WPI-CS-TR-WPI-CS-TR-09-03 March 2009

Skyline and Mapping Aware Query Evaluation Across Disparate Data Sources

by

Venkatesh Raghavan
Shweta Srivastava
Elke A. Rundensteiner

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Skyline and Mapping Aware Query Evaluation Across Disparate Data Sources*

Venkatesh Raghavan, Shweta Srivastava and Elke A. Rundensteiner
Department of Computer Science, Worcester Polytechnic Institute, USA
{venky, shweta.sriv, rundenst}@cs.wpi.edu

ABSTRACT

Growing interests in multi-criteria decision support applications have resulted in a flurry of efficient skyline algorithms. In practice, real-world decision support applications require to access data from disparate sources. Existing techniques define the skyline operation to work on a single set, and therefore, treat skylines as an "add-on" on top of a traditional Select-Project-Join query plan. In many real world applications, the skyline dimensions can be anti-correlated (e.g., attribute pairs {price, mileage} for cars or {price, distance} for hotels). Anti-correlated data are skyline-unfriendly and thus ignored by existing techniques. In this work, we propose a robust execution framework called *SKIN* to evaluate skyline over joins. The salient features of *SKIN* are: (a) effective in reducing the two primary costs, namely the cost of generating the join results and the cost of dominance comparisons to compute the final skyline of join results, (b) shown to be robust for both skyline-friendly (independent and correlated) as well as skyline-unfriendly (anti-correlated) data distributions. *SKIN* is effective in exploiting the skyline knowledge in both local (individual data source), as well as across disparate sources – to significantly reduce the above mentioned two primary costs incurred during the evaluation of skyline over join. Our experimental study demonstrates the superiority of our proposed approach over state-of-the-art techniques to handle a wide variety of data distributions.

1. INTRODUCTION

The intuitive nature of specifying user preferences has made skyline computation critical for many multi-criteria decision support (MCDS) applications. Skyline evaluation is characterized by the following features: (1) the user is interested in minimizing (or maximizing) a variety of criteria, and (2) the underlying data set may not contain a single overall best match [2]. Traditional queries return only exact matches. Instead, the goal of MCDS queries is to return a set of non-dominated results meaning each result is better than the others in at least one criterion.

*This work is supported by the National Science Foundation under Grant No. IIS-0633930 and CRI-0551584.

1.1 Motivation

In recent years several skyline algorithms have been proposed [2, 6, 11] to provide this capability. The skyline operation, similar to aggregate computations, is traditionally evaluated last after joins and group-bys in a query plan, thus assuming its input to be a *single set* of homogeneous data [2]. However, this common assumption is rather limiting since a many MCDS applications in practice do not operate on just a single source. Instead, they are required to: (1) access data from disparate sources with varying schemas, and (2) combine several attributes across these sources by possibly complex user-defined functions (mappings) to characterize the final composite product. In many real world applications the attributes values exhibit anti-correlation, for example in a hotel reservation application the cost of the hotel increases with the nearness to popular tourist interests, and in an automobile purchase system the mileage on the car is inversely proportional to its asking price. Existing techniques proposed to handle skyline over joins [9, 19] are not viable for such anti-correlated data sets. For this reason, existing techniques in the literature simply do not test against data sets that exhibit such correlations, and rather they limit themselves to the skyline-friendly correlated and independent data sets. The focus of this work is to present a robust evaluation strategy that handles all three extreme distributions. We first substantiate these requirements by drawing from a diverse set of applications.

Supply-Chain Management. A manufacturer in the supply chain aims to maximize profit and market share while minimizing overhead and delays. This is achieved by structuring an optimal production and distribution plan through the investigation of various alternatives. $Q1$ identifies suppliers that can produce *"100K"* units of part *"P1"* and pairs them with transporters who can deliver it. The preference is to minimize both costs and delays.

```
Q1: SELECT R.id, T.id, (T.shipTime) as delay
(R.uPrice + T.uShipCost) as tCost,
FROM Suppliers R, Transporters T
WHERE R.country = T.country AND
'P1' in R.suppliedParts AND R.manCap >= 100K
PREFERRING LOWEST(tCost) AND LOWEST(delay)
```

Internet Aggregators. The rapid increase in the number of online vendors has resulted in Internet aggregators such as Froogle[1] for durable goods, and Kayak[2] for travel services, are fast growing in popularity. Aggregators access and combine data form several sources to produce complex results that are then pruned by the skyline operation. Consider the query $Q2$, where the user is planning

[1]http://froogle.google.com/shoppinglist, [2]www.kayak.com

a holiday in Europe visiting both Rome and Paris. The user has different preferences in each leg of the journey. For instance since Rome is an ancient city, the user is willing to walk twice as much in Rome than in Paris. In addition, the user has a cumulative goal of minimizing the total cost of the trip.

```
Q2: SELECT R.id Rome, T.id Paris,
(R.price + T.price) as tCost,
(2 * R.distance+T.distance) as tDistance
FROM RomeHotels R, ParisHotels T
PREFERRING LOWEST(tCost) AND
LOWEST(tDistance)
```

Drug Discovery. The life cycle for drug discovery spans over several years starting with the synthesis of the compound first in the laboratory until the finished product is introduced in the market. Molecular modeling plays a vital role in drug discovery and is used to identify protein-ligand pairs that can point to potential directions of further investigation. This involves screening large data banks of ligands against a protein, and then ranking the protein-ligand pair interactions according to scoring functions based on structure, energy forces or empirical data of each pair with the goal of maximizing the intermolecular interaction energy between the molecules [5].

To summarize, in this work we target user queries that involve combining both skyline and mapping operations over disparate data sources, here known as *SkyMapJoin* queries.

1.2 State-of-the-Art Approach

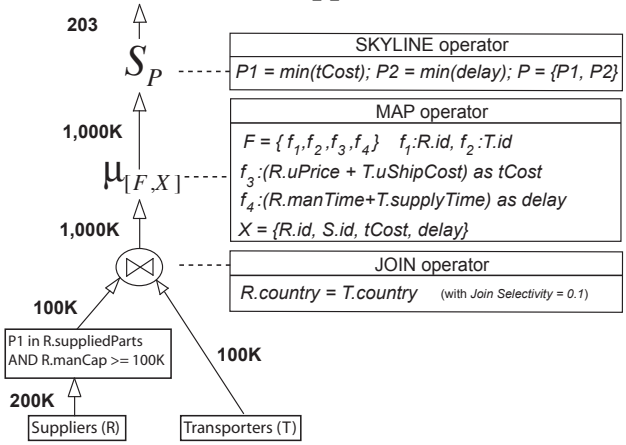


Figure 1: Traditional Query Plan For Motivating Query $Q1$

The traditional approach is to view skylines as an independent operation to join evaluation that is, a *join-first, skyline-later (JF-SL)* paradigm. [2, 12] tends to employ this paradigm by dividing the query execution into disjoint steps. As illustrated in Figure 1, the objects in relation R which satisfy the filter conditions are first selected, followed by the join evaluation. The join results are then transformed by the mapping operator. Finally, the skyline evaluation returns the output skyline of supplier-transporter pairs. In our particular data sets with $|R|$=200K and $|T|$=100K objects, the selectivity of the filter conditions is 0.5 and join selectivity = 0.1. JF-SL first generates all 1 million join results. The skyline evaluation requires $\approx$ 1 million comparisons.

Discussion. To draw a parallel to *Select* in *Select-Project-Join* (SPJ) queries, there are scenarios when the skyline functionality can be *pushed inside* as well as sometimes *pushed-through* joins [2, 8]. Such rewrite rules aim to facilitate processing. For instance, pushing Select inside the Cartesian product results in the theta-join operator, for which then numerous efficient strategies such as index-join, sort-merge join, etc., have been devised in the literature. We observe that the skyline operation can be viewed as a complex and expensive filter operation. In our motivating query $Q1$, the attribute value of each skyline dimension is generated on the fly by the join and mapping operations. [12] noted that skylines cannot be correctly pushed-through in many scenarios. In traditional filters increasing the number of conditions will usually increase its pruning capacity. However, adding a preference to a skyline operation will reduce their filtering capacity and increase the cardinality of its result set up to the size of the entire relation [4]. Therefore, the *partial push-through* of skylines can be an expensive and sometimes ineffective proposition. [9, 19] proposed techniques that rely heavily on the effectiveness of making local *partial push-through* decisions on each individual data source. By relying primarily on the principle of *partial push-through*, [9, 19] are *unable to see the forest from the individual trees*. Thus by only exploiting the partial push-through they suffer from the following two limitations: (1) not being robust for a wide variety of data sets as they themselves report [19], and (2) for the generated join results [9, 19] cannot further optimize the skyline evaluation even when such opportunities can be found provided the knowledge of the *forest* is exploited.

1.3 Problem Definition

In this work, we aim to design a query execution framework that is able to: (1) leverage the skyline knowledge at various steps of query evaluation, (2) minimize the number of join results generated, (3) also reduce the number of skyline comparisons over this reduced set of join results, and (4) exhibit competitive performance for all distributions, including the skyline-unfriendly anti-correlated data sets. The **optimization-mantra** in this work is to "*avoid joining objects that will not result in a skyline result and avoid evaluating skylines for objects that will not join.*"

1.4 Proposed Approach

We propose **SKIN** *(SKyline INside Join)* an efficient methodology to evaluate SkyMapJoin queries. Guided not only by the input space of each data source, *SKIN* is able to look ahead into the final output space. This allows us to determine the interrelations between the two spaces and identify optimization opportunities missed by current techniques. In addition, *SKIN* is able to efficiently evaluate the query at various levels of data abstraction. As a result of these principles, we significantly reduce the expensive object-level processing and achieve our optimization goals. In this effort, we form a multi-dimensional, higher-level abstraction of both the input and output data spaces. Thereafter, we exploit the insight that the join, skyline and mapping operations can now be performed in the coarser granularity data spaces. For our example with $|R|$=200K and $|T|$= 100K objects, *SKIN* achieves nearly 2 orders of magnitude reduction in the number of join results generated to $\approx$14K from 1 million pairs as generated by JF-SL.

In addition, we succeed to minimize the skyline computation costs to generated the join results and then mapped results (here known as combined objects) by piggy backing the knowledge about the output space obtained from the previous step. For each generated combined object, we are now able to restrict the skyline comparisons to only those combined objects mapped to a subset of output

partitions that they can potentially dominate and vice-versa.

Our proposed *SKIN* execution strategy is shown to successfully reduce the total number of skyline comparisons in several cases by orders of magnitude against state-of-the-art approaches, for the various data distributions as generated by the de-facto skyline stress test [2]. For our example, we only require a manageable 11K dominance comparisons instead of 1.2 million comparisons required by JF-SL (Figure 1). This represents 2 orders of magnitude reduction of skyline computations for this particular scenario.

1.5 Contributions

The contributions of this work include:

1. Propose *SKIN*, a robust methodology to process SkyMapJoin queries. To our best knowledge, *SKIN* is the first algorithm to efficiently exploit the insight that the join, skyline and mapping operations can not only be performed at different levels of data abstraction, but also be simultaneously performed in both input and output spaces.

2. Existing state-of-the-techniques [9, 19] do not test their approach over anti-correlated data sets, rather they restrict themselves to only the skyline friendly distributions such independent and correlated. In this work, we provide a comprehensive experimental evaluation of the existing techniques over all distributions commonly used in skyline literature as a stress test [2].

3. Our performance analysis demonstrates the superiority of our proposed approach for many cases (such as anti-correlated and some independent data sets) by 1-2 orders of magnitude faster over state-of-the-art methods. For the skyline friendly data sets such as correlated data, SKIN has similar performance as state-of-the-art techniques [9, 12, 19]. In addition, we report that *SKIN* on an average produces 50% less number of join results in comparison to state-of-the-art techniques and achieves 1-2 orders of magnitude fewer skyline comparisons to produce the final result.

1.6 Organization

The rest of the paper is organized as follows. In Section 2 we review the preference model used in skyline queries, and the mapping operator. In Section 2.3 we introduce the proposed extensions to the algebra model. We present a brief overview of our *PB-SMJ* methodology in Section 3, while details of our techniques are presented in Sections 4-8. In Section 9 we discuss our performance study. The survey of related works is presented in Section 10, and Section 11 concludes the paper.

2. PRELIMINARIES

In this section, we review the *preference model* [10] and the *algebra model* used to represent an SMJ query such as $Q1$. Each d-dimensional object is defined by a set of attributes $A = \{a_1, \ldots, a_d\}$. For a given object r_i, the value of the attribute a_k can be accessed as $r_i[a_k]$. $Dom(a_k)$ is domain of the attribute a_k and $Dom(A) = Dom(a_1) \times \ldots \times Dom(a_d)$.

2.1 Mapping Functions and Map Operator

The *map operator* (μ) is defined based on a set of k mapping functions. For each input object r_i the mapping function f_j, in a set of k mapping functions $\mathcal{F}$, takes as input a set of distinct attributes $B_j \subseteq A$ and returns a newly computed attribute x_j. That is, $f_j : Dom(B_j) \rightarrow Dom(x_j)$ and $\mathcal{F} = \{f_1, f_2, \ldots f_k\}$.

Map Operator ($\mu_{[\mathcal{F},X]}(R)$) applies a set of k mapping functions $\mathcal{F}$ to transform each d-dimensional input object $r_i \in R$ into a k-dimensional output object r'_i defined by a set of attributes $X = \{x_1, \ldots, x_k\}$, where x_i is generated by the function $f_i \in \mathcal{F}$.

2.2 Preference Model and Skyline Operator

Given a set of attributes $E \subseteq A$, the preference P_i over the set of tuples R is defined as $P_i := (E, \succ_P)$ where $\succ_P$ is a *strict partial order* on the domain of E. Given a set of preferences $\{P_1, \ldots, P_m\}$, their combined Pareto preference P is defined as a set of equally important preferences.

DEFINITION 1. *For a set of d-dimensional tuples R and preference $P = (E, \succ_P)$ over R, a tuple $r_i \in R$* **dominates** *tuple $r_j \in R$ based on the preference P (denoted as $r_i \succ_P r_j$), iff* $(\forall(a_k \in E)\ (r_i[a_k] \succeq r_j[a_k]) \wedge \exists(a_l \in E)\ (r_i[a_l] \succ r_j[a_l]))$.

Skyline Operator ($\mathcal{S}_P(R)$), given a set of objects R and a preference P, returns a subset of non-dominated objects in R.

2.3 Extended Algebra Model

To facilitate the pushing of the skyline operation into the map and/or the join, we introduce three new operators, namely *SkyMap* ($\widehat{\mu}$), *SkyJoin* ($\widehat{\bowtie}$) and *SkyMapJoin* ($\widehat{\Psi}$).

SkyMap ($\widehat{\mu}_{[\mathcal{F},X,P]}$) performs the following operations in order: (1) apply the set of k mapping functions $\mathcal{F}$ to transform each d-dimensional object $r_i \in R$ into a k-dimensional object r'_i defined by the set of attributes X, and then (2) generate the skyline of transformed objects by the preference $P = (E, \succ_P)$, where $E \subseteq X$.

SkyJoin ($\widehat{\bowtie}_{[\mathcal{C},P]}$) combines objects from its input data sets based on the conditions in $\mathcal{C}$ and returns a set of non-dominated combined-objects based on the preference P. If $\mathcal{C} = \phi$ then the operator returns the set of non-dominated Cartesian product results.

SkyMapJoin ($\widehat{\Psi}_{[\mathcal{C},\mathcal{F},X,P]}$) performs the following operations in order: (1) combine objects from the input data sets based on the conditions in $\mathcal{C}$, (2) apply the set of mapping functions $\mathcal{F}$ to transform each combined-object to generate a transformed combined-object with attributes X, and (3) generate the skyline of combined-objects by the preference $P = (E, \succ_P)$, where $E \subseteq X$.

Example 2: Figure 2.a depicts the traditional query plan for $Q2$ using canonical algebra operators where the results of the Cartesian product are fed to the map operator. For each Rome and Paris hotel pair, the map operator calculates $tCost$ and $tDistance$. The transformed combined-objects are then given as inputs to the skyline operator to generate the final result. Figures 2.b and 2.c represent the equivalent *SkyMap*- and *SkyMapJoin*-based query plans of $Q2$ generated by pushing first the skyline operation into the map operator, and then pushing both into the join.

3. SKIN: THE PROPOSED APPROACH

In the remainder of this paper, we introduce our proposed **SKIN** (SKyline INside Join) methodology for evaluating the *SkyMapJoin*

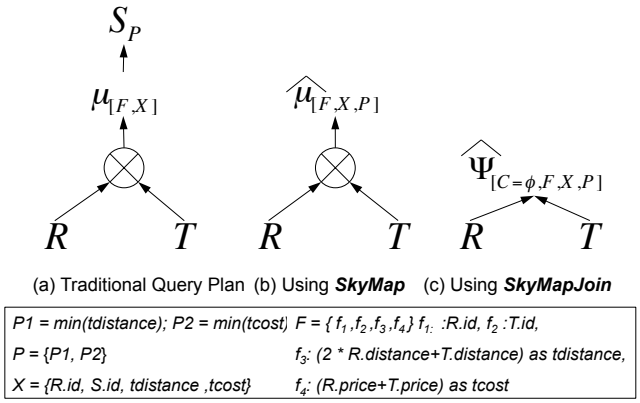


Figure 2: Multiple Equivalent Plans for *SkyMapJoin* query Q2

operator. The *SkyMap* operation is straightforward and thus omitted for conciseness. The *SkyJoin* operator can be viewed as a special case of the *SkyMapJoin* operator where each of the k mapping functions is a trivial projection.

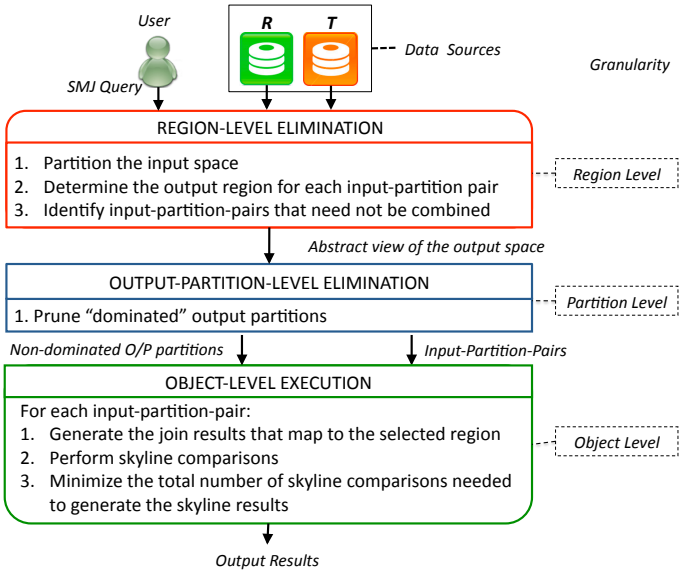


Figure 3: Overview of the SKIN Approach

In Figure 3 we present the overview of the *foundation* of our approach called *SKIN*. For each input data source, we form an m-dimensional abstraction where m is the number of skyline dimensions in the combined object. Then, the first phase, called **region-level elimination** phase, targets to avoid the generation of combined objects altogether. Given a pair of input partitions from R and T, we determine: (1) whether the join operation between the objects in the input partition pairs will result in at least one combined-object (see details in Sections 4–7), and (2) the region in the mapped output space into which the future combined-objects will fall during the actual object-level evaluation. Next, we identify output regions that are dominated by other regions. As we will show in Section 4 (Lemma 1), dominated regions are guaranteed to not contribute to the final skyline. Therefore, the join evaluation that generates objects that map to such dominated output regions need not be performed altogether, thereby saving on combined-object generation costs as well as dominance comparison costs (see Section 4).

Partition-level elimination, our second phase aims to reduce dominance comparisons between combined-objects. Here, we perform query evaluation at the abstraction of partitions in the output space. In other words, we partition the output space such that the each output regions is composed of a set of output partitions. We observe that for some regions, a subset of their output partitions are dominated by other regions. Unlike in the above *region-level elimination* such partially dominated regions cannot be entirely discarded. However, we show by Lemma 2 that such dominated output partitions are guaranteed to not contribute to the final skyline. Therefore, combined-objects that map to these dominated output partitions can be immediately discarded without conducting any skyline comparisons (Section 5). In summary, *region-* and *partition-level elimination* phases eliminate many output regions and partitions respectively without any object-level access.

Finally, the third phase, called **object-level execution**, further reduces the total number of dominance comparisons needed to generate the final skyline. For each generated combined-object $r_f t_g$ we minimize the number of comparisons by: (1) eliminating all output partitions dominated by $r_f t_g$, and (2) restricting the object-level skyline comparisons to only a small subset of partitions containing combined-objects, namely those in output partitions that it can potentially dominate, and vice-versa (see Section 6). This third phase piggybacks the former steps by reusing the partition information produced by them. We present the details; both underlying theory as well as concrete algorithms for each of the three phases in Sections 4, 5, and 6 respectively.

The principle of skyline *partial push-through* is complimentary to our approach and is applied as a pre-filtering phase to our core idea (see Section 9.3).

Notations:
• Each input partition in R is denoted as I_i^R
• Each output partition is denoted as O_i
• $\mathcal{I}^R$ is a set of all input partitions in R
• $\mathcal{O}$ is a set of all output partitions
• $[I_i^R, I_j^T]$ input-partition pairs whose combined-objects map to region $\mathcal{R}_{i,j}$
• $\mathfrak{R}$ is a set of all output regions called as *Region Collection*
• **LOWER**(X)**:** Returns the *lower-bound* point of the region or partition X
• **UPPER**(X)**:** Returns the *upper-bound* point of the region or partition X
• **MAP_OBJECT**($r_c t_d$, $\mathcal{F}$, δ)**:** Returns the output partition to which $r_c t_d$ maps to.
• **MAP_REGION**($\mathcal{R}_{i,j}$, $\mathcal{F}$, δ)**:** Returns a set of output partitions that $\mathcal{R}_{i,j}$ maps to.
• **MARK**(O_i)**:** Marks the partition O_i as "*non-contributing*"
• **IS_MARKED**(O_i)**:** Return *true* if output partition O_i is marked, *false* otherwise.

Table 1: Notations Used In This Work

4. PHASE I: REGION-LEVEL ELIMINATION

We now present the first phase of our *SKIN* methodology named *region-level optimization*. This phase avoids the generation of many

combined-objects. To easily highlight the core areas of optimization we first consider the motivating query $Q2$ where the join condition $\mathcal{C}$=ϕ. In Section 7 we slightly extend the core approach to handle general join predicates, as in query Q1.

We first partition each of the input data source. For the remainder of this elaboration we employ an m-dimensional grid for partitioning the data space, with m being the number of skyline dimensions. In Section 8 we provide a more general discussion on partitioning methods. In Figure 4.b the input data set T is partitioned into a 2-D grid. Each partition is uniquely identified by its bottom-left coordinates, the *lower bound* retrieved by the function LOWER(I_i^T). We define $\mathcal{I}^T$ as the **set of all non-empty input partitions** for T and $\mathcal{I}^T$=$\{I_1^T[(1,5)(2,6)], I_2^T[(3,1)(4,2)], I_3^T[(5,0)(6,1)]\}$.

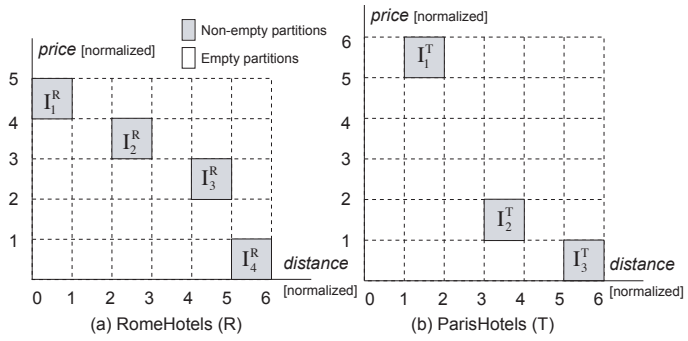


Figure 4: Partitioning Of Input Datasets

In *region-level elimination*, for each non-empty partition, $I_i^R \in \mathcal{I}^R$ and each non-empty partition $I_j^T \in \mathcal{I}^T$, we determine the region of the output mapped space to which their join results will map to. This is achieved by applying the set of k mapping functions $\mathcal{F}$ to *lower-* and *upper-* bounds of the two input partitions respectively. The region corresponding to an input-partition-pair $[I_i^R, I_j^T]$ is called an **output region** (denoted as $\mathcal{R}_{i,j}$).

To elaborate, for query $Q2$ the mapping functions f_1 and f_2 compute two attributes, namely (f_1:$R.price$+$T.price$) computes *tprice* and ($2 * R.distance + T.distance$:$f_2$) *tdistance*. The objects that map to the partition $I_1^R[(0,4)(1,5)]$ in Figure 4.a when combined with objects in partition $I_2^T[(3,1)(4,2)]$ in Figure 4.b resulting combined-objects that are guaranteed to fall into the region bounded by the points $b(3,5)$ and $B(6,7)$ in Figure 5.

We observe that: (1) the mapping and skyline functionality can now be applied at this higher level of abstraction, (2) output regions can always be determined *a priori* without any object-level data access, and (3) as long as both of the input partitions are non-empty, the corresponding output region is guaranteed to be populated. The set of all populated regions is called the **Region Collection** ($\mathfrak{R}$).

In query $Q2$ the preference is to minimize all skyline-dimensions. In a pessimistic scenario for each output region $\mathcal{R}_{i,j}$, all the combined-objects that map to it would lie on the upper-bound point of $\mathcal{R}_{i,j}$. We introduce the notion of the **pessimistic output skyline**, denoted as $\mathcal{S}_{pes}$ to identify the dominated output regions.

DEFINITION 2. *For the preference P, the region collection $\mathfrak{R}$, $U = \{UPPER(\mathcal{R}_{i,j}) \mid \forall(\mathcal{R}_{i,j} \in \mathfrak{R})\}$, i.e., the set of upper-bounds of all output regions in $\mathfrak{R}$. Pessimistic skyline $\mathcal{S}_{pes}$ is defined as the skyline over U based on P, i.e., $\mathcal{S}_{pes} = \mathcal{S}_P(U)$.*

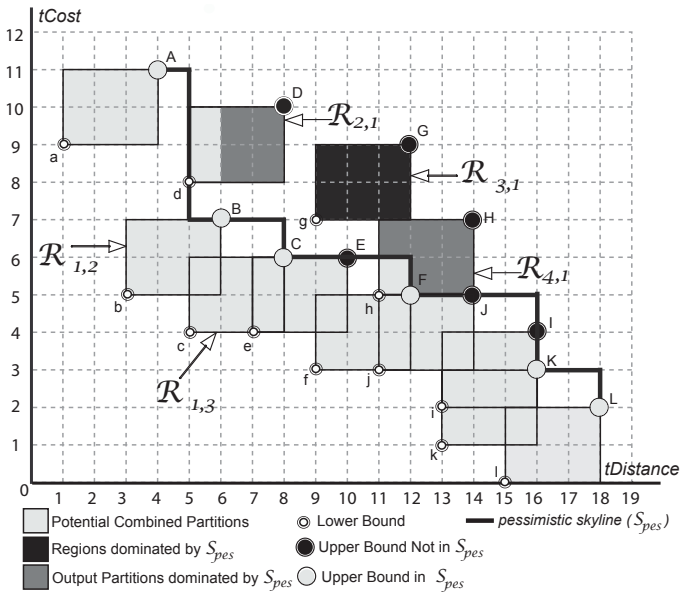


Figure 5: Pessimistic Skyline, S_{pes}

In Figure 5 the region $\mathcal{R}_{1,2}$ with the upper bound $B(6,7)$ clearly dominates $G(12,9)$, the upper bound of the region $\mathcal{R}_{3,1}$. Therefore H is not in the pessimistic skyline, $\mathcal{S}_{pes}$ = { A, B, C, E, F, K, L }.

Lemma 1. *For a region collection $\mathfrak{R}$, its pessimistic skyline S_{pes}, preference P and an output region $\mathcal{R}_{i,j} \in \mathfrak{R}$, if $\exists s \in S_{pes}$ such that $s \succ_P$ LOWER($\mathcal{R}_{i,j}$) then no combined-object $r_f t_g \in \mathcal{R}_{i,j}$ can be contained in the output skyline.*

Proof: Proof by contradiction. Assume that $\exists(r_f t_g \in \mathcal{R}_{i,j})$, $\exists s \in S_{pes}$ and $s \succ$ LOWER($\mathcal{R}_{i,j}$), but $r_f t_g$ in the output skyline. By Definition 2, $\exists\, \mathcal{R}_{x,y}$ s.t. s =UPPER($\mathcal{R}_{x,y}$). Since $\mathcal{R}_{x,y} \neq \phi$, let $r_b t_c \in \mathcal{R}_{x,y}$. Since $s \succ$ LOWER($\mathcal{R}_{i,j}$), $r_b t_c \succ r_f t_g$ and thus $r_f t_g$ is not in the output skyline. This is a contradiction. Therefore, the output region $\mathcal{R}_{i,j}$ will not contribute to the output skyline. ■

RULE 1. **Region Elimination.** *For a region collection $\mathfrak{R}$, its pessimistic skyline S_{pes}, and preference P and a region $\mathcal{R}_{i,j} \in \mathfrak{R}$, if $\exists s \in \mathcal{S}_{pes}$ s.t., $s \succ$ LOWER($\mathcal{R}_{i,j}$), then by Lemma 1, the objects in the input partitions I_i^R need never be combined with those in I_j^T since the resulting combined-objects are guaranteed to not be in the final skyline.*

For example, in Figure 5 the output region $\mathcal{R}_{3,1}$ with the lower-bound $g(9,7)$ is dominated by the pessimistic skyline points $B(6,7)$ as well as $C(8,6)$. By Lemma 1 the combined-objects generated from the input partition pair $[I_3^R, I_1^T]$ are guaranteed to not contribute to the output skyline, thus need not be generated. Algorithm 1 is the pseudo-code for the region-level elimination.

To summarize, the properties of region-level elimination are:

1. Only the output regions that are guaranteed to be populated are considered for join evaluation.

2. Dominated regions are guaranteed to not contribute to the final results and therefore need not be considered for join evaluation.

Algorithm 1 Region-Level Elimination

Input: $\mathcal{F}, P, \mathcal{I}^R, \mathcal{I}^T$
Output: $\mathfrak{R}$ {Region Collection}
1: $\mathfrak{R} = \phi$; $U = \phi$
2: **for** each partition $I_i^R \in \mathcal{I}^R$ **do**
3: **for** each partition $I_j^T \in \mathcal{I}^T$ **do**
4: $\mathcal{R}_{i,j} \leftarrow$ Associated output region for $[I_i^R, I_j^T]$
5: Add UPPER($\mathcal{R}_{i,j}$) to U; Add $\mathcal{R}_{i,j}$ to $\mathfrak{R}$
6: **end for**
7: **end for**
8: $\mathcal{S}_{pes} = \mathcal{S}_P(U)$
9: **for** each output region $\mathcal{R}_{i,j} \in \mathfrak{R}$ **do**
10: **if** $\exists(s \in \mathcal{S}_{pes})$ $(s \succ_P$ LOWER($\mathcal{R}_{i,j}$)) **then**
11: Remove $\mathcal{R}_{i,j}$ from $\mathfrak{R}$ {By Lemma 1, Rule 1}
12: **end if**
13: **end for**
14: **return** $\mathfrak{R}$

Time Complexity. The total number of input partitions for the input data sets R and T is denoted by n^R and n^T respectively. If $n^R = n^T = n$, then the time complexity to determine all n^2 output regions is $\mathcal{O}(n^2)$. The time complexity of generating the pessimistic skyline is in the worst case $\mathcal{O}(n^4)$ based on the Block-Nested-Loop (BNL) skyline algorithm [2]. Therefore, region-level elimination has the time complexity of $\mathcal{O}(n^2 + n^4) \approx \mathcal{O}(n^4)$. This optimization is beneficial because it: (1) is at the granularity of output regions and does not require any object-level data access, (2) has the potential to eliminate $n^2 - 1$ out of n^2 output regions in the best case scenario, and (3) has a significantly cheaper time complexity than $\mathcal{O}(N^4)$ for popular skyline algorithms [2], where $|R|$=$|T|$=N. Since typically, $n << N$, $\mathcal{O}(n^4) << \mathcal{O}(N^4)$.

5. PHASE II: OUTPUT-PARTITION LEVEL ELIMINATION

The **partition-level elimination** phase aims to reduce dominance comparisons needed to produce the final skyline. Each region $\mathcal{R}_{i,j}$ is mapped to one or possibly several output partitions. In Figure 5, $\mathcal{R}_{1,1}$ maps to the set of output partitions {O[(1,9)(2,10)], O[(1,10)(2,11)], O[(2,9)(3,10)], O[(2,10)(3,11)], O[(3,9)(4,10)], O[(3,10)(4,11)]}. Different regions may map to common output partitions. For example, $\mathcal{R}_{1,2}$ and $\mathcal{R}_{1,3}$ in Figure 5 share the output partition $O[(5,5)(6,6)]$. The *set of all output partitions* in the mapped output space is denoted as $\mathcal{O}$.

We observe that for some regions, only a subset of the output partitions they contain are dominated by the pessimistic skyline $\mathcal{S}_{pes}$. In Figure 5 for $\mathcal{R}_{2,1}$ the output partitions: O[(6,8)(7,9)], O[(6,9)(7,10)], O[(7,8)(8,9)] and O[(7.9)(7,10)] are dominated by point $B(6,7)$ $\in \mathcal{S}_{pes}$. Prior to the actual generation of combined-objects that map to $\mathcal{R}_{2,1}$ we cannot easily determine which of the output partitions for $\mathcal{R}_{2,1}$ the combined-objects will fall into. For this reason, unfortunately we cannot entirely discard the partially dominated output region $\mathcal{R}_{2,2}$ as in Section 4. Therefore, we have to first generate the combined-objects that map to such partially dominated regions. As we show next, we can however exploit this fact of partially dominated output regions. The intuition here is that for a point B to be in $\mathcal{S}_{pes}$ there must exist an output region, here $\mathcal{R}_{1,2}$, such that UPPER($\mathcal{R}_{1,2}$) = B. All output regions are guaranteed to be populated i.e., $\mathcal{R}_{1,2} \neq \phi$ and $\exists r_b t_c \in \mathcal{R}_{1,2}$ such that $r_b t_c$ dominates a subset of output partitions mapped to $\mathcal{R}_{2,1}$. Therefore, combined-objects that map to output partitions: O[(6,8)(7,9)], O[(6,9)(7,10)], O[(7,8)(8,9)] and O[(7.9)(7,10)] are guaranteed to not contribute to the final skyline.

Lemma 2. *Given a region collection $\mathfrak{R}$, its pessimistic skyline S_{pes}, preference P and an output partition $O_l \in \mathcal{O}$ s.t., $\exists(s \in \mathcal{S}_{pes})$ $(s \succ_P$ LOWER(O_l)), then no combined-object $r_f t_g \in O_l$ can be contained in the output skyline.*

We omit the proof for By Lemma 2 for conciseness. Combined objects that map to dominated output partitions during actual object-level evaluation can safely discard thereby avoid performing any skyline comparisons on such combined-objects. In addition, we mark all such dominated output partitions as **non-contributing**. Algorithm 2 is the pseudo-code for the partition-level elimination. For each region $\mathcal{R}_{i,j}$, it determines its corresponding partitions by the function MAP_REGION($\mathcal{R}_{i,j}, \mathcal{F}, \delta$) (Line: 2). For each output partition O_l we determine if it is dominated by a pessimistic skyline point (Line: 4-5). All dominated output partitions are marked as "non-contributing" by the function MARK(O_l) (Line: 6).

Algorithm 2 Output Partition-Level Elimination

Input: $\mathfrak{R}$ {Region Collection}, S_{pes} {Pessimistic Skyline}
Output: $\mathcal{O}$ {Set of output partitions}
1: **for** each output region $\mathcal{R}_{i,j} \in \mathfrak{R}$ **do**
2: **for** each o/p partition $O_l \in$ MAP_REGION($\mathcal{R}_{i,j}, \mathcal{F}, \delta$) **do**
3: Add O_l to $\mathcal{O}$
4: **end for**
5: **end for**
6: **for** each output partition $O_l \in \mathcal{O}$ **do**
7: **if** $\exists(s \in \mathcal{S}_{pes})$ $(s \succ$ LOWER(O_l)) **then** MARK(O_l)
8: **end for**
9: **return** $\mathcal{O}$

Time Complexity. This phase eliminates the skyline comparisons for combined-objects that map to any dominated output partitions. In the worst case scenario, the region-level elimination is unsuccessful in eliminating any output region. Then the pessimistic skyline has n^2 points that correspond to the upper bounds of all n^2 output regions. The total number of output partitions is denoted as n_o. The time complexity of the partition-level elimination is $\mathcal{O}(n_o^2)$. This overhead is small because, (1) it is at the granularity of output partitions and does not require any object-level data access, and (2) in a typical database $n_o^2 << N^2$.

6. PHASE III: OBJECT-LEVEL EXECUTION

We now introduce the processing logic for the actual generation of combined-objects and skyline evaluation assuming the above pruning steps have been carried out. For each remaining input partition pair $[I_i^R, I_j^T]$ the object-level execution logically involves three steps. First, each object $r_f \in I_i^R$ is combined with each object $t_g \in I_j^T$ to generate the combined object $r_f t_g$. Second, the newly generated combined object $r_f t_g$ is then mapped to its corresponding output partition O_h. Finally, $r_f t_g$ is compared against other existing combined-objects to generate the output skyline. We optimize the skyline comparison step as follows: first, if the newly generated combined object $r_f t_g$ that maps to an output partition that is marked as "non-contributing" we can safely discard with

out further processing. If $r_f t_g$ maps to an output partition is not marked "non-contributing" we cannot discard $r_f t_g$ without having to perform skyline comparisons. Second, in such scenarios we next mark all partitions dominated by $r_f t_g$ as being "non-contributing" by following the principle stated in Lemma 2. Please note that this sub-task of marking dominated partitions is done only for the first combined object falls in the O_h. In Figure 6, the combined-object $r_f t_g \in O_h$ clearly dominates the output partitions in the top-right corner, i.e., every $O_q \in \mathcal{O}$ where $r_f t_g \succ_P$ LOWER(O_q) holds. Third, we minimizing the total number of combined-object comparisons during skyline computation. We exploit the knowledge gained from the output space by the ordering the sequence in which the input partition pair $[I_i^R, I_j^T]$ are considered for object-level execution based on its nearness to origin given that we assume that the preference is to minimize across all dimensions.

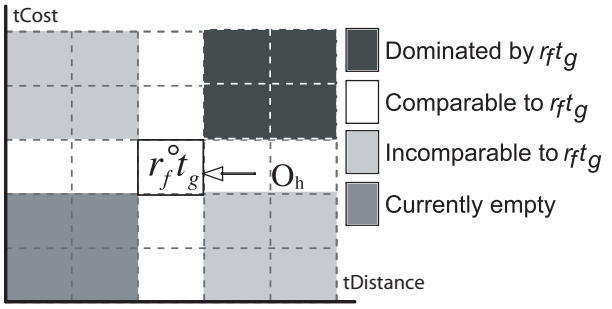


Figure 6: Object-Level Comparison Criteria

We tackle the last goal of reducing the number of skyline comparisons. In Figure 6 we observe that (1) combined-objects that map to output partitions in the top-left and the bottom right corner of O_h cannot dominate $r_f t_g$ and vice versa. Thus, such comparisons can be avoided. (2) Partitions in the bottom-left corner of O_h are guaranteed to be empty, else O_h would have been be marked as "non-contributing" in an earlier iteration. (3) $r_f t_g \in O_h$ can be only dominated by combined objects that map to the slice of partitions that either have the same $price$ or $distance$ as O_h.

RULE 2. **Comparable Partitions.** *Given a newly generated combined-object $r_f t_g$, let MAP_OBJECT($r_f t_g, \mathcal{F}, \delta$) = O_h, and let A = LOWER(O_h). Then, $r_f t_g$ needs to only be compared against combined-objects in output partition $O_q \in \mathcal{O}$ with B = LOWER(O_q), where* $\exists(1 \leq v \leq m)(A[l_v] = B[l_v])$.

Algorithm 3 lists the pseudo-code for the object-level execution. For each generated combined-object $r_f t_g$, we first identify the partition O_h to which it maps by the function MAP_OBJECT (Line: 4). If O_h is marked "non-contributing" we immediately discard $r_f t_g$ without any further processing (Line: 5-6). Otherwise, we mark all partitions $O_q \in \mathcal{O}$ which are dominated by O_h as "non-contributing" Line (9-11). Next, we begin the comparisons of $r_f t_g$ by first comparing it against other existing combined-objects that are mapped to the same O_h (Line: 12). If $r_f t_g$ is dominated in O_h we stop and move to the generation of the next combined-object. Otherwise, we compare $r_f t_g$ against combined-objects that are mapped to "*comparable partition*" as defined by Rule 2 (Line 13-14). After all these comparisons if $r_f t_g$ remains not dominated then and only then we insert $r_f t_g$ into O_h.

Optimization Benefits. Let each dimension in the output space be

Algorithm 3 Object-Level Execution

Input: $\mathcal{I}^R, \mathcal{I}^T, \mathcal{F}, \mathfrak{R}, P$
Output: Set of non-dominated combined-objects
1: **for** each output region $\mathcal{R}_{i,j} \in \mathfrak{R}$ **do**
2: **for** each object $r_f \in I_i^R$ and $t_g \in I_j^T$ **do**
3: Generate combined object $r_f t_g$
4: $O_h \leftarrow$ MAP_OBJECT($r_f t_g, \mathcal{F}, \delta$)
5: **if** IS_MARKED(O_h) **then**
6: discard $r_f t_g$
7: **else**
8: **for** each $O_q \in \mathcal{O}$ **do**
9: **if** $r_f t_g \succ_P$LOWER(O_q) **then** MARK(O_q)
10: **end for**
11: Call UpdatePartition($O_h, r_f t_g, P$)
12: **for** each O_q satisfying Rule 2 AND $r_f t_g$ not dominated **do**
13: Call UpdatePartition($O_q, r_f t_g, P$)
14: **end for**
15: Add $r_f t_g$ to O_h if $r_f t_g$ is still not dominated
16: **end if**
17: **end for**
18: **end for**
19: **return** all combined objects mapped to all output partitions
20: **procedure** UpdatePartition($O_q, r_f t_g, P$)
21: **for** each $r_x t_y \in O_q$ **do**
22: **if** $r_x t_y \succ_P r_f t_g$ **then** discard $r_f t_g$
23: **else if** $r_f t_g \succ_P r_x t_y$ **then** remove $r_x t_y$ from O_q
24: **end for**

partitioned into k partitions, so the d-dimensional grid has k^d output partitions. For any skyline algorithm in the worst case scenario all objects are in the final skyline. A naïve approach in the worst case scenario will need to compare against objects in all k^d partitions. Instead, for each newly generated object in the worst case we only perform dominance comparisons against objects that are mapped to a smaller set of $[k^d - (k-1)^d]$ partitions.

7. HANDLING JOIN PREDICATES

We now illustrate the full solution of our approach to also handle join predicates in queries such as $Q1$ (in Figure 1) where suppliers and transporters have to be from the same country. Unlike the scenario in Section 4 where we utilize all input partition pairs, we now only need to consider those input partition pairs that are guaranteed to have at least one tuple each with matching attributes values. That is, they indeed join and populate the corresponding region. Clearly, otherwise no input objects will find matching join partners, and thus the region will be empty. Intuitively, for this determination we perform join evaluation at a higher-level abstraction instead of actual object-level joins. In our example as in Figure 7, the input partition I_2^R shares with input partition I_2^T the domain values {Brazil, China, Mexico}, while both input partitions I_1^T and I_3^T share the attribute value {China}. Therefore, objects in I_2^R are guaranteed to find at least one join pair in each of the partitions I_1^T, I_2^R and I_3^T. Conversely, in Figure 7, we do not consider the pairs [I_3^R,I_1^T], [I_3^R,I_2^T] and [I_3^R,I_3^T], since there is no partition in T that shares any domain value with I_3^R, in this case *Indonesia*.

To determine if an input-partition shares at least a single domain value, we maintain a signature for each partition to capture the domain values of its member objects. This signature could either be a Bloom Filter, bit vector, etc. In Figure 7 for each partition in R

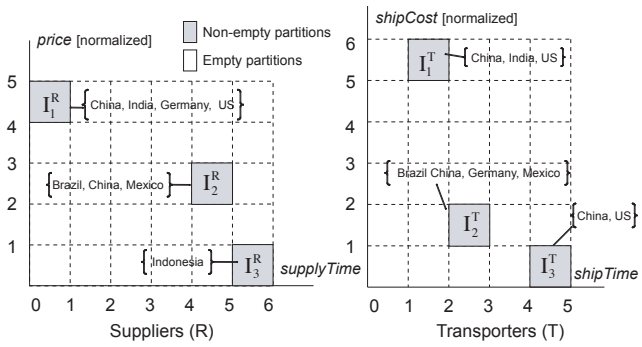


Figure 7: Partitioning For Suppliers (R) and Transporters (T)

(or T) we maintain a list of countries to which the suppliers (or transporters) belong to, e.g., I_3^T has suppliers from *China* and *US*.

For finite domains, to efficiently maintain the occurrence of join values within a partition we use a bit vector with size equal to the cardinality of the domain of the join attribute. In the motivating example as in Figure 7, $Dom(country)$= {Brazil, China, Germany, India, Indonesia, Mexico, USA}, and input partition I_1^R has a bit vector $B_1^R(0111001)$. For our bit vector method, to determine if the input partition pair [I_i^R,I_j^T] will generate even a single join result, a simple bit-wise AND operation between B_i^R and B_j^T is sufficient. If the resulting bit vector $B_i^R \wedge B_j^T$ is greater than 0, the region $\mathcal{R}_{i,j}$ is guaranteed to be populated. Otherwise, the output region $\mathcal{R}_{i,j}$ is guaranteed to be empty and so we move on to the next input partition pair. In our motivating example, $B_1^R(0111001) \wedge B_1^T(0101001)$=$(0101001) > 0$ and therefore we generate region $\mathcal{R}_{1,1}$. Conversely, $B_3^R(0000100) \wedge B_1^T(0101001)$ = (0000000) and we do not generate region $\mathcal{R}_{3,1}$.

Region- and Output Partition-level Elimination:. When an output region or partition is populated by at least one member, we proceed with the techniques described in Section 4-5 without any modification. This is because both output region- and partition-level reasoning are independent of domain values and only concerns itself exclusively with dominance properties of output regions or partitions that are known to contain some join pairs.

Object-level execution. We now only generate combined objects that satisfy the join condition. The strategy presented in Section 6 is used as is to generate the final skyline.

For infinite domains, we start by assuming that no region is guaranteed to be populated. Then, we iteratively trigger the region and partition-level eliminations when a priori empty output region or output partition becomes populated.

8. DISCUSSIONS

8.1 Handling Larger Data Sets

Our experimental evaluation reveal that a main memory based implementation of *SKIN* can safely handle SkyMapJoin queries over data sources of 500K each. We now briefly discuss an adaptation to *SKIN* when the available main memory is not sufficient.

Pre-processing Phase: The objects associated with the same input partition are clustered together in one or possibly multiple disk pages. The index loading time is similar to that of other indexing techniques such as *R-Trees*, *Bit Map-Index*, etc. For each source, we maintain the input-partition abstraction in main memory, i.e., the meta-data describing each of its input partitions such as the identifier, upper and lower bounds and number of tuples.

Region- and Partition- Level Elimination: To determine which region is dominated, the region-level elimination phase solely needs to access the meta-data for each input source. Similarly, the partition-level elimination phase can be performed without any disk access.

Object-Level Execution: For each input partition pair [I_i^R, I_j^T], during the join evaluation, we only fetch pages associated with the two current input partitions I_i^R and I_j^T and the output partitions that map to the region $\mathcal{R}_{i,j}$. Next, to efficiently handle skyline comparisons we exploit the concept described in Section 6 to only fetch pages that are associated with the comparable output partitions.

To summarize, our proposed SKIN approach can naturally handle scenarios when the raw data resides on disk.

8.2 Handling Higher-Dimensional Data Sets

In this work, we target multi-dimensional applications (such as those outlined in the introduction) where the number of skyline dimensions is in the order of 2 to 6 dimensions. The reasoning here is that as the number of attributes on which the skyline is applied to increases the cardinality of the output results also increases drastically. Human analyst cannot make effective decisions when faced with such high cardinality result sets. Thus, further processing such as linear ranking of results or clustering, would be needed to be employed [3]. Increasing usability of the results in an orthogonal problem to the one addressed in this work. In addition to the above concern, high-dimensional skyline evaluation (even for single sets) has exponential time complexity [2]. Therefore, to extend this effort to high-dimensional applications with say 100s of dimensions is clearly a challenging task. However, the focus of this work is low-dimensional applications, we leave high-dimensional applications for our future work.

8.3 Adaptive Spatial Partitioning

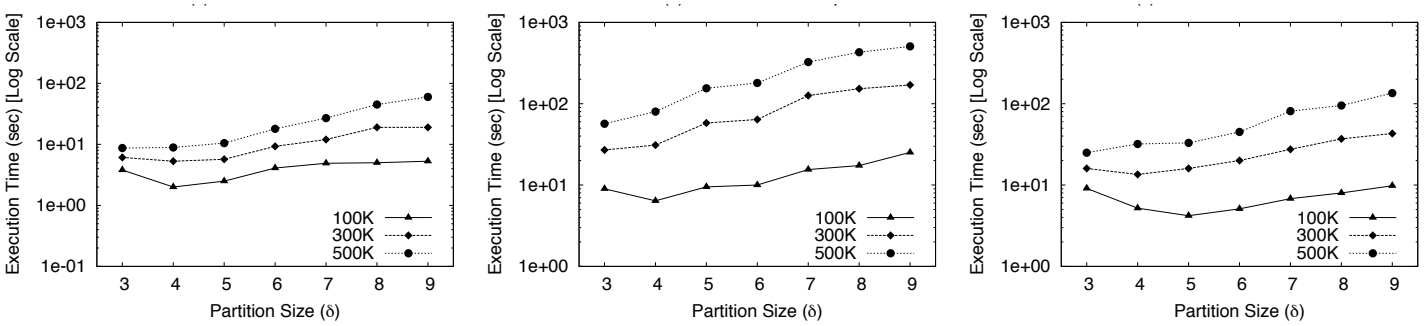
In this work, we employ a traditional grid strategy for partitioning the data space and forming higher-level abstractions as platform for dominance reasoning. And, as our experiments demonstrate, this simple strategy has indeed shown to be extremely effective. However, it would be interesting in the future to also explore alternate partitioning methods. For instance, we could explore variable-sized partitioning, considering aspects of the population of partitions. This could be achieved either with an adaptive re-partitioning method triggered by observed potential benefit for dominance-driven region purging. Alternatively, consider distribution-sensitive preprocessing to determine optimal partitioning based on criteria such as data density, relative closeness to the origin, etc. We leave those interesting challenges for the future.

9. EXPERIMENTAL EVALUATION

In this section, we verify the effectiveness and efficiency of our proposed approach to handle skyline queries over disparate sources.

9.1 Experimental Setup

Proposed Techniques. The principle of skyline *partial push-through* [2, 8] is complimentary to the core approach presented in Section 3–7 and is incorporated by pruning each individual data source by



(a) Correlated Distribution (b) Independent Distribution (c) Anti-correlated Distribution

Figure 8: Effects of Partition Size (δ) on the performance of SKIN ($d = 2$; σ = 0.1)

first applying Algorithm 4 on them separately.

Competitor Techniques. State-of-the-art techniques that handle skylines over joins are as follows: first, *JF-SL* with the improvement of incrementally maintaining the skyline of join results and using a hash-based join implementation [12]. Second, an optimized *JF-SL*$^+$ which uses the principle of skyline partial push-through to prune each individual data source. Third, *Skyline-Sort-Merge-Join* (*SSMJ*) proposed by [9]. SSMJ maintains for each data source two active lists of objects: (1) those objects that are in the set-level skyline generated by ignoring the join condition, and (2) the objects that are in group-level skyline for each join attribute value (as in Algorithm 4). Next, these lists are given for join evaluation, set-level lists first followed by group-level lists. The skyline is then computed over the join results to return the final query results. Fourth, [19] noted that for very low join selectivity of ≤ 0.000001, *SSMJ* is ineffective in pruning many objects both at the set- and group-level of each data source. To handle such scenarios, [19] proposed an *Iterative* (IT) technique. Fifth, *SAJ* [12] extended the popular Fagin technique [7] following the JF-SL paradigm.

Algorithm 4 Skyline Partial Push-Through

Input: Input Set R; Skyline dimensions $\{a_1, \ldots a_d\}$; Join attribute a_{d+1}
Output: Set of objects which are non-dominated by other objects with the same join attribute value a_{d+1}
1: Group objects in input set R into groups $\mathcal{G}_i$ by the join attribute a_{d+1}
2: **for** each group $\mathcal{G}_i$ **do**
3: LR_i = Generate local skyline on attributes $\{a_1, \ldots a_d\}$
4: **end for**
5: **return** $LR_o \cup LR_1 \cup \ldots LR_k$

Experimental Platform. All experiments are conducted on a Linux machine with AMD 2.6GHz Dual Core CPUs and 4GB memory. All algorithms are implemented in Java 1.5.0_16.

Evaluation Metrics. For each algorithm we measure: (1) the total execution time, (2) the total number of intermediate combined-objects generated, and (3) the total number of domination comparisons required to generate the final skyline. In addition, we measure the time taken by each phase of our approach.

Data Sets. We conducted our experiments using data sets that are the *de-facto* standard for stress testing skyline algorithms in the literature [2]. The data sets contain three extreme attribute correlations, namely *independent*, *correlated*, or *anti-correlated*. For each data set R (and T), we vary the cardinality N [10K–500K] and the # of skyline dimensions d. The attribute values are real numbers in the range [1–100]. The join selectivity σ is varied in the range $[10^{-4}$–$10^{-1}]$. The mapping function used is an addition operation between the attribute-values of the corresponding dimensions similar to those in our motivating queries. We set $|R| = |T| = N$.

9.2 Experimental Analysis of SKIN

Purpose. We study the robustness of our approach by varying: (1) partition sizes δ, (2) data distributions, (3) cardinality N , and (4) dimensions d. For a dimension d, data distribution and cardinality N, we measure the execution time for each partition size δ.

Partition Size (δ). The effectiveness of SKIN depends on the ratio between the object-level vs. the partition-level granularity. Figures 8.a–c show the execution time of SKIN for dimension d=2, and for the data sizes N= 100K, 300K and 500K. Smaller partition sizes will result in many sparsely populated input partitions, and therefore the overhead costs of region-level elimination will outweigh its benefits. Alternatively, as δ is increased the execution costs of region-level eliminations is reduced. Larger δ however may only marginally reduce the number of combined objects generated but will increase the number of combined objects to be compared against the output space. This insight is confirmed in Figures 8.a–c for $\delta \geq 6$ for all distributions. In Figures 8.a–c and for all N, we observe that $\delta = 4$ for all three distributions has the smallest execution costs. Similarly, for the dimension d=3 and then d=4 we observe similar trends for picking δ =12 or δ=17 respectively for all distributions. In summary, selection of a suitable δ helps in effectively reducing both the number of combined-objects generated as well as number of skyline comparisons, while still keeping the overhead costs of optimization small.

Execution Time Analysis of SKIN steps. In Figure 9 we present the CPU processing time incurred by the three pipelined steps in our proposed *SKIN* methodology. In Figure 9, the time required to generate the abstract-level output space and to evaluate the dominance criteria on them (*region-level elimination*) is fairly cheap in comparison to the object-level execution. For a given distribution, we confirm the intuition that query processing at the higher

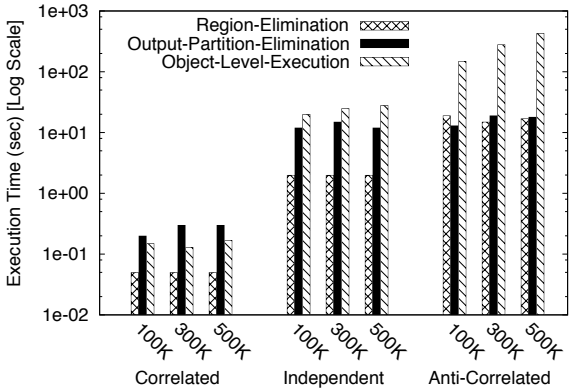


Figure 9: Execution time for the phases in SKIN; d=4, σ=0.01

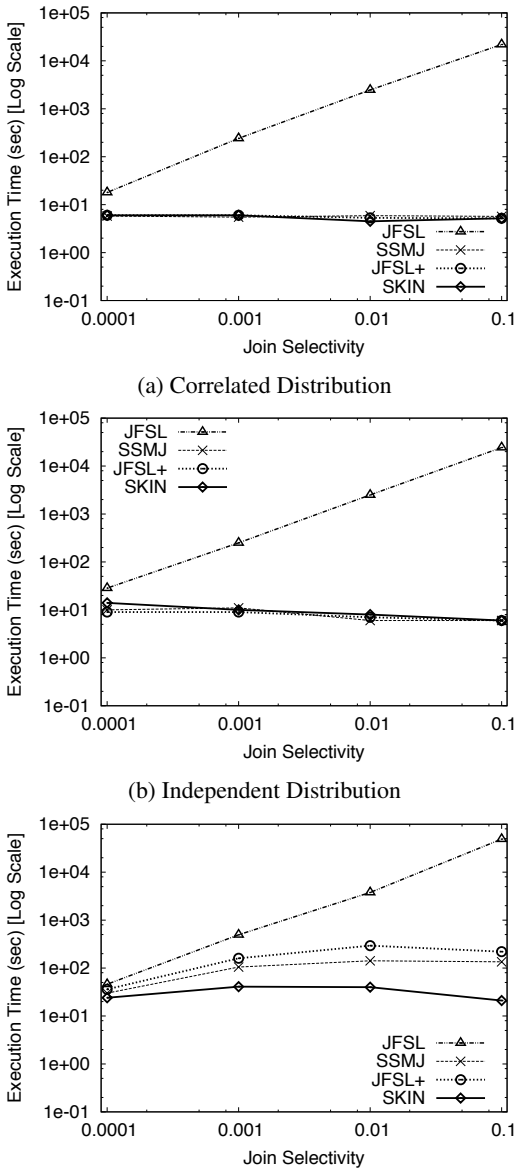
abstractions, namely the *region-level elimination* as well as *output partition-level-elimination* steps, are consistent across different cardinalities of the input data sets. In contrast, the time consumed for object-level processing increases with N specifically for anti-correlated and independent data sets.

Cardinality (N). In Figures 8a–c, as N increases from 100K to 500K the execution cost of SKIN increases gradually. In Figure 9, we present a closer investigation of the time required to perform the various phases of SKIN. Among the three optimization steps, both region- and partition- level execution time are relatively smaller than the actual object-level evaluation. Here, we observe) within each distribution the region-level and partition-level is relatively insensitive to N. As expected for anti-correlated and independent data sets the object-level execution rises in processing time proportional to N.

Distributions. Correlated distribution is specially geared for skyline operation, since a few objects can potentially dominate the remainder [2]. Efficient skyline algorithms tend to do the best for such correlated data sets [2, 12, 16]. Figures 8.b, 10.a and 11.a show clearly that our approach is efficient in handling correlated data for all cardinalities and dimensions. Anti-correlated data usually does not favor skyline algorithms since they have large skyline results [2]. It is interesting to observe in Figures 10.a and 11.a that our method however is robust and has significantly better performance that the state-of-the-art techniques in this most challenging case scenario. The detailed comparison of the proposed approach against state-of-the-art techniques, see Figure 12, confirms that for anti-correlated data our optimization phases are highly effective in reducing the number of skyline comparisons.

9.3 Comparisons with State-of-the-art

Purpose. We compare our approach against existing techniques on three factors: (1) execution time, (2) the number of intermediate join results (combined objects) generated, and (3) the number of skyline comparisons. [12] acknowledged that *SAJ* is beneficial only for correlated data sets while *JF-SL*-based techniques exhibit better performance for all distributions. [19] noted that *IT* has identical performance characteristics to *SSMJ* for all join selectivities $\sigma \geq 10^{-5}$. Therefore, we focus our detailed comparative study on *JF-SL*, *JF-SL* and *SSMJ*.



(a) Correlated Distribution

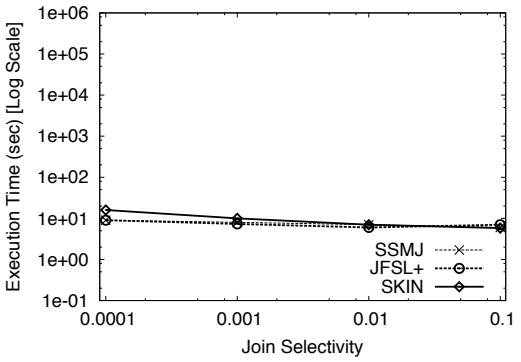
(b) Independent Distribution

(c) Anti-Correlated Distribution

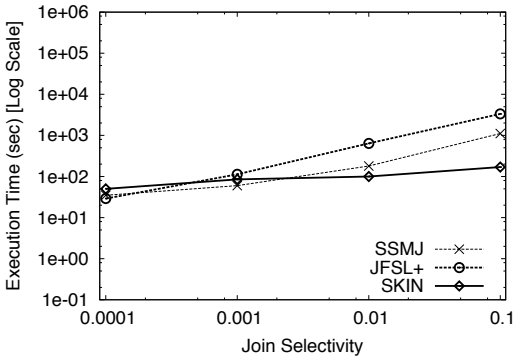
Figure 10: Performance comparisons with state-of-the-art JF-SL, JF-SL$^+$ [12], and SSMJ [9] for $d = 3$; N=500K

Execution Time. Figures 10 and 11 compare the execution times of the different techniques for d=3, and d=5 respectively. *JF-SL* generates all possible join results which for most of cases range in the order of several million combined-objects. Due to this drawback, *JF-SL* exhibits inferior performance as observed in Figures 10.a-c. For d=4 and anti-correlated data set, *JF-SL* ran out memory space and failed to return results. Therefore, for $d \geq 4$ we only compare our technique against *JF-SL*$^+$ and *SSMJ*.

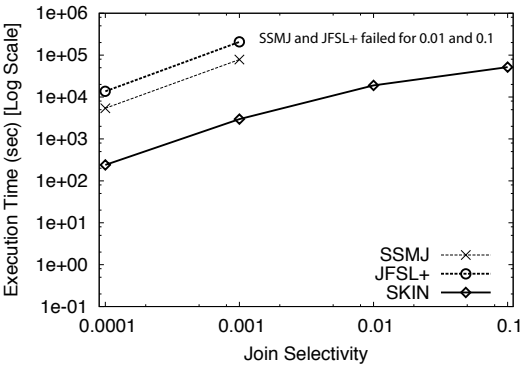
For **d=3** and the data distribution – correlated data and independent, *SKIN* has identical performance as *JF-SL*$^+$ and *SSMJ* as shown in Figures 10.a and 10.b. For the tougher anti-correlated data, *SKIN* is effective and outperform both *JF-SL*$^+$ and *SSMJ* on an average 1 order of magnitude for the join selectivity of $\sigma \geq 0.001$. It is important to note that even though *JF-SL*$^+$, *SSMJ*, and *SKIN* all receive the same set of pruned pre-filtered sources, and our proposed approach *SKIN* is able to perform better than others due to



(a) Correlated Distribution

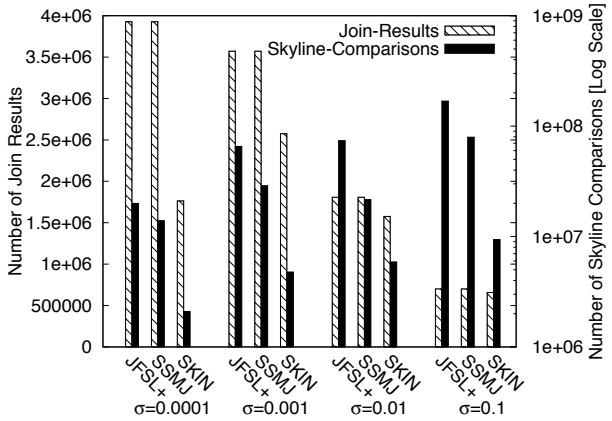


(b) Independent Distribution

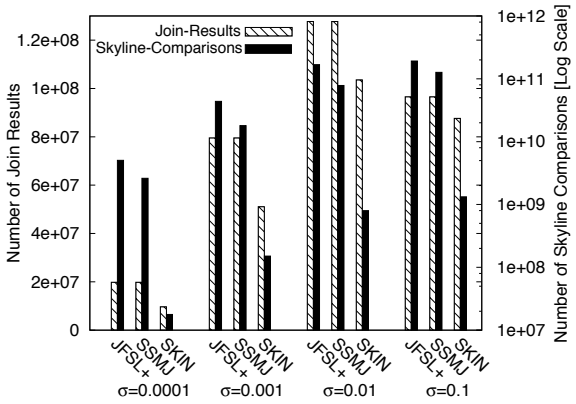


(c) Anti-Correlated Distribution

Figure 11: Closer Investigation with only JF-SL$^+$ [12], and SSMJ [9] for $d = 5$; N=500K



(a) Independent Distribution



(b) Anti-Correlated Distribution

Figure 12: Number of join results generated, and number of skyline comparisons for d=4 and N=500K

its underlying principle of query processing at different data abstractions. From Figure 10 we can therefore conclude that for d=3 and all distributions *SKIN* is shown to be robust. More specifically, for $\sigma \geq 0.001$, *SKIN* outperforms other techniques by an average of 1 order of magnitude. Henceforth, we focus our discussion on the tougher anti-correlated and independent data sets.

For **d=5** and independent data set in Figure 11.b we observe that for the join selectivity of $\sigma = 0.0001$ we observed that *SSMJ* is 15 seconds faster than *SKIN*. The cross over point is $\sigma > 0.001$ when *SKIN* outperforms *SSMJ* by slightly less than 1 order of magnitude and 2 folds better for $\sigma = 0.1$ and $\sigma = 0.001$ respectively. For **d=5** and anti-correlated data set, as shown in Figure 11 we observe the following: (1) for join selectivity $\sigma > 0.001$, both *JF-SL*$^+$ and *SSMJ* fail to return results even after several hours, (2) Since *SKIN* can exploit the knowledge of the output space it can further optimize the generation of combined objects and minimize the number of dominance comparisons, and (3) for $\sigma = 0.0001$ and $\sigma = 0.001$ *SKIN* outperforms both *SSMJ* and *JFSL*$^+$ by a factor larger than 1 order of magnitude. We therefore conclude that for $d = 5$ *SKIN* is robust and effective across all distributions in comparison to its competitors. For **d = 4** we observed similar performance benefits of our *SKIN* approach.

No. of join results (combined objects) generated. For correlated data sets a few tuples can dominate a large portion of the input space. Thus the *partial push through* principle when used as a pre-filtering phase is very effective. Therefore, correlated data sets are less interesting. In Figure 12 we compare the number of intermediate join results generated by the different algorithms on the $y - axis$. Here we observe that *SKIN* produces the least number of join results across all join selectivities and for both independent as well as anti-correlated data sets. This is because *SKIN* effectively exploits the *region- and partition-level* elimination techniques proposed in Sections 4 and 5 respectively. In 12.a for independent data sets *SKIN* has an average performance benefit of producing 45% fewer intermediate join results across various join selectivities when compared against both *SSMJ* and *JFSL*$^+$. More specifically, the minimum benefit is 7% lesser join results for $\sigma = 0.1$ and a maximum benefit of 122% for $\sigma = 0.0001$. For anti-correlated data sets, as in Figure 12.b, the average benefit is to produce 50%

lesser intermediate join results with a minimum of 10% fewer intermediate join results for $\sigma = 0.1$ and a maximum of 105% fewer intermediate join results for $\sigma = 0.0001$.

No. of skyline comparisons. In Figure 12, we compare the number of skyline (dominance) comparisons on the secondary *y-axis* needed by the compared algorithms. For anti-correlated data, *SKIN* achieves 1 and 2 orders of magnitude fewer skyline comparisons than *SSML* and *JFSL*$^+$ respectively, to generate the final skyline. In contrast, for independent data sets *SKIN* requires 1 order of magnitude fewer skyline comparisons than both *SSMJ* and *JFSL*$^+$.

9.4 Summary of Experimental Conclusions

1. The *SKIN* approach is robust across all distributions, cardinality and join factors.
2. The principle of skyline *partial push-thorough* is complimentary to our work.
3. For all three data distributions, cardinalities and join selectivities, *SKIN* outperforms both *JS-FL* and *SSMJ*. More specifically, for skyline non-friendly anti-correlated data *SKIN* outperforms by 1-2 orders of magnitude in execution times. In addition, for skyline-friendly correlated data sets *SKIN* has competitive performance with respect to both *SSMJ* and *JFSL*.
4. For d=5 existing techniques are unable to produce results when the data set is anti-correlated and selectivity $\sigma > 0.001$. In contrast, *SKIN* is shown to perform well for such data sets.
5. *SKIN* produces fewer numbers of intermediate join results than the state-of-the-art techniques. More precisely, on average 50% fewer intermediate join results.
6. *SKIN* is more effective in performing fewer number of dominance comparisons to generate the final skyline than *JS-FL*$^+$ or *SSMJ*. More specifically, requires 1-2 orders of magnitude fewer skyline comparisons to generate the final results.

10. RELATED WORK

10.1 Skyline Algorithms over Single Relation

Majority of research on skylines has focused on the efficient computation of a skyline over a single set [1, 2, 6, 11, 15] and can be broadly categorized as *non-index* and *index-based* solutions. *Block nested loop* (BNL) [2] is the straightforward non-index based approach that compares each new object against the skyline of objects considered so far. The *Sort Filter Skyline* (SFS) [6] improves on BNL by first sorting the input data by a monotonic function. Nearest Neighbor (NN) [11] and Branch & Bound Search (BBS) [15] are indexed based algorithms. *Sort Filter Skyline* (SFS) [6] improves on BNL by first sorting the input data.

10.2 Skyline over Disparate Sources

In the context of returning meaningful results by relaxing user queries, [12] presented various strategies that follow the join-first, skyline-later (JF-SL) paradigm (see Section 1). This approach does not consider mapping functions, and is attractive only for correlated data where the combined-object generation can be stopped early [2]. In our detailed technical report [18], we show for all data distributions even the naïve *JF-SL* approach achieves several orders of magnitude improvement against *SAJ*. This result confirms the findings presented in [12]. [9, 19] proposed techniques to handle skylines over join by primarily exploiting the principle of *skyline partial push-through*. This approach suffers from the following three drawbacks. First, *SSMJ* is only benefitial when the local level pruning decisions can successfully prune a large number of objects, like skyline friendly data sets such as correlated and independent data sets or very high selectivity [19]. In our experimental study we show that even in data sets where *SSMJ* has good performance our proposed *SKIN*$^+$ performs equally well. Second, since they do not have any knowledge of the mapped output space, similar to *JF-SL*$^+$, *SSMJ* is unable to exploit this knowledge to reduce the number of dominance comparisons. Third, the guarantee that objects in the set-level skyline of an individual table are guaranteed to be in the output no longer holds since they do not consider mapping functions which can affect dominance characteristics.

10.3 Other Related Techniques

Top-K or Ranked Queries retrieve the best K objects that minimize a user-defined *scoring function* to name a few [7, 13, 14]. That is, from a *totally ordered* set of objects such queries fetch the top K objects, where the ordering criterion is a *single* scoring function. In contrast, the skyline operator returns a set of non-dominated objects based on *multiple* criteria in a multi-dimensional space and from a *strict partially ordered* set of objects. Therefore, the objects returned by Top-K queries may not be part of the skyline [15].

Convex-Hull also known as *convex envelope*, for a given set X of multi-dimensional points is the minimal convex set containing X [17]. The convex hull is a different problem than skyline since in the geometric space some skyline points can be hidden behind a convex segment. In theory, the computation of convex hull has a higher time complexity than skyline evaluation [2].

11. CONCLUSION

The efficient evaluation of skyline over joins requires avoiding two primary costs, namely the cost of generating the intermediate join results and the cost of dominance comparisons to compute the final skyline of join results. State-of-the-art techniques handle this by primarily relying on making local pruning decisions at each source, and are therefore shown to be not robust for a wide variety of data. To address this shortcoming, we propose our *SKIN* approach that supports the robust and efficient evaluation of *SkyMapJoin* queries. We achieve this by taking advantage of optimization opportunities that are available by looking ahead into the mapped output space and exploiting this knowledge at the level of both individual sources and the complete query. We demonstrate the superiority of our approach over state-of-the-art methods by consistently outperforming them in many cases by several orders of magnitude, for a wide range of data sets, confirming the robustness of our methodology.

Acknowledgment

This work is supported by the National Science Foundation under Grant No. IIS-0633930 and CRI-0551584. We thank Dr. Donald Kossmann for the synthetic data generator, which is the de-facto benchmark data sets for skyline evaluation.

12. REFERENCES

[1] I. Bartolini, P. Ciaccia, and M. Patella. Salsa: computing the skyline without scanning the whole sky. In *CIKM*, pages 405–414, 2006.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[3] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[4] S. Chaudhuri, N. N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE*, page 64, 2006.

[5] R. Chen, L. Li, and Z. Weng. Zdock: An initial-stage protein docking algorithm. *Proteins*, 52(1), 2003.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–816, 2003.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[8] B. Hafenrichter and W. Kießling. Optimization of relational preference queries. In *ADC*, pages 175–184, 2005.

[9] W. Jin, M. Ester, Z. Hu, and J. Han. The multi-relational skyline operator. In *ICDE*, pages 1276–1280, 2007.

[10] W. Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

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

[12] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210, 2006.

[13] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD*, pages 131–142, 2005.

[14] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, pages 281–290, 2001.

[15] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[16] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

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

[18] V. Raghavan, S. Srivastava, and E. Rundensteiner. Skyline and mapping aware evaluation over disparate sources. Technical report, Dept. of Computer Science, Worcester Polytechnic Institute, 2009.

[19] D. Sun, S. Wu, J. Li, and A. K. H. Tung. Skyline-join in distributed databases. In *ICDE Workshops*, pages 176–181, 2008.