Probabilistic Contextual Skylines

Dimitris Sacharidis [#1], Anastasios Arvanitis [#*2], Timos Sellis [#*3]

[#]*Institute for the Management of Information Systems — "Athena" R.C., Greece*
[1]dsachar@imis.athena-innovation.gr
[3]timos@imis.athena-innovation.gr

[*]*National Technical University of Athens, Greece*
[2]anarv@dblab.ntua.gr

***Abstract*— The skyline query returns the most interesting tuples according to a set of explicitly defined preferences among attribute values. This work relaxes this requirement, and allows users to pose meaningful skyline queries without stating their choices. To compensate for missing knowledge, we first determine a set of uncertain preferences based on user profiles, i.e., information collected for previous contexts. Then, we define a probabilistic contextual skyline query (p-CSQ) that returns the tuples which are interesting with high probability. We emphasize that, unlike past work, uncertainty lies within the query and not the data, i.e., it is in the relationships among tuples rather than in their attribute values. Furthermore, due to the nature of this uncertainty, popular skyline methods, which rely on a particular tuple visit order, do not apply for p-CSQs. Therefore, we present novel non-indexed and index-based algorithms for answering p-CSQs. Our experimental evaluation concludes that the proposed techniques are significantly more efficient compared to a standard block nested loops approach.**

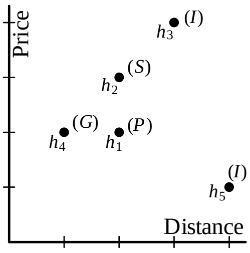
I. Introduction

Given a set of preferences, a skyline query [1] returns the non-dominated records. A tuple *dominates* another if it is at least as good (i.e., preferred) in all attributes and strictly better in at least one. For example, consider a database containing information about hotels. A skyline query returns those hotels for which there is no cheaper and, at the same time, closer to the beach alternative. In many cases, it is meaningful to specify preferences with respect to the *context*, i.e., the current query situation, and pose dynamic skyline queries [2], [3], [4], [5], [6], [7]. To emphasize that preferences and dominance relationships are defined relative to a context, we adopt the term *contextual skyline queries* (CSQ). Returning to the hotel example, a user for her/his business trip may opt for hotels that are close to the airport and, further, provide good service. The same user for her/his vacation (another context) would prefer a hotel close to the beach with a low price.

All previous works assume that the user explicitly states her/his preferences for each CSQ posed. In this paper, we relax this assumption and allow users to pose skyline queries *without stating their preferences*. As a running example, we use the hotels dataset illustrated in Figure 1(a). The table contains information about Price, Distance to the city center and Amenity. Note that the latter is a set-valued attribute, since a hotel can offer multiple amenities. For ease of presentation, we assume hotels with a single amenity; our methods, however, apply to the most general case. Lower values are preferred on the first two attributes, whereas for Amenity, preferences depend on the situation. Figure 1(b) draws hotels on the Price–Distance plane; Amenity values are shown next to each tuple. Assuming all amenities are equally attractive, the conventional skyline contains hotels h_4, h_5, for which there is no cheaper and closer alternative.

Hotel	Price	Distance	Amenity
h_1	200	10	Pool (P)
h_2	300	10	Spa (S)
h_3	400	15	Internet (I)
h_4	200	5	Gym (G)
h_5	100	20	Internet (I)

(a) Dataset



(b) 2d representation

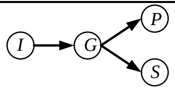
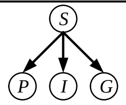
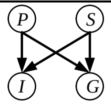
Fig. 1. Hotels example

Table I exemplifies CSQs for the three contexts C_1–C_3, shown in the first column. The second column contains the Hasse diagrams of the contextual preferences, while the third presents the resulting skylines. Initially, consider that the user prefers Internet (I) over Gym (G), and both over any other amenity, when s/he is on a Business trip in June (context C_1). Based on these preferences, hotels h_3, h_4, h_5 are the results to the CSQ for C_1, as shown in the first row of Table I. Although h_3 is more expensive and distant than h_1, h_2, h_4, it offers a more desirable amenity, I, and hence is not dominated. In addition, the user has specified preferences for contexts C_2 (Vacation trip) and C_3 (in the Summer), with the corresponding skylines included in Table I.

Examine now situation C_q (fourth row in Table I), where the user plans a Business trip in the Summer but states no preferences. To understand the resulting uncertainty in C_q preferences, consider amenities Internet I and Pool P. Should (i) I be preferred over P as in C_1, (ii) P be preferred over I as in C_3, or (iii) I and P be equally favorable as in C_2? In fact, all three cases hold with a probability that depends on the similarity of C_q to C_1, C_2, C_3. Furthermore, the uncertainty propagates to the dominance relationships, i.e., each hotel dominates every other with a probability that depends on context.

TABLE I

CONTEXTS, PREFERENCES AND CONTEXTUAL SKYLINES

Context	Preferences	Skyline
C_1: Business, June		h_3, h_4, h_5
C_2: Vacation		h_2, h_4, h_5
C_3: Summer		h_1, h_2, h_4, h_5
C_q: Business, Summer	—	?

Motivated by the previous example, we propose a framework that compensates for missing knowledge in user preferences. In particular, we solve two distinct sub-problems. (i) We determine a set of *uncertain preferences* suitable for the current query context based on users' profiles. (ii) We address uncertainty in dominance relationships, with a *probabilistic contextual skyline query* (p-CSQ) that returns the tuples not dominated with high probability.

For the first sub-problem, we borrow ideas from personalization systems [8] and, in particular, the preference reconciliation of [9] and context resolution of [10]. The user has specified (or, the system has collected) a profile, that is, preferences for a set of characteristic contexts C_is, such as a business trip or a vacation. Based on this information, we assess the similarity of the current query context C_q to each C_i, and use it to assign probabilities to preferences. The uncertainty of preferences also affects dominance relationships. For the second sub-problem, the probability of a tuple to belong to the skyline is set to the probability that it is not dominated by any other tuple. p-CSQ returns all records whose aforementioned probability is above a certain threshold. Note that if all preferences are certain, e.g., when C_q exactly matches one C_i, p-CSQ reduces to a conventional contextual skyline query.

While variations of the first sub-problem have been studied, to the best of our knowledge no previous work discusses the second. Note that probabilistic skylines have appeared in the past to handle *uncertainty in the tuples' values* [11], [12]. Here, they serve a fundamentally different purpose as they account for *uncertainty in the users' preferences*. Existing approaches do not apply to p-CSQs for the following reason. Almost all skyline algorithms (with the exception of the block nested loops (BNL) algorithm) visit tuples in a monotonic order from more to less preferred attribute values. Such an order exists even in the case of [11], [12], given that their values can be bounded. This reduces the average number of dominance checks and allows progressive output of results. However, when preferences are uncertain, a monotonic order does not exist due to the lack of transitivity, as discussed in Section IV-A. Therefore, p-CSQ necessitates novel query processing methods.

Our main contributions include the following.

- We introduce uncertain preferences and define probabilistic contextual skyline queries (p-CSQ).
- Given a current context C_q and a set of preferences for contexts C_is, we propose a simple methodology that derives probabilities for the uncertain preferences in C_q.
- We devise non-indexed and index-based algorithms for processing p-CSQs that are significantly faster than a standard BNL approach.
- We perform an extensive experimental evaluation verifying the efficiency of the proposed algorithms.

The remainder of this paper is structured as follows. Section II reviews relevant bibliography. Section III defines the problem and describes an example. Sections IV and V present non-indexed and index-based algorithms, respectively, for answering p-CSQs. Section VI presents the experimental results and Section VII concludes this paper.

II. RELATED WORK

Section II-A reviews bibliography regarding personalization systems and contextual preferences. Section II-B overviews methods for skylines queries.

A. Personalization Systems

There are two ways to define interest on attribute values and tuples. *Quantitative* preferences, used in [13], [14], [8], assign numeric scores to values via a *scoring function* imposing a total order. For example, values a, b, c are assigned scores 0.9, 0.7, 0.1, respectively. On the other hand, *qualitative* preferences, used in [15], [16], are specified using binary predicates and induce strict partial orders. For example, value a is preferred over b and c, but b, c are indifferent. In this paper, we focus on qualitative preferences as they are more generic: quantitative can be expressed as qualitative, but not the other way around.

The general goal of personalization systems is to offer custom-tailored services based on collected user profiles. Personalized database systems, e.g., [17], [13], [18], [14], [15], [16], [8], provide ranked query results by combining user preferences. The work in [17] augments queries with a prefer clause that functions as a soft constraint; if no tuples satisfy it, the clause is relaxed. In [13] generic functions that merge quantitative preferences are presented. The works in [18], [14] deal with linear combinations of preference scores and propose index and view based techniques for ranking tuples. For qualitative preferences, [15], [16] introduce a framework for composing or accumulating interests. Among the discussed methods is the Pareto composition, which corresponds to the skyline query reviewed in Section II-B. The work in [8] provides personalized answers by considering preferences specified on attribute values and join conditions.

More recently, focus has turned to context-aware personalization systems, where *contextual preferences* that only apply to a particular situation, are defined. The work in [9] is the most relevant to ours. Given a set of contextual qualitative preferences, the authors provide a context-aware ranking of query results. For every stored context C_i, they initially compute the similarity to the query context C_q. Then, they assign to each possible ordering of query results a score that reflects the degree of agreement between the preferences of C_i and the order. The best order is the one that has the highest (weighted by the similarities) sum of scores across all contexts. The authors turn to heuristic approaches for finding the best order, as the problem is NP-hard. For the case of a single user, the work in [10] solves a similar context-aware ranking problem involving quantitative contextual preferences. In this paper, rather than computing a *global* order for all tuples, we compute a number of *local* probabilistic orders (the uncertain preferences) for the values of each attribute; the final ranking is due to the skyline.

B. Skyline Query Processing

The skyline query returns the set of not dominated tuples. If records are seen as points in a multi-dimensional space, the skyline query returns the maximal elements, a problem which has been extensively studied in computational geometry literature, e.g., [19]. The seminal work of [1] presents various external memory algorithms. The most well-known method is Block Nested Loops (BNL), which checks each point for dominance against the entire dataset. Furthermore, the authors describe an index (B-tree) based approach, as well as an extension of the main memory multidimensional divide and conquer algorithm of [20]. Tan et al. [21] introduce techniques, which progressively output skyline tuples without having to scan the entire dataset. The work in [22] observes that examining points according to a monotone (in all attributes) preference function reduces the average number of dominance checks. Based on this fact, the authors propose the Sort-first Skyline algorithm (SFS), which is similar to BNL but includes a presorting step. Several optimizations to the SFS algorithm, e.g., [23], [24], increase its efficiency. There are attributes, e.g., nominal, hierarchical, set-valued, etc., whose values cannot be sorted from most to least preferred, i.e., a total order on preferences does not exist. This occurs because certain values cannot be compared with each other, i.e., preference among them constitutes a (strict) partial order. To handle such attributes, the work in [25] proposes an algorithm based on a stronger notion of dominance, which however causes false positives in the skyline and requires an additional filtering step. The work in [26] identifies the minimal set of preferences that cause the exclusion of a given point from the skyline.

Multidimensional (spatial) indexes are used to guide the search for skyline points and prune large parts of the space. The Nearest Neighbor (NN) algorithm [27] uses an R-tree to index tuples and performs nearest neighbor search on non-dominated areas. Assuming small values are preferred in all attributes, the basic idea of NN is that the record closest to the lower corner of a non-dominated area always belongs to the skyline. Note that the algorithm must perform duplicate result elimination, because the examined areas are overlapping. On the other hand, the Branch and Bound Skyline (BBS) method [2], which also uses an R-tree, is shown to be I/O optimal. BBS maintains (i) a heap of R-tree entries sorted in ascending order of their minimum distance (MINDIST) to the axes origin, and (ii) a list of skyline tuples found so far. Upon deheaping an entry, the lower corner of its MBB is checked for dominance against the list. If it is dominated, the entire subtree is pruned. Otherwise, its children are examined, and those not dominated are inserted into the heap. Execution terminates when the heap is depleted. Similar to SFS, BBS visits points according to a monotone preference function, but in addition disregards large sets of points (inside a subtree) without even accessing them. Analogous results hold when records are packed according to the z-order space filling curve [28].

Several extensions and related concepts to the skyline query have been studied. The k-*skyband* [2] contains the tuples dominated by less than k other records; the skyline is the 1-skyband. The *skycube query* [29] returns the tuples not dominated in a specified subset of the dimensions. In the dynamic, or contextual skyline query (CSQ) [2], preferences among attribute values can vary and are specified at query time. Two recent works [30], [7] discuss CSQs in the presence of attributes with partially ordered domains. In the simplest CSQ form, an exemplar record q is provided so that all dominance relationships are defined *relative* to q, rather than the axes origin. The work in [6] presents techniques for caching past results to expedite processing of future queries. The *multi-source skyline query* [5], [3] retrieves the records not dominated with respect to a set of exemplar tuples. The *reverse skyline* of p [4], [12] contains points p' such that p is in the relative skyline w.r.t. p'. Seen from a data-warehouse perspective, the work in [31] studies dominance relationships beyond skylines.

The work in [32] combines top-k with skyline queries, using aggregate R-trees to rank tuples based on the number of records they dominate. The work in [33] deals with the problem that the skyline in high dimensional spaces is too large. For this purpose, it relaxes the notion of dominance to k-dominance, so that more points are dominated. The k most representative skyline operator is proposed in [34]. This selects a set of k skyline points, so that the number of points dominated by at least one of them is maximized.

The notion of probabilistic skyline queries was introduced in [11] to deal with uncertain databases, where each object (tuple) corresponds to a collection of independent instances. An uncertain object is represented as a minimum bounding box (MBB) that encloses all instances. Dominance relationships among objects are probabilistic, as MBBs may overlap each other in one or multiple dimensions. The skyline probability of an object is equal to the probability of its instances not being dominated by any other object. The p-skyline contains objects with skyline probability above p. Based on the aforementioned definitions, the work in [12] adapts the notion of reverse skyline queries for uncertain objects. Both works define probabilistic dominance relationships to handle uncertainty in the database. On the other hand this work deals with uncertainty present in the preference relationships and, therefore, the techniques of [11], [12] do not apply.

III. Problem Definition

Section III-A introduces contexts, preferences and the skyline query. Section III-B extends the previous definitions in the case of uncertainty and formalizes the problem. Section III-C discusses extraction of uncertain preferences and Section III-D presents an example. Table II contains the frequently used notation and its meaning.

TABLE II

Notation

Symbol	Definition
$\mathcal{D}$	dataset
$\mathrm{dom}(A_j)$	domain of attribute A_j
C_i	context
$u \succ_{A_j} v \mid C_i$	value u is preferred to v in C_i, for $u, v \in \mathrm{dom}(A_j)$
CSQ($\mathcal{D} \mid C_q$)	contextual skyline query
$Pr[u \succ_{A_j} v \mid C_i]$	probability that $u \succ_{A_j} v \mid C_i$ holds
$t \succ t'$	tuple t dominates t'
$Pr[t \succ t' \mid C_q]$	probability that tuple t dominates t' in C_q
$P_{sky}^{C_q}(t)$	probability that tuple t belongs to the skyline
p-CSQ($\mathcal{D} \mid C_q$)	probabilistic contextual skyline query
$\mathcal{G}_i$	group of $\mathcal{D}$
e_j^i, e_j^{i-}, e_j^{i+}	a node of aR-tree T_i, its lower and upper corner
$p(e_j^i, t)$	expected probability that e_j^i's tuples dominate t

A. Contextual Skylines

Consider a relation $\mathcal{D}$ with attributes $A_1 \ldots A_d$. A *context* C is a particular situation or state associated with the user/query, and is represented as a set of parameter-value pairs [9], [10]. For example, context C_1 of Table I is expressed as {Purpose=Business, Period=June}. We denote attributes for which preferences are explicitly defined with respect to a context as *relatively preferred* (RP). On the other hand, attributes for which preferences are fixed and independent of contexts are called *statically preferred* (SP). Referring to Figure 1(a), Price and Distance are SP, since lower values are always better, whereas Amenity is an RP attribute.

A *preference* applies to a specific attribute A_j and has the form $u \succ_{A_j} v$, where $u, v \in \mathrm{dom}(A_j)$. This implies that if for tuples t, t' it holds that $t.A_j = u$, $t'.A_j = v$ and $t.A_k = t'.A_k$ for all $k \neq j$, then t is preferred to t'. For RP attributes, preferences are *contextual*, i.e., they only hold for a specific context C_i, and are denoted as $u \succ_{A_j} v \,|\, C_i$. Since preferences for SP attributes hold for all contexts, we unify notation using $u \succ_{A_j} v \,|\, C_i$ for any RP or SP A_j. Continuing our example, the user has specified preferences $I \succ_A G \,|\, C_1$, $I \succ_A P \,|\, C_1$, $I \succ_A S \,|\, C_1$, $G \succ_A P \,|\, C_1$, $G \succ_A S \,|\, C_1$, regarding the Amenity attribute (abbreviated as A) for context C_1. We assume that preferences for a particular context C_i and attribute A_j are non-conflicting, i.e., for any two values $u \neq v \in \mathrm{dom}(A_j)$, $u \succ_{A_j} v \,|\, C_i$ and $v \succ_{A_j} u \,|\, C_i$ cannot simultaneously hold. This means that the set of preferences for A_j in context C_i defines a (strict) partial order; Table I depicts the Hasse diagrams of the partial orders corresponding to the defined preferences.

We say that a record t *dominates* another t' in context C_i, denoted as $t \succ t' \,|\, C_i$, if t is preferred or equal to t' in all attribute values and preferred in at least one, i.e., $\forall j \;\, t.A_j \succeq_{A_j} t'.A_j \,|\, C_i$[1] and $\exists k \;\, t.A_k \succ_{A_k} t'.A_k \,|\, C_i$. The *contextual skyline query* (CSQ) for context C_i, denoted as CSQ($\mathcal{D}|C_i$), returns the tuples not dominated by any other in C_i. Note that the above definitions are in accordance to existing skyline literature. Conventional skyline queries need not be associated with a specific query context, as all attributes are statically preferred. On the other hand, for dynamic skyline queries (i.e., when RP attributes exist), past works assume a query context for which preferences are stated, but do not explicitly describe it. In this work, we treat all skyline queries as contextual, i.e., they are associated with a particular situation, which is formally stated as a context.

B. Probabilistic Contextual Skylines

All past works deal with CSQs, where user preferences for all RP attributes are concretely stated along the query. This section shows that it is possible to provide meaningful results to skyline queries without requiring the user to specify choices. The only requirement is that information regarding preferences for other contexts exists. Note that in the absence of such data, other users' preferences can be used instead. Let the *profile* of a user refer to the set of contextual preferences specified/collected in the past. The goal of this work can be roughly stated as: given the current context C_q and the user's profile, determine the non-dominated tuples in C_q. To solve this, we identify and formulate two sub-problems discussed in the following.

When the current situation perfectly matches with only one of the contexts included in the profile, the problem naturally reduces to a contextual skyline query. Interesting challenges arise when C_q is not in the profile; what should the user's preferences be in this case? Since C_q preferences are missing, the system has to interpolate them based on the profile, a process which entails uncertainty. We model uncertainty using probabilities. For context C_i and two values $u, v \in \mathrm{dom}(A_j)$, an *uncertain contextual preference* states predisposition of u over v that holds with probability $Pr[u \succ_{A_j} v \,|\, C_i]$. In the absence of uncertainty, the probability is either 0 or 1. Similar to concrete preferences, we do not allow conflicts. This translates to the following condition for any two values $u \neq v \in \mathrm{dom}(A_j)$, $Pr[u \succ_{A_j} v \,|\, C_i] \leq 1 - Pr[v \succ_{A_j} u \,|\, C_i]$; the inequality accounts for the possibility that u, v are incomparable. We are now ready to state the first sub-problem, further discussed in Section III-C.

Problem 1 [Uncertain Preferences Extraction] Based on the user's profile, derive a set of uncertain preferences for the current context C_q.

The uncertainty associated with preferences leads to *uncertain dominance relationships*. Assuming independence among

[1] The shorthand $u \succeq_A v$ stands for $u \succ_A v \vee u = v$.

attribute preferences, the probability that tuple t dominates t' in the context C_i is

$$Pr[t \succ t' \,|\, C_i] = \begin{cases} \prod_j Pr[t.A_j \succeq_{A_j} t'.A_j \,|\, C_i], & \text{if } t \neq t' \\ 0, & \text{if } t = t', \end{cases} \tag{1}$$

where the first case applies only when tuples do not have the same values in all attributes. It is important to note that this definition reduces to the conventional notion of dominance when all preferences are certain, i.e., the probability is 1 if $t \succ t' \,|\, C_i$ and 0 otherwise. This holds because the condition $\forall j\; t.A_j \succeq_{A_j} t'.A_j \,|\, C_i$ suffices for a tuple t to dominate $t' \neq t$.

Based on the previous discussion, we now adapt the definition of skyline queries. Intuitively, a tuple is in the skyline when it is not dominated; since dominance is uncertain, this is a probabilistic event. The *skyline probability* of a tuple t is defined as

$$P_{sky}^{C_i}(t) = \prod_{t' \neq t} \left(1 - Pr[t' \succ t \,|\, C_i]\right). \tag{2}$$

In accordance to the deterministic case, when all preferences are certain, $P_{sky}^{C_i}(t)$ is 1 if t is not dominated and 0 otherwise. We now state the second sub-problem.

Problem 2 [Probabilistic Contextual Skyline Query (p-CSQ)] Given a database and a set of uncertain preferences, return the tuples t whose skyline probability is above a threshold, i.e., $t \in p\text{-CSQ}(\mathcal{D}|C_i) \Leftrightarrow P_{sky}^{C_i}(t) \geq p$.

We introduce efficient algorithms for Problem 2 in Sections IV, V, but first we elaborate on Problem 1 and present a concrete example.

C. Extracting Uncertain Preferences

This section presents a simple interpretation and solution to Problem 1; although others could apply, examining them is beyond the scope of this paper.

Initially, we define a measure of context similarity. Assume that for each context parameter X_i, there exists a function sim_{X_i} that assesses the similarity between two values of its domain $\mathrm{dom}(X_i)$. This function takes values in $[0, 1]$, where higher ones express greater similarity. Depending on the domain type, different functions apply. For numerical domains $\mathrm{sim}_{X_i}(a, b) = 1 - \frac{|a-b|}{M-m}$, where M (m) is the maximum (minimum) value in $\mathrm{dom}(X_i)$. For categorical/hierarchical domains, $\mathrm{sim}_{X_i}(a, b) = \frac{|\mathrm{lvs}(a) \cap \mathrm{lvs}(b)|}{|\mathrm{lvs}(a) \cup \mathrm{lvs}(b)|}$, i.e., the Jaccard coefficient, where $\mathrm{lvs}(a)$ denotes the set of leaves under a. For nominal domains, $\mathrm{sim}_{X_i}(a, b) = 1$ when $a = b$, 0 otherwise. Based on these functions, we define similarity between contexts C, C' as $\mathrm{sim}(C, C') = \prod_i \mathrm{sim}_{X_i}(c.X_i, c'.X_i)$. Note that a context parameter value not specified completely matches any value from the same domain.

We apply the previous definitions to compute the similarity of current context C_q to all contexts present in the user's profile, and use them to extract the uncertain preferences. Intuitively, we require a value u to be preferred over v in C_q with a probability that increases with the number of $u \succ v$ occurrences in the contexts, and with the similarity of those contexts to C_q. For each pair of $u, v \in \mathrm{dom}(A_i)$, we define:

$$Pr[u \succ_{A_i} v \,|\, C_q] = \frac{\sum_j \left(\mathrm{sim}(C_q, C_j) \cdot |u \succ_{A_i} v \,|\, C_j|\right)}{\sum_j \mathrm{sim}(C_q, C_j)}, \tag{3}$$

where $|u \succ_{A_i} v \,|\, C_j|$ is 1 if such a preference exists, and 0 otherwise.

Equation 3 has the following properties, for $u, v \in \mathrm{dom}(A_i)$. When C_q matches exactly one C_i, the probability Pr[$u \succ_{A_i} v \,|\, C_q$] is 1 if $u \succ_{A_i} v \,|\, C_i$ holds, 0 otherwise. When $u \succ v$ in all contexts, Pr[$u \succ_{A_i} v \,|\, C_q$] is 1. Finally, since there is no conflict in the input preferences, the derived uncertain preferences are non-conflicting, i.e., it holds that $Pr[u \succ_{A_i} v \,|\, C_q] \leq 1 - Pr[v \succ_{A_i} u \,|\, C_q]$.

D. A Concrete Example

We demonstrate the definitions presented in the previous sections, using the example of Section I. Consider a probabilistic contextual skyline query that requests the tuples with skyline probability above 1/2.

Problem 1 First, we assess the similarity of contexts C_1, C_2, C_3 to the query context C_q shown in Table I. Assuming similarities among parameter values $\mathrm{sim}_{\text{Purpose}}$(Business, Vacation) $= 0$, $\mathrm{sim}_{\text{Period}}$(June, Summer) $= 1/3$, we obtain $\mathrm{sim}(C_q, C_1) = 1/3$, $\mathrm{sim}(C_q, C_2) = 0$ and $\mathrm{sim}(C_q, C_3) = 1$. Then, based on Equation 3, we compute the uncertain preferences depicted on Table III; the number inside a cell is the probability that the value corresponding to the row is preferred over the value in the column. Consider values I, P, for example; we have $Pr[I \succ_A P \,|\, C_q] = \frac{1/3+0+0}{1+0+1/3} = 1/4$ and $Pr[P \succ_A I \,|\, C_q] = \frac{0+0+1}{1+0+1/3} = 3/4$.

TABLE III

PREFERENCE PROBABILITIES $Pr[u \succ_A v \,|\, C_q]$ BASED ON TABLE I

v u	I	G	P	S
I	—	1/4	1/4	1/4
G	0	—	1/4	1/4
P	3/4	3/4	—	1/4
S	3/4	3/4	0	—

Problem 2 To obtain the probabilistic contextual skyline, we need to compute the uncertain dominance relationships among the tuples shown in Figure 1(a). The respective dominance probabilities are depicted in Table IV; the number inside a cell is the probability that the tuple corresponding to the row dominates the one in the column. Consider hotel h_4. Figure 1(a) shows that h_4 does not dominate h_5 with respect to the statically preferred attributes. It follows that h_4 can only have 0 probability of dominating h_5 when all attributes are considered. Figure 1(a) also shows that h_4 dominates h_1, h_2, h_3 on the SP attributes. Since h_4's amenity value G is preferred with 1/4 probability to h_1's P, we obtain $Pr[h_4 \succ h_1 \,|\, C_q] = 1 \cdot 1 \cdot 1/4 = 1/4$. Similarly, $Pr[h_4 \succ h_2 \,|\, C_q] = 1 \cdot 1 \cdot 1/4 = 1/4$ because $Pr[G \succ_A S \,|\, C_q] = 1/4$.

On the other hand, G is not preferred over I and hence, $Pr[h_4 \succ h_3 \,|\, C_q] = 1 \cdot 1 \cdot 0 = 0$. Finally, h_4 cannot dominate itself according to Equation 1.

TABLE IV
DOMINANCE PROBABILITIES $Pr[t' \succ t \,|\, C_q]$ FOR FIGURE 1(A)

t' \ t	h_1	h_2	h_3	h_4	h_5
h_1	0	0	3/4	0	0
h_2	0	0	3/4	0	0
h_3	0	0	0	0	0
h_4	1/4	1/4	0	0	0
h_5	0	0	0	0	0
$\boldsymbol{P_{sky}^{C_q}(t)}$	**3/4**	**3/4**	**1/16**	**1**	**1**

The final step involves computing the skyline probability for each tuple. Let us consider hotel h_3. Table IV shows that h_3 has a non-negative probability of being dominated by only h_1, h_2. Equation 2 implies that $P_{sky}^{C_q}(h_3) = (1 - 3/4) \cdot (1 - 3/4) \cdot 1 \cdot 1 = 1/16$. In a similar manner, the skyline probabilities for all hotels, shown in the last row of Table IV, are computed. Therefore, hotels h_1, h_2, h_4, h_5 are the result to 0.5-CSQ($D|C_q$).

IV. NON-INDEXED ALGORITHMS

This section discusses algorithms for answering p-CSQs that do not use index structures. Section IV-A presents a straightforward method similar to the BNL algorithm [1], whereas Section IV-B makes various useful observations that increase efficiency.

A. Basic Iterative Algorithm

The majority of skyline algorithms visit tuples in a monotonic order from more to less preferred attribute values, either explicitly, e.g., by pre-sorting [21], [22], [23], [11], [24], or implicitly, e.g., using a heap [27], [2], [12], [7]. Note that even when tuples are uncertain, such an order exists if one considers the minimum bounding box of each record [11], [12]. Due to the transitivity property of the dominance relationship ($t_1 \succ t_2, t_2 \succ t_3 \Rightarrow t_1 \succ t_3$), a monotonic order reduces the average number of dominance checks and allows progressive output of results.

In the following, we demonstrate that transitivity and monotonicity do not hold in the case of uncertain preferences and thus neither in dominance relationships. Consider the uncertain preferences in Table III among values S, I, P for a given context C_q. Value S is preferred to I with probability 3/4 and I to P with 1/4, as shown in Figure 2(a). If transitivity held, we would expect S to be preferred over P (the grey arrow in Figure 2(a)) with some probability, e.g., 3/16. However, $Pr[S \succ_A P \,|\, C_q] = 0$, i.e., the probability is less than expected. Similarly, while $Pr[S \succ_A P \,|\, C_q] = 0$ and $Pr[P \succ_A I \,|\, C_q] = 3/4$, we get $Pr[S \succ_A I \,|\, C_q] = 3/4$, i.e., more than expected, as shown in Figure 2(b). Similar results hold for the not-preferred probability $1 - Pr[\cdot \succ_A \cdot \,|\, C_q]$, as demonstrated by the dashed arrows in Figures 2(c) and 2(d). Based on this example, no (partial) order from more to less preferred values exists, when preferences are uncertain.

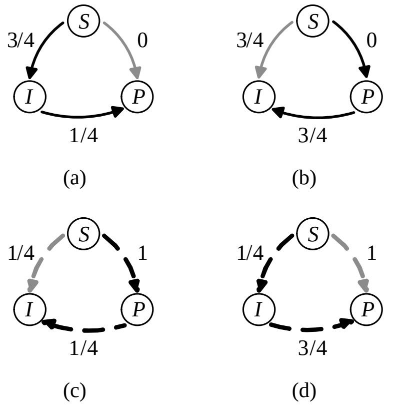


Fig. 2. Intransitivity and non-monotonicity; preferred probability decreases (a), increases (b); not-preferred probability decreases (c), increases (d)

The lack of transitivity and monotonicity, suggests that a sort-based algorithm (e.g., like SFS) does not exist for p-CSQs. In the following, we present a baseline solution termed Basic Iterative Algorithm (BIA). Assuming, tuples are stored in consecutive disk blocks, the main idea of BIA is to compute the skyline probability for each tuple by scanning the entire dataset in a block nested loops paradigm. More specifically, let M, N denote the available memory and dataset size, respectively, measured in disk blocks. BIA partitions the dataset into $\frac{N}{M-1}$ batches of $M - 1$ blocks and examines them in sequence. For each batch, BIA loads it into memory and repeats the following procedure.

BIA initializes the skyline probability of each batch tuple t to $P_{sky}^{C_q}(t) = 1$. Then, it scans the entire database loading one block at a time in memory. For each tuple t in the batch and each t' in the retrieved block, BIA computes the probability that t' dominates t and updates t's skyline probability, i.e., $P_{sky}^{C_q}(t) := P_{sky}^{C_q}(t) \cdot (1 - Pr[t' \succ t \,|\, C_q])$. As soon as $P_{sky}^{C_q}(t)$ falls below the desired threshold, t is excluded from further dominance checks. If all tuples in the current batch are eliminated, BIA continues with the next. On the other hand, once the entire database is scanned, the skyline probabilities of the non-eliminated tuples are above the threshold and finalized; hence, all remaining batch tuples are returned.

It is straightforward to see that the Basic Iterative Algorithm answers probabilistic contextual skyline queries correctly with an $O\left(\frac{N^2}{M}\right)$ worst case I/O complexity.

B. Candidate Selection Algorithm

This section presents two preprocessing steps that significantly reduce the necessary pairwise dominance checks. Let $\{\mathcal{G}_i\}$ denote the set of relations that correspond to a group-by RP attributes statement on the dataset $\mathcal{D}$, after projecting on the SPs. Therefore, each $\mathcal{G}_i$ contains tuples for a particular value combination of the relative preferred attributes. As an example, consider the dataset of Figure 1. There is a single

SP with four distinct values, and hence four groups exist $\mathcal{G}_I = \{h_3, h_5\}$, $\mathcal{G}_S = \{h_2\}$, $\mathcal{G}_P = \{h_1\}$, $\mathcal{G}_G = \{h_4\}$. Let CSQ($\mathcal{G}_i$) be the skyline (on the SP attributes) of tuples in $\mathcal{G}_i$; the following holds.

Lemma 1: For any probability threshold $p > 0$ and context C_q, *p-CSQ*$(\mathcal{D}|C_q) \subseteq \bigcup_i$ *CSQ*$(\mathcal{G}_i)$.

Proof: We prove by contradiction. Assume that a tuple t exists such that $t \in p$-CSQ$(\mathcal{D}|C_q)$ but $t \notin \bigcup_i$ CSQ$(\mathcal{G}_i)$, for some p and C_q. Further, let $\mathcal{G}_k$ be the group that t belongs to. Since $t \notin$ CSQ$(\mathcal{G}_k)$, there exists another tuple $t' \in \mathcal{G}_k$, such that t' dominates t with respect to the SP attributes, i.e., $t' \succ_{\text{SP}} t$. Since t, t' are in the same group, they have equal values in all RP attributes, and thus t' dominates t in any context, i.e., $Pr[t' \succ t \,|\, C_q] = 1$. As a result $P_{sky}^{C_q}(t) = 0$ and $t \notin p$-CSQ$(\mathcal{D}|C_q)$, a contradiction. ■

Returning to our example, no tuple can be eliminated as they all belong to their respective group skyline.

Lemma 2: For any probability threshold p and context C_q, if for a tuple $t \in$ CSQ$(\bigcup_i \mathcal{G}_i)$ there exists no $t' \in \mathcal{D}$ such that t, t' have equal SP attribute values, then $t \in p$-CSQ$(\mathcal{D}|C_q)$.

Proof: Consider a tuple t such that $t \in$ CSQ$(\bigcup_i \mathcal{G}_i)$ and there is no $t' \in \mathcal{D}$ such that t, t' have equal SP attribute values. Note that the first condition implies that t belongs to the skyline of the entire dataset with respect to the SP attributes. We will argue that for all $t^* \neq t \in \mathcal{D}$ we have $Pr[t^* \succ t \,|\, C_q] = 0$. Assume otherwise; then a t^* must be preferred (with non-zero probability) or be equal to t in all attributes. However, this cannot hold for the SP attributes: (i) there is no tuple with equal SP values (due to t's uniqueness property), and (ii) no tuple is preferred to t in the SPs (because t is in the skyline w.r.t. SPs). The assumption is wrong and $Pr[t^* \succ t \,|\, C_q] = 0$ holds for all $t^* \neq t \in \mathcal{D}$. Therefore, Equation 2 gives $Pr_{sky}^{C_q}(t) = 1$, which implies that t is in p-CSQ$(\mathcal{D}|C_q)$ for any p, C_q. ■

In the example depicted in Figure 1(b), tuples h_4, h_5 are in the skyline w.r.t. the SP attributes. Hence, they belong in the skyline for any p-CSQ, as shown in Tables I and IV.

Note that the skyline of the union of groups is the same as the skyline computed over the union of the group skyline points, i.e., CSQ$(\bigcup_i \mathcal{G}_i)$ = CSQ$\left(\bigcup_i \text{CSQ}(\mathcal{G}_i)\right)$. Therefore, CSQ$(\bigcup_i \mathcal{G}_i) \subseteq \bigcup_i$ CSQ$(\mathcal{G}_i)$. In other words, among the set of candidates of Lemma 1, Lemma 2 identifies those tuples that are definitely in the result set of any probabilistic contextual skyline query and can be immediately returned. It is important to note that the skyline within each group CSQ$(\mathcal{G}_i)$ depends only on the (static) preferences on the SP attributes, and thus is the same for any query. Therefore, both lemmas can be used as a preprocessing step.

The Candidate Selection Algorithm (CSA) uses the above results to expedite query processing. Let $\mathcal{C}$ denote the set of candidate tuples, i.e., all tuples t in $\bigcup_i$ CSQ$(\mathcal{G}_i)$ excluding those identified by Lemma 2 to have $Pr_{sky}^{C_q}(t) = 1$. CSA is identical to BIA except for the batch creation process. Instead of partitioning the entire database, CSA sorts $\mathcal{C}$ using the Hilbert curve and partitions it into batches that fit in main memory. CSA then calculates the probability of each tuple in $\mathcal{C}$. We emphasize that, similar to BIA, CSA needs to scan the entire database (and not just $\mathcal{C}$) for each batch, as tuples outside $\mathcal{C}$ can dominate those in $\mathcal{C}$ with non-zero probability. Note that when all candidate skyline points fit in main memory, the I/O cost of CSA becomes $O(N)$.

V. Index-based Algorithms

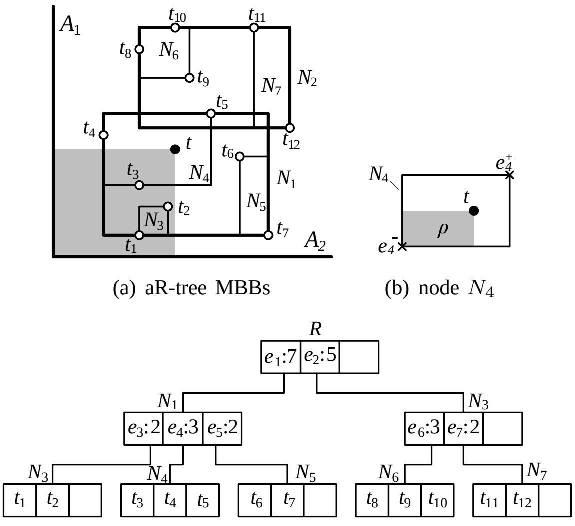
This section discusses methods for p-CSQs that utilize index structures. Unlike the boolean case of conventional skylines (is tuple t dominated?), a p-CSQ needs to find out *how many* tuples, and *with what probability*, dominate t. Therefore, all three algorithms discussed below employ index structures with aggregate information. We note that the index structures described in the following are built independent of the current context and its preferences and thus remain valid for all possible p-CSQs.

A. Basic Group Counting

The Basic Group Counting (BGC) algorithm depends on two key ideas. The first is to decouple SP and RP dominance, which is possible due to the distributive property of Equation 1. This implies that for a tuple t' to dominate t with non-zero probability, t', t should be distinct (i.e., with different attribute values) and, further, t' should dominate t with respect to the statically preferred attributes. The second is the observation that all tuples t' that belong to group $\mathcal{G}_i$ have the same probability of dominating t w.r.t. the relatively preferred attributes. Combining the two, BGC's goal is to count the number of tuples that dominate t w.r.t. the SP attributes, for each group $\mathcal{G}_i$.

To obtain quick counts per group, BGC builds a COUNT aggregate R-tree (aR-tree) to index tuples that belong to the same group. Similar to an R-tree, the structure groups tuples together and assigns them into leaf nodes. Then, the minimum bounding boxes (MBBs) of non-leaf nodes are hierarchically grouped together to produce higher level nodes, according to a maximum capacity. An aR-tree node contains entries of the form $\langle e_i, \text{MBB}_i, c_i \rangle$ for its children nodes N_i; e_i is a pointer to N_i, MBB$_i$ is the N_i's MBB and c_i is the aggregate information, i.e., the number of tuples located at the subtree rooted at N_i. In the following, MBB$_i$ is represented by its lower e_i^- and upper corner e_i^+. Figure 3 shows an aR-tree with node capacity 3 for a group of 12 tuples. In particular, Figure 3(a) draws the MBBs, Figure 3(b) zooms in on node N_4 displaying the lower and upper corner points, and Figure 3(c) shows the structure of the node entries.

Following the discussion of Section IV-B, BGC calculates the skyline probability only for the candidate tuples. Note that the group skylines, used to extract the candidate set $\mathcal{C}$, can be computed using the BBS method [2] on the aR-trees (the aggregate information is simply ignored). In the following, we fix a tuple $t \in \mathcal{C}$ and let $\mathcal{G}_t$ be the group that t belongs to. We use the notation $\succ_{\text{SP}}$ ($\succ_{\text{RP}}$) to indicate dominance w.r.t. the SP (RP) attributes; the corresponding probabilities are computed by Equation 1, when iterating only through the SP (RP) A_js. Without loss of generality, SP attributes are numerical



(a) aR-tree MBBs

(b) node N_4

(c) aR-tree structure

Fig. 3. Aggregate R-tree example

```
Basic/Super Group Counting
   Input: $\mathcal{C}$, $p$, $C_q$, aR-trees $\{T_i\}$
   Output: $S$ the answer to $p$-CSQ($\mathcal{D}|C_q$)
   Variables: $\mathcal{H}$ a minheap with entries $\langle e, p(e,t)\rangle$ and key $p(e,t)$
1  begin
2      $S := \varnothing$
3      foreach $t \in \mathcal{C}$ do
4          $Pr^{C_q}_{SKY}(t) := 1$
5          foreach $T_i$ do
6              $p(e^i_R) :=$ ComputeProb ($e^i_R$, $t$, $C_q$)   // for the root $e^i_R$ of $T_i$
7              enheap $\langle e^i_R, p(e^i_R, t)\rangle$
8          while $\mathcal{H}$ not empty and $Pr^{C_q}_{SKY}(t) \geq p$ do
9              deheap $\langle e^i_j, p(e^i_j, t)\rangle$
10             if $e^{i+}_j \succ_{SP} t$ then
11                 $Pr^{C_q}_{SKY}(t) := Pr^{C_q}_{SKY}(t) \cdot p(e^i_j, t)$
12             else
13                 foreach child $e^i_k$ of $e^i_j$ do
14                     if $e^{i-}_k \succ_{SP} t$ then
15                         $p(e^i_k, t) :=$ ComputeProb ($e^i_k$, $t$, $C_q$)
16                         enheap $\langle e^i_k, p(e^i_k, t)\rangle$
17         if $Pr^{C_q}_{SKY}(t) \geq p$ then
18             insert $t$ in $S$
19     return $S$
20 end
```

Fig. 4. BGC/SGC Algorithm

and small values are more preferred. Furthermore, we write $Pr[\mathcal{G}_{t'} \succ_{\text{RP}} \mathcal{G}_t \,|\, C_q]$ to denote the probability $Pr[t' \succ_{\text{RP}} t \,|\, C_q]$ that any tuple $t' \in \mathcal{G}_{t'}$ RP dominates any $t \in \mathcal{G}_t$.

A straightforward implementation of BGC computes t's skyline probability visiting all aR-trees in sequence; initially $Pr^{c_q}_{sky}(t) = 1$. For each aR-tree T_i, which contains tuples from group $\mathcal{G}_i$, a range-count query is issued to obtain the number n_i of tuples that dominate t w.r.t. SP. The shaded region in Figure 3(a) corresponds to such a range query; after traversing nodes R, N_1 and N_4, the correct answer 3 is computed (t_3 and two tuples inside N_3). Then, t's skyline probability is updated $Pr^{c_q}_{sky}(t) := Pr^{c_q}_{sky}(t) \cdot (1 - Pr[\mathcal{G}_i \succ_{\text{RP}} \mathcal{G}_t \,|\, C_q])^{n_i}$.

While the above procedure is correct, it may incur unnecessary I/Os for a tuple t that does not satisfy the p-CSQ. In these cases, it is crucial to quickly disqualify t. Therefore, BGC visits nodes, across aR-trees, in an order that increases the chances of t reaching a skyline probability below the threshold. We note that this effects only the visit order of BGC and not the set of visited nodes.

The BGC algorithm, shown in Figure 4, repeats the following procedure for each candidate tuple t. BGC maintains a minheap $\mathcal{H}$ with entries $\langle e^i_j, p(e^i_j, t)\rangle$, where e^i_j corresponds to a node in the aR-tree T_i and $p(e^i_j, t)$ is the key of $\mathcal{H}$, which portrays the contribution of node e^i_j to t's skyline probability. In particular, $p(e^i_j, t)$ is the expected probability that tuples in e^i_j dominate t; we discuss its computation (function ComputeProb in Figure 5) in the sequel. Initially, the skyline probability of t is set to 1 (Line 4), and an entry $\langle e^i_R, p(e^i_R, t)\rangle$ is created and enheaped for the root node e^i_R of each tree T_i (Lines 5–7)[2].

The algorithm proceeds (Lines 8–16) examining entries until either the heap is depleted, in which case t is inserted in the result set (Lines 17–18), or the skyline probability drops below the threshold. Let $\langle e^i_j, p(e^i_j, t)\rangle$ be the deheaped entry, i.e., the one with the minimum key (Line 9). If e^i_j's upper corner dominates t w.r.t. the SP attributes (Line 10), then all records within it dominate t. In this case, the expected probability value $p(e^i_j, t)$ is exact and thus t's skyline probability is updated by that quantity (Line 11). Otherwise, BGC needs to retrieve the node e^i_j and examine its children (Lines 13–16). A heap entry for child node e^i_k is created only if it contains a tuple that can dominate t, i.e., if e^i_j's lower corner SP dominates t (Line 14). Then, node e^i_k's expected dominance probability is calculated (Line 15) and the appropriate entry in enheaped (Line 16).

The final issue that remains is the computation of $p(e,t)$, i.e., the expected probability by which a node e dominates t. Each tuple t' of e that dominates t w.r.t. the SPs contributes by $1 - Pr[\mathcal{G}_e \succ_{\text{RP}} \mathcal{G}_t \,|\, C_q]$ to t's skyline probability. The question is how many such tuples t' exist. Given only the node's MBB, and assuming uniformity within the node, it is reasonable to assume that the number of dominating tuples is analogous to the volume of the space they can exist in. Figure 3(b) shows an example for tuple t and node N_4; any tuple in the shaded region can dominate t. Let $\rho = \prod_k \max\left\{\frac{\min\{t.A_k, e^+.A_k\} - e^-.A_k}{e^+.A_k - e^-.A_k}, 0\right\}$ be the volume fraction of the dominating space. Then, there are $\rho \cdot c$ expected tuples within the space, where c is the node's count. Therefore, the expected probability is $p(e,t) = (1 - Pr[\mathcal{G}_e \succ_{\text{RP}} \mathcal{G}_t \,|\, C_q])^{\rho \cdot c}$, as computed by function ComputeProb shown in Figure 5.

B. Super Group Counting

The BGC algorithm's performance degrades as the number of groups increases. When the number of tuples per aR-trees decreases, the space becomes sparse and nodes occupy larger volumes. Therefore, it becomes less likely for an entire node to dominate t, which means fewer subtree prunings.

[2] To avoid blowing up heap space when the number of aR-trees is large, we examine aR-trees in batches instead of concurrently.

```
ComputeProb
Input: $e$, $t$, $C_q$
Output: $p(e,t)$ the expected $\prod_{t' \in e}(1 - Pr[t' \succ t])$
Variables: $c$ the count associated with entry $e$,
           $\mathcal{G}_e$ the group of tuples contained in $e$,
           $\mathcal{G}_t$ the group of $t$,
           $\rho$ the fraction of $e$ volume that dominates $t$ w.r.t. SP
begin
    $\rho := 1$
    foreach SP attribute $A_k$ do
        $\rho := \rho \cdot \max\left\{\frac{\min\{t.A_k, e^+.A_k\} - e^-.A_k}{e^+.A_k - e^-.A_k}, 0\right\}$
    $p(e,t) := (1 - Pr[\mathcal{G}_e \succ_{\text{RP}} \mathcal{G}_t \mid C_q])^{\rho \cdot c}$
    return $p(e,t)$
end
```

Fig. 5. ComputeProb Function for BGC

To address this issue we propose the Super Group Counting (SGC) algorithm, which assigns groups to supergroups and builds a modified aR-tree per supergroup.

Since an aR-tree contains tuples from different groups, the aggregate information stored must be properly adapted. The entry for a node N_i has the form $\langle e_i, \text{MBB}_i, c_i[] \rangle$, where $c_i[]$ is an array containing the number of tuples beneath N_i for each group; i.e., $c_i[j]$ corresponds to the count for G_j tuples. The SGC algorithm operates exactly like BGC (Figure 4). However, the ComputeProb function changes considering counts for multiple groups (Figure 6).

```
ComputeProb
Input: $e$, $t$, $C_q$
Output: $p(e,t)$ the expected $\prod_{t' \in e}(1 - Pr[t' \succ t])$
Variables: $c[]$ the count array associated with entry $e$,
           $\{\mathcal{G}_j\}$ the groups of tuples contained in $e$,
           $\mathcal{G}_t$ the group of $t$,
           $\rho$ the fraction of $e$ volume that dominates $t$ w.r.t. SP
begin
    $\rho := 1$
    foreach SP attribute $A_k$ do
        $\rho := \rho \cdot \max\left\{\frac{\min\{t.A_k, e^+.A_k\} - e^-.A_k}{e^+.A_k - e^-.A_k}, 0\right\}$
    $p(e,t) := 1$
    foreach group $G_j \neq G_t$ do
        $p(e,t) := p(e,t) \cdot (1 - Pr[\mathcal{G}_j \succ_{\text{RP}} \mathcal{G}_t \mid C_q])^{\rho \cdot c[j]}$
    return $p(e,t)$
end
```

Fig. 6. ComputeProb Function for SGC

C. Batch Counting Algorithm

The previous methods share a disadvantage: they examine aR-tree nodes multiple times, one for each tuple. The Batch Counting Algorithm (BCA) offers a more efficient approach that processes multiple records concurrently. We assume that all candidate tuples fit in main memory; otherwise, BCA partitions the set of candidates into batches, similar to CSA, and proceeds for each batch independently. Note that since BCA can use either an aR-tree per group or super group, we do not distinct between the two options.

Examining tuples in batch introduces additional challenges. Consider tuples t, t' and let e be the aR-tree node currently under examination. Assume that the node's upper corner e^+ SP dominates t but not t', i.e., only e^- SP dominates t'. BCA updates the skyline probability for t (using the ComputeProb function). Note that the subtree rooted at e cannot be disregarded, because it contains tuples that SP dominate t'. Therefore, BCA needs to make sure that e's children do not contribute to t's skyline probability, as their contribution has already been accounted for. A straightforward approach would be to explicitly associate each node e with the set of tuples to examine. However, even if compressed bitmaps (e.g., Bloom filters) are used, the space overhead is large. BCA takes a different direction. It associates with each node e, the upper corner of its parent b^+. If b^+ SP dominates a tuple t, e's parent completely SP dominates t. Thus, e is not considered for t, as the contribution of all e's tuples has been accounted for.

```
Batch Counting Algorithm
    Input: $\mathcal{C}$, $p$, $C_q$, aR-trees $\{T_i\}$
    Output: $S$ the answer to $p$-CSQ($\mathcal{D}|C_q$)
    Variables: $\mathcal{H}$ a minheap with entries $\langle e, b^+ \rangle$ and key MINDIST($e$)
 1  begin
 2      $S := \mathcal{C}$
 3      foreach $t \in S$ do
 4          $Pr^{C_q}_{SKY}(t) := 1$
 5      foreach $T_i$ do
 6          enheap $\langle e^i_R, e^{i+}_R \rangle$                    // for the root $e^i_R$ of $T_i$
 7      while $\mathcal{H}$ and $S$ not empty do
 8          deheap $\langle e^i_j, b^+ \rangle$
 9          foreach $t \in S$ do
10              if $e^{i+}_j \succ_{SP} t$ and $b^+ \not\succ_{SP} t$ then
11                  $p(e^i_j, t) :=$ ComputeProb $(e^i_j, t, C_q)$
12                  $Pr^{C_q}_{SKY}(t) := Pr^{C_q}_{SKY}(t) \cdot p(e^i_j, t)$
13                  if $Pr^{C_q}_{SKY}(t) < p$ then
14                      remove $t$ from $S$
15              else if $e^{i-}_j \succ_{SP} t$ then
16                  foreach child $e^i_k$ of $e^i_j$ do
17                      if $e^{i-}_k \succ_{SP} t$ and $\langle e^i_k, e^{i+}_j \rangle$ not in $\mathcal{H}$ then
18                          enheap $\langle e^i_k, e^{i+}_j \rangle$
19      return $S$
20  end
```

Fig. 7. BCA Algorithm

The pseudocode for BCA is given in Figure 7. Similar to BGC and SGC, BCA uses a minheap $\mathcal{H}$ with entries for nodes that may belong to different aR-trees. Note that BCA's heap entries have the form $\langle e, b^+ \rangle$ and are sorted by MINDIST(e), where e is an aR-tree node and b^+ is the upper corner of e's parent. Initially the skyline probability of each candidate is set to 1 (Lines 3–4) and an entry $\langle e^i_R, e^{i+}_R \rangle$ is enheaped for each root node (Lines 5–6). Since the root has no parent, its own upper corner e^{i+}_R is used as the b^+ point.

BCA proceeds iteratively until either the heap is depleted or there is no candidate tuple left (Lines 7–18). Let $\langle e^i_j, b^+ \rangle$ be the entry with the minimum MINDIST (Line 8). All remaining candidate skyline tuples are examined in sequence; let t denote the current. If e^{i+}_j SP dominates t and b^+ does not (Line 10), then all tuples beneath e^i_j should contribute to t's skyline probability as they have not been considered in e^i_j's parent. Thus, $Pr^{C_q}_{SKY}(t)$ is properly updated (Lines 11–12); if it drops below the threshold, t is disqualified from S (Lines 13–14). When only e^i_j's lower corner SP dominates t (Line 15–18), the node's children e^i_k are examined in turn (Lines 16–18). If e^i_k's lower corner SP dominates t then the entry $\langle e^i_k, e^{i+}_j \rangle$ is enheaped only if it is not already in $\mathcal{H}$ (Lines 17–18). The non eliminated tuples are the answer to p-CSQ (Line 20).

VI. Experimental Study

In this section, we evaluate the proposed methods using synthetically generated data, assuming the preference probabilities for the current context have been extracted. The examined algorithms along with their acronyms and the section they are discussed in are shown in Table V. All techniques were implemented in C++, compiled with gcc and executed on a 2Ghz Intel Core 2 Duo CPU. The page block size is 4096 bytes, and each random I/O costs 10 msecs. A memory buffer equivalent to 100 pages (410 Kbytes) was allotted to all algorithms. CSA and BCA use this buffer to process candidate tuples in batches. On the other hand, since SGC processes a single candidate at each iteration, the buffer is used to cache aR-tree nodes.

TABLE V
Proposed algorithms

Algorithm	Acronym	Section
Basic Iterative Algorithm	BIA	IV-A
Candidate Selection Algorithm	CSA	IV-B
Super Group Counting	SGC	V-A, V-B
Batch Counting Algorithm	BCA	V-C

We use a publicly available generator[3] to construct synthetic datasets. We distinguish two classes based on the distribution of the statically preferred attributes. In Independent, SP attribute values are drawn from a uniformly random distribution. In Anti-correlated, tuples with preferable SP values in one dimension are more likely to have non preferable values in the others, e.g., hotels closer to the city center are more expensive. The size of SP domain is set to 10000, whereas that of RP varies from 8 up to 128 values. The cardinality of the dataset N varies from 100K up to 10M points. The number of SP and RP attributes is $d_{SP} = \{2, 3, 4\}$ and $d_{RP} = \{1, 2\}$, respectively. Each tuple has a fixed size of 100 bytes. The preference probabilities are provided as input to the algorithms for each tested scenario.

For the index-based methods, we investigate the effect of super groups and vary $|sg|$, the number of groups assigned to each super group, from 1 up to 32. Note that when $|sg| = 1$ SGC reduces to BGC. The examined probability threshold values p for the p-CSQ range from 0.1 to 0.9. In each experimental setup, we vary a single parameter while setting the remaining to their default values. Table VI displays the parameters under investigation and their corresponding ranges; default values are shown bold.

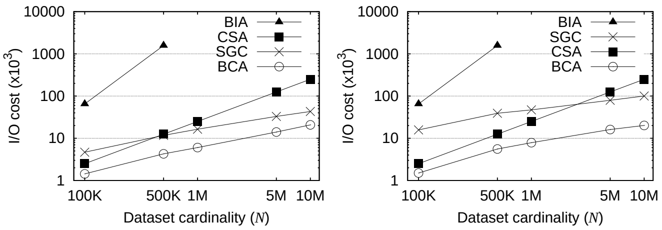
Scalability vs. Dataset Cardinality In the first set of experiments we investigate the performance of all methods with respect to the dataset cardinality. In particular, we vary N from 100K up to 10M tuples and measure the number of I/O operations, the time spent on CPU and the total query processing time. The results are depicted in Figures 8, 9 and 10, respectively.

TABLE VI
Parameters and values

Parameter	Range
Data cardinality (N)	100K, 500K, **1M**, 5M, 10M
SP dimensionality (d_{SP})	2, **3**, 4
RP dimensionality (d_{RP})	**1**, 2
RP domain size ($\|RP\|$)	8, 16, **32**, 64, 128
Groups per super group ($\|sg\|$)	1, 4, **8**, 16, 32
Probability threshold (p)	0.1, 0.3, **0.5**, 0.7, 0.9

Consider Figure 8; all methods incur higher I/O costs, since the number of blocks occupied by the dataset increases with N. The naive block nested loops variant BIA has a quadratic dependence on N and thus quickly becomes infeasible for datasets with more than 500 thousand tuples. On the other hand, all other methods examine only candidate skyline points (following the lemmas of Section IV-B) and scale almost linearly. In fact, when all candidates fit in memory, the non-indexed method CSA has exactly linear dependence on N as it scans once over the entire database.

Figure 8(a) shows that CSA, SGC, BCA have over two orders of magnitude improvement over BIA on 500K tuples for the Independent dataset. Among the three, the aR-tree based algorithms SGC, BCA are significantly more efficient, exhibiting around an order of magnitude less I/Os for the largest dataset; 43,123 and 20,741, respectively, versus 250,000 I/Os for CSA. Regarding SGC and BCA, the latter incurs on average 2 times less I/Os in all but the smallest dataset. Figure 8(b) shows similar trends for BIA, CSA, BCA. Note, however, that the performance of SGC degrades for the Anti-correlated dataset, as there are many more candidate skyline tuples. Recall that SGC's I/O complexity critically depends on the size of $\mathcal{C}$, since it has to traverse the aR-trees once per candidate.



(a) Independent

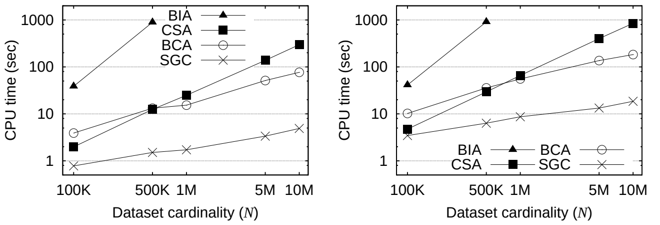
(b) Anti-correlated

Fig. 8. I/O cost vs. N

Figure 9 plots the CPU processing cost for all methods as N increases. The non-indexed methods, BIA and CSA, scale similarly to Figure 8. On the other hand, SGC has little processing overhead, whereas BCA is very CPU intensive. The latter is attributed mainly to BCA's large heap. In the default scenario, the maximum heap size for BCA is 6,036 entries (60 Kbytes), whereas for SGC is 57 entries (0.4 Kbytes). This occurs because in BCA, an aR-tree node is enheaped when it dominates *at least one* candidate, i.e., a very frequent event. Furthermore, a deheaped entry $\langle e, b^{+} \rangle$ requires dominance checks against every candidate for e as well as b^{+}. Hence

[3] http://randdataset.projects.postgresql.org

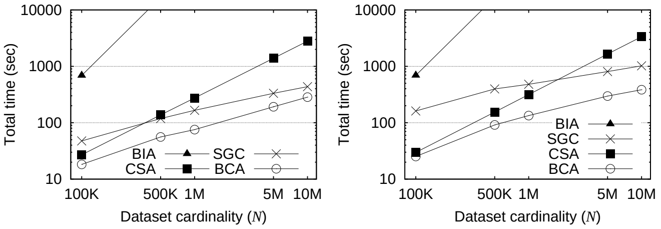
a heap entry incurs around $3 \cdot |S|$ checks, where S is the current candidate set. In the Anti-correlated dataset shown in Figure 9(b), tuples are harder to dominate each other and the number of dominance checks and the required CPU cycles increases. Therefore, the required CPU time of all methods grows.



(a) Independent (b) Anti-correlated

Fig. 9. CPU time vs. N

Figure 10 draws the total processing time (CPU and I/O cost) as the dataset cardinality increases. Notice that among the two individual costs, the required number of I/Os is the dominating factor. Figure 10(a) shows that the index-based algorithms are significantly faster than CSA following the trends of Figure 8(a). Similarly, Figure 10(b) exhibits the trends of Figure 8(b) for Anti-correlated datasets. BCA is up to 6.4 and 8.7 times faster than CSA and SGC, respectively.



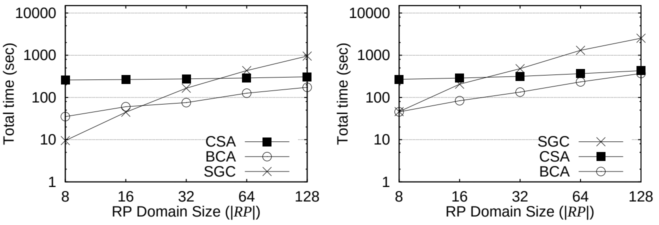
(a) Independent (b) Anti-correlated

Fig. 10. Total time vs. N

In what follows, we exclude the naive Basic Iterative Algorithm from the figures.

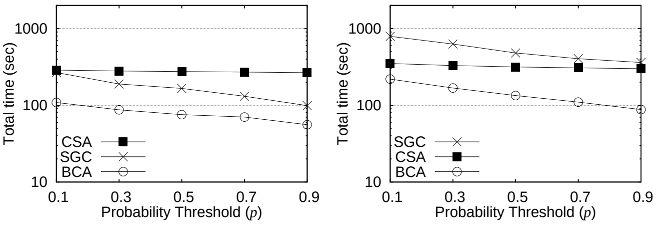
Scalability vs. RP attributes domain size Figure 11 plots the total processing time as a function of the domain size of the RP attributes. The cost for the non-indexed approach CSA remains largely unaffected by the increase in $|RP|$. This occurs, because even though the candidate skyline tuples increase with $|RP|$, they can still fit in main memory. Thus, CSA still performs only a single linear scan on the database. On the other hand, BCA performs more I/O operations and CPU cycles as $|RP|$ increases, since the increased number of candidate skylines leads to more entries being inserted into the heap. As discussed in the context of Figure 8(b), SGC is very sensitive to large $\mathcal{C}$ sizes, which leads to a dramatic total time increase in both the Independent and Anti-correlated dataset.

Scalability vs. probability threshold Figure 12 draws the query time against the probability threshold p. Note that varying p has no effect on the number of candidate skyline tuples. Larger threshold values mean that candidate tuples can be quickly disqualified. This is the trend exhibited by SGC and BCA in both distributions. Regarding CSA, however, note that when there exists even one candidate tuple such that it belongs in the skyline, CSA needs to scan the entire dataset. This is the case with all scenarios examined in Figure 12. As a result only CSA's CPU time can decrease, which accounts for its marginal total time decrement as p increases.



(a) Independent (b) Anti-correlated

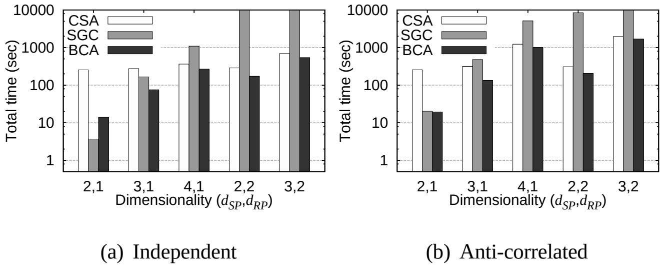
Fig. 11. Total time vs. $|RP|$



(a) Independent (b) Anti-correlated

Fig. 12. Total time vs. p

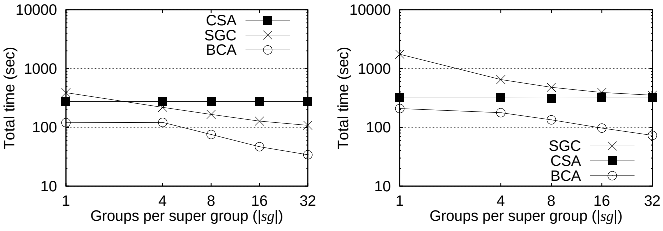
Scalability vs. dimensionality Figure 13 examines the effect of dimensionality in total processing time. When the number of SP attributes d_{SP} increases, the number of candidate (i.e., local skyline) tuples grows due to the curse of dimensionality. However, when d_{RP} increases the candidates multiply at a higher rate, since the number of groups rises exponentially. SGC is mostly affected because of its sensitivity in the number of candidates. This is demonstrated in Figure 13, where although SGC is the most efficient method in low dimensionality, it quickly becomes impractical for more than four dimensions. On the other hand, CSA and BCA achieve reasonable execution times even for large dimensionalities.



(a) Independent (b) Anti-correlated

Fig. 13. Total time vs. d_{SP}, d_{RP}

Scalability vs. number of groups per super group Finally, Figure 14 portrays the effect of the total processing time for the index-based methods as we vary the number of groups we include in a super group. For ease of comparison, we also include the time for CSA, which is unaffected by the $|sg|$ factor. Note that $|sg| = 1$ corresponds to a single group per super group, which implies that SGC degenerates to the BGC algorithm. It becomes apparent that in any case BGC's performance is inferior to SGC's for $|sg| > 1$. In general, both SGC and BCA are benefited by fewer aR-trees as explained in Section V-B. Because the RP domain size is 32 and there is a single RP attribute in the default scenario, the case of $|sg| = 32$ suggests that only a single aR-tree exists, which indexes all points. In this extreme setting, SGC and BCA achieve their maximum efficiency.



(a) Independent

(b) Anti-correlated

Fig. 14. Total time vs. $|sg|$

In conclusion, we make the following remarks regarding the proposed algorithms for processing probabilistic contextual skyline queries. The naive BIA algorithm is impractical in all settings. Although SGC exhibits low CPU time, it is an attractive solution only when the number of candidates is small, i.e., in independent low cardinality and dimensionality datasets. The other non-indexed approach, CSA performs relatively well for medium-sized datasets and remains efficient even for very large RP domain sizes. The BCA algorithm is the most practical of the pack, as it exhibits solid performance in all scenarios tested.

VII. Conclusion

We introduced a methodology that allows the expression of skyline queries without explicitly stating preferences among attribute values. To handle the case of missing information, we derive a set of uncertain preferences based on users' profiles, i.e., from stated preferences for past situations or contexts. As a result, the dominance relationships among tuples become uncertain, which gives rise to probabilistic contextual skyline queries (p-CSQ). We introduced several non-indexed and index-based algorithms for processing p-CSQs, which are experimentally shown to significantly outperform a naive block nested loops approach.

In the future we plan to follow two directions for further work on this subject. The first is to develop techniques for efficiently processing top-k queries, where tuples are ranked based on their skyline probability. The second is to design methods that are cache-aware, i.e., use past query results in order to expedite processing of current p-CSQs.

References

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

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

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

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

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

[6] D. Sacharidis, P. Bouros, and T. Sellis, "Caching dynamic skyline queries," in *SSDBM*, 2008.

[7] D. Sacharidis, S. Papadopoulos, and D. Papadias, "Topologically sorted skylines for partially ordered domains," in *ICDE*, 2009.

[8] G. Koutrika and Y. E. Ioannidis, "Personalization of queries in database systems," in *ICDE*, 2004, pp. 597–608.

[9] R. Agrawal, R. Rantzau, and E. Terzi, "Context-sensitive ranking," in *SIGMOD*, 2006, pp. 383–394.

[10] K. Stefanidis, E. Pitoura, and P. Vassiliadis, "Adding context to preferences," in *ICDE*, 2007, pp. 846–855.

[11] J. Pei, B. Jiang, X. Lin, and Y. Yuan, "Probabilistic skylines on uncertain data," in *VLDB*, 2007, pp. 15–26.

[12] X. Lian and L. Chen, "Monochromatic and bichromatic reverse skyline search over uncertain databases," in *SIGMOD*, 2008, pp. 213–226.

[13] R. Agrawal and E. L. Wimmers, "A framework for expressing and combining preferences," in *SIGMOD*, 2000, pp. 297–306.

[14] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "Prefer: A system for the efficient execution of multi-parametric ranked queries," in *SIGMOD*, 2001, pp. 259–270.

[15] W. Kießling, "Foundations of preferences in database systems," in *VLDB*, 2002, pp. 311–322.

[16] J. Chomicki, "Preference formulas in relational queries," *ACM Transactions on Database Systems*, vol. 28, no. 4, pp. 427–466, 2003.

[17] M. Lacroix and P. Lavency, "Preferences: Putting more knowledge into queries," in *VLDB*, 1987, pp. 217–225.

[18] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, "The onion technique: Indexing for linear optimization queries," in *SIGMOD*, 2000, pp. 391–402.

[19] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM*, vol. 22, no. 4, pp. 469–476, 1975.

[20] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Springer, 1985.

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

[22] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *ICDE*, 2003, pp. 717–816.

[23] P. Godfrey, R. Shipley, and J. Gryz, "Algorithms and analyses for maximal vector computation," *The International Journal on Very Large Data Bases*, vol. 16, no. 1, pp. 5–28, 2007.

[24] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Transactions on Database Systems*, vol. 33, no. 4, pp. 1–45, 2008.

[25] C. Y. Chan, P.-K. Eng, and K.-L. Tan, "Stratified computation of skylines with partially-ordered domains," in *SIGMOD*, 2005, pp. 203–214.

[26] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang, "Mining favorable facets," in *KDD*, 2007, pp. 804–813.

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

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

[29] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang, "Efficient computation of the skyline cube," in *VLDB*, 2005, pp. 241–252.

[30] R. C.-W. Wong, A. W.-C. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu, "Efficient skyline querying with variable user preferences on nominal attributes," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1032–1043, 2008.

[31] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang, "Dada: a data cube for dominant relationship analysis," in *SIGMOD*, 2006, pp. 659–670.

[32] M. L. Yiu and N. Mamoulis, "Efficient processing of top-k dominating queries on multi-dimensional data," in *VLDB*, 2007, pp. 483–494.

[33] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, "Finding k-dominant skylines in high dimensional space," in *SIGMOD*, 2006, pp. 503–514.

[34] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *ICDE*, 2007, pp. 86–95.