Look Before You Leap: An Adaptive Processing Strategy For Multi-Criteria Decision Support Queries

by

Shweta Srivastava

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

April 2011

APPROVED:

Professor Elke A. Rundensteiner, Thesis Advisor

Professor George T. Heineman, Reader

Professor Craig E. Wills, Head of Department

Abstract

In recent years, we have witnessed a massive acquisition of data and increasing need to support multi-criteria decision support (MCDS) queries efficiently. Pareto-optimal also known as skyline queries is a popular class of MCDS queries and has received a lot of attention resulting in a flurry of efficient skyline algorithms. The vast majority of such algorithms focus entirely on the input being a single data set. In this work, we provide an adaptive query evaluation technique — *AdaptiveSky* that is able to reason at different levels of abstraction thereby effectively minimizing the two primary costs, namely the cost of generating join results and the cost of dominance comparisons to compute the final skyline of the join results. Our approach hinges on two key principles. First, in the input space – we determine the abstraction levels dynamically at run time instead of assigning a static one at compile time that may or may not work for different data distributions. This is achieved by adaptively partitioning the input data as intermediate results are being generated thereby eliminating the need to access vast majority of the input tuples. Second, we incrementally build the output space, containing the final skyline, without generating a single join result. Our approach is able to reason about the final result space and selectively drill into regions in the output space that show promise in generating result tuples to avoid generation of results that do not contribute to the query result. In this effort, we propose two alternate strategies for reasoning, namely the *Euclidean Distance* method and the cost-benefit driven *Dominance Potential* method for reasoning. Our experimental evaluation demonstrates that AdaptiveSky shows superior performance over state-of-the-art techniques over benchmark data sets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time applications, ranging from Internet aggregators, business intelligence to data warehouse systems, need for supporting complex multi-criteria decision support (MCDS)queries. The intuitive nature of specifying a set of user preferences has made Pareto-optimal (or *skyline*) queries a popular class of MCDS queries [2]. Unlike traditional queries, whose goal is to return only exact matches, skyline queries return a set of non-dominated results meaning each result is better than the others in at least one criterion.

Recently, we have witnessed a flurry of techniques [1, 2, 11] that evaluate skyline queries over a *single* homogeneous data set. The common assumption of viewing skyline as an operator on top of the traditional SPJ *select project join* queries applied on homogeneous data sets is rather limiting since a vast majority of MCDS applications in practice do not operate on just a single source. Instead, they require to (1) access data from disparate sources via joins, and (2) combine several attributes across these sources through possibly complex user-defined functions to characterize the final composite product (as substantiated below).

Supply-Chain Management. A manufacturer in a supply chain aims to maximize profit, market share, etc., and minimize overhead, delays, etc. This is achieved by structuring an optimal production and distribution plan through the evaluation of various alternatives.

```
Q1: SELECT R.id, T.id,  (R.uPrice + T.uShipCost) as tCost,
(2 * R.manTime + T.shipTime) as delay
FROM Suppliers R, Transporters T WHERE R.country=T.country AND
P1′ in R.suppliedParts AND R.manCap >=100000
PREFERRING LOWEST(tCost) AND LOWEST(delay)
```

$Q1$ identifies the suppliers that can produce "100K" units of the part "*P1*" and couples them with transporters that deliver it. The preference is to minimize both total cost ($tCost$) as well as delays ($delay$). In this work, we target such queries which combine the skyline and mapping functions over the join, here known as **SkyMapJoin** (SMJ) queries.

Internet Aggregators. The rapid increase in the number of on-line 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 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 cost,
(2 * R.distance + T.distance) as distance
FROM RomeHotels R, ParisHotels T
PREFERRING LOWEST(tCost) AND LOWEST(tDistance)
```

In addition to there are numerous other applications in the field of *drug discovery*, and *query relaxation* where queries with a similar skeleton structure is encountered. In this

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

work we target queries involving both skyline and mapping operations over disparate data sources, known as *SkyMapJoin* queries introduced by [12, 13].

State-of-the-art Techniques. Existing techniques view the skyline operation as an add-on after the join and thus follow a *join-first skyline-later* (JF-SL) paradigm [2]. JFSL divides query evaluation into two steps, namely first produce all possible join results, and second perform skyline evaluation over the entire join results. This approach misses several optimization opportunities. JFSL spends precious resources in producing join results that may not belong to the final skyline result set. Further more it also bears the cost of comparing all the unwanted join objects. In [13] we introduced a method called *SKIN* that leverages this opportunity by considering both skyline dominance and mapping function transformation as part of the join processing logic. SKIN applies a partition-based solution has been shown to outperform existing methods in some case as much as several orders of magnitude. SKIN however suffers from the deficiency that it pre-determines, from empirical results, the level of abstraction. While this is critical for the success of SKIN, it is extremely difficult to decide a priori. In this we show that a bad abstraction level can hinder the performance of an otherwise efficient framework.

Our Proposed Solution. In this work, we propose **AdaptiveSky** that successfully leverages the strengths of SKIN while at the same time alleviates its weaknesses by being dynamic in its decision making. *AdaptiveSky* works on two data space namely, the *input space* – containing the input tuples and an *output space* containing the intermediate join results – some of which belong to the final skyline result. *AdaptiveSky* rests on key principle of *"look before you leap"* and our intuition can be summarized as follows. First, do not jump into generating numerous intermediate results. Instead, at run-time dynamically determine level of abstraction at which the input space needs to be investigated. Second, build the output space from a higher-level abstraction of regions to lower-level abstractions of mapped join results. This can be done heterogeneously across both spaces instead of uniform level of abstraction over the input or output space. The main contributions of our approach can be summarized as follows: (1) We propose *AdaptiveSky* that provides an adaptive execution strategy for evaluating *SkyMapJoin* queries. (2) We present two alternate strategies, a naiive *euclidean distance* method and the other more complex cost-vs-benefit driven *dominance potential* method. These strategies aid AdaptiveSky in further exploring the input space based on the feedback from the abstract output space. (3) Our experimental evaluation demonstrates that AdaptiveSky shows superior performance over state-of-the-art techniques.

Chapter 2

Background: The SKIN Approach

In this chapter, we briefly discuss the fundamental aspects of the SKIN approach proposed in our prior work [13]. The key idea is to have two fixed layers of abstraction, namely *macro-level processing* and **micro-level processing**. Table 2.1 summarizes the notation used here.

Table 2.1: Notations Used In This Work

Notation	**Meaning**
I_i^R	Input grid in R
$\mathcal{I}^R$	Set of all input grids in R
$r_f t_g$	Join result, $r_f \in R$; $t_g \in T$
$\mathcal{R}_{i,j}$	Region of output space that map the join results from the input partitions $[I_i^R, I_j^T]$
$\mathfrak{R}$	Set of all regions in the output space
LOWER(X)	*Lower-bound* point of a region or partition
UPPER(X)	*Upper-bound* point of a region or partition

Macro Level Processing. The aim of this step is to perform query execution at a higher granularity and generate the abstract output space. For this, SKIN assumes the input is uniformly partitioned into equi-sized grids also known as partitions. For a pair of input partitions, one from each table $I_a^R \in R$ and $I_b^T \in T$, we determine: (1) if tuples in these partitions produce at least one join result, and (2) once generated the *region* of the output space into which the generated join results will fall (denoted as $\mathcal{R}_{a,b}$). To illustrate, assume that the domain values of the join attributes are finite and known. In such a scenario, for each input partition we maintain a list of domain values of the join attribute(s) for the tuples mapped into that particular partition. Therefore, if two partitions share at least one join domain value we can guarantee that their join will result in at least one join result. The full treatment of joins is described in [13]. output regions which are guaranteed to be populated for further processing.

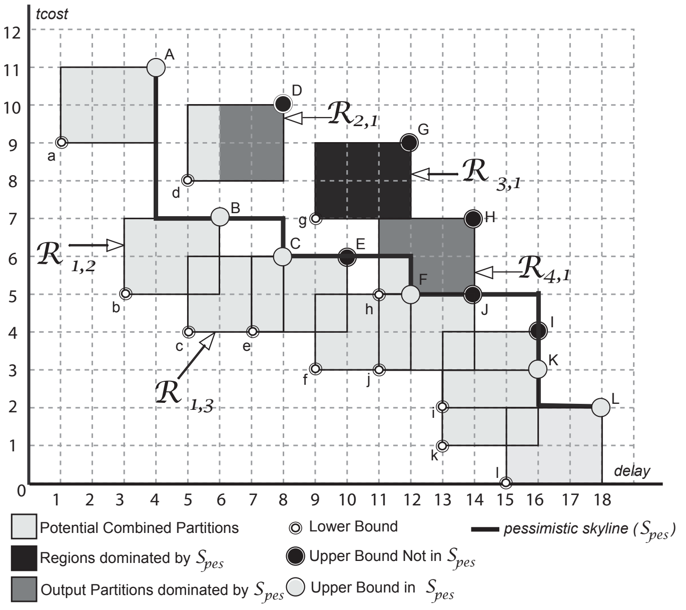


Figure 2.1: Macro-Level Processing: Avoid Join and/or Skyline Costs

Example: Tuples in the input partition (a.k.a. grid) from Supplier (R), I_1^R with bounds $[(0,4)(1,5)]$, when joined with tuples in input partition $I_2^T[(0,4)(1,5)]$ from Transporter (T), will result in join results that will fall in the region bounded by the lower-bound point $b(3,5)$ and the upper-bound point $B(6,7)$ in Figure 2.1. This output region is denoted as $\mathcal{R}_{1,2}$.

In our motivating query $Q1$, the preference is to minimize all skyline-dimensions. In a pessimistic scenario for each output region $\mathcal{R}_{i,j}$, all the intermediate join and then mapped results would lie on the upper-bound point of $\mathcal{R}_{i,j}$. $SKIN$ introduces the notion of the **pessimistic output skyline**, denoted as $\mathcal{S}_{pes}$ to identify the dominated output regions.

Any region in $\mathcal{R}_{i,j} \in \mathfrak{R}$ if $\exists s \in S_{pes}$ such that $s \succ_P$ LOWER($\mathcal{R}_{i,j}$) then no intermediate result $r_f t_g \in \mathcal{R}_{i,j}$ can be contained in the output skyline. Thus, the pessimistic skyline limits the comparisons of output regions generated to only those regions that belong to S_{pes}.

Micro Level Processing. Having eliminated all possible higher abstractions, this step executes tuple level joins and performs tuple level dominance comparisons to give the resulting skyline. While SKIN has introduced its own method, one can practically use any state-of-the-art approach to compute skyline over joins.

Chapter 3

Motivation For An Adaptive Framework

The fundamental approach in SKIN is to solve the skyline problem at a higher level of abstraction before solving it at the expensive level of individual tuples. In this section, we take look closer at the pros and cons of applying the reasoning at a static abstraction level. Based on this analysis we then propose our strategy to successfully address SKIN's drawbacks.

Cost of Macro Level Processing: This phase incurs the cost of comparing every output region that is generated with all regions that lie in the pessimistic skyline across all skyline dimensions. This is found to be $O(N_{total} \cdot |S_{pes}| \cdot d)$, where N_{total} is the total number of output regions generated, $|S_{pes}|$ is the number of regions in the pessimistic skyline and d is the number of attributes in the skyline result as stated by the users query. For the sake of simplicity we consider that d attributes from each relation combine to form d skyline attributes as in Q1. If k_i is the number of partitions that each of the dimensions in the input is divided into, then the total number of partitions (or grids) in each relation is k_i^d. Therefore, $N_{total} = \sigma k_i^{2d}$ where σ is the join factor. Next, the cost of computing the pessimistic skyline is $O(k_i^{4d} \cdot d)$. Thus, the total cost of performing *macro level processing* is the cost of creating the output regions and maintaining the pessimistic skyline maintaining $|S_{pes}|$ which is given as follows:

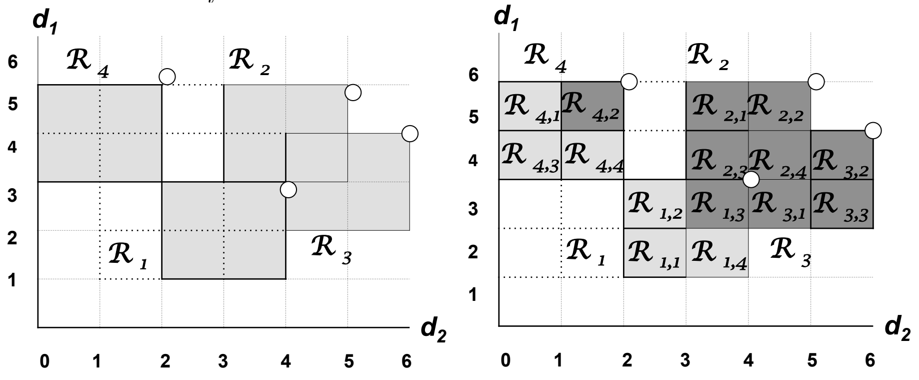
$$O\bigg(\sigma k_i^{2d} \cdot \sigma k_i^{2d} \cdot d\bigg) + O\bigg(k_i^{4d} \cdot d\bigg) = O\bigg(k_i^{4d} \cdot d\bigg) \tag{3.1}$$

Cost of Micro Level Processing: This phase incurs the cost of processing an output region at the level of individual tuples. This includes creating combined objects from the two input partitions I_a^R and I_b^T that form output region $R_{a,b}$ and subsequently performing skyline comparisons on these objects to produce the final skyline result set. The cost incurred during *micro-level-processing* for processing a single output region $\mathcal{R}_{a,b}$ is as follows:

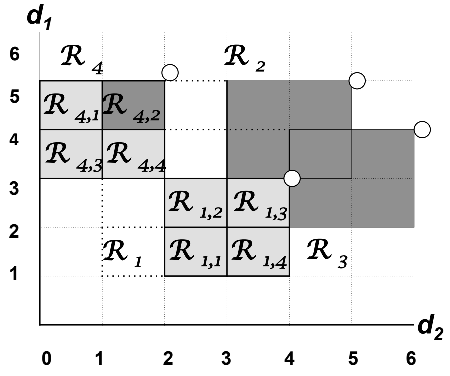
$$\begin{aligned} Cost(\mathcal{R}_{a,b}) &= C_{join}(\mathcal{R}_{a,b}) + C_{map}(\mathcal{R}_{a,b}) + C_{sky}(\mathcal{R}_{a,b}) \\ &= O(\sigma \cdot n^2) + O(\sigma \cdot n^2 \cdot d) + O(\sigma \cdot n^4 \cdot d) = O(\sigma \cdot n^4 \cdot d) \end{aligned} \tag{3.2}$$

where $n = |I_a^R| = |I_b^T|$ in a uniform distribution. Therefore the cost of materializing all N_{remain} output regions is $N_{remain} \cdot Cost(\mathcal{R}_{a,b})$. Here N_{remain} is the number of output regions that remain to be materialized after pruning all dominated regions during *macro level processing*.

Next, we shed light on the dependencies on some of the important factors that affect adversely the execution cost of SKIN. First, it is evident that the total number of regions created, N_{total}, depends on k_i which in turn is determined by the partition size δ that the input is uniformly divided into. Second, $|S_{pes}|$ mainly depends on the data distribution. Third, the cardinality n of each input partition. As described above, *macro level processing* is quadratic in the number of partitions k_i while *micro level processing* is quadratic in the number of tuples in each partition n. Therefore to reduce the total cost the goal should be to minimize both k_i and n. However, these two values are inversely proportionate to one another i.e. in an independent distribution, fewer number of tuples in a partition means that each partition covers a smaller area and therefore number of partitions are higher than a scenario where the number of tuples are high. From this analysis it is evident that if we try to decrease costs of *micro level processing* by reducing n the cost of performing *macro level processing* increases and vice versa. Since a reduction of both cannot be achieved at the same time, one of the goals in this work is to strike the right balance between n and k_i.



(a) Output space with pre-determined δ (b) Output space with smaller δ



(c) Output space with heterogeneous δ

Figure 3.1: Effect of varying δs on the output space

We will now show with the help of case examples how selecting an appropriate δ can affect the cost of skyline computation.

Case I - Larger δ in SKIN (Static Approach) Let Figure 3.1.a denote the output space after *macro-level processing* where $N_{total} = N_{remain}$ in other words the regions are fairly large and no region can be eliminated by *macro-level processing*. For this example, SKIN will have to bear an additional cost to perform *macro level processing* without seeing any of its benefits. In addition, since the regions are at a higher granularity the cost of *micro level processing* is going to be high as the number of tuples in its partitions n will be high (assuming independent distribution). The question is to find out if it is possible to change the granularity of the partitions reduce the overall cost.

Case II - Smaller δ SKIN (Static Approach) Figure 3.1.b shows us the same output space with smaller regions. This implies that the δ used to partition datasets R and T is smaller as compared to that used in Case I. In this scenario, several output regions are completely dominated and therefore can be safely eliminated. Although more number of regions go into *micro level processing* as compared to Case I, their respective cardinality is smaller which implies that the cost of processing each one of them have been drastically reduced. However, this reduction of cost has not come without shedding extra resources. *Macro level processing* has to do more work to generate increased number of regions and perform more number of skyline comparisons to maintain the pessimistic skyline $\mathcal{S}_{pes}$ as compared to Case I. It is also evident that attaining the right granularity of output regions is an important factor in achieving effective elimination and reduced costs. In other words determining the most appropriate level of abstraction a-priori is an ideal scenario. However selection of a correct δ for partitioning the input side without any knowledge of the output remains a challenging problem and therefore SKIN assumes that a δ is given a priori.

Case III - Heterogeneous δ AdaptiveSky Approach In the previous example the additional cost of maintaining $\mathcal{S}_{pes}$ may not be worth bearing if it does not translate into more benefit. Furthermore for certain data distribution, lower granularity of input partitions does not necessarily increase the pruning capacity of all output regions in the same proportion. In other words for some areas in the output space making smaller regions may not reap any benefits of elimination. In Figure 3.1.c that represents Case III the first visual difference that comes is that unlike Cases I and II, where all regions were of the same size, here the regions are of variable size. Output regions of heterogeneous sizes are a result of heterogeneously partitioned input space. In Case III, while the area eliminated is the same as Case II, the cost of *macro level processing* is lower since the number of total output regions created and compared for elimination are fewer. Since the cost of *macro level processing* is affected quadratically by the number of input partitions, this is a considerable cost savings.

The two key observations of the above analysis are as follows. First, finer granularity output regions increase elimination potential thereby reducing the subsequent *micro level processing* costs. However the associated overhead can be cost prohibitive. Thus it is important to find the right granularity of regions. That is, small enough to be able to eliminate other regions and large enough to keep its cost low. Second, since the static approach has no feedback mechanism it keeps making incorrect choices in partitioning. A feedback driven approach is crucial to in deciding which areas in the output and input spaces to drill into thereby limiting the cost incurred while increasing pruning capacity. We complement our framework with two heuristic-based reasoning techniques.

Chapter 4

Our Proposed AdaptiveSky Framework

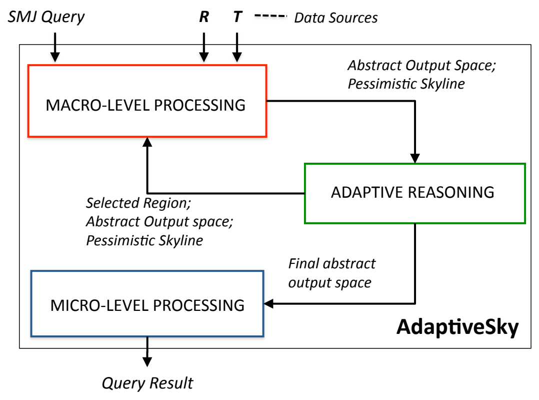
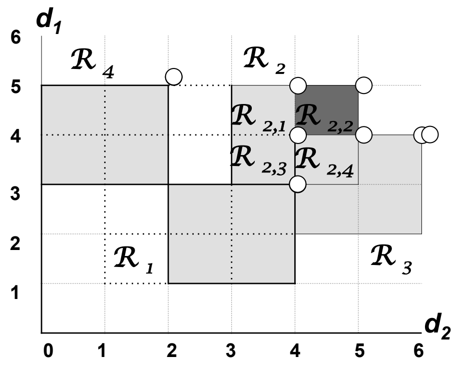

SMJ Query
R
T
Data Sources
MACRO-LEVEL PROCESSING
Abstract Output Space; Pessimistic Skyline
ADAPTIVE REASONING
Selected Region; Abstract Output space; Pessimistic Skyline
Final abstract output space
MICRO-LEVEL PROCESSING
AdaptiveSky
Query Result

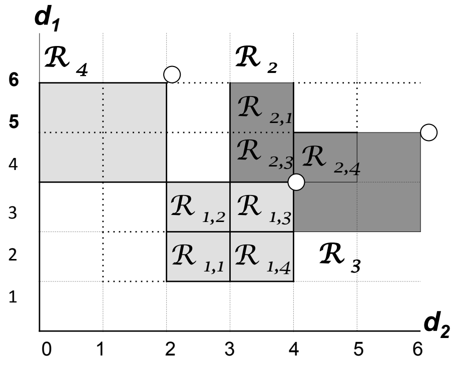
Figure 4.1: Adaptive Evaluation Framework

In this section, we present a brief overview of our *adaptive evaluation framework* called **AdaptiveSky** as illustrated in Figure 4.1. Our proposed framework builds on top of SKIN making it δ insensitive by empowering its *macro-level processing* to arrive at the right granularity or in other words heterogeneous δ depending on the result distribution in the output space. The *macro-level processing* takes as input the datasets (R and T), the preference query and a *coarse* δ (for instance $\delta = 50$). Next, the *adaptive reasoning* module looks ahead into the coarse grained output space to iteratively pick the next output region to zoom in. The selected region when made finer will result in the maximum pruning among all its peers. Armed with this knowledge, the *macro-level processing* creates smaller output regions by repartitioning its associated input partitions and updating the output space. This iterative process continues till the abstract processing cannot yield more pruning and the overhead costs outweighs its benefits from elimination. At this point, the output space is shipped for *micro level processing*. In a nutshell, we adapt selected areas of the output space over several iterations to obtain maximum possible pruning in a course abstraction of the output space to minimize the cost of *micro level processing*.

In the following discussion, we introduce four components added to the framework for adaptive reasoning that attempt to address two key questions that our approach poses: (1) *which* output region to zoom into. and (2) *when* to stop abstract-level processing and initiate *micro-level processing*.



(a) Region R_2 selected for repartitioning



(b) Region R_1 selected for repartitioning

Figure 4.2: Effects of Region Selection

Region Selector. In each iteration, this component selects the next most-beneficial region to adapt. Figure 4.2.a motivates the importance of selecting regions in an appropriate order consider the following example. If we employed a random selection process and pick region R_2, after repartitioning we added to the space $R_{2,1}$, $R_{2,2}$, $R_{2,3}$ and $R_{2,4}$. Since $R_{2,2}$ is the only region completely dominated by another region ($R_{2,3}$), it is eliminated. In contrast, let us assume we choose R_1 instead, as it is evident in Figure 4.2.b that on repartitioning R_1 more of the output space can be pruned. In next section, we present the two metrices that are used in selecting the next region for further processing. They are the *euclidean distance* metric and the cost-vs-benefit driven *dominance potential* metric .

Threshold Analyzer addresses the second problem of keeping regions small enough to facilitate more elimination but large enough to keep the cost of repartitioning low. The *threshold analyzer* uses the cardinality of the participating input partitions of a region as a good metric to stop repeated macro-level processing. The reasoning is that a small but dense output region has more potential for pruning therefore worthwhile for further investigation. Once all output regions reach the threshold limit the *micro-level processing* takes over.

Output Region Generator is a part of *macro-level processing* that updates output space after every feedback provided by the abstract reasoning. It first combines input partitions based on their join attributes by applying *mapping functions*. Next, it maintains the pessimistic skyline and eliminates regions based on dominance comparisons.

Adaptor component does the actual work of creating smaller output regions. It repartitions the input partitions of the selected region and sends the newly created input partitions to the output region generator. This process continues in an iterative manner until all regions have not reached the threshold size. The reason why we adapt the input space is because if we simply divide output regions without considering the input space, there is no way of knowing whether a dominating region will not be empty when we start *micro-level processing*. In case a dominating region is empty, it might end up eliminating a region that could have potential skyline candidates resulting in an incorrect result set.

Chapter 5

Technical Details

We will now begin discussing our framework in greater technical depth. In this section we describe the control and data flow among the components introduced in the previous section. In the following three sections we will discuss each component individually in technical detail.

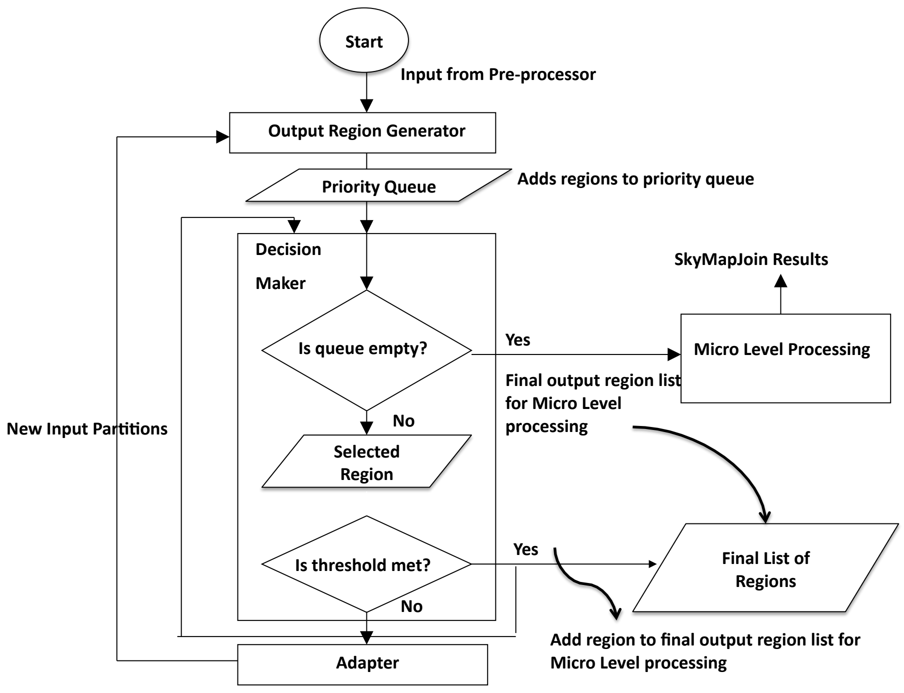


Figure 5.1: Adaptive Evaluation Framework

Figure 5.1 illustrates the control and data flow in our proposed framework. As part of *macro-level processing* the *output region generator* creates an abstract output space starting with the coarse δ. These generated regions are stored in a central priority queue that we refer to as P based on the metric chosen by the *region selector*. The value for a given region indicates pruning benefits in the output space than another region. The *region selector* sends the output region at the top of the queue to the *threshold analyzer*. The job of the analyzer is to verify on the basis of the metric chosen for itself whether or not the region has crossed the threshold. If the region is within the threshold, it is sent to the Adaptor to repartition its input cells and passes it on to the *output region generator*. This process is repeated till no regions need to investigated. In the scenario where the selected region has crossed the threshold, it is removed from the queue. This indicates that the overhead cost of adapting the region is more than its benefits. This region is now transferred to a list called R_{final} that is eventually passed on to micro-level processing at the end of all iterations. Now that the region has been removed other regions will get an opportunity to be investigated. This process continues till the queue becomes empty. This indicates that all regions have crossed the threshold and will no longer provide more benefits from pruning than the overhead cost of repartitioning. At this point R_{final} is passed on to micro-level processing for computing the final skyline result set. We will now describe the technical details of each component of our framework.

Chapter 6

Decision Making Components

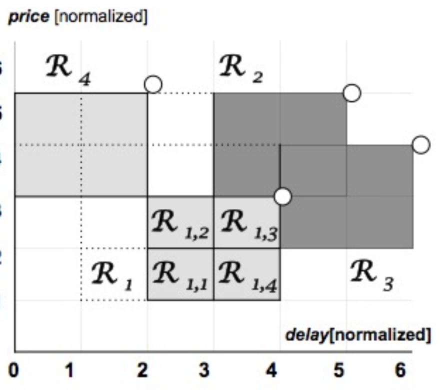
In the following discussion, we describe the two decision making components of our framework in technical depth. These components are the *region selector* and *threshold analyzer*. While the *region selector* decides *which* output region to adapt, the job of the *threshold analyzer* is to decide *when* to stop abstract-level processing and initiate *micro-level processing*.

Region Selector : Before diving into the details of how the Region Selector works let us understand how it impacts the cost of the entire adaptive process. In order to repartition a region, its input partitions are repartitioned using a smaller δ. The cost of this repartitioning can be given as $O((n_a^R \cdot d) + (n_b^T \cdot d))$ where n_a^R and n_b^T are the number of tuples in the input partitions I_a^R and I_b^T respectively (Refer Table 1 for notations). Thereafter the cost of combining input partions to create new output regions is $O(\sigma k_{new}^{2d} \cdot d) = O(N_{new} \cdot d)$ where N_{new} is the number of new output regions created after repartitioning, k_{new} is the number of new partitions that each dimension of existing partition is divided into using the new partition size δ_{new}. Having created the regions they now need to be compared for elimination. The cost of comparing N_{new} with the pessimistic skyline, S_{pes}, for elimination can be modeled as $O(N_{new} \cdot |S_{pes}| \cdot d)$. Now consider if *all* output regions of a given iteration, lets call it $N_{fPrevious}$ were to be re-partitioned in the following one, the total cost of generating an entirely new output space with finer granularity would be $O(N_{fPrevious} \cdot ((|I_a^R| \cdot d) + (|I_b^T| \cdot d)) + (N_{new} \cdot d) + (N_{new} \cdot |S_{pes}| \cdot d) +))$.

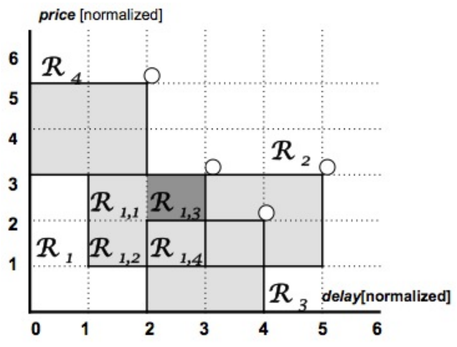
By the above analysis it is evident that fewer the number of regions repartitioned, lesser is the overhead cost. The *region selector* selects the output regions that would give maximum benefits of elimination. In doing so, it uses certain characteristics or metrics associated with the region that provides knowledge of its pruning capacity. This technique is a cost saver for the following two reasons. One, by repartitioning only selected regions in iteration $i{+}1$, all resources spent on generating the output space in the previous iteration i is not lost. Two, in any given iteration no resources need be spent on regions that will result in zero or limited number of regions being eliminated.

We shall now discuss how the *region selector* goes about selecting the right regions and provide us cost savings. At the end of the *one pass scan* all regions from R_{temp} and S_{pes} are added to the priority queue P that is part of the *region selector* component. These are the regions that have remained un-dominated and will now be considered for repartitioning. In every iteration one region will be selected to be repartitioned. In order to make this selection we rank all regions based on certain parameters or metrics. In this work we use alternate metrices, namely *eudlidean distance* and *dominance potential*. In the remainder of this section we shall discuss the intent behind each of these metrices, understand their computation and maintenance within the framework and anaylze the costs incurred in doing so.

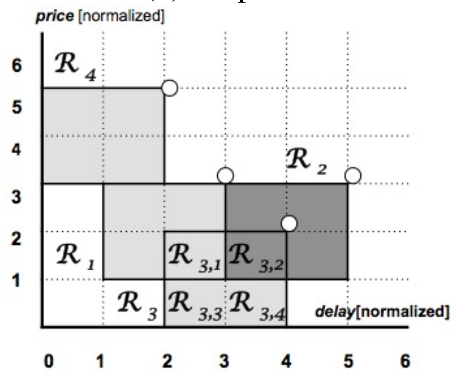
Euclidean Distance This metric is based on the fundamental intuition behind skylines: a point closer to the origin has greater probablity of dominating another point farther away from the origin than vice-versa. In Figure 6.1.a euclidean distance is applied on the output space and region R_1 is selected for repartitioning. The euclidean distance of region R_1 is computed as the euclidean distance of between the lower bound of region



(a) Euclidean Distance heuristic



(b) Output scenario where Euclidean distance is not effective



(c) Dominance Potential Heuristic

Figure 6.1: Heuristics used by Region Selector

R_1 or $LOWER(R_1)$ (Refer table 1 for notations) and the origin. The cost of computing the distance is constant for a given input of d dimensions and mapping functions. As the distance of a region from the origin is never going to change their is no overhead cost of maintaining this metric. The only cost incurred is the cost of adding these regions in every iteration which is given by $O\Bigg(log(|N|)\Bigg)$ where N is the number of regions to be added to the queue in that iteration.

Dominance Potential As nearness to origin is an intuitive metric and does not take into account any cost based metric, it may not always be correct. Figure 6.1.b shows a scenario in which Euclidean distance would have selected region R_1 for repartitioning. However, 6.1.c shows that choosing R_3 over R_1 will prune more area. Therefore we have come up with the *dominance potential* driven metric that takes into account the number of regions a given region can potentially dominate. Although we have introduced two metrices in this work, one can plugin any metric that can help order the regions and reduce costs. We will now look into the details of this metric to understand how it is computed and what are the overhead costs associated with their maintenance. *Dominance potential* is defined as the number of regions dependent on it. Higher the number of dependents higher is the region's dominance potential. A region R_1 is said to be *dependent* on region R_2 if LOWER($\mathcal{R}_2$) $\succ$ LOWER($\mathcal{R}_1$) and UPPER($\mathcal{R}_2$) $\succ$ UPPER($\mathcal{R}_1$). Here region R_2 becomes a dependee of R_1. We will now discuss the computation of this metric and its maintenance cost. When the output space is computed for the first time each region is compared with every other region to create a list of dependents and dependees for every region. Based on count of the dependents list the regions are stored in P to determine their priority for selection. The cost incurred in the first iteration is $O(N_f^2 \cdot d)$ where N_f is the number of regions after all possible elimination in the first iteration. Thereafter, in subsequent iterations, the *one pass scan* of the *output region generator* is replaced by a *Dependent Region Elimination Scan*. Like the one pass scan, this too is performed after the incremental maintenance of S_{pes}. However, in this case, instead of comparing all output regions with S_{pes} we compare the dependents of the region with the newly created regions. Whenever a dependent region is eliminated, we create a ripple effect such that the count of all dependents and dependees of this eliminated region is reduced by one. The cost of performing this maintenance operation for a selected region is $N_{dep}(O(N_{dependent} + N_{dependee}))$ where N_{dep} is the number of dependents of the region and $N_{dependent}$ and $N_{dependee}$ are the number of dependents and dependees of the eliminated region. In the process of comparing new regions to their parent region's dependents their own dependent list is generated. The cost of computing dependents of these newly created regions is $O(N_{dep} \cdot N_{new} \cdot d)$ where N_{new} is the number of new regions that remain un-dominated by S_{pes}. The benefit obtained from this exercise is $O(N_{dep} \cdot Cost(\mathcal{R}_{a,b}))$ where $Cost(\mathcal{R}_{a,b})$ is the cost of performing micro-level processing on a region.

Threshold Analyzer. This is the second decision making component of our framework. Once a region is selected, the *threshold analyzer* component verifies it against the threshold value. In this work, we define *threshold* in terms of the cardinality input partitions of a region. If the cardinality of the input partitions is smaller than the *threshold* the output region is removed from the priority queue. It is then passed on to the *adaptor* for repartitioning its input partitions. There is practically no cost over head in maintaining this metric value.

Chapter 7

Output Region Generator

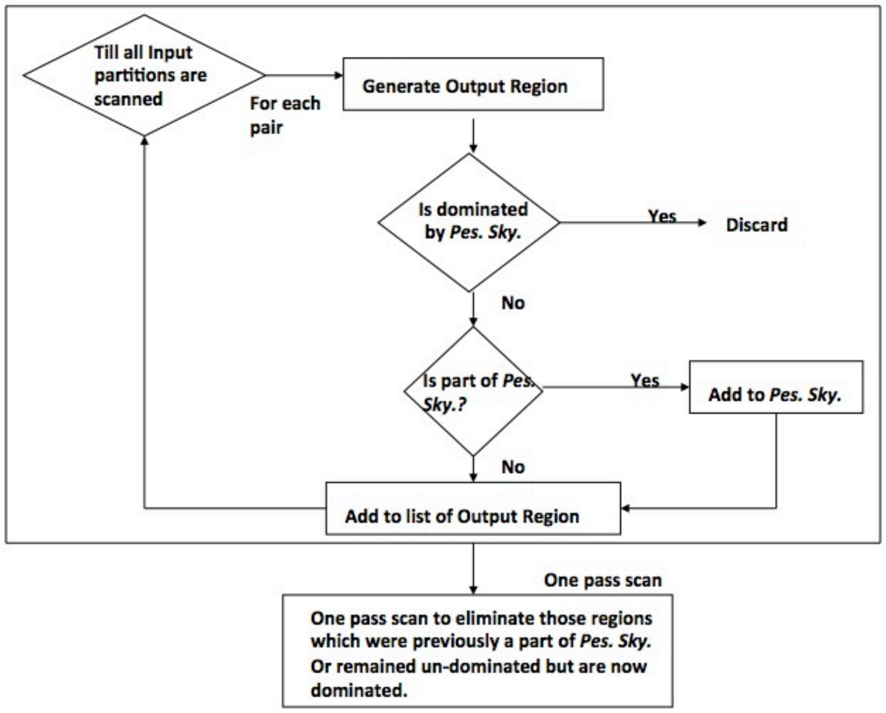

Till all Input partitions are scanned
For each pair
Generate Output Region
Is dominated by Pes. Sky.
Yes
Discard
No
Is part of Pes. Sky.?
Yes
Add to Pes. Sky.
No
Add to list of Output Region
One pass scan
One pass scan to eliminate those regions which were previously a part of Pes. Sky. Or remained un-dominated but are now dominated.

Figure 7.1: Output Region Generator

Output Region Generator accomplishes the operations of eliminating dominated output regions with the help of the pessimistic skyline, S_{pes}. It incrementally maintains the pessimistic skyline and at the same time eliminates regions that are dominated by S_{pes}.

Figure 7.1 illustrates the control flow in the Output Region Generator. Any region that is not part of S_{pes} but remains un-dominated is stored in a temporary output region list called R_temp. The first pass generates output regions and performs incremental S_{pes} maintenance. By incremental maintenance we mean the following. Each newly generated region is only compared with regions in S_{pes} to check whether it is going to be eliminated. If it is not then, there is a possibility that it may be added to S_{pes}. Furthermore, it is also possible that it displaces an output region from S_{pes} to R_{temp}. At the end of this first pass we get the final S_{pes} as all newly generated regions have been compared against it. At this point there may be some output regions in R_{temp} that are by now completely dominated but are still not eliminated because they were never compared to the incoming regions. The second pass which we call the *one pass scan* identifies these dominated regions and eliminates them by performing dominance comparisons between all regions in R_{temp} with S_{pes}.

Chapter 8

The Adaptor

This component is reponsible for repartitioning the selected regions. In order to understand the process let us first revisit how a region is formed in the first place. Consider figure 8.1 that shows two input partitions I^R and I^T. These two partitions need to be joined using the following mapping functions:

$f_x : R.distance + T.distance = tdistance$ and $f_y : R.price + T.price = tprice$

Applying the above mapping functions to I^R and I^T we obtain the output region R_1 as shown in Figure 8.1. Thus we perform the join operation at a higher level of abstraction.

Now that we have revisited how regions are formed will start understanding *how regions are adapted?* Consider that during one of the iterations region R_1 created above is selected for adapting or repartitioning. Below are the 2 steps taken to adapt the region:

Step 1: To divide a region first divide its input partitions

Inorder to divide the partitions we first need to determine a smaller δ. For the sake of simplicity in this work we have chosen the new δ also called δ' to be half of the previous δ . i.e. δ' = $\delta/2$. Note that now that we have built a framework, any complex metric for selecting δ' can be plugged into this. After repartitioning the new input partitions that are created are shown in Figure 8.2.b.

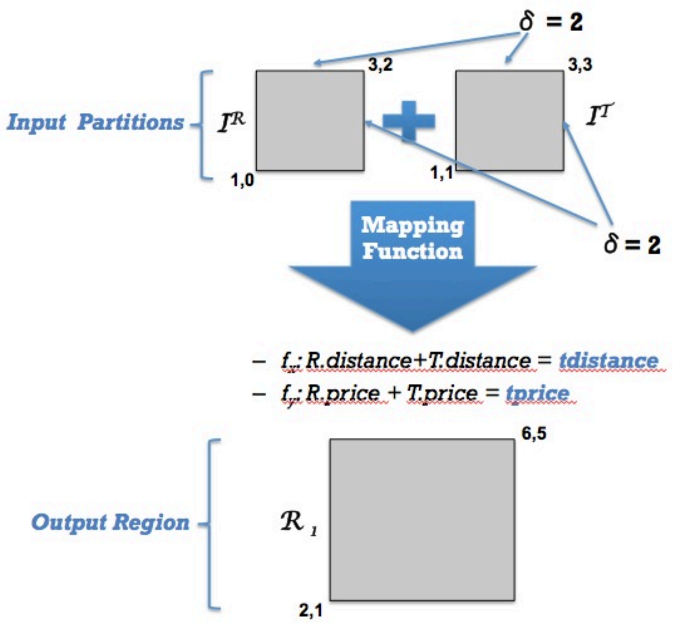
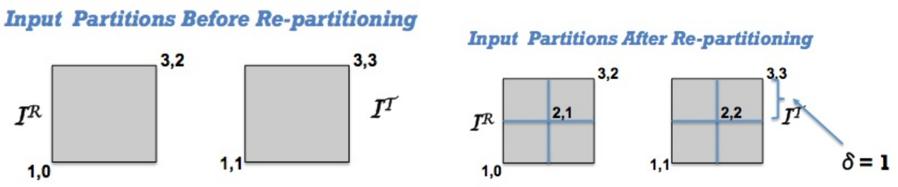


Figure 8.1: How Input Partitions are combined to form Output Region by applying Mapping Functions


Input Partitions Before Re-partitioning
3,2
3,3
I^R
I^T
1,0
1,1
Input Partitions After Re-partitioning
3,2
3,3
I^R
2,1
2,2
I^T
1,0
1,1
δ = 1

(a) Input partitions before repartitioning (b) Input partitions after repartitioning

Figure 8.2: Step 1: Repartition the input partitions that form the selected output region

Note that since R_1 is now adapted, we delete it from memory. However that does not imply that the original input partitions of R_1 can also be deleted. We store the child partitions in a hierarchical structure. This is done for the following reasons. Each input partition in relation R combines with all input partitions of region T to create output regions. Therefore it is possible that even after elimination of regions an input partition I^R may still belong to more than one output region. Let us assume that the Decision Maker selects one of the regions R_1 to which I^R belongs. Now, there may be another region R_2 that still refers to I^R. Therefore it cannot be removed from the system. Also, if susbequently R_2 is selected for repartitioning it need not re-create the child partitions of

I_a^R, but can directly access its children created earlier.

Step 2: Combine all child partitions to form new but smaller output regions

This step is performed by the *Output Region Generator* but for the sake of continuity it is important to discuss it here. As we discussed in Section 2, these newly created input partitions will join based on the join attributes. For simplicity if we assume that all input partitions combine with one another in both data sets, we get a new set of output regions to replace the repartitioned parent region.

Having discussed out framework in technical depth we will now analyze its performance with respect to SKIN.

Chapter 9

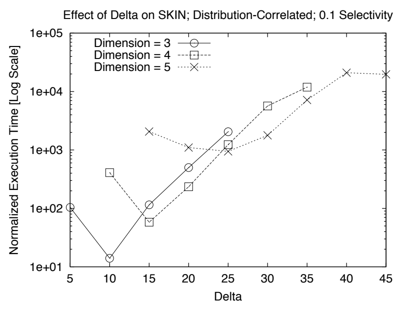
Experimental Evaluation

In this section, we verify the effectiveness of our proposed *AdaptiveSky* approach in comparison to the *SKIN*. A detailed comparison of *SKIN* against state-of-the-art techniques can be found in our recently published work [13].

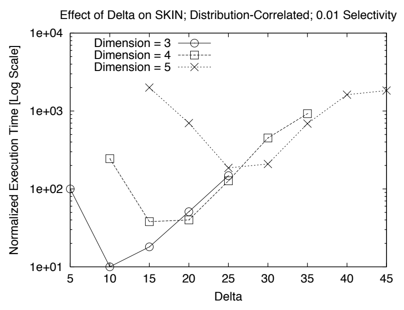
Experimental Platform. All experiments were conducted on a Linux machine(s) with AMD 2.6GHz Dual Core Opteron CPUs with 8 GB of memory. The algorithms were implemented in Java. In our analysis we use the total execution time as the comparison metric.

Benchmark Data. We conduct our experiments using data sets generated by [2]– *de-facto* standard for stress testing skyline algorithms. The data sets contain three extreme attribute correlations, namely *independent*, *correlated*, or *anti-correlated*. For each data set (R and T), we vary the number of skyline dimensions d while keeping the cardinality N constant at [500K]. The attribute values are real numbers in the range of [1–100]. The join selectivity σ is varied in the range $[10^{-3}$–$10^{-1}]$. We set $|R| = |T| = N$.

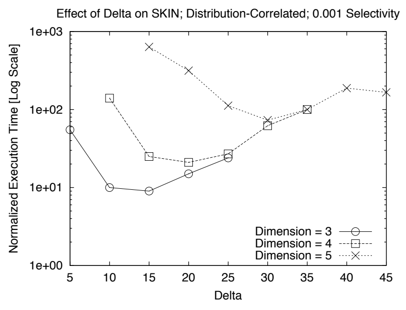
Experimental Analysis of SKIN: Effect of δ We first experimentally show the effects of varying partition sizes δ on SKIN approach. As illustrated earlier in Section **??** the effectiveness of SKIN greatly depends on the ratio between the tuple-level vs. the



(a) $\sigma = 0.1$



(b) $\sigma = 0.01$



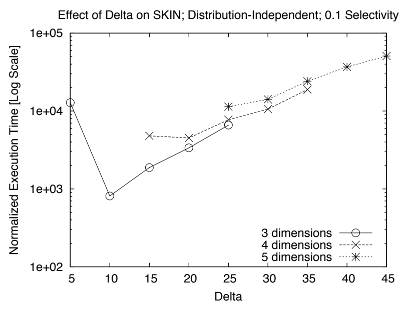
(c) $\sigma = 0.001$

Figure 9.1: Effect Partition Size δ on SKIN executed on Correlated data set

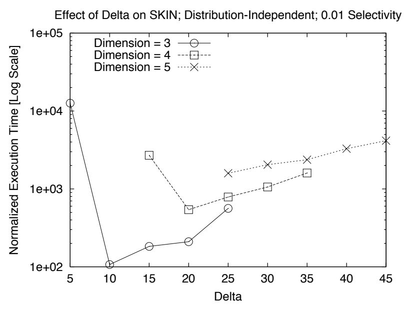
abstract-level granularity. Figures 9.1, 9.2 and 9.3 show the execution time of SKIN for correlated, independent and anti-correlated data sets respectively for varying join selectivities. Smaller partition sizes result in many sparsely populated input partitions, and therefore the overhead costs of macro-level processing will out-weigh its benefits. This is evident for all three distributions. Alternatively, as δ is increased the execution costs of macro-level processing is reduced, reflected by the dip in execution time. 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 is depicted by the slow rise in execution costs as δ increases. Results in Figures 9.1, 9.2 and 9.3 imply that SKIN is only as good as its δ.

Experimental Analysis of AdaptiveSky: Effect of threshold

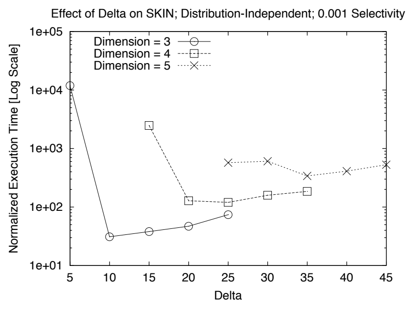
In this experiment, we study the effect of varying the threshold value on AdaptiveSky approach. Besides the threshold we vary (1) data distributions, (2) dimensions d and (3) join factor σ. For a given dimension d, data distribution, join factor σ and cardinality



(a) $\sigma = 0.1$



(b) $\sigma = 0.01$

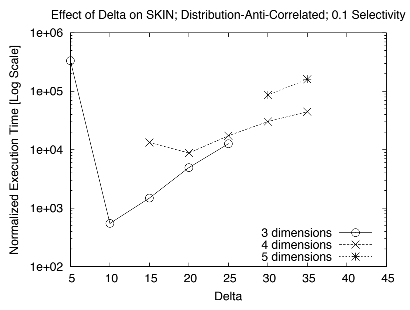


(c) $\sigma = 0.001$

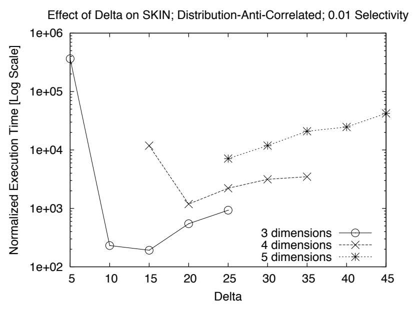
Figure 9.2: Effect Partition Size δ on SKIN executed on Independent data set

N=500K, we measure the normalized execution time for each threshold. A meaningful stopping condition is at the heart of any iterative technique. In this work the threshold value is used as our stopping condition. In this evaluation we have presented threshold as a percentage of the size of the input data sets that are participating in the join query. When the cardinality of input partitions of an output region is less than threshold% of $|N|$ then the output region is no longer considered for repartitioning. In Figures 9.4, 9.5 and 9.6 AdaptiveSky displays consistent performance across the selected threshold values for each data distribution.

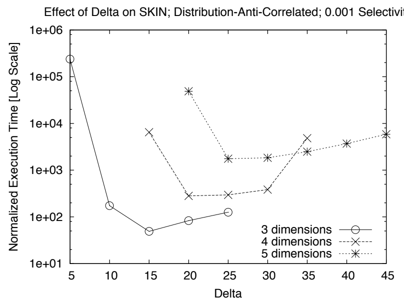
AdaptiveSky vs. SKIN. Next, we compare SKIN against our proposed AdpativeSky approach. In the case of SKIN we have measured its execution time with two δ. SKIN (H) is measured with a high performing due to a good pick of δ while SKIN (L) is measured with a low performing sub-optimal δ. This is to show that SKIN's performance is solely dependent on the selected δ and can be easily surpassed if not selected on the basis on empirical knowledge. Figures 9.7,9.8, and 9.9 compare the execution time of AdaptiveSky



(a) $\sigma = 0.1$



(b) $\sigma = 0.01$

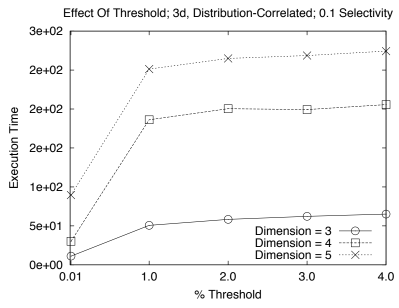


(c) $\sigma = 0.001$

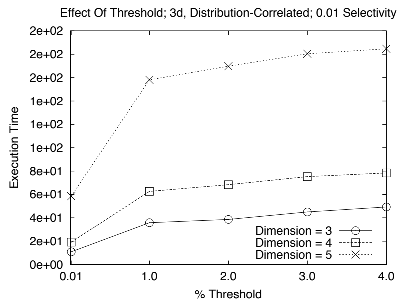
Figure 9.3: Effect Partition Size δ on SKIN executed on Anti-Correlated data set vs. SKIN for varying dimensions $d[3-5]$, data distributions and join selectivities.

For correlated data , AdaptiveSky performs greater than 1 order of magnitude faster than SKIN (L) irrespective of dimensions. In the case of SKIN (H), as the dimensions increases picking an uniform delta affects performance making the AdaptiveSky approach 1 order of magnitude faster. For independent distribution and $d = 3$, AdaptiveSky is found to be at most 1 order of magnitude faster than low performing SKIN (L). In comparisons to SKIN (H) and $d = 3$, the adaptive approach is 1.5 folds faster. In the case of $d = 5$, the adaptive approach is about 20% faster than SKIN (L) and marginally better than SKIN (H). For anti-correlated data, AdaptiveSky performs markedly better than SKIN (L) while being comparable to SKIN's performance when an optimal δ is chosen.

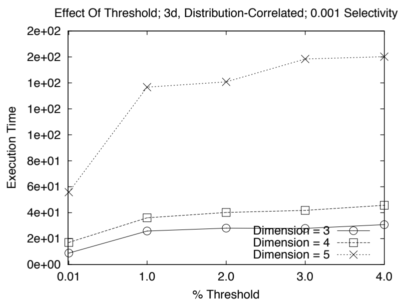
No. of combined objects generated and No. of dominance comparisons. In Figures 9.10, 9.11 and 9.12 we have compared the number of combined objects between AdaptiveSky and high performing SKIN. The observations are in line with what would be expected from the execution time comparisons. For correlated distribution AdaptiveSky



(a) $\sigma = 0.1$



(b) $\sigma = 0.01$

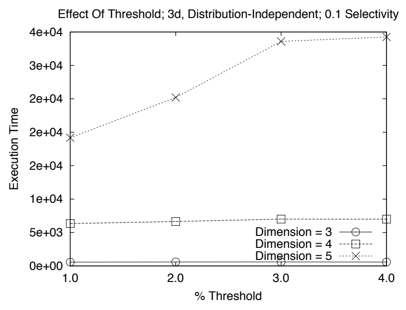


(c) $\sigma = 0.001$

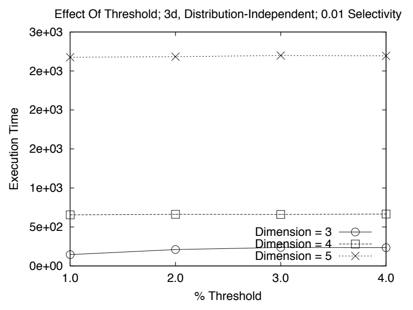
Figure 9.4: Effects of Threshold on AdaptiveSky executed on Correlated data set

has been able to minimize both parameters by 1-2 orders of magnitude thus resulting in significant performance gain as shown in 9.7. In case of independent and anti-correlated data sets both high performing SKIN and AdaptiveSky perform neck to neck. In this experiment too the No. of combined objects generated and No. of dominance comparisons are at similar levels for $d = 4$ and $d = 5$.

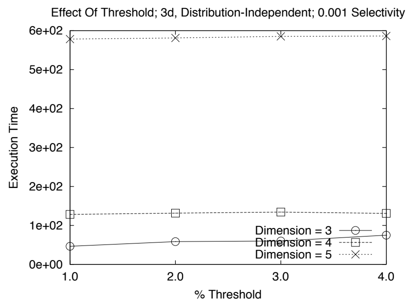
Experimental Conclusions. The main findings of our performance study can be summarized as follows: (1) our AdaptiveSky is robust to all three distributions. (2) AdaptiveSky outperform the low performing SKIN (L) in many cases by several orders of magnitude. It also outperforms the high performing SKIN (L) for correlated data sets. (3) Our proposed approach has made the process independent of a pre-determined δ that is shown to have a very high impact on performance.



(a) $\sigma = 0.1$

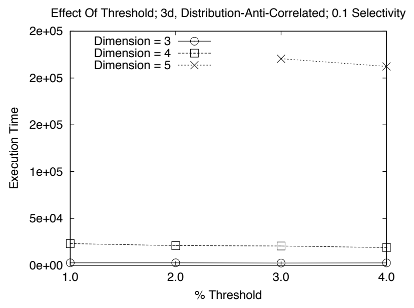


(b) $\sigma = 0.01$

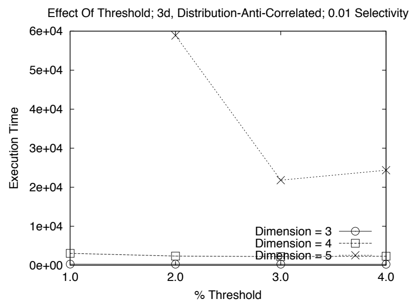


(c) $\sigma = 0.001$

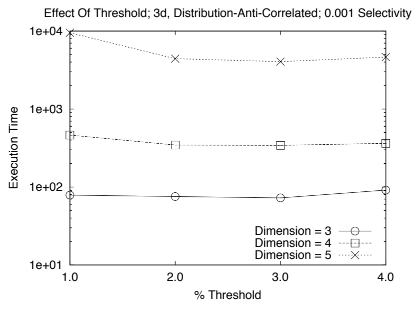
Figure 9.5: Effects of Threshold on AdaptiveSky executed on Independent data set



(a) $\sigma = 0.1$

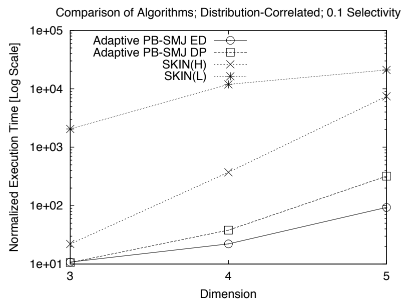


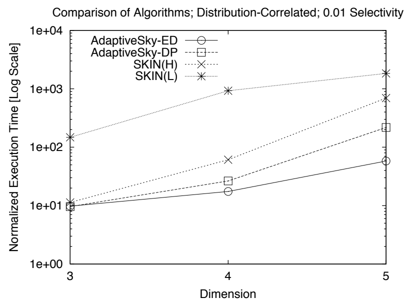
(b) $\sigma = 0.01$



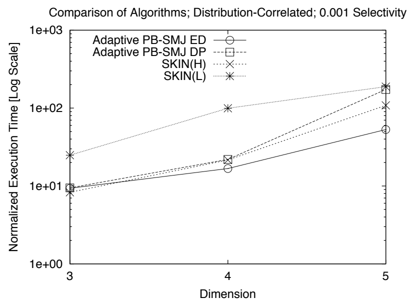
(c) $\sigma = 0.001$

Figure 9.6: Effects of Threshold on AdaptiveSky executed on Anti-Correlated data set



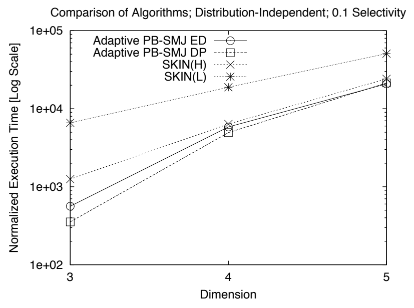


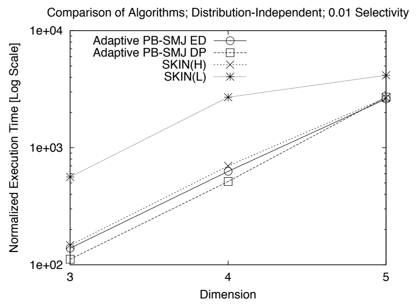
(a) $\sigma = 0.1$ (b) $\sigma = 0.01$



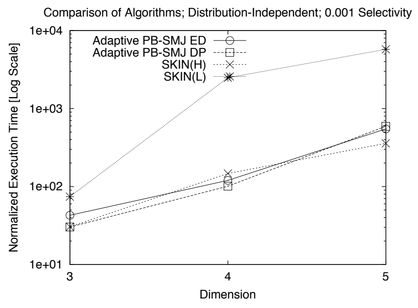
(c) $\sigma = 0.001$

Figure 9.7: Execution Time comparisons between AdaptiveSky and SKIN in Correlated data



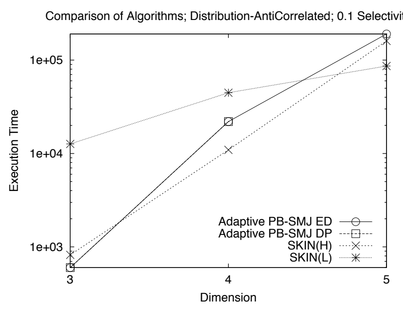


(a) $\sigma = 0.1$ (b) $\sigma = 0.01$

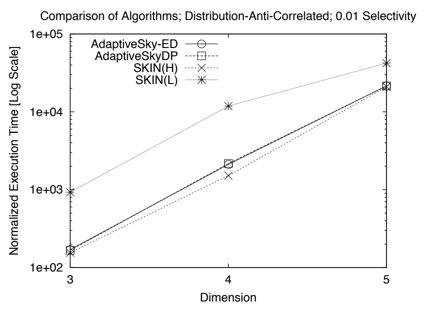


(c) $\sigma = 0.001$

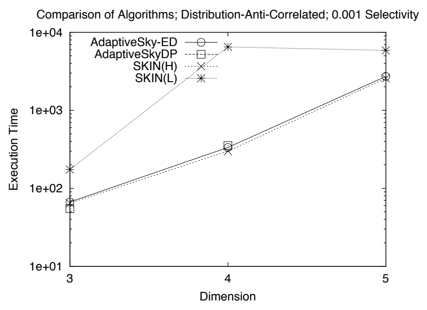
Figure 9.8: Execution Time comparisons between AdaptiveSky and SKIN in Independent data



(a) $\sigma = 0.1$

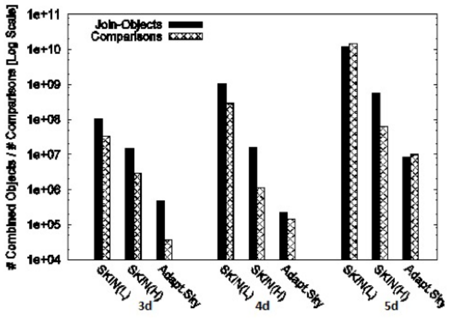


(b) $\sigma = 0.01$

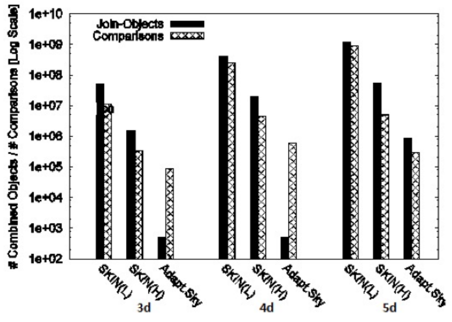


(c) $\sigma = 0.001$

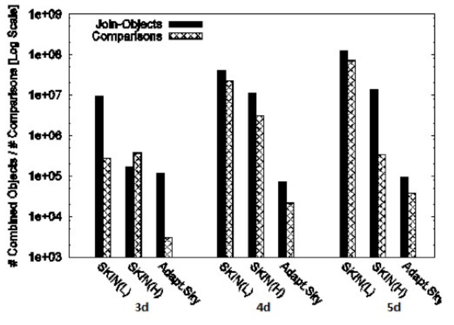
Figure 9.9: Execution Time comparisons between AdaptiveSky and SKIN in Anti-Correlated data



(a) $\sigma = 0.1$

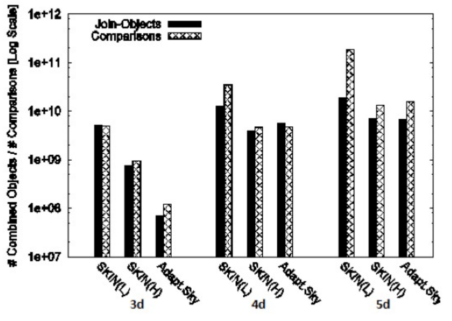


(b) $\sigma = 0.01$

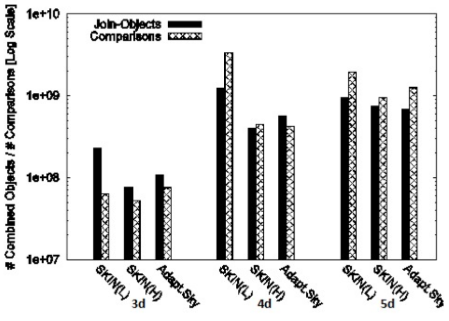


(c) $\sigma = 0.001$

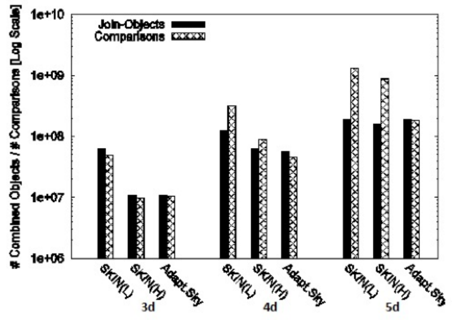
Figure 9.10: No. of combined objects and dominance comparison count between SKIN and AdaptiveSky for Correlated data set



(a) $\sigma = 0.1$

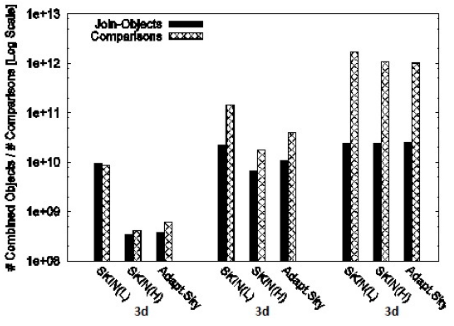


(b) $\sigma = 0.01$

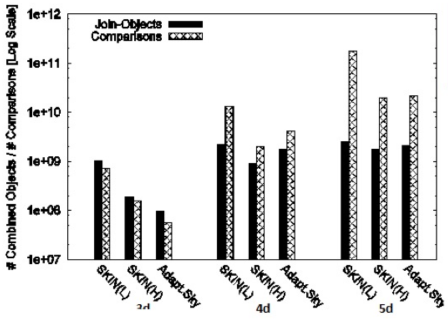


(c) $\sigma = 0.001$

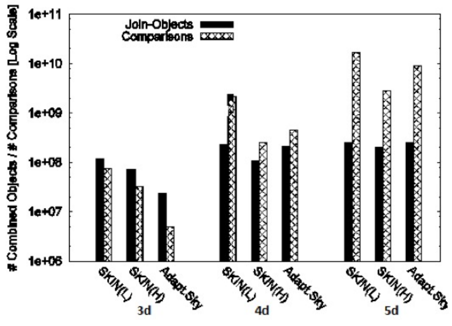
Figure 9.11: No. of combined objects and dominance comparison count between SKIN and AdaptiveSky for Independent data set



(a) $\sigma = 0.1$



(b) $\sigma = 0.01$



(c) $\sigma = 0.001$

Figure 9.12: No. of combined objects and dominance comparison count between SKIN and AdaptiveSky for Anti-Correlated data set

Chapter 10

Related Work

Skyline Algorithms over Single Relation. The majority of research on skylines has focused on the efficient computation of a skyline over a single set [1, 2, 11].

Skyline Algorithms over Disparate Sources. In the context of returning meaningful results by relaxing user queries, [8] presented various strategies that follow the join-first, skyline-later (JF-SL) paradigm (see Section **??**). This approach does not consider mapping functions. In fact, it is shown to be effective only for correlated data where the combined-object generation can be stopped early [2] confirming the findings presented in [8]. [5, 6] proposed *SSMJ* (*Skyline Sort Merge Join*) technique 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 [14]. 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 clearly contribute to be in the output no longer holds here. This is so because they do not consider mapping functions which can affect dominance characteristics. Following the principles proposed in [6], [7] recently have proposed extensions in Postgres to support for a variety of preference queries.

Adaptive techniques in Skylines. [15] proposes an adaptive algorithm for controlling the degree of parallelism and the required network traffic while computing skyline result in a distributed environment. [3] too addresses the skyline computation for single data sets in a distributed environment but for higher dimensional data sets. They use dimension reduction techniques to reduce the dimensions to 1 single dimension. These techniques however limit themselves to processing skylines over a single data set.

Top-K or Ranked Queries retrieve the best K objects that minimize a user-defined *scoring function* to name a few [4, 9, 10]. 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 may not be part of the skyline [11], or vice versa.

Chapter 11

Conclusion

The efficient evaluation of skyline over disparate sources is burdened by two primary component cost factors, 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 do not consider scenarios when attributes across these sources through user-defined mapping functions to characterize the final result. Although SKIN overcomes these drawbacks, it suffers from the dependence of its performance on a pre-determined abstraction level which may lead to poor performance if chosen level is poor. In this work, we proposed AdaptiveSky that dynamically chooses the abstraction level of output spaces based on knowledge of their pruning capacity. This helps in attaining a balance between the benefits of effective elimination in the output space and the cost of adapting abstraction levels. We demonstrate the superiority of our approach over SKIN confirming the effectiveness of our methodology.

Chapter 12

Future Work

One of the key challenges that we address to achieve effective elimination is which otuput region should be selected and how it should be repartitioned. In our current approach the 'which' question is answered by the Euclidean distance and Dominance potential methods of selection. Another, parameter that could play an important role in determining the effectiveness of elimination in skewed data sets is the data population of the regions being eliminated. Let us assume that, according to the Dominance Potential method we select region $R_{a,b}$ for repartitioning and that $R_{a,b}$'s dependent count is N_{dep}. In a uniform distribution the number of output ojects pruned by the elimination of these N_{dep} is proportional to N_{dep}, i.e. $O(N_{dep} \cdot (\sigma) \cdot n_a^R \cdot n_b^T)$. However this is not the case in a skewed data set like an anti-correlated one. In such data sets if the dependent regions being pruned are scarcely populated, the cost of repartitioning the region may not be worth the small benefit achieved. Therefore, for such scenarios having a Data Population based metric for the priority queue could be a viable option.

In our approach the second question of 'how' to repartition is addressed by considering δ_{new} to be $1/2(\delta)$ for a given partition. This is a rather simplistic approach to address the issue. A more sophisticated and effective method of determining a new δ could be as follows. Instead of having a uniform δ_{new} for all dimensions of the parent partition, we could have a different δ_{new} for each dimension. Inorder to find δ_{new} for each dimension we need to maintain the minimum partition size such that the new region dominates all its dependent regions in that dimension. On one hand it may increase the computation cost for maintaining this minimum size, however it will also gaurantee that by repartitioning a region once, all its dependents are eliminated.

Another opportunity of optimization is to to use data structures different than the priority queue for selecting regions. Because the purpose is basically to retrieve the best region in that particular iteration a *Heap* could be used for better performance.

An important area of improvement is using this framework for higher dimensions. Currently our framework only supports 3 to 5 dimensions. However by using some hierarchical grid based data structures specially made for higher dimension data like X-trees, our AdaptiveSky framework could be extended for higher dimensions.

So far we have been focussing more on the algorithmic side of the framework. An important improvement that can be made to this framework is on its design. Inorder to get the maximum benefits of a plug and play model for using a variety of metrices applying a components based design is the best way forward.

Acknowledgment

This work is supported by the National Science Foundation under Grant No. IIS-0633930 and CRI-0551584. I thank Dr. Donald Kossmann for the synthetic data generator, which is the de-facto benchmark data sets for skyline evaluation. I also thank Professor Elke A. Rundensteiner and Venkatesh Raghavan for their invaluable inputs, unending support and guidance.

Bibliography

[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] L. Chen, B. Cui, H. Lu, L. Xu, and Q. Xu. isky: Efficient and progressive skyline computing in a structured p2p network. In *ICDCS*, pages 160–167, 2008.

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

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

[6] W. Jin, M. Morse, J. Patel, M. Ester, and Z. Hu. Evaluating skylune in the presence of equi-joins. In *ICDE*, pages 249–260, 2010.

[7] M. Khalefa, M. F. Mokbel, and J. Levandoski. Prefjoin: An efficient preference-aware join operator. In *ICDE*, 2011, To Appear.

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

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

[10] 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.

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

[12] V. Raghavan and E. A. Rundensteiner. Progressive result generation for multi-criteria decision support queries. In *ICDE*, pages 733–744, 2010.

[13] V. Raghavan, S. Srivastava, and E. Rundensteiner. Skyline and mapping aware join query evaluation. *Information Systems (2011)*, To Appear. Technical Report: WPI-CS-TR-09-03.

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

[15] G. Valkanas and A. N. Papadopoulos. Efficient and adaptive distributed skyline computation. In *SSDBM*, pages 24–41, 2010.