-tree [3], one of the most popular indexes, supports the ranking of objects based on a weighted sum of spatial distance and textual relevance. In our work, we use the aggregate distance proposed in [4], which is parameter-free and has no normalization problems compared to weight sum. Several variants to the basic spatial-keyword search problem, include boolean spatial keyword search [35], [36], collective spatial keyword search [37], joint spatial keyword queries [1], and continuously moving spatial keyword queries [4]. The similarity join was extended in [38] to have spatio-textual relevance as the join predicate. Our spatio-textual skyline query takes as input a set of query locations and a set of query keywords and returns the objects which are not dominated by others based on their distance to the query locations and their relevance to the query keywords. Our problem definition and algorithms differ significantly from previously studied spatial keyword search problems and their solutions. To the best of our knowledge, no previous work has studied the combination of spatial skyline queries and keyword search.

7 Conclusion

In this paper, we proposed a novel Spatial-Textual Skyline (STS) operator, which integrates textual relevance into the spatial skyline, so that the retrieved objects are spatially close and textually relevant to the query input. To implement STS, we propose and study three models DDA, KBFF, and STD. After a thorough analysis of the results produced by these models, we recommend STD as the most effective one. Specifically, the skyline objects returned by STD are spatially close to the query locations and textually relevant to the query keywords. Moreover, the skyline size of STD is significantly smaller compared to the ones of the other models (and the spatial-only skyline). Besides showing the effectiveness of STD, we also propose algorithms for computing the STD-based skyline efficiently. A basic algorithm BSTD adopts a hierarchical hybrid index (IR-tree is used as a demonstration) and pruning rules to discover skyline objects progressively. We also propose an advanced algorithm ASTD that uses a main-memory skyRtree to organize the discovered skyline objects and employs a number of pruning rules to reduce the dominance tests and I/O accesses incurred by BSTD. Our experimental evaluation confirms the efficiency of ASTD over BSTD.

In the definition of STS studied in this paper, all query locations share the same set of query keywords $Q.\psi$. In the future, we plan to consider a variant of STS, where each query location has its own query keywords and the textual relevance to each query location is computed independently. In this case, textual relevance varies in each of the derived dimensions. It would be interesting to compare this variant with our STS definition qualitatively and also extend our pruning rules and algorithms to apply for this variant. In addition, we plan to explore alternative DDA-like models which augment to SSQ alternative dimensions for textual relevance and test their effectiveness.

REFERENCES

[1] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE TKDE*, vol. 24, no. 10, pp. 1889–1903, 2012.

[2] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *PVLDB*, vol. 6, no. 3, pp. 217–228, 2013.

[3] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.

[4] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *ICDE*, 2011, pp. 541–552.

[5] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *The VLDB Journal*, vol. 21, no. 6, pp. 797–822, Dec. 2012.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001, pp. 421–430.

[7] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *VLDB*, 2006, pp. 751–762.

[8] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[9] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[10] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR*, 1998, pp. 275–281.

[11] H. Choi, H. Jung, K. Y. Lee, and Y. D. Chung, "Skyline queries on keyword-matched data," *Inf. Sci.*, vol. 232, pp. 449–463, 2013.

[12] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting: Theory and optimizations," in *IIPWM*, 2005, pp. 595–604.

[13] P. Godfrey, R. Shipley, and J. Gryz, "Algorithms and analyses for maximal vector computation," *VLDB J.*, vol. 16, no. 1, pp. 5–28, 2007.

[14] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Trans. Database Syst.*, vol. 33, no. 4, 2008.

[15] S. Zhang, N. Mamoulis, and D. W. Cheung, "Scalable skyline computation using object-based space partitioning," in *SIGMOD*, 2009, pp. 483–494.

[16] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.

[17] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *VLDB*, 2002, pp. 275–286.

[18] K. C. K. Lee, B. Zheng, H. Li, and W. Lee, "Approaching the skyline in Z order," in *VLDB*, 2007, pp. 279–290.

[19] K. Tan, P. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *VLDB*, 2001, pp. 301–310.

[20] L. Chen and X. Lian, "Dynamic skyline queries in metric spaces," in *EDBT*, 2008, pp. 333–343.

[21] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *VLDB*, 1997, pp. 426–435.

[22] K. Mouratidis, Y. Lin, and M. L. Yiu, "Preference queries in large multi-cost transportation networks," in *ICDE*, 2010, pp. 533–544.

[23] K. Deng, X. Zhou, and H. T. Shen, "Multi-source skyline query processing in road networks," in *ICDE*, 2007, pp. 796–805.

[24] L. Zou, L. Chen, M. T. Özsu, and D. Zhao, "Dynamic skyline queries in large graphs," in *DASFAA*, 2010, pp. 62–78.

[25] W. Son, M. Lee, H. Ahn, and S. Hwang, "Spatial skyline queries: An efficient geometric algorithm," in *SSTD*, 2009, pp. 247–264.

[26] W. Son, S. Hwang, and H. Ahn, "MSSQ: manhattan spatial skyline queries," *Inf. Syst.*, vol. 40, pp. 67–83, 2014.

[27] D. Fuhry, R. Jin, and D. Zhang, "Efficient skyline computation in metric space," in *EDBT*, 2009, pp. 1042–1051.

[28] Q. Lin, Y. Zhang, W. Zhang, and A. Li, "General spatial skyline operator," in *DASFAA*, 2012, pp. 494–508.

[29] A. Regalado, M. Goncalves, and S. Abad-Mota, "Evaluating skyline queries on spatial web objects," in *DEXA*, 2012, pp. 416–423.

[30] X. Guo, Y. Ishikawa, and Y. Gao, "Direction-based spatial skylines," in *MobiDE*, 2010, pp. 73–80.

[31] K. Kodama, Y. Iijima, X. Guo, and Y. Ishikawa, "Skyline queries based on user locations and preferences for making location-based recommendations," in *LBSN*, 2009, pp. 9–16.

[32] K. Deng, S. W. Sadiq, X. Zhou, H. Xu, G. P. C. Fung, and Y. Lu, "On group nearest group query processing," *IEEE TKDE*, vol. 24, no. 2, pp. 295–308, 2012.

[33] E. Dellis and B. Seeger, "Efficient computation of reverse skyline queries," in *VLDB*, 2007, pp. 291–302.

[34] D. Sacharidis, P. Bouros, and T. K. Sellis, "Caching dynamic skyline queries," in *SSDBM*, 2008, pp. 455–472.

[35] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 656–665.

[36] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search," in *ICDE*, 2013, pp. 901–912.

[37] C. Long, R. C. Wong, K. Wang, and A. W. Fu, "Collective spatial keyword queries: a distance owner-driven approach," in *SIGMOD*, 2013, pp. 689–700.

[38] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-textual similarity joins," *PVLDB*, vol. 6, no. 1, pp. 1–12, 2012.

Jieming Shi is a PhD candidate at the Department of Computer Science, University of Hong Kong. He received his Bachelor's Degree of Science from the Department of Computer Science and Technology in Nanjing University, 2011. His research interests include query processing on spatial-textual data, and geo-social network mining and management.

Dingming Wu received the bachelor's and the master's degrees in computer science from the Huazhong University of Science and Technology and Peking University in 2005 and 2008, respectively. She received the PhD degree in computer science from Aalborg University in 2011. She is a post-doc fellow at the University of Hong Kong. She was a post-doc teaching fellow at Hong Kong Baptist University from 2011 to 2013. Her research concerns spatial keyword search, geo-social networks, and recommendation systems.

Nikos Mamoulis received his diploma in computer engineering and informatics in 1995 from the University of Patras, Greece, and his PhD in computer science in 2000 from the Hong Kong University of Science and Technology. Since 2001, he is a professor at the Department of Computer Science, University of Hong Kong. His research focuses on the management and mining of complex data types, privacy and security in databases, and uncertain data management.