Output-sensitive Evaluation of Prioritized Skyline Queries

Niccolò Meneghetti
Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000
niccolom@buffalo.edu

Denis Mindolin
Bloomberg L.P.
731 Lexington Avenue
New York, NY 10022
dmindolin1@bloomberg.net

Paolo Ciaccia
DISI
University of Bologna
Mura Anteo Zamboni, 7
40126 Bologna Italy
paolo.ciaccia@unibo.it

Jan Chomicki
Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000
chomicki@buffalo.edu

ABSTRACT

Skylines assume that all attributes are equally important, as each dimension can always be traded off for another. Prioritized skylines (*p-skylines*) take into account *non-compensatory* preferences, where some dimensions are deemed more important than others, and trade-offs are constrained by the relative importance of the attributes involved.

In this paper we show that querying using non-compensatory preferences is computationally efficient. We focus on preferences that are representable with *p-expressions*, and develop an efficient in-memory divide-and-conquer algorithm for answering p-skyline queries. Our algorithm is *output-sensitive*; this is very desirable in the context of preference queries, since the output is expected to be, on average, only a small fraction of the input. We prove that our method is well behaved in both the worst- and the average-case scenarios. Additionally, we develop a general framework for benchmarking p-skyline algorithms, showing how to sample prioritized preference relations uniformly, and how to highlight the effect of data correlation on performance. We conclude our study with extensive experimental results.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query languages*; H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Experimentation, Performance

Keywords

preference, preference query, skyline, p-skyline, Pareto accumulation, Prioritized accumulation

1. INTRODUCTION

Output-sensitive algorithms are quite popular in computational geometry. Starting from the classical results on convex hulls by Kirkpatrick and Seidel [26] and Chan [9], output-sensitive solutions have been explored in several problem domains. Let's denote the input- and the output-size by n and v, respectively. An output-sensitive algorithm is designed to be efficient when v is a small fraction of n. More specifically, its asymptotic complexity should depend explicitly on v, and gracefully degrade to the level of the best known output-*insensitive* algorithms when $v \in \Omega(n)$. Preference queries [11, 12, 22] provide an interesting domain for output-sensitive algorithms. Starting from a large set of tuples, the goal is to extract those that maximize a given binary preference relation. Preference queries are not aimed at retrieving tuples that perfectly match user's interests, but rather at filtering out those that clearly do not. Hence, if preferences are modeled properly, the user should expect to see only a fairly limited number of results.

The most basic preferences a user can express are those that depend on a single attribute: for example, a user may state that she prefers cheap cars over expensive ones, or that she is more comfortable driving with the manual transmission rather than with the automatic one. Single-attribute preferences are easy to elicit and we will assume they are known a priori.

Several ways exist to combine simple, one-dimensional preferences into composite, multi-dimensional ones. A popular approach is to look for tuples that are *Pareto*-optimal. A tuple is Pareto-optimal when no other tuple Pareto-dominates it, being better in one dimension and no worse in all the other dimensions. Within the database community, the set of tuples satisfying Pareto-optimality is called a *skyline* [7, 13]. It is important to notice that skylines always allow to trade off one dimension for another: even if a tuple is dominated in several dimensions, it can still be part of the skyline as long as it guarantees an arbitrarily small advantage on some other dimensions. In other words, one dimension can always compensate for another. It has been proved that skylines contain all the tuples maximizing any arbitrary scoring function[1]. Hence, if we assume that users make their decisions according to some hidden utility function, under reasonable assumptions we are guaranteed that their preferred tuple will belong to the skyline. The first output-sensitive algorithm for skyline queries is due to Kirkpatrick and Seidel [25].

Prioritized-skylines [22, 29] take a rather different approach: single-dimensional preferences are composed according to a specific priority order, that is dictated by the user. For example: let's assume a customer is looking for a cheap, low-mileage vehicle, and she would prefer to have a manual transmission *as long as there is no extra charge for it.* Such customer exhibits preferences over three dimensions (price, mileage and transmission), but the preference on price is deemed more important than the one on transmission type. In other words, the user is not willing to negotiate on price if the only reward is to obtain a better transmission. Notice that we have no reason to assume that the preference on mileage is either more or less important than any of the other two. Prioritized skylines are a special-case of *non-compensatory decision making*, a phenomenon that has been extensively studied in economics and psychology [31, 16, 17]. A non-compensatory preference arises whenever a user refuses to negotiate a trade-off between attributes (for example: a higher price for a better transmission), deeming one dimension infinitely more important than the other. P-skylines allow to model this kind of preferences naturally. Semantically, they generalize skylines: when the user considers all the attributes to be equally important, the p-skyline contains only Pareto-optimal tuples. A p-skyline query always returns a subset of the skyline; in practice, it usually returns just a small portion of it.

The main contribution of this paper is to develop a novel output-sensitive algorithm for p-skyline queries. We show the problem is $\mathcal{O}\left(n \log^{d-2} v\right)$ in $d \geq 4$ dimensions, and $\mathcal{O}\left(n \log v\right)$ in two and three dimensions. Hence, our work generalizes the results in [25] to the context of p-skylines. Our solution differs significantly from [25], as we show how to exploit the semantics of prioritized preferences for devising an effective divide-and-conquer strategy. Additionally, we prove the algorithm is $\mathcal{O}\left(n\right)$ in the average case.

We conclude our work presenting extensive experimental results on real-life and synthetic data. Apart from the nice asymptotic properties, our algorithm proves to be practical for processing realistically sized data sets. For our evaluations we use data sets with up to one million records and up to 20 attributes. To the best of our knowledge our benchmarks are in line with most of the studies of skyline queries in the literature.

2. NOTATION AND RELATED WORK

In this document we adopt the following typographic conventions: sets of tuples (or operators returning set of tuples) are written as capital letters (like D, B, W or $M(D)$), sets of attributes are written in calligraphic font (like $\mathcal{A}$, $\mathcal{C}$ or $\mathcal{E}$), actual attributes are written in boldface font (for example: $\mathbf{A}_1, \mathbf{A}_2, \ldots$).

[1]Assuming the function is defined on all dimensions.

$\mathcal{A}$	a relation schema
D	a relation instance
$\mathbf{A}_1, \mathbf{A}_2, \ldots$	attributes
$t_1, t_2, \ldots$	tuples
π	a p-expression
$\succ_\pi$	the strict partial order induced by π
$M_{\text{sky}}(D)$	the skyline of D
$M_\pi(D)$	the p-skyline of D, w.r.t. π
n	the size of the input (# of tuples)
v	the size of the output (# of tuples)
d	the number of relevant attributes
$W \not\succeq_\pi B$	no tuple in W is better than (or indistinguishable from) any tuple in B
Γ_π, Γ_π^r	the p-graph of π and its trans. reduction
$\mathcal{V}ar(\pi)$	the attributes appearing in π
$\mathcal{S}\text{ucc}_\pi(\mathbf{A}_i)$	immediate successors of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{D}\text{esc}_\pi(\mathbf{A}_i)$	descendants of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{P}\text{re}_\pi(\mathbf{A}_i)$	immediate predecessors of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{A}\text{nc}_\pi(\mathbf{A}_i)$	ancestors of $\mathbf{A}_i$ in Γ_π^r
$\mathcal{R}\text{oots}_\pi$	attributes having no ancestors in Γ_π^r
$\mathcal{B}\text{etter}_\pi(t', t)$	attributes where t' is preferred to t
$\mathcal{T}op_\pi(t', t)$	topmost attributes in Γ_π^r where t and t' disagree
$d_{\mathbf{A_i}}$	the depth of $\mathbf{A}_i$, the length of the longest path in Γ_π^r from any root to $\mathbf{A}_i$
$C_d(v, n)$	worst-case complexity of skyline queries
$F_d(b, w)$	w.c. complexity of screening queries
$C_d^*(v, n)$	w.c. complexity of p-skyline queries
$F_d^*(b, w)$	w.c. complexity of p-screening queries

For the convenience of the reader the above table summarizes the most common notations used throughout the paper. All notational conventions are formally introduced and explained in more details in the following sections.

2.1 Modeling Preferences

Several frameworks have been proposed for modeling preferences in databases, many of which are surveyed in [34]. In this paper we follow the *qualitative* approach [11, 22, 12], as we assume user's wishes are modeled as strict partial orders. We denote by $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \ldots\}$ a finite set of attributes defining a relation schema, and by U the set of all possible tuples over such schema. Without lack of generality, we assume attributes can be either discrete or range over the set of rational numbers. A *preference* is a strict partial order $\succ$ over U, a subset of $U \times U$ being transitive and irreflexive. The assertion $t' \succ t$ (t' *dominates* t) means the user prefers t' over t, i.e. she's always willing to trade the tuple t for the tuple t' if she is given the chance. Given a relation instance $D \subseteq U$ and a preference $\succ$, a *preference query* retrieves all the maximal elements of the partially ordered set $(D, \succ)$. Following the notation of [25], we denote by $M_\succ(D)$ the set of all these maximal tuples:

$$M_\succ(D) = \{t \in D \mid \nexists\, t' \in D \text{ s.t. } t' \succ t\}$$

Every preference $\succ$ induces an indifference relation ($\sim$): $t' \sim t$ holds whenever $t' \not\succ t$ and $t \not\succ t'$. A preference $\succ$ is a *weak order* when $\sim$ is transitive. A weak order extension of $\succ$ is simply an arbitrary weak order containing $\succ$. Two tuples t' and t are *indistinguishable* with respect to $\succ$ when they agree on all attributes that are relevant for deciding $\succ$. In such case we write $t' \approx t$. We denote by $t' \succeq t$ the fact that t' is either better than or indistinguishable from t. If D and D' are two relation instances, we write $D' \not\succeq D$ when there is no pair of tuples (t', t) in $D' \times D$ such that $t' \succeq t$.

Declaring a preference by enumerating its elements is impractical in all but the most trivial domains. Kießling [22] suggested to define preferences in a constructive fashion, by iteratively composing single-attribute preferences into more complex ones. From his work we borrow two composition operators, namely the *Pareto accumulation* ($\otimes$) and the *Prioritized accumulation* (&). Denote $\succ_1$ and $\succ_2$ two strict partial orders; their Pareto accumulation $\succ_{1\otimes 2}$ is defined as follows

$$\forall\, t', t \quad t' \succ_{1\otimes 2} t \iff \begin{array}{l}\big(t' \succ_1 t \wedge t' \succeq_2 t\big) \vee \\ \big(t' \succ_2 t \wedge t' \succeq_1 t\big)\end{array}$$

In other words, $t' \succ_{1\otimes 2} t$ holds when t' is better according to one of the preferences $\succ_1$ and $\succ_2$, and better or indistinguishable from t according to the other. Hence, the two preferences are equally important. The prioritized composition of $\succ_1$ and $\succ_2$ is defined as follows

$$\forall\, t', t \quad t' \succ_{1\&2} t \iff t' \succ_1 t \vee \big(t' \approx_1 t \wedge t' \succ_2 t\big)$$

Clearly $\succ_{1\&2}$ gives more weight to the first preference, as $\succ_2$ is taken into consideration only when two tuples are indistinguishable w.r.t. $\succ_1$. Both operators are associative and the Pareto accumulation is commutative [22]. Several standard query languages have been extended to support preference constructors like the Pareto- and the Prioritized accumulation, including SQL [24], XPATH [23], and SPARQL [33]. To the best of our knowledge we are the first to develop an output-sensitive algorithm supporting both of these constructors.

A *p-expression* [29] is a formula denoting multiple applications of the above operators. P-expressions respect the following grammar:

$$\begin{array}{rcl} \textit{pExpr} & \rightarrow & \textit{paretoAcc} \mid \textit{prioritizedAcc} \mid \textit{attribute} \\ \textit{paretoAcc} & \rightarrow & \textit{pExpr} \otimes \textit{pExpr} \\ \textit{prioritizedAcc} & \rightarrow & \textit{pExpr}\ \&\ \textit{pExpr} \\ \textit{attribute} & \rightarrow & \mathbf{A}_1 | \mathbf{A}_2 | \ldots \end{array}$$

where all non-terminal symbols are lower-case and each terminal symbol is either a composition operator ($\otimes$ or &) or a single-attribute preference, with the restriction that no attribute should appear more than once. A single-attribute preference (also denoted by $\succ_{\mathbf{A}_i}$) is simply an arbitrary total order defined over the attribute's domain. Without lack of generality, we will assume users rank values in natural order (i.e. they prefer small values to larger ones), unless stated differently. With a little abuse of notation, we will use single-attribute preferences for ranking either tuples or values, depending on the context.

Example 1. *Assume that a dealer is offering the following cars*

id	**P** *(price)*	**M** *(mileage)*	**T** *(transmission)*
t_1	\$ 11,500	50,000 miles	*automatic*
t_2	\$ 11,500	60,000 miles	*manual*
t_3	\$ 12,000	50,000 miles	*manual*
t_4	\$ 12,000	60,000 miles	*automatic*

and that we are looking for a cheap vehicle, with low mileage, possibly with manual shift. Notice that while the first car is Pareto-optimal for price and mileage, if we want a manual transmission we need to give up either on getting the best price or the best mileage. Depending on our priorities, we can model our preferences in different ways. All the following are well-formed and meaningful p-expressions:

1. $\mathbf{P}$

2. $(\mathbf{P} \otimes \mathbf{M})\ \&\ \mathbf{T}$

3. $(\mathbf{P}\ \&\ \mathbf{T}) \otimes \mathbf{M}$

4. $\mathbf{M}\ \&\ \mathbf{T}\ \&\ \mathbf{P}$

Expression (1) states that we care only about price. If that is the case, we should buy either t_1 or t_2. Expression (2) states that we are looking for cars that are Pareto-optimal w.r.t. price and mileage, and that we take into consideration transmission only to decide between cars that are indistinguishable in terms of price and mileage. In this case t_1 is the best option. Expression (3) is more subtle: we are looking for cheap cars, with low mileage and manual transmission, but we are not willing to pay an extra price for the manual transmission. In this case we should buy either t_1 or t_2, since t_1 dominates t_3 and t_4. Finally, expression (4) denotes a lexicographic order: amongst the cars with the lowest mileage, we are looking for one with manual transmission, and price is the least of our concerns. In this case we should buy t_3.

Given a p-expression π we denote by $\succ_\pi$ the preference relation defined by it. Notice that $\succ_\pi$ is guaranteed to be a strict partial order [22]. We denote by $\mathcal{V}ar(\pi)$ the set of attributes that appear inside π, i.e. those that are relevant for deciding $\succ_\pi$. Notice that $t' \approx_\pi t$ holds iff t' and t agree on every attribute in $\mathcal{V}ar(\pi)$; in the following we will simply say that t and t' are indistinguishable w.r.t. attributes in $\mathcal{V}ar(\pi)$.

Definition 1. Given a relation instance $D \subseteq U$ and a p-expression π, a *p-skyline query* returns the set $M_\pi(D)$ of the maximal elements of the poset $(D, \succ_\pi)$.

The computational complexity of p-skyline queries depends strongly on the size of the input, the size of the output, and the number of attributes that are relevant to decide $\succ_\pi$. Hence, our analysis will take in consideration mainly three parameters: n, the number of tuples in the input, v the number of tuples that belong to the p-skyline, and d, the cardinality of $\mathcal{V}ar(\pi)$.

Every p-expression π implicitly induces a priority order over the attributes in $\mathcal{V}ar(\pi)$. Mindolin and Chomicki [29] modeled these orders using p-graphs, and showed how they relate to the semantics of p-skyline preferences. We will follow a similar route designing our divide-and-conquer strategy for p-skyline queries. Hence, we need to introduce some of the notation used in [29].

Definition 2. A p-graph Γ_π is a directed acyclic graph having one vertex for each attribute in $\mathcal{V}ar(\pi)$. The set $E(\Gamma_\pi)$ of all edges connecting its vertices is defined recursively as follows:

- if π is a single-attribute preference, then $E(\Gamma_\pi) \equiv \emptyset$
- if $\pi = \pi_1 \otimes \pi_2$ then $E(\Gamma_\pi) \equiv E(\Gamma_{\pi_1}) \cup E(\Gamma_{\pi_2})$
- if $\pi = \pi_1\ \&\ \pi_2$ then $E(\Gamma_\pi) \equiv E(\Gamma_{\pi_1}) \cup E(\Gamma_{\pi_2}) \cup (\mathcal{V}ar(\pi_1) \times \mathcal{V}ar(\pi_2))$

Intuitively, a p-graph Γ_π contains an edge from $\mathbf{A}_i$ to $\mathbf{A}_j$ iff the preference on $\mathbf{A}_i$ is more important than the one on $\mathbf{A}_j$. Notice that p-graphs are transitive by construction and, since p-expressions do not allow repeated attributes, they are guaranteed to be acyclic. In order to simplify the notation in the following sections we will not use p-graphs directly, but we will refer mostly to their transitive reductions[2] Γ^r_π (see Figure 1). In relation to Γ^r_π we define the following sets of attributes:

$\mathcal{S}\mathrm{ucc}_\pi(\mathbf{A}_i)$	immediate successors of $\mathbf{A}_i$
$\mathcal{D}\mathrm{esc}_\pi(\mathbf{A}_i)$	descendants of $\mathbf{A}_i$
$\mathcal{P}\mathrm{re}_\pi(\mathbf{A}_i)$	immediate predecessors of $\mathbf{A}_i$
$\mathcal{A}\mathrm{nc}_\pi(\mathbf{A}_i)$	ancestors of $\mathbf{A}_i$
$\mathcal{R}\mathrm{oots}_\pi$	nodes having no ancestors

We will denote by $d_{\mathbf{A_i}}$ the depth of $\mathbf{A}_i$, i.e. the length of the longest path in Γ^r_π from any root to $\mathbf{A}_i$ (roots have depth 0).

EXAMPLE 2. *A customer is looking for a low-mileage* ($\mathbf{M}$) *car; amongst barely used models, she is looking for a car that is available near her location* ($\mathbf{D}$) *for a good price* ($\mathbf{P}$)*, possibly still under warranty* ($\mathbf{W}$)*. In order to obtain a comprehensive warranty she is willing to pay more, but not to drive to a distant dealership, since regular maintenance would require her to go there every three months. All else being equal, she prefers cars equipped with heated seats* ($\mathbf{H}$) *and manual transmission* ($\mathbf{T}$)*. Her preferences can be formulated using the following p-expression:*

$$\mathbf{M}\ \&\ ((\mathbf{D}\&\mathbf{W}) \otimes \mathbf{P})\ \&\ (\mathbf{T} \otimes \mathbf{H})$$

Figure 1(a) shows the corresponding p-graph and Figure 1(b) its transitive reduction. Notice the p-graph is not a weak order, thus the attributes cannot be simply ranked.

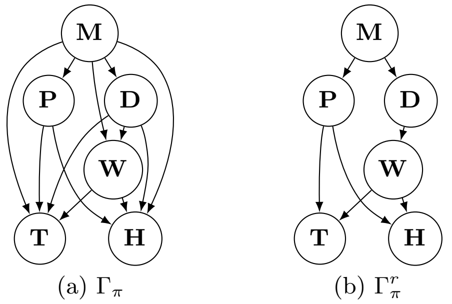


Figure 1: (a) the p-graph of the expression $\mathbf{M}\ \&\ ((\mathbf{D}\&\mathbf{W}) \otimes \mathbf{P})\ \&\ (\mathbf{T} \otimes \mathbf{H})$ and (b) its transitive reduction.

The following result from [29] highlights the relation between p-graphs and the semantics of p-skylines:

PROPOSITION 1. *[29] Denote by t and t' two distinct tuples, by $\mathcal{B}etter_\pi(t', t)$ the set of attributes where t' is preferred to t, and by $\mathcal{T}op_\pi(t', t)$ the topmost elements in Γ^r_π where t and t' disagree. The following assertions are equivalent:*

1. $t' \succ_\pi t$
2. $\mathcal{B}etter_\pi(t', t) \supseteq \mathcal{T}op_\pi(t', t)$
3. $\mathcal{D}esc_\pi(\mathcal{B}etter_\pi(t', t)) \supseteq \mathcal{B}etter_\pi(t, t')$

In other words, $t' \succ_\pi t$ holds iff the two tuples are distinguishable and every node in the p-graph for which t is preferred has an ancestor for which t' is preferred. We will take advantage of this result in Sections 4 and 6.

[2]Since every p-graph is a finite strict partial order, the transitive reduction Γ^r_π is guaranteed to exist and to be unique: it consists of all edges that form the only available path between their endpoints.

2.2 Skyline queries

A *skyline query* can be seen as a special case of p-skyline query: the Pareto-optimality criterion over an arbitrary set of attributes $\{\mathbf{A}_i, \mathbf{A}_j, \mathbf{A}_k, \ldots\}$ can be formulated as follows

$$\mathbf{A}_i \otimes \mathbf{A}_j \otimes \mathbf{A}_k \otimes \ldots$$

From now on we will denote by *sky* the above expression, and by $M_{\text{sky}}(D)$ the result of a skyline query. Clearly, the meaning of this notation will depend on the content of $\mathcal{V}ar(\text{sky})$.

PROPOSITION 2. *[29] Let π and π' be two p-expressions such that $\mathcal{V}ar(\pi) = \mathcal{V}ar(\pi')$. The following containment properties hold*

$$\succ_\pi \subset \succ_{\pi'} \Leftrightarrow E(\Gamma_\pi) \subset E(\Gamma_{\pi'}) \quad (1)$$

$$\succ_\pi = \succ_{\pi'} \Leftrightarrow E(\Gamma_\pi) = E(\Gamma_{\pi'}) \quad (2)$$

From Proposition 2 we can directly infer that $M_\pi(D) \subseteq M_{\text{sky}}(D)$ as long as $\mathcal{V}ar(\pi) = \mathcal{V}ar(sky)$.

Skyline queries have been very popular in the database community. Over several years a plethora of algorithms have been proposed, including BNL [7], SFS [14], LESS [20], SALSA [2], BBS [30], and many others. For the purpose of this paper we briefly review BNL, together with its extensions. BNL (block-nested-loop) allocates a fixed-size memory buffer able to store up to k tuples, the *window*, and repeatedly performs a linear scan of the input; during each iteration i each tuple t is compared with all the elements currently in the window. If t is dominated by any of those, it is immediately discarded, otherwise all the elements of the window being dominated by t are discarded and t is added to the window. If there is not enough space, t is written to a temporary file T_i. At the end of each iteration i all the tuples that entered the window when T_i was empty are removed and added to the final result; the others, if any, are left in the window to be used during the successive iteration $(i+1)$, that will scan the tuples stored in T_i. The process is repeated until no tuples are left in the temporary file.

SFS (sort-filter-skyline) improves BNL with a pre-sorting step; at the very beginning the input is sorted according to a special ranking function, ensuring that no tuple dominates another that precedes it. The resulting algorithm is pipelineable and generally faster than BNL. LESS (Linear Elimination Sort for Skyline [20]) and SALSA (Sort and Limit Skyline algorithm [2]) improve this procedure by applying an elimination filter and an early-stop condition.

Skylines have been studied for several decades in computational geometry, as an instance of the *maximal vector problem* [28, 6, 25]. The first divide and conquer algorithm is due to Kung, Luccio and Preparata [28] and is similar to the one used in this paper: the general idea is to split the data set in two halves, say B and W, so that no tuple in W is indistinguishable from or dominates any tuple in B ($W \not\succeq_{\text{sky}} B$); then, the skyline of both halves is computed recursively, obtaining $M_{\text{sky}}(B)$ and $M_{\text{sky}}(W)$. The last step is to remove from $M_{\text{sky}}(W)$ all tuples dominated by some element in $M_{\text{sky}}(B)$. This operation is called *screening*. Let W' be the set of tuples from $M_{\text{sky}}(W)$ that survive the screening, the algorithm returns the set $M_{\text{sky}}(B) \cup W'$, containing each and every skyline point. The base case for the recursion is when B and W are small enough to make the computation of the skyline trivial.

In [3] Bentley et al. developed a similar, alternative algorithm, and in [4] proposed a method that is provably fast in the average-case. Kirkpatrick and Seidel [25] were the first to propose an output-sensitive procedure. More recently, [32] developed a divide-and-conquer algorithm that is efficient in external memory, while several results have been obtained using the word-RAM model [18, 10, 1]. Following [25] we denote by $C_d(v,n)$ the worst-case complexity of skyline queries, and by $F_d(b,w)$ the worst-case complexity of screening, assuming $b = |B|$ and $w = |W|$. The following complexity results were obtained in [25, 27, 28]. In this paper we will prove similar results in the context of p-skylines.

PROPOSITION 3. *[25] The following upper-bounds hold on the complexity of the maximal vector problem:*

1. $C_d(v,n) \leq \mathcal{O}(n)$ *for* $d = 1$
2. $C_d(v,n) \leq \mathcal{O}(n \log v)$ *for* $d = 2,3$
3. $C_d(v,n) \leq \mathcal{O}\left(n \log^{d-2} v\right)$ *for* $d \geq 4$
4. $C_d(v,n) \leq \mathcal{O}(n)$ *when* $v = 1$

PROPOSITION 4. *[25, 27, 28] The following upper-bounds on the complexity of the screening problem hold:*

1. $F_d(b,w) \leq \mathcal{O}(b+w)$ *for* $d = 1$
2. $F_d(b,w) \leq \mathcal{O}((b+w)\log b)$ *for* $d = 2,3$
3. $F_d(b,w) \leq \mathcal{O}\left((b+w)\log^{d-2} b\right)$ *for* $d \geq 4$
4. $F_d(b,w) \leq \mathcal{O}(w)$ *when* $b = 1$

3. OUTPUT-SENSITIVE P-SKYLINE

In this section we present our output-sensitive algorithm for p-skyline queries. We first introduce a simple divide and conquer algorithm, named DC, showing the problem is $\mathcal{O}\left(n \cdot \log^{d-2} n\right)$. Later we extend it, making it output-sensitive and ensuring an asymptotic worst-case complexity of $\mathcal{O}\left(n \cdot \log^{d-2} v\right)$. Before discussing our algorithms in detail, we need to generalize the concept of screening to the context of p-skylines:

Definition 3. Given a p-expression π and two relation instances B and W, such that $W \not\succeq_\pi B$, *p-screening* is the problem of detecting all tuples in W that are not dominated by any tuple in B, according to $\succ_\pi$.

Extending the notation of [27], we denote by $\left[{B \atop W}\right]_\pi$ the problem of p-screening B and W, and by $F^*_d(b,w)$ its worst-case complexity[3]. In Section 4 we will show $F^*_d(b,w)$ is $\mathcal{O}\left((b+w)\cdot(\log b)^{d-2}\right)$. For the moment we take this result as given. We denote by $C^*_d(v,n)$ the worst-case complexity of p-skylines, where n and v are the input- and the output-size.

The complete code of DC is given in the Figure on the right. Similarly to the divide and conquer algorithm by [28], the strategy of DC is to split the input data set into two chunks, B and W, so that no tuple in W dominates or is indistinguishable from any tuple in B ($W \not\succeq_\pi B$). The algorithm then computes recursively the p-skyline of B, $M_\pi(B)$, and performs the p-screening of W against $M_\pi(B)$, i.e. it prunes all tuples in W that are dominated by some element of $M_\pi(B)$. Eventually, it recursively computes the p-skyline of the tuples that survived the p-screening, and returns it, together with $M_\pi(B)$, the p-skyline of B.

[3]As in Section 2: $d = |\mathcal{V}ar(\pi)|$, $b = |B|$ and $w = |W|$.

Algorithm Dc (divide & conquer)

Input: a p-expression π, a relation instance D_0
Output: the p-skyline $M_\pi(D_0)$

```
 1: procedure DC($\pi$, $D_0$)
 2:     return DCREC($\pi, D_0, \mathcal{R}oots_\pi, \emptyset$)

 3: procedure DCREC($\pi$, $D$, $\mathcal{C}$, $\mathcal{E}$)
 4:     if $\mathcal{C} = \emptyset$ or $|D| \leq 1$ then return $D$
 5:     select an attribute $\mathbf{A}$ from the candidates set $\mathcal{C}$
 6:     if all tuples in $D$ assign the same value to $\mathbf{A}$ then
 7:         $\mathcal{E}' \leftarrow \mathcal{E} \cup \{\mathbf{A}\}$
 8:         $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{\mathbf{A}\}$
 9:         $\mathcal{C}'' \leftarrow \mathcal{C}' \cup \{\mathbf{V} \in \mathcal{S}ucc_\pi(\mathbf{A}) : \mathcal{P}re_\pi(\mathbf{V}) \subseteq \mathcal{E}'\}$
10:         return DCREC($\pi, D, \mathcal{C}'', \mathcal{E}'$)
11:     else
12:         $(B, W, m^*) \leftarrow$ SPLITBYATTRIBUTE($D, \mathbf{A}$)
13:         $B' \leftarrow$ DCREC($\pi, B, \mathcal{C}, \mathcal{E}$)
14:         $W' \leftarrow$ PSCREEN($\pi, B', W, \mathcal{C} \setminus \{\mathbf{A}\}, \mathcal{E}$)
15:         $W'' \leftarrow$ DCREC($\pi, W', \mathcal{C}, \mathcal{E}$)
16:         return $B' \cup W''$

17: procedure SPLITBYATTRIBUTE(D, $\mathbf{A}$)
18:     Select $m^*$ as the median w.r.t. $\succ_\mathbf{A}$ in $D$
19:     Compute the set $B = \{t \in D \mid t \succ_\mathbf{A} m^*\}$
20:     Compute the set $W = \{t \in D \mid m^* \succeq_\mathbf{A} t\}$
21:     return $(B, W, m^*)$
```

In order to understand how DC works, it is important to understand how the input data set is split into B and W: the goal is to ensure that no tuple in B is dominated by (or indistinguishable from) any tuple in W. The general idea is to select an attribute from $\mathcal{V}ar(\pi)$, say $\mathbf{A}$, and compute the median tuple m^* w.r.t. $\succ_\mathbf{A}$, over the entire data set; then we can put in B all tuples that assign to $\mathbf{A}$ a better value than the one assigned by m^*, and put in W all the other tuples. If we want to be sure that $W \not\succeq_\pi B$ holds, we need to make sure that the preference on attribute $\mathbf{A}$ is not overridden by some other, higher-priority attribute. Hence, we need to choose $\mathbf{A}$ so that all tuples in both B and W agree w.r.t. all the attributes in $\mathcal{A}nc_\pi(\mathbf{A})$. In order to choose $\mathbf{A}$ wisely, DC keeps track of two sets of attributes, namely $\mathcal{E}$ and $\mathcal{C}$, ensuring the following invariants hold:

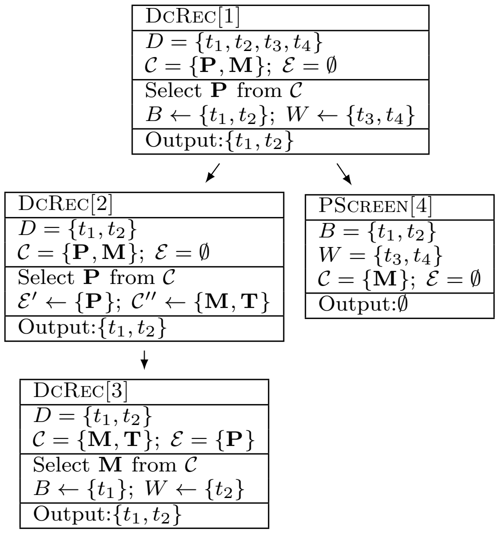
I1 : If an attribute belongs to $\mathcal{E}$ then no pair of tuples in D can disagree on the value assigned to it. In other words: all tuples in D are indistinguishable w.r.t. all attributes in $\mathcal{E}$.

I2 : An attribute in $\mathcal{A} \setminus \mathcal{E}$ belongs to $\mathcal{C}$ if and only if all its ancestors belong to $\mathcal{E}$.

Clearly, attribute $\mathbf{A}$ is always chosen from $\mathcal{C}$. Let's see how DC works in more detail: at the beginning $\mathcal{E}$ is empty and $\mathcal{C}$ contains all the root nodes of Γ^r_π; at every iteration the algorithm selects some attribute $\mathbf{A}$ from $\mathcal{C}$ and finds the median tuple m^* in D w.r.t. $\succ_\mathbf{A}$. If all tuples in D assign to $\mathbf{A}$ the same value, the algorithm updates $\mathcal{C}$ and $\mathcal{E}$ accordingly and recurs (lines 7-10). Otherwise (lines 12-16) m^* is used to split D into B and W: B contains all the tuples preferred to m^*, W those that are indistinguishable from or dominated by m^*, according to $\succ_\mathbf{A}$. Clearly $W \not\succeq_\mathbf{A} B$ holds by construction, and $W \not\succeq_\pi B$ is a direct consequence of invariants I1 and I2. Hence, the algorithm can compute $M_\pi(B)$ recursively (line 13), perform the p-screening $\left[{M_\pi(B) \atop W}\right]_\pi$ (line

14), and compute the p-skyline of the remaining tuples (line 15). The procedure PScreen at line 14 will be discussed in Section 4.

Example 3. *Let's see how* Dc *would determine the p-skyline for the data set of Example 1, with respect to the p-expression* $\pi = (\mathbf{P} \,\&\, \mathbf{T}) \otimes \mathbf{M}$. *The p-graph of* π *contains three nodes and only one edge, from* $\mathbf{P}$ *to* $\mathbf{T}$. *The following diagram shows the invocation trace of the procedure* DcRec. *Each box represents an invocation, its input, output and a short explanation of the actions performed. For the lack of space we omit the invocations that process only one tuple.*


DcRec[1]
$D = \{t_1, t_2, t_3, t_4\}$
$\mathcal{C} = \{\mathbf{P}, \mathbf{M}\}$; $\mathcal{E} = \emptyset$
Select $\mathbf{P}$ from $\mathcal{C}$
$B \leftarrow \{t_1, t_2\}$; $W \leftarrow \{t_3, t_4\}$
Output:$\{t_1, t_2\}$
DcRec[2]
$D = \{t_1, t_2\}$
$\mathcal{C} = \{\mathbf{P}, \mathbf{M}\}$; $\mathcal{E} = \emptyset$
Select $\mathbf{P}$ from $\mathcal{C}$
$\mathcal{E}' \leftarrow \{\mathbf{P}\}$; $\mathcal{C}'' \leftarrow \{\mathbf{M}, \mathbf{T}\}$
Output:$\{t_1, t_2\}$
PScreen[4]
$B = \{t_1, t_2\}$
$W = \{t_3, t_4\}$
$\mathcal{C} = \{\mathbf{M}\}$; $\mathcal{E} = \emptyset$
Output:$\emptyset$
DcRec[3]
$D = \{t_1, t_2\}$
$\mathcal{C} = \{\mathbf{M}, \mathbf{T}\}$; $\mathcal{E} = \{\mathbf{P}\}$
Select $\mathbf{M}$ from $\mathcal{C}$
$B \leftarrow \{t_1\}$; $W \leftarrow \{t_2\}$
Output:$\{t_1, t_2\}$

During the first invocation, the original set of cars is split into two halves, with respect to price. *The algorithm then recurs on the first half,* $\{t_1, t_2\}$, *in order to compute its p-skyline. The second invocation performs no work, except updating* $\mathcal{C}$ *and* $\mathcal{E}$: *attribute* $\mathbf{P}$ *is moved from* $\mathcal{C}$ *to* $\mathcal{E}$, *and attribute* $\mathbf{T}$ *enters* $\mathcal{C}$. *The third invocation computes the p-skyline of* $\{t_1, t_2\}$, *splitting w.r.t.* mileage *(no tuple is pruned). Back to the first invocation of* DcRec, *the procedure* PScreen *is used for removing from* $\{t_3, t_4\}$ *all tuples dominated by some element in* $\{t_1, t_2\}$. *Both* t_3 *and* t_4 *are pruned. Since no tuple survived the screening, the algorithm directly returns* $\{t_1, t_2\}$ *without making an additional recursive call.*

The complexity analysis for Dc is straightforward. If we denote by $T(n)$ its running time and we assume $|B| \simeq |W|$ at every iteration, the following upper bound holds for some fixed constant k_0

$$T(n) \leq k_0 n + 2 \cdot T\left(\frac{n}{2}\right) + F^*_{d-1}\left(\frac{n}{2}, \frac{n}{2}\right)$$

where the linear term $k_0 n$ models the time spent by the SplitByAttribute procedure; notice the p-screening operation at line 14 doesn't need to take attribute $\mathbf{A}$ into consideration, hence only $d-1$ attributes need to be taken into account. Since we assumed $F^*_d(b,w)$ is $\mathcal{O}\left((b+w) \cdot (\log b)^{d-2}\right)$, we can apply the master theorem [5, 15] and conclude that

$$T(n) \leq \mathcal{O}\left(n \cdot \log^{d-2} n\right)$$

We show now how to make Dc output-sensitive. Before moving to the algorithm, we introduce some basic complexity results for $C^*_d(v,n)$ when $v = 1$ and for $F^*_d(b,w)$ when $b = 1$.

Algorithm Osdc (output-sensitive divide & conquer)

Input: a p-expression π, a relation instance D_0
Output: the p-skyline $M_\pi(D_0)$

```
 1: procedure Osdc($\pi$, $D_0$)
 2:     return OsdcRec($\pi, D_0, \mathcal{R}oots_\pi, \emptyset$)

 3: procedure OsdcRec($\pi$, $D$, $\mathcal{C}$, $\mathcal{E}$)
 4:     if $\mathcal{C} = \emptyset$ or $|D| \leq 1$ then return $D$
 5:     select an attribute $\mathbf{A}$ from the candidates set $\mathcal{C}$
 6:     if all tuples in $D$ assign the same value to $\mathbf{A}$ then
 7:         $\mathcal{E}' \leftarrow \mathcal{E} \cup \{\mathbf{A}\}$
 8:         $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{\mathbf{A}\}$
 9:         $\mathcal{C}'' \leftarrow \mathcal{C}' \cup \{\mathbf{V} \in \mathcal{S}ucc_\pi(\mathbf{A}) : \mathcal{P}re_\pi(\mathbf{V}) \subseteq \mathcal{E}'\}$
10:         return OsdcRec($\pi, D, \mathcal{C}'', \mathcal{E}'$)
11:     else
12:         $(B, W, m^*) \leftarrow$ SplitByAttribute($D, \mathbf{A}$)
13:         $p^* \leftarrow$ PSkylineSP($\pi, B$)
14:         $B' \leftarrow$ PScreenSP($\pi, \{p^*\}, B \setminus \{p^*\}$)
15:         $W' \leftarrow$ PScreenSP($\pi, \{p^*\}, W$)
16:         $B'' \leftarrow$ OsdcRec($\pi, B', \mathcal{C}, \mathcal{E}$)
17:         $W'' \leftarrow$ PScreen($\pi, B'', W', \mathcal{C} \setminus \{\mathbf{A}\}, \mathcal{E}$)
18:         $W''' \leftarrow$ OsdcRec($\pi, W'', \mathcal{C}, \mathcal{E}$)
19:         return $\{p^*\} \cup B'' \cup W'''$
```

Lemma 1. *Given a relation instance* $D \subseteq U$ *and a p-expression* π, *locating a single, arbitrary element of* $M_\pi(D)$ *takes linear time.*

Proof. We can use an arbitrary weak order extension of $\succ_\pi$ and locate a maximal element $p^* \in D$ in linear time. It is easy to see p^* must belong to $M_\pi(D)$. From now on, we will denote this procedure as PSkylineSP(π, D). ☐

As a corollary to Lemma 1 we can state that $C^*_d(1,n) \leq \mathcal{O}(n)$. A similar result holds for p-screening:

Lemma 2. $F^*_d(1,w) \leq \mathcal{O}(w)$ *for any positive* d *and* w.

Proof. If B contains only one element, then we can easily perform p-screening in linear time: we only need one dominance test for each element of W. From now on, we will refer to this procedure as PScreenSP(π, B, W). ☐

Now that we have defined the two procedures PScreenSP and PSkylineSP, we can introduce our output-sensitive algorithm, Osdc (see the Figure above). The divide and conquer strategy is similar to the one in Dc, except for the look-ahead procedure at lines 13-15: at each recursion the algorithm spends linear time to extract a single p-skyline point p^* and prune from both B and W all tuples dominated by it. As a consequence, if at some point in the execution either B or W contains only a single p-skyline point, then either B' or W' will be empty, and the corresponding recursive call (line 16 or 18) will terminate immediately (line 4). We use Algorithm Osdc for proving the following theorem:

Theorem 1. $C^*_d(v,n)$ *is* $\mathcal{O}\left(n \cdot \log^{d-2} v\right)$.

Proof. If we run Osdc, the following upper bound holds on $C^*_d(v,n)$, for some fixed constant k_0

$$C^*_d(v,n) \leq k_0 n + C^*_d\left(v', \frac{n}{2}\right) + C^*_d\left(v'', \frac{n}{2}\right) + F^*_{d-1}\left(v', \frac{n}{2}\right)$$

where $v' + v'' + 1 = v$. The linear term $k_0 n$ models the time spent at lines 12-15, the second and the third terms the recursive calls at lines 16 and 18, while the last term is the p-screening operation at line 17. In order to keep track of the partitioning of v into smaller chunks during the recursion, we need to introduce some additional notational conventions. Let's denote by $v_{\ell,j}$ the size of the $j-th$ portion of v obtained at depth ℓ into the recursion. We can organize the different $v_{\ell,j}$ variables into a binary tree, as in Figure 2. Clearly, only

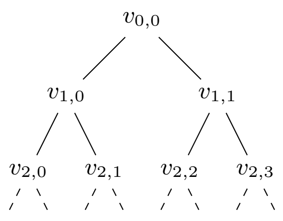


Figure 2

a finite subtree rooted in $v_{0,0}$ will cover the variables having a strictly positive value. The root $v_{0,0}$ is equal to v and, for all ℓ and j, $v_{\ell,j}$ is either $1 + v_{\ell+1,2j} + v_{\ell+1,2j+1}$ or zero. Notice there are exactly v assignments to (ℓ, j) such that $v_{\ell,j} > 0$, and the recursion proceeds up to a level $\ell_{\max}$ such that $v_{\ell_{max}+1,j} = 0$ for all j. Using this notation:

$$C_d^*(v,n) \le \sum_{\ell=0}^{\ell_{\max}} \sum_{\{j:v_{\ell,j}>0\}} k_0 \frac{n}{2^\ell} + F_{d-1}^*\left(v_{\ell+1,2j}, \frac{n}{2^{\ell+1}}\right)$$

We don't know $\ell_{\max}$ a priori; depending on the input it could be anywhere between $\log v$ and $\min(v, \log n)$. On the other hand, we do know the above sum has exactly v terms, and that their cost increases when the value of ℓ decreases. Therefore, in the worst-case scenario only the first $\log v$ levels of recursion are used, i.e. $\ell_{\max} = \log v$.

$$C_d^*(v,n) \le \sum_{\ell=0}^{\log v} \left[k_0 n + 2^\ell \cdot F_{d-1}^*\left(v_{\ell+1,2j}, \frac{n}{2^{\ell+1}}\right)\right]$$

Assuming $F_d^*(b,w)$ is $\mathcal{O}\left((b+w)\cdot(\log b)^{d-2}\right)$, there is a constant k_1 such that

$$C_d^*(v,n) \le k_0 n \log v + k_1 n \cdot \sum_{\ell=0}^{\log v - 1} \left(\log\left(v_{\ell+1,2j}\right)\right)^{d-3}$$

Since in the worst-case scenario only the first $\log v$ levels of recursion are used, we have that $v_{\ell,j} \le v/2^\ell \le n/2^\ell$, hence

$$C_d^*(v,n) \le k_0 n \log v + k_1 n \cdot \sum_{\ell=0}^{\log v - 1} \left(\log(v) - (\ell+1)\right)^{d-3}$$

If we set $h = \log(v) - (\ell+1)$, we obtain

$$C_d^*(v,n) \le k_0 n \log v + k_1 n \cdot \sum_{h=0}^{\log v - 1} h^{d-3}$$

We can conclude that $C_d^*(v,n) \le \mathcal{O}\left(n \cdot (\log v)^{d-2}\right)$ □

4. COMPLEXITY OF P-SCREENING

We can now analyze the complexity of p-screening. We start from the simple case $d \le 3$. When $d = 1$ the problem coincides with regular screening, and it is linear. When $d = 2$ or $d = 3$ the following lemmas apply

LEMMA 3. *$F_2^*(b,w) \le \mathcal{O}\left((b+w)\log b\right)$ for any $b > 1$ and $w > 0$.*

PROOF. A p-expression with only two attributes can be either a lexicographical order or a regular skyline. In the first case p-screening takes $\mathcal{O}(b+w)$ time, in the second it takes $\mathcal{O}((b+w)\log b)$ time, as per Proposition 4. □

LEMMA 4. *$F_3^*(b,w) \le \mathcal{O}\left((b+w)\log b\right)$ for any $b > 1$ and $w > 0$.*

PROOF. A p-expression π over three attributes can come in five possible forms:

Case 1 When $\pi = \mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \mathbf{A}_3$ p-screening consists of a regular screening in three dimensions: its complexity is $\mathcal{O}((b+w)\cdot\log b)$, as per Proposition 4.

Case 2 When $\pi = \mathbf{A}_1 \,\&\, \mathbf{A}_2 \,\&\, \mathbf{A}_3$, p-screening can be done in $\mathcal{O}(b+w)$ time, since π represents a lexicographical order. In $\mathcal{O}(b)$ time we can find a maximal element p^* in B, in additional $\mathcal{O}(w)$ time we can check for each $t \in W$ whether $p^* \succ_\pi t$ holds.

Case 3 When $\pi = \mathbf{A}_1 \,\&\, (\mathbf{A}_2 \otimes \mathbf{A}_3)$ we can proceed as follows: let a_1^* be the best value for attribute $\mathbf{A}_1$ amongst all tuples in B, let W_b, W_e and W_w contain all the tuples in W assigning to $\mathbf{A}_1$ a value respectively better, equal and worse than a_1^*; we have that

$$\begin{bmatrix} B \\ W \end{bmatrix}_\pi = W_b \cup \begin{bmatrix} M_{\mathbf{A}_1}(B) \\ W_e \end{bmatrix}_{\mathbf{A}_2 \otimes \mathbf{A}_3}$$

Since $M_{\mathbf{A}_1}(B)$, W_b, W_e and W_w can be computed in linear time, the overall complexity is dominated by the two-dimensional screening, that takes $\mathcal{O}((b+w)\log b)$, as per Proposition 4.

Case 4 When $\pi = (\mathbf{A}_1 \otimes \mathbf{A}_2) \,\&\, \mathbf{A}_3$ we can proceed as follows: first we compute the set

$$W' = \begin{bmatrix} B \\ W \end{bmatrix}_{\mathbf{A}_1 \otimes \mathbf{A}_2}$$

in $\mathcal{O}((b+w)\log b)$ time, then we sort B by the lexicographic order $\succ_{\mathbf{A}_1 \& \mathbf{A}_2 \& \mathbf{A}_3}$; given an assignment to $\mathbf{A}_1$ and $\mathbf{A}_2$ finding the best value for $\mathbf{A}_3$ in B takes only $\mathcal{O}(\log b)$, therefore pruning all the remaining dominated points from W' takes $\mathcal{O}(w \log b)$.

Case 5 When $\pi = (\mathbf{A}_1 \,\&\, \mathbf{A}_2) \otimes \mathbf{A}_3$ we can proceed as follows: let k be the number of distinct values for $\mathbf{A}_1$ in B, we can partition both B and W into k subsets, such that $B_k \not\succeq_{\mathbf{A}_1} B_{k-1} \not\succeq_{\mathbf{A}_1} \ldots \not\succeq_{\mathbf{A}_1} B_1$ and[4] $B_{i+1} \not\succeq_{\mathbf{A}_1} W_i \not\succeq_{\mathbf{A}_1} B_{i-1}$, for each i in $\{1,\ldots,k\}$; then, for each i and j in $\{1,\ldots,k\}$ such that $i \le j$ we perform the screening of W_j against B_i. When $i = j$ the screening is 2-dimensional, and takes $\mathcal{O}(|B_i \cup W_j| \log |B_i|)$ time; when $i < j$ the screening is 1-dimensional and takes $\mathcal{O}(|B_i \cup W_j|)$ time. All the 1-dimensional screenings take $\mathcal{O}(b+w)$ overall, while the 2-dimensional ones take $\mathcal{O}((b+w)\log b)$.

□

[4]Here we assume $B_0 = B_{k+1} = \emptyset$

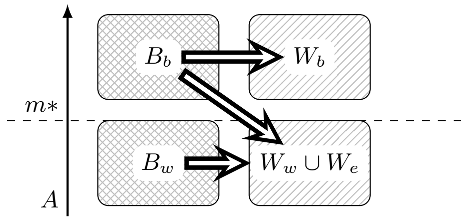


Figure 3: Dividing p-screening into simpler subproblems. Each box represents a set of tuples, each arrow a p-screening operation.

Algorithm PScreen (see the Figure on the right) shows how to perform p-screening in $\mathcal{O}\left((b+w)\log^{d-2} b\right)$ when $d \geq 3$. The algorithm is inspired by the one proposed by Kung, Luccio and Preparata [28]: in order to perform the p-screening $\left[\begin{smallmatrix}B\\W\end{smallmatrix}\right]_\pi$, we split B in two halves, namely B_b and B_w, ensuring that no tuple in B_w dominates or is indistinguishable from any tuple in B_b ($B_w \not\succeq_\pi B_b$). We can obtain B_b and B_w with the same strategy we used in Dc: we select an attribute $\mathbf{A}$, making sure that all tuples in *both* B *and* W agree w.r.t all the attributes in $\mathcal{A}\text{nc}_\pi(\mathbf{A})$. Then we find the median tuple m^* w.r.t. $\succ_\mathbf{A}$ and split B accordingly. Clearly $B_w \not\succeq_\pi B_b$ holds, as all tuples in B_b are better than all tuples in B_w w.r.t. $\succ_\mathbf{A}$, while the preference on $\mathbf{A}$ is not overridden by other higher-priority attributes. Then we proceed splitting W into three chunks: the set W_b of tuples being preferred to m^* (according to $\succ_\mathbf{A}$), those being indistinguishable from m^* (W_e) and those being dominated (W_w). Since we chose $\mathbf{A}$ so that all tuples in *both* B and W agree w.r.t. all attributes in $\mathcal{A}\text{nc}_\pi(\mathbf{A})$, we are guaranteed that no tuple in B_w can dominate (or be indistinguishable from) any tuple in W_b[5]. Hence, the problem of p-screening B and W is reduced to following three smaller sub-problems, as depicted in Figure 3:

$$\text{(i)} \begin{bmatrix}B_b\\W_b\end{bmatrix}_\pi \quad \text{(ii)} \begin{bmatrix}B_b\\W_w \cup W_e\end{bmatrix}_\pi \quad \text{(iii)} \begin{bmatrix}B_w\\W_w \cup W_e\end{bmatrix}_\pi$$

We can solve these three sub-problems by recursion. The recursion will stop when we run out of attributes ($d \leq 3$) or tuples ($|B| \leq 1$). Notice that for sub-problem (ii) we do not need to take attribute $\mathbf{A}$ into consideration, hence, for that recursion branch, we reduced the dimensionality by one unit. In order to choose attribute $\mathbf{A}$ properly, PScreen adopts the same strategy of Dc: it keeps track of two sets of attributes, $\mathcal{C}$ and $\mathcal{E}$, ensuring the following invariants:

- **I1** : If an attribute belongs to $\mathcal{E}$ then no pair of tuples in $B \cup W$ can disagree on the value assigned to it. That is, all tuples in $B \cup W$ are indistinguishable with respect to all attributes in $\mathcal{E}$.
- **I2** : An attribute in $\mathcal{A} \setminus \mathcal{E}$ belongs to $\mathcal{C}$ if and only if all its ancestors belong to $\mathcal{E}$.

We can now analyze the pseudo-code of PScreen in more detail. Lines 19-24 define the core logic of the recursion, lines 4-17 handle the base-cases, while lines 25-29 contain auxiliary methods. The algorithm takes as input two sets of tuples, B and W, such that $W \not\succeq_\pi B$. At each iteration the algorithm selects an attribute $\mathbf{A}$ from $\mathcal{C}$ (line 9), and tests whether the tuples in B are distinguishable w.r.t. $\mathbf{A}$. If they are, the block at lines 19-24 is executed, otherwise the one at lines 11-17.

Algorithm PScreen

```
Input:
    a p-expression $\pi$
    two relation instances, $B$ and $W$, s.t. $W \not\succeq_\pi B$
Output: the set $W'$ of all tuples in $W$ that are not dominated
    by any tuple in $B$

 1: procedure PScreen($\pi$, $B$, $W$)
 2:     return PScreenRec($\pi, B, W, \mathcal{R}\text{oots}_\pi, \emptyset$)

 3: procedure PScreenRec($\pi$, $B$, $W$, $\mathcal{C}$, $\mathcal{E}$)
 4:     if $\mathcal{C}$ is empty then return $\emptyset$
 5:     else if $|B| = 1$ then
 6:         return PScreenSP($\pi, B, W$)
 7:     else if $|\mathcal{C} \cup \mathcal{D}\text{esc}_\pi(\mathcal{C})| \leq 3$ then
 8:         apply Lemma 4 and return
 9:     select an attribute $\mathbf{A}$ from the candidates set $\mathcal{C}$
10:     if all tuples in $B$ assign the value $a$ to $\mathbf{A}$ then
11:         $(W_b, W_e, W_w) \leftarrow$ SplitByValue($W, \mathbf{A}, a$)
12:         $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{\mathbf{A}\}$
13:         $W'_w \leftarrow$ PScreenRec($\pi, B, W_w, \mathcal{C}', \mathcal{E}$)
14:         $\mathcal{E}' \leftarrow \mathcal{E} \cup \{\mathbf{A}\}$
15:         $\mathcal{C}'' \leftarrow \mathcal{C}' \cup \{\mathbf{V} \in \mathcal{S}\text{ucc}_\pi(\mathbf{A}) : \mathcal{P}\text{re}_\pi(\mathbf{V}) \subseteq \mathcal{E}'\}$
16:         $W'_e \leftarrow$ PScreenRec($\pi, B, W_e, \mathcal{C}'', \mathcal{E}'$)
17:         return $W'_w \cup W'_e \cup W_b$
18:     else
19:         $(B_b, B_w, m^*) \leftarrow$ SplitByAttribute($B, \mathbf{A}$)
20:         $(W_b, W_e, W_w) \leftarrow$ SplitByValue($W, \mathbf{A}, m^*$)
21:         $W'_b \leftarrow$ PScreenRec($\pi, B_b, W_b, \mathcal{C}, \mathcal{E}$)
22:         $W'_w \leftarrow$ PScreenRec($\pi, B_w, W_w \cup W_e, \mathcal{C}, \mathcal{E}$)
23:         $W''_w \leftarrow$ PScreenRec($\pi, B_b, W'_w, \mathcal{C} \setminus \{\mathbf{A}\}, \mathcal{E}$)
24:         return $W'_b \cup W''_w$

25: procedure SplitByValue(D, $\mathbf{A}$, $m^*$)
26:     Compute the set $D_b = \{t \in D \mid t[\mathbf{A}] \succ_\mathbf{A} m^*\}$
27:     Compute the set $D_e = \{t \in D \mid t[\mathbf{A}] \approx_\mathbf{A} m^*\}$
28:     Compute the set $D_w = \{t \in D \mid m^* \succ_\mathbf{A} t[\mathbf{A}]\}$
29:     return $(D_b, D_e, D_w)$
```

Let's analyze the first case. At line 19 B is split into B_b and B_w, at line 20 W is split into W_b, W_e and W_w. At lines 21, 22 and 23 the algorithm recurs three times, in order to solve the three sub-problems discussed above. We can now analyze the base-cases: when $\mathcal{C}$ is left empty (line 4), all tuples in W are dominated by all tuples in B, so the algorithm returns the empty set; when B contains only one element, the algorithm applies the procedure from Lemma 2 (lines 5-6); when only three attributes are left to be processed, the algorithm applies Lemma 4. When it is not possible to split B, since all its tuples agree on some value a for $\mathbf{A}$, the algorithm splits W into three sets (W_b, W_e and W_w) containing tuples that are respectively better, indistinguishable from and worse than a, w.r.t. $\succ_\mathbf{A}$. Then it proceeds to solve $\left[\begin{smallmatrix}B\\W_w\end{smallmatrix}\right]_\pi$ and $\left[\begin{smallmatrix}B\\W_e\end{smallmatrix}\right]_\pi$. Notice $\mathcal{C}$ and $\mathcal{E}$ are updated at lines 12, 14 and 15 in order to satisfy the invariants I1 and I2.

Theorem 2. *$F^*_d(b+w) \leq \mathcal{O}\left((b+w)\log^{d-2} b\right)$ for any $d > 3$, b and w.*

Proof. We show that the above upper bound holds if we apply Algorithm PScreen to perform p-screening. The proof is by induction on d, and will use the simplifying assumptions that (i) $b = 2^m$ for some positive m and (ii) the splitting operation at line 19 divides B into two almost equally sized halves. Since we proved the base case $d = 3$ in

[5]Notice that for every pair of tuples (t', t) in $B_w \times W_b$ attribute $\mathbf{A}$ belongs to both $\mathcal{T}op_\pi(t', t)$ and $\mathcal{B}\text{etter}_\pi(t, t')$. From Proposition 1 we can conclude that $B_w \not\succeq_\pi W_b$ holds.

Lemma 4, the rest of the proof has a similar structure to the one proposed in [27, 28]. During each recursion Algorithm PScreen splits the set W in two parts; using a notation similar to the one presented in Figure 2, from now on we will denote by $w_{\ell,j}$ the size of the j-th portion of W obtained at depth ℓ in the recursion. Clearly $w_{0,0} = |W|$, and for each ℓ in $\{0, 1, \ldots, \log b - 1\}$ the sum $\sum_{j=1}^{2^\ell} w_{\ell,j}$ is equal to $|W|$. It is easy to see the following upper bound holds on F_d^*, for some fixed constant k_0

$$\begin{aligned} F_d^*(b, w_{0,0}) &\leq k_0(b + w_{0,0}) + F_{d-1}^*(b, w_{0,0}) \\ &+ F_d^*\left(\frac{b}{2}, w_{1,0}\right) + F_d^*\left(\frac{b}{2}, w_{1,1}\right) \end{aligned} \quad (3)$$

the linear term $k_0(b+w_{0,0})$ models the time spent for finding the median m^* and split B and W accordingly (lines 19 and 20), and the other three terms model the recursive calls at lines 21,22, and 23. Inequality 3 holds during each recursion, therefore

$$\begin{aligned} F_d^*\left(\frac{b}{2^\ell}, w_{\ell,j}\right) &\leq k_0\left(\frac{b}{2^\ell} + w_{\ell,j}\right) + F_{d-1}^*\left(\frac{b}{2^\ell}, w_{\ell,j}\right) \\ &+ F_d^*\left(\frac{b}{2^{\ell+1}}, w_{\ell+1,2j}\right) \\ &+ F_d^*\left(\frac{b}{2^{\ell+1}}, w_{\ell+1,2j+1}\right) \end{aligned} \quad (4)$$

We can apply (4) to the last two terms of (3) and repeat the operation until we obtain the following

$$\begin{aligned} F_d^*(b, w) &\leq \sum_{\hat{\ell}=0}^{\log b-1} \sum_{k=1}^{2^{\hat{\ell}}} k_0\left(\frac{b}{2^{\hat{\ell}}} + w_{\hat{\ell},k}\right) \\ &+ \sum_{\hat{\ell}=0}^{\log b-1} \sum_{k=1}^{2^{\hat{\ell}}} F_{d-1}^*\left(\frac{b}{2^{\hat{\ell}}}, w_{\hat{\ell},k}\right) \\ &+ \sum_{k=0}^{2^{\log b-1}} F_d^*(1, w_{\log b-1,k}) \end{aligned}$$

By inductive hypothesis, we can assume

$$F_{d-1}^*(b, w) \leq \mathcal{O}\left((b + w)(\log b)^{d-3}\right)$$

Also, from Lemma 2 we know $F_d^*(1, w)$ is $\mathcal{O}(w)$. Therefore, there exist constants k_0, k_1 and k_2 such that, for every b and w, the following holds

$$\begin{aligned} F_d^*(b, w) &\leq k_0 \cdot (b + w) \log b \\ &+ k_1 \cdot \sum_{\hat{\ell}=0}^{\log b-1} \sum_{j=1}^{2^{\hat{\ell}}} \left(\frac{b}{2^\ell} + w_{\hat{\ell},j}\right) \cdot \left(\log \frac{b}{2^\ell}\right)^{d-3} \\ &+ k_2 \cdot w \end{aligned}$$

Solving the sum over j, we get

$$\begin{aligned} F_d^*(b, w) &\leq k_0 \cdot (b + w) \log b \\ &+ k_1 \cdot (b + w) \cdot \sum_{\hat{\ell}=0}^{\log b-1} \left(\log b - \hat{\ell}\right)^{d-3} + k_2 \cdot w \end{aligned}$$

If we set $h = \log b - \hat{\ell}$, we obtain

$$\begin{aligned} F_d^*(b, w) &\leq k_0 \cdot (b + w) \log b \\ &+ k_1 \cdot (b + w) \cdot \sum_{h=1}^{\log b} h^{d-3} + k_2 \cdot w \end{aligned}$$

Since $\sum_{h=1}^{\log b} h^{d-3}$ is $\mathcal{O}\left((\log b)^{d-2}\right)$, it follows that

$$F_d^*(b, w) \leq \mathcal{O}\left((b + w) \cdot (\log b)^{d-2}\right)$$

This concludes our proof. □

5. AVERAGE-CASE ANALYSIS

The average-case performance for regular skyline algorithms has been studied extensively in several papers [20, 4, 6]. The most usual assumption is that the data is distributed so that each attribute is independent from the others (*statistical independence*) and the probability of two tuples agreeing on the same value for the same attribute is negligible (*sparseness*). These two assumptions are usually called, collectively, *component independence* (or CI, as in [6]). Under the CI assumption Less has been shown to be $\mathcal{O}(n)$ in the average-case, while Sfs resulted to be $\mathcal{O}(n \log n)$ [20]. In this section we show how to modify Osdc, in order to ensure a linear average-case complexity. We start by making two key observations:

Observation 1 Over all the preference relations that can be expressed using p-expressions, the skyline relation $\succ_{\text{sky}}$ represents a worst-case scenario for an output-sensitive algorithm like Osdc. This follows directly from the containment property (Proposition 2): it is easy to see that for every $\succ_\pi$ we have that $M_\pi(D) \subseteq M_{\text{sky}}(D)$ holds as long as $\mathcal{V}ar(\pi) = \mathcal{V}ar(sky)$.

Observation 2 Buchta [8] proved that, under the CI assumption, the expected size of $M_{\text{sky}}(D)$ is $H_{d-1,n}$, the $(d-1)$-th order harmonic of n. Godfrey [19] observed that, if we drop the assumption of sparseness, the size of $M_{\text{sky}}(D)$ is likely to *decrease*.

Hence, computing regular skylines over data sets respecting the CI assumption is a corner-case scenario for an output-sensitive p-skyline algorithm, in the sense that introducing priorities between attributes or dropping the sparseness assumption would only improve the algorithm's performance. In order to show that our procedure is well-behaved in the average-case, we will prove it is well-behaved under the assumptions discussed above. More specifically, we will show how to make a simple modification to Osdc and ensure an average performance of $\mathcal{O}(n)$ for skyline queries under the CI assumption. The general idea is to follow a two-stage approach, similar to the one proposed in [4]:

1. During the first phase a linear scan prunes all the points dominated by a virtual tuple t^*, that is chosen so that the probability that no point dominates it is less than $1/n$, and the average number of points not dominated by it is $o(n)$. Such t^* can be chosen using the strategy presented in [4].
2. With probability $(n-1)/n$ the virtual tuple t^* is dominated by a real tuple in D, and so the algorithm can compute the final answer by running Osdc only on the $o(n)$ points that survived the initial linear scan.

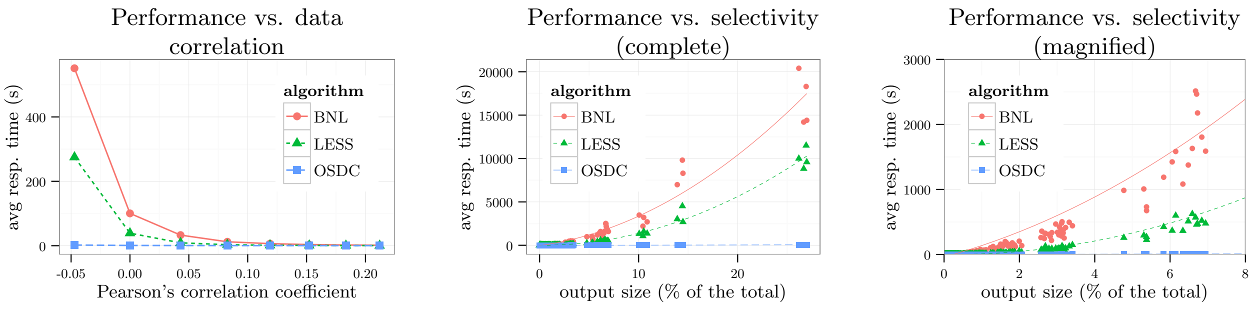


Figure 4: The effect of data correlation and query selectivity on performance (synthetic data sets)

3. With probability $1/n$ the final answer needs to be computed by running OSDC over the whole data set D.

It is easy to see that the amortized, average cost of this procedure is $\mathcal{O}(n)$, while the worst-case complexity is still $\mathcal{O}\left(n \log^{d-2} v\right)$.

6. P-SKYLINES IN EXTERNAL MEMORY

In the past scan-based skyline algorithms like BNL, SFS, SaLSa or LESS have generated a considerable interest in the database community. While all these algorithms exhibit a sub-optimal worst-case performance of $\mathcal{O}(n^2)$, they are known to be well-behaved in the average-case scenario, and, for larger values of d, to be even more practical than divide-and-conquer algorithms like [6] (we refer the reader to [20] for an exhaustive discussion of the topic). Additionally, scan-based algorithms are easy to embed into existing database systems, as they are easily pipelinable, and support execution in external memory (where they still exhibit a sub-optimal, quadratic worst-case performance, as discussed in [32]). In this section we show how to adapt two scan-based skyline algorithms, SFS and LESS, in order to support p-skyline queries. Our goal is twofold: on one hand we want to show that OSDC is faster than the scan-based solutions, a part for its nice asymptotic properties (this will be done in the Section 7); on the other hand we want to develop a p-skyline algorithm that supports execution in external memory.

Both SFS and LESS sort the input dataset so that no tuple can dominate another preceding it; to achieve a similar result with prioritized preferences, we propose to presort the input w.r.t. the following weak order extension of $\succ_\pi$

$$\succ_\pi^{\text{ext}} = \succ_{\text{sum}_0} \ \& \ \succ_{\text{sum}_1} \ \& \ \ldots \ \& \ \succ_{\text{sum}_{d-1}} \qquad (5)$$

Each $\succ_{\text{sum}_i}$ is defined as follows:

$$t' \succ_{\text{sum}_i} t \iff \sum_{\mathbf{A} \in \mathcal{V}ar(\pi): d_{\mathbf{A}}=i} t'[\mathbf{A}] < \sum_{\mathbf{A} \in \mathcal{V}ar(\pi): d_{\mathbf{A}}=i} t[\mathbf{A}]$$

That is, $t' \succ_{\text{sum}_i} t$ holds when the sum over all attributes at depth i computed for t' is lower than the same sum computed for t. After the sorting step, if $t' \succ_\pi^{\text{ext}} t$ holds tuple t' is going to be processed before tuple t.

THEOREM 3. *The relation $\succ_\pi^{ext}$ defined above is a weak order extension of $\succ_\pi$.*

PROOF. First we want to show that for each pair of tuples (t', t) in U^2, $t' \succ_\pi^{\text{ext}} t$ implies $t \not\succ_\pi t'$. We can denote by i^* the smallest index i such that $(t' \succ_{\text{sum}_{i^*}} t)$. Since the sum over all attributes at depth i^* is smaller for t' rather than for t, there must be at least one attribute $\mathbf{A}$ at depth i^* favoring t' over t. It is easy to see that such $\mathbf{A}$ belongs to $\mathcal{T}op_\pi(t', t)$; from Proposition 1, point 2, we can conclude that $t \not\succ_\pi t'$. We are left to prove that $\succ_\pi^{\text{ext}}$ is a weak order: this follows directly from (5), noting that each $\succ_{\text{sum}_i}$ is a weak order, and the prioritized composition of weak orders is a weak order itself. $\square$

7. EXPERIMENTAL RESULTS

To the best of our knowledge there is no published work on measuring p-skyline queries performance. The problem is difficult, since the response time depends on many factors, including the topology of the p-graph and data properties like size, correlation and likelihood of duplicated values.

In the following sections we try to address this issue by proposing a novel p-skyline testing framework. First we show how to sample random p-expressions from a uniform distribution, and how to generate meaningful synthetic data sets. Later we present our experimental results from both real and synthetic data sets.

7.1 Sampling random p-expressions

P-expressions can encode a wide variety of preferences: they can represent lexicographic orders, classical skylines, or any combination of the two. In order to keep evaluations fair and unbiased we should not polarize benchmarks on specific preferences. Instead, our goal is to randomly sample p-expressions from a uniform distribution, ensuring all preferences are equally represented.

Given the number of attributes d, sampling a random p-expression means building a random p-graph over d vertices, ensuring that all legal p-graphs have the same probability of being generated. We present a result from [29] to characterize the set of p-graphs we want to sample from.

THEOREM 4. *[29] Given a set of d attributes $\mathcal{A}$, a graph Γ over $\mathcal{A}$ is a p-graph if and only if:*

1. *Γ is transitive and irreflexive.*
2. *Γ respects the* envelope *property:* $\forall \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$ *all different in $\mathcal{A}$,* $(\mathbf{A}_1, \mathbf{A}_2) \in \Gamma \wedge (\mathbf{A}_3, \mathbf{A}_4) \in \Gamma \wedge (\mathbf{A}_3, \mathbf{A}_2) \in \Gamma \Rightarrow (\mathbf{A}_3, \mathbf{A}_1) \in \Gamma \vee (\mathbf{A}_1, \mathbf{A}_4) \in \Gamma \vee (\mathbf{A}_4, \mathbf{A}_2) \in \Gamma$.

Iterating over all graphs that respect the above constraints is practical only for small values of d. For larger values we

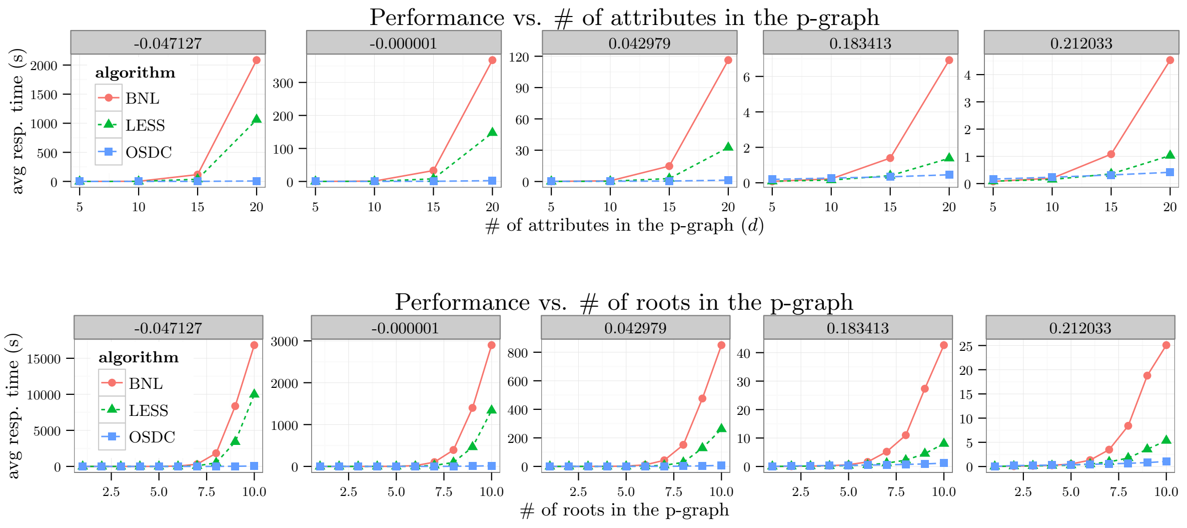


Figure 5: The effect of the p-graph's topology on performance (synthetic data sets)

use the following strategy: we convert the constraints into a boolean satisfaction problem and sample from its solutions near-uniformly using SampleSAT [35]. SampleSAT performs a random walk over the possible solutions of a SAT problem, alternating greedy WalkSAT moves with simulated annealing steps. The ratio of the frequencies of the two kinds of steps, denoted by f, determines the trade-off between the uniformity of the sampling and the time spent to obtain it. For our tests on synthetic data we used $f = 0.5$ for generating 200 p-expressions, with d ranging from 5 to 20 attributes.

7.2 Synthetic data sets

Several papers, starting from [7], showed how data correlation affects the performance of skyline queries. We wanted to test whether similar considerations apply to p-skylines. Notice the role of correlation in this context is subtle: depending whether two variables have the same priority or not, a correlation between them may have different effects. For this reason we decided to generate synthetic data sets where each pair of dimensions exhibits approximatively the same linear correlation. Let's denote by $\vec{\mathbf{1}}$ a d-dimensional all-ones vector $(1, 1, \ldots, 1)$, and by M a $d \times d$ matrix whose rows form an orthonormal basis for $\mathbb{R}^d$, the first one being parallel with $\vec{\mathbf{1}}$. Notice that M represents a rotation centered on the origin. Let M_D be a $d \times d$ diagonal matrix, having $(\alpha, 1, \ldots, 1)$ as its main diagonal. We propose to test p-skyline algorithms over a multivariate Gaussian distribution $\mathcal{N}_\alpha$, centered on the origin, with covariance matrix $\Sigma_\alpha = M \times M_D \times M^{-1}$. According to this distribution each pair of distinct dimensions exhibits the same correlation, determined by the parameter α. It is important to notice that $\mathcal{N}_\alpha$, amongst all the distributions where all pairs of variables have the same correlation, is the one with *maximum entropy*, given the parameter α.

For our tests we sampled several data sets, varying the value of α; each set contains one million tuples over $d = 20$ attributes. Since p-skylines make sense only when some tuples agree on some attributes, we rounded the data off to four decimal digits of precision, in order to ensure the presence of duplicated values. As a result, the uncorrelated data sets (those with $\alpha = 1$) have approximatively 7,000 distinct values in each column. We compared the performance of OSDC against BNL and LESS. To keep the comparison fair we implemented an in-memory version of BNL, setting the size of the window to be large enough to store the whole input. This way, the algorithm could answer each query with a single iteration. We adapted LESS using the strategy discussed in Section 6. We ran it using several different thresholds on the size of the elimination filter, ranging between 50 and 10,000 tuples. For each experiment we report only the fastest response times. In order to avoid any overhead, we precomputed the ranking $\succ_\pi^{\text{ext}}$. All the algorithms were implemented using Java, and tested on an Intel Core i7-2600 (3.4 GHz) machine equipped with 8 GB of RAM. We ran all the experiments using the Java Runtime Environment version 1.7.0, limiting the maximum heap size to 4 GB.

On the average we observed OSDC to be significantly faster than LESS and BNL. Here we analyze these results in relation with data correlation, and we study how the topology of p-expressions affects the performance of each algorithm. Figure 4 (left) focuses on the effect of data correlation. We average the response time over all queries, and plot it against the observed Pearson's correlation coefficient[6]. The Figure shows that LESS and BNL compete with OSDC in presence of positive data correlation, but their performance decreases quickly on anti-correlated data. OSDC, on the other hand, remains mostly unaffected by data correlation.

In Figure 5 we investigate the relation between performance and the topology of p-graphs. We group queries according to the number of attributes (top) and roots (bottom) in their p-graphs, and we aggregate response times w.r.t. data correlation. For lack of space we report only five levels of correlation, the most significant ones. Independently from data correlation, OSDC exhibits a distinct performance advantage on queries with more than 10 attributes,

[6]The correlation coefficient was measured after rounding the data sets.

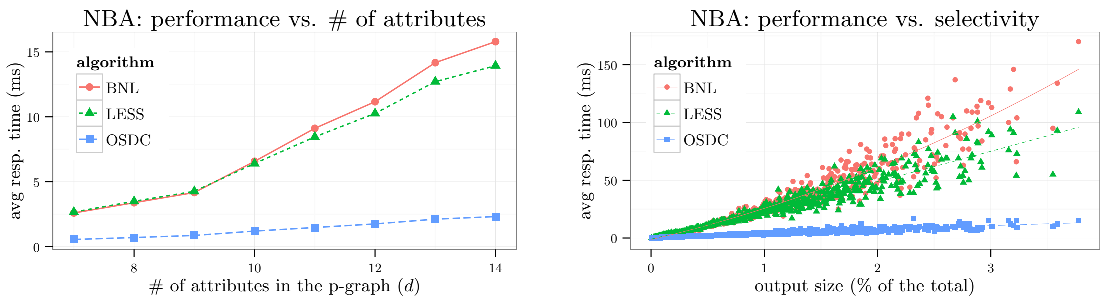


Figure 6: NBA data set (21,959 tuples over 14 attributes)

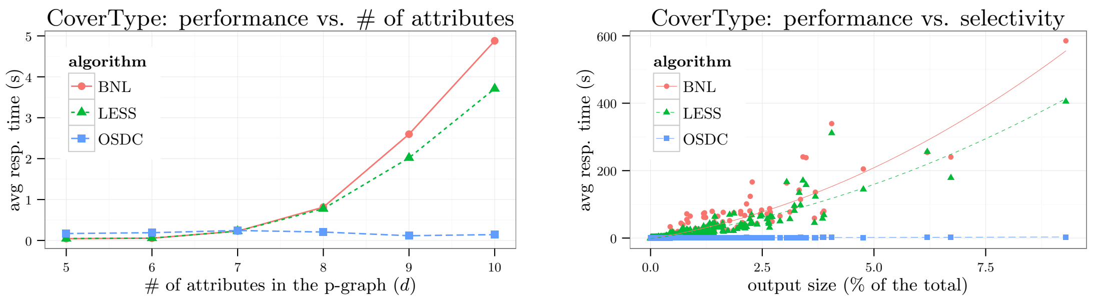


Figure 7: CoverType data set (581,012 tuples over 10 attributes).

especially if there are more than five roots; Less shows a similar advantage in presence of positive data correlation, while Bnl results are competitive mostly on queries with less than five roots. During our experiments we observed that both p-graph topology and data correlation have a direct influence on the size of the output: highly-prioritized p-expressions (those with few roots) are likely to produce smaller p-skylines; similarly, positively correlated data is likely to produce smaller result-sets. Therefore, we summarize our results by plotting the average response time against the size of the output (Figure 4, on the right). As expected, Osdc and Less show a clear advantage for large result-sets while Bnl remains competitive only for queries returning few tuples. The lines on the graph represent second-order polynomial regressions.

7.3 Real data sets

We tested our algorithms over the following real, publicly available data sets:

NBA NBA[7] is a very popular data set for evaluating skyline algorithms. We used the following regular season statistics: *gp*, *minutes*, *pts*, *reb*, *asts*, *stl*, *blk*, *turnover*, *pf*, *fga*, *fta*, *tpa*, *weight*, *height*. After dropping null values, the data set contains 21,959 tuples. We generated 8,000 random p-expressions with d ranging from 7 to 14. For this data set we used the assumption that larger values are preferred.

CoverType Forest Covertype[8] contains a collection of cartographic observations performed by the US Forest Service and the US Geological Survey. We extracted a data set of 581,012 tuples over 10 attributes. We generated 6,000 random p-expressions with d ranging from 5 to 10. For this data set we used the assumption that smaller values are preferred.

Our results are presented in Figures 6 and 7. In the graphs on the left response times are aggregated by the number d of attributes in each p-expression. In the plots on the right response times are put in relation with the size of the output. On both data sets our findings confirmed our average-case analysis and the results we obtained from synthetic data: Osdc outperforms Less and Bnl, especially when the output-size is over 1% of the input-size.

8. CONCLUSIONS AND FUTURE WORK

In this paper we generalized the results of [25] to the context of p-skylines. We proved that p-skylines can be computed in $\mathcal{O}\left(n \log^{d-2} v\right)$ in the worst-case scenario, and in $\mathcal{O}(n)$ in the average-case. Additionally, we proposed a novel framework for benchmarking p-skyline queries, showing how to sample p-expressions uniformly, with the purpose of running unbiased tests. We designed our divide-and-conquer strategy assuming the input data always fits in the main memory; it would be interesting to verify whether we can drop this assumption, and develop an output-sensitive algorithm that runs efficiently in external memory, taking inspiration from [32, 21]. Another interesting aspect to investigate concerns the estimation of the expected size of the output. Can we exploit the semantics of p-skylines for predicting the expected output-size of a query? This would be helpful for choosing the most convenient algorithm for answering it, on a case-by-case basis. We leave the answers to all these questions open for future work.

[7]www.databasebasketball.com
[8]archive.ics.uci.edu/ml/datasets/Covertype

9. REFERENCES

[1] P. Afshani. Fast computation of output-sensitive maxima in a word RAM. In C. Chekuri, editor, *SODA*, pages 1414–1423. SIAM, 2014.

[2] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. *ACM Trans. Database Syst.*, 33(4):31:1–31:49, Dec. 2008.

[3] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[4] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In D. S. Johnson, editor, *SODA*, pages 179–187. SIAM, 1990.

[5] J. L. Bentley, D. Haken, and J. B. Saxe. A general method for solving divide-and-conquer recurrences. *SIGACT News*, 12(3):36–44, Sept. 1980.

[6] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, 2001.

[8] C. Buchta. On the average number of maxima in a set of vectors. *Inf. Process. Lett.*, 33(2):63–65, 1989.

[9] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[10] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 1–10, New York, NY, USA, 2011. ACM.

[11] J. Chomicki. Querying with intrinsic preferences. In C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Böhm, and M. Jarke, editors, *EDBT*, volume 2287 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2002.

[12] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, Dec. 2003.

[13] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline queries, front and back. *SIGMOD Record*, 42(3):6–18, 2013.

[14] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[16] S. A. Drakopoulos. Hierarchical choice in economics. *Journal of Economic Surveys*, 8(2):133–153, 1994.

[17] P. C. Fishburn. Axioms for lexicographic preferences. *The Review of Economic Studies*, pages 415–419, 1975.

[18] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In R. A. DeMillo, editor, *STOC*, pages 135–143. ACM, 1984.

[19] P. Godfrey. Skyline cardinality for relational processing. In D. Seipel and J. M. T. Torres, editors, *FoIKS*, volume 2942 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2004.

[20] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors, *VLDB*, pages 229–240. ACM, 2005.

[21] X. Hu, C. Sheng, Y. Tao, Y. Yang, and S. Zhou. Output-sensitive skyline algorithms in external memory. In S. Khanna, editor, *SODA*, pages 887–900. SIAM, 2013.

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

[23] W. Kießling, B. Hafenrichter, S. Fischer, and S. Holland. Preference XPATH: A query language for e-commerce. In H. U. Buhl, A. Huther, and B. Reitwiesner, editors, *Wirtschaftsinformatik*, page 32. Physica Verlag / Springer, 2001.

[24] W. Kießling and G. Köstler. Preference SQL - design, implementation, experiences. In *VLDB*, pages 990–1001. Morgan Kaufmann, 2002.

[25] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Symposium on Computational Geometry*, pages 89–96, 1985.

[26] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986.

[27] H. T. Kung. On the computational complexity of finding the maxima of a set of vectors. In *SWAT (FOCS)*, pages 117–121, 1974.

[28] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[29] D. Mindolin and J. Chomicki. Preference elicitation in prioritized skyline queries. In *VLDB J.*, pages 157–182, 2011.

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

[31] A. Scott. Identifying and analysing dominant preferences in discrete choice experiments: an application in health care. *Journal of Economic Psychology*, 23(3):383–398, 2002.

[32] C. Sheng and Y. Tao. Worst-case i/o-efficient skyline algorithms. *ACM Transactions on Database Systems (TODS)*, 37(4):26, 2012.

[33] W. Siberski, J. Z. Pan, and U. Thaden. Querying the semantic web with preferences. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 612–624. Springer, 2006.

[34] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19:1–19:45, Aug. 2011.

[35] W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 670–676. AAAI Press / The MIT Press, 2004.