Reconciling Multiple Categorical Preferences with Double Pareto-based Aggregation

Nikos Bikakis[1,2]*, Karim Benouaret[3], and Dimitris Sacharidis[2]

[1]National Technical University of Athens, Greece
[2]IMIS, "Athena" Research Center, Greece [3]Inria Nancy – Grand Est, France

Abstract. Given a set of objects and a set of user preferences, both defined over a set of categorical attributes, the *Multiple Categorical Preferences* (MCP) problem is to determine the objects that are considered preferable by all users. In a naïve interpretation of MCP, matching degrees between objects and users are aggregated into a single score which ranks objects. Such an approach, though, obscures and blurs individual preferences, and can be unfair, favoring users with precise preferences and objects with detailed descriptions. Instead, we propose an objective and fair interpretation of the MCP problem, based on two Pareto-based aggregations. We introduce an efficient approach that is based on a transformation of the categorical attribute values and an index structure. Moreover, we propose an extension for controlling the number of returned objects. An experimental study on real and synthetic data finds that our index-based technique is an order of magnitude faster than a baseline approach, scaling up to millions of objects.

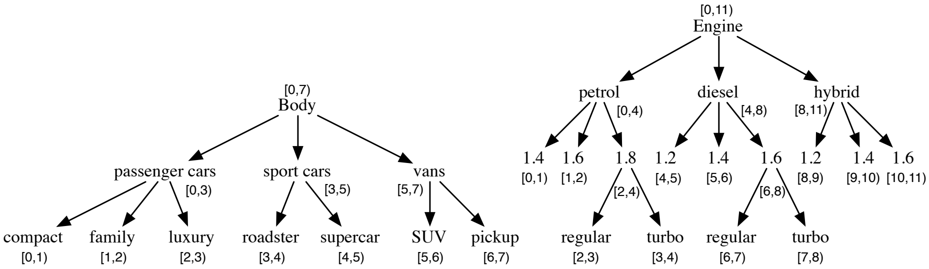
Keywords: Group preferences, Group recommendation, Categorical attributes, Rank aggregation, Skyline queries, Collective dominance.

1 Introduction

Given a collection of *objects* and a user's *preference*, both defined on a set of *attributes*, the *general recommendation problem* is to identify those objects that are most aligned to the user's preference. Several instances of this generic problem have appeared over the past few years in the Information Retrieval and Database communities; e.g., see [7,26]. This paper deals with an instance of the above class, termed the *Multiple Categorical Preferences* (MCP) problem. MCP has three characteristics. (1) Objects are described by a set of categorical attributes. (2) User preferences are defined on a subset of the attributes. (3) There are multiple users with distinct, possibly conflicting, preferences. The MCP problem may appear in several scenarios; for instance, colleagues arranging for a dinner at a restaurant, friends selecting a vacation plan for a holiday break.

To illustrate MCP, consider the following example. Assume that a three-member family is looking to buy a new car. Assume a list of available cars, where each is characterized by two categorical attributes, Body and Engine. Figure 1 depicts the hierarchies for these two attributes; Body is a three-level, and Engine is a four-level hierarchy. Table 1 shows the attribute values of four cars, and the family members' preferences. For

* This work is partially supported by the EU/Greece funded KRIPIS: MEDA Project & the FP7 project DIACHRON



(a) Body attribute (b) Engine attribute

Fig. 1. Attribute hierarchies

Table 1. Objects, Users & Matching vectors

Car	**Body**	**Engine**
o_1	family	hybrid 1.4
o_2	roadster	petrol 1.8 turbo
o_3	SUV	diesel 1.6
o_4	compact	petrol 1.4

(a) Objects

User	**Preference**
u_1	{passenger cars, petrol}
u_2	{sport cars}
u_3	{petrol 1.8}

(b) Users

	User		
Car	u_1	u_2	u_3
o_1	$\langle 1/3, 0\rangle$	$\langle 0, 0\rangle$	$\langle 0, 0\rangle$
o_2	$\langle 0, 1/4\rangle$	$\langle 1/2, 0\rangle$	$\langle 0, 1/2\rangle$
o_3	$\langle 0, 0\rangle$	$\langle 0, 0\rangle$	$\langle 0, 0\rangle$
o_4	$\langle 1/3, 1/4\rangle$	$\langle 0, 0\rangle$	$\langle 0, 0\rangle$

(c) Matching vectors

instance, member u_1 prefers passenger cars with petrol engines, while u_2 likes sport cars but states no preference on the engine type.

Observe that if we look at a particular family member, it is straightforward to determine her/his ideal car based on existing methods. For instance, u_1 clearly prefers o_4, which is a passenger car with a petrol engine, while u_2 clearly favors o_2, which is a sport car. These conclusions are reached using the following reasoning. Each preference attribute value $u_j.A_k$ is *matched* with the corresponding object attribute value $o_i.A_k$ using, e.g., the Jaccard coefficient $\frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k \cup u_j.A_k|}$, and a matching degree per preference attribute is derived. Given these degrees, the next step is to "compose" them into an overall matching degree between a user u_j and an object o_i. Note that several techniques are proposed for "composing" matching degrees; e.g., see [16,26]. The simplest option is to compute a linear combination, e.g., the sum, of the individual degrees. Returning to our example, the matching degrees of user u_1 are: $\langle 1/3, 0\rangle$ for car o_1, $\langle 0, 1/4\rangle$ for car o_2, $\langle 0, 0\rangle$ for car o_3, and $\langle 1/3, 1/4\rangle$ for car o_4. Note that independently of the "composition" method employed, o_4 is the most favorable car for user u_1. Using similar reasoning, car o_2, is ideal for both users u_2, u_3.

When all users are taken into consideration, as required by the MCP problem, several questions arise. Which is the best car that satisfies the entire family? And more importantly, *what does it mean to be the best car*? A simple answer to the latter, would be the car that has the highest "composite" degree of match to all users. Using a similar method as before, one can define a collective matching degree that "composes" the overall matching degrees for each user. This interpretation, however, enforces an additional level of "composition", the first being across attributes, and the second across users. These compositions obscure and blur the individual preferences per attribute of each user.

To some extent, the problem at the first "composition" level can be mitigated by requiring each user to manually define an importance weight among her/his specified attribute preferences. On the other hand, it is not easy, if possible at all, to assign weights to users, so that the assignment is fair. There are two reasons for this. First, *users may specify different sets of preference attributes*, e.g., u_1 specifies Body and Engine, while u_2 only Body. Second, even when considering a particular preference attribute, e.g., Engine, *users may specify values at different levels of the hierarchy*, e.g., u_1 specifies a petrol Engine, while u_3 a petrol 1.8 Engine, which is one level below. Similarly, objects can also have attribute values defined at different levels. Therefore, any "composition" is bound to be *unfair*, as it may favor users with specific preferences and objects with detailed descriptions, and disfavor users with broader preferences and objects with coarser descriptions. This is an inherent difficulty of the MCP problem.

In this work, we introduce the *double Pareto-based aggregation*, which provides an objective and fair interpretation to the MCP problem without "compositing" across preference attributes and users. Under this concept, the matching between a user and an object forms a *matching vector*. Each coordinate of this vector corresponds to an attribute and takes the value of the corresponding matching degree. The first Pareto-based aggregation is defined over attributes and induces a partial order on these vectors. Intuitively, *for a particular user*, the first partial order objectively establishes that an object is *better*, i.e., more preferable, than another if it is better on all attributes.

Then, the second Pareto-based aggregation, defined across users, induces the second partial order on objects. According to this order, an object is better than another, if it is more preferable according to *all users*. The answer to the MCP problem is the set of *maximal objects* under the second partial order. Note that since this order is only partial, i.e., two objects may not be comparable, there may exist multiple objects that are maximal; recall, that an object is maximal if there exists no other object that succeeds it in the order considered. In essence, it is the fact that this order is partial that guarantees objectiveness.

There exists a plethora of main-memory algorithms for finding the maximal elements according to some partial order, e.g., [17]. More recently, since [8], the problem has received great attention in the data management community, rechristened as the *skyline query*, for which several secondary memory algorithms have been proposed. Therefore, it is possible to adapt an existing algorithm to solve the MCP problem, as we discuss in Section 2.2. However, such an approach faces two performance limitations. First, it needs to compute the matching degrees and form the matching vectors for all objects, before actually executing the algorithm. Second, it makes little sense to apply index-based methods, which are known to be the most efficient, e.g., the state-of-the-art method of [23]. The reason is that the entries of the index depend on the specific MCP instance, and need to be rebuilt from scratch when the user preferences change, even though the description of objects persists.

To address these limitations, we introduce a novel index-based approach for solving the MCP problem. The key idea is to index the set of objects that, unlike the set of matching vectors, remains constant across MCP instances, and defer expensive computation of matching degrees. To achieve this, we apply a simple transformation of the categorical attribute values to intervals, so that each object translates to a rectangle in the

Euclidean space. Then, we can employ a space partitioning index, e.g., an R*-Tree, to hierarchically group the objects. We emphasize that this transformation and index construction is a one-time process, whose cost is amortized across MCP instances, since the index requires no maintenance, as long as the collection of objects persists. Based on the transformation and the hierarchical grouping, it is possible to efficiently compute upper bounds for the matching degrees for *groups* of objects. We then introduce an algorithm that uses these bounds to guide the search towards objects that are more likely to belong to the answer set, and at the same time avoid computing unnecessary matching degrees.

There exists a potential issue of our MCP interpretation when the number of users becomes very large. In this case, the number of conflicting preferences also increases, which makes it harder to differentiate among objects. As a result, the cardinality of the result set increases, which is the cost to pay for being objective. To address this phenomenon, we relax our requirement for unanimity in the second Pareto-based aggregation, and require only a percentage $p\%$ of users to agree. We refer to this extension as the p-MCP problem; naturally, for $p = 100\%$, it reduces to the regular MCP.

2 Problem Statement

2.1 Definitions

Table 2. Notation

Symbol	Definition
$\mathcal{A}$, d	Set of attributes, number of attributes ($\lvert\mathcal{A}\rvert$)
A_k, $\lvert A_k\rvert$	Attribute, number of distinct values in A_k
$\mathcal{H}(A_k)$, $\lvert\mathcal{H}(A_k)\rvert$	Hierarchy of A_k, number of hierarchy nodes
$\mathcal{O}$, o_i	Set of objects, an object
$\mathcal{U}$, u_j	Set of users, a user
$o_i.A_k$, $u_j.A_k$	Value of attribute A_k in object o_i, user u_j
$o_i.I_k$, $u_j.I_k$	Interval representation of the value of A_k in o_i, u_j
m_i^j	Matching vector of object o_i to user u_j
$m_i^j.A_k$	Matching degree of o_i to user u_j on attribute A_k
$o_a \succ o_b$	Object o_a is collectively preferred over o_b
$\mathcal{T}$	The R*-Tree that indexes the set of objects
N_i, e_i	R*-Tree node, the entry for N_i in its parent node
$e_i.ptr$, $e_i.mbr$	The pointer to node N_i, the MBR of N_i
M_i^j	Maximum matching vector of entry e_i to user u_j
$M_i^j.A_k$	Maximum matching degree of e_i to user u_j on A_k

Table 2 shows the most important symbols and their definition. Consider a set of d categorical *attributes* $\mathcal{A} = \{A_1, \ldots, A_d\}$. The domain of each attribute A_k is a *hierarchy* $\mathcal{H}(A_k)$. A hierarchy $\mathcal{H}(A_k)$ defines a tree, where a leaf corresponds to a lowest-level value, and an internal node corresponds to a category, i.e., a set, comprising all values within the subtree rooted at this node. The root of a hierarchy represents the category covering all lowest-level values. We use the symbol $|A_k|$ (resp. $|\mathcal{H}(A_k)|$) to denote the number of leaf (resp. all hierarchy) nodes. With reference to Figure 1a, consider the "Body" attribute. The node "sport cars" is a category and is essentially a shorthand for the set {"roadster", "supercar"}, since it contains the two leaves, "roadster" and "supercar".

Assume a set of *objects* $\mathcal{O}$. An object $o_i \in \mathcal{O}$ is defined over *all* attributes, and the value of attribute $o_i.A_k$ is one of the nodes of the hierarchy $\mathcal{H}(A_k)$. For instance, in Table 1, the value of the "Body" attribute of object o_1, is the leaf "family" in the hierarchy of Figure 1a.

Further, assume a set of *users* $\mathcal{U}$. A user $u_i \in \mathcal{U}$ is defined over a *subset* of the attributes, and for each *specified* attribute $u_i.A_j$, its value in one of the hierarchy $\mathcal{H}(A_j)$

nodes. For all unspecified attributes, we say that user u_i is *indifferent* to them. Note that, a user may specify multiple values for each attribute (see [6] for details).

Given an object o_i, a user u_j, and a specified attribute A_k, the *matching degree* of o_i to u_j with respect to A_k, denoted as $m_i^j.A_k$, is specified by a *matching function* $\mathbf{M}\colon dom(A_k) \times dom(A_k) \to [0, 1]$. The matching function defines the *relation* between the user's preferences and the objects attribute values. For an indifferent attribute A_k of a user u_j, we define $m_i^j.A_k = 1$.

Note that, different matching functions can be defined per attribute and user; for ease of presentation, we assume a single matching function. Moreover, note that this function can be *any user defined function* operating on the cardinalities of intersections and unions of hierarchy attributes. For example, it can be the Jaccard coefficient, i.e., $m_i^j.A_k = \frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k \cup u_j.A_k|}$. The numerator counts the number of leaves in the intersection, while the denominator counts the number of leaves in the union, of the categories $o_i.A_k$ and $u_j.A_k$. Other popular choices are the Overlap coefficient: $\frac{|o_i.A_k \cap u_j.A_k|}{\min(|o_i.A_k|,|u_j.A_k|)}$, and the Dice coefficient: $2\frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k|+|u_j.A_k|}$.

In our running example, we assume the Jaccard coefficient. Hence, the matching degree of car o_1 to user u_1 w.r.t. the "Body" attribute is $\frac{|\text{“family”}\cap\text{“passenger cars”}|}{|\text{“family”}\cup\text{“passenger cars”}|} = \frac{|\{\text{“family”}\}|}{|\{\text{“compact”,“family”,“luxury”}\}|} = \frac{1}{3}$, where we substituted "passenger cars" with the set {"compact", "family","luxury"}.

Given an object o_i and a user u_j, the *matching vector* of o_i to u_j, denoted as m_i^j, is a d-dimensional point in $[0, 1]^d$, where its k-th coordinate is the matching degree with respect to attribute A_k. Furthermore, we define the norm of the matching vector to be $\|m_i^j\| = \sum_{A_k \in \mathcal{A}} m_i^j.A_k$. In our example, the matching vector of car o_1 to user u_1 is $\langle 1/3, 0\rangle$. All matching vectors of this example are shown in Table 1c.

In the following, we consider a particular user u_j and examine the matching vectors. The *first Pareto-based aggregation* across the attributes of the matching vectors, induces the following partial and strict partial "preferred" orders on objects. An object o_a is *preferred* over o_b, for user u_j, denoted as $o_a \succeq^j o_b$ iff for every specified attribute A_k of the user it holds that $m_a^j.A_k \geq m_b^j.A_k$. Moreover, object o_a is *strictly preferred* over o_b, for user u_j, denoted as $o_a \succ^j o_b$ iff o_a is preferred over o_b and additionally there exists a specified attribute A_k such that $m_a^j.A_k > m_b^j.A_k$. Returning to our example, consider user u_1 and its matching vector $\langle 1/3, 1/4\rangle$ for o_4, and $\langle 1/3, 0\rangle$ for o_1. Observe that o_4 is strictly preferred over o_1.

We now consider all users in $\mathcal{U}$. The *second Pareto-based aggregation* across users, induces the following strict partial "collectively preferred" order on objects. An object o_a is *collectively preferred* over o_b, if o_a is preferred over o_b for all users, and there exists a user u_j for which o_a is strictly preferred over o_b. From Table 1c, it is easy to see that car o_4 is collectively preferred over o_1 and o_3, because o_4 is preferred by all three users, and strictly preferred by user u_1.

The *collectively maximal objects* in $\mathcal{O}$ with respect to users $\mathcal{U}$, is defined as the set of objects for which there exists no other object that is collectively preferred over them. In our example, o_4 is collectively preferred over o_1, and all other objects are collectively preferred over o_3. There exists no object which is collectively preferred over o_2 and o_4, and thus are the collectively maximal objects.

We next formally define the MCP problem.

Problem 1. [MCP] Given a set of objects $\mathcal{O}$ and a set of users $\mathcal{U}$ defined over a set of categorical attributes $\mathcal{A}$, the *Multiple Categorical Preference* (MCP) problem is to find the collectively maximal objects of $\mathcal{O}$ with respect to $\mathcal{U}$.

2.2 Baseline MCP Algorithm

The MCP problem can be transformed to a maximal elements problem, or a skyline query, where the input elements are the matching vectors. Note, however, that the MCP problem is different than computing the conventional skyline, i.e., over the object's attribute values.

The *Baseline* (BSL) method, whose pseudocode is depicted in Algorithm 1, takes advantage of this observation. The basic idea of BSL is for each object o_i (loop in line 1) and for all users (loop in line 2), to compute the matching vectors m_i^j (line 3). Subsequently, BSL constructs a $|\mathcal{U}|$-dimensional tuple r_i (line 4), so that its j-th entry is a composite value equal to the matching vector m_i^j of object o_i to user u_j. When all users are examined, tuple r_i is inserted in the set $\mathcal{R}$ (line 5).

Algorithm 1: BSL

```
Input: objects $\mathcal{O}$, users $\mathcal{U}$
Output: $CM$ the collectively maximal
Variables: $\mathcal{R}$ set of intermediate records
1 foreach $o_i \in \mathcal{O}$ do
2     foreach $u_j \in \mathcal{U}$ do
3         compute $m_i^j$
4         $r_i[j] \leftarrow m_i^j$
5     insert $r_i$ into $\mathcal{R}$
6 $CM \leftarrow$ POSkylineAlgo($\mathcal{R}$)
```

The next step is to find the maximal elements, i.e., compute the skyline over the records in $\mathcal{R}$. It is easy to prove that tuple r_i is in the skyline of $\mathcal{R}$ iff object o_i is a collectively maximally preferred object of $\mathcal{O}$ w.r.t. $\mathcal{U}$. Notice, however, that due to the two Pareto-based aggregations, each attribute of a record $r_i \in \mathcal{R}$ is also a record that corresponds to a matching vector, and thus is partially ordered according to the preferred orders defined in Section 2.1. Finally, in order to compute the skyline of $\mathcal{R}$, we need to apply a skyline algorithm (line 6), such as [8,23,14].

The computational cost of BSL is the sum of two parts. The first is computing the matching degrees, which takes $O(|\mathcal{O}| \cdot |\mathcal{U}|)$ time. The second is computing the skyline, which requires $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ comparisons, assuming a quadratic time skyline algorithms is used. Therefore, BSL takes $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ time.

3 Index-based MCP Algorithm

3.1 Hierarchy Transformation

This section presents a simple method to transform the hierarchical domain of a categorical attribute into a numerical domain. The rationale is that numerical domains can be ordered, and thus tuples can be stored in multidimensional index structures. The index-based algorithm of Section 3.2 takes advantage of this transformation.

Consider an attribute A and its hierarchy $\mathcal{H}(A)$, which forms a tree. We assume that any internal node has at least two children; if a node has only one child, then this node and its child are treated as a single node. Furthermore, we assume that there exists an ordering, e.g., the lexicographic, among the children of any node. Observe that such an ordering, totally orders all leaf nodes.

The hierarchy transformation assigns an interval to each node, similar to labeling schemes such as [1]. The i-th leaf of the hierarchy (according to the ordering) is assigned the interval $[i-1, i)$. Then, each internal node is assigned the smallest interval that covers the intervals of its children. Figure 1 depicts the assigned intervals for all nodes in the two car hierarchies.

Following this transformation, the value on the A_k attribute of an object o_i becomes an interval $o_i.I_k = [o_i.I_k^-, o_i.I_k^+)$. The same holds for a user u_j. Therefore, the transformation translates the hierarchy $H(A_k)$ into the numerical domain $[0, |A_k|]$.

An important property of the transformation is that it becomes easy to compute matching degrees for metrics that are functions on the cardinalities of intersections or unions of hierarchy attributes. This is due to the following properties, which use the following notation: for a closed-open interval $I = [\alpha, \beta)$, define $\|I\| = \beta - \alpha$. Note that the proofs can be found in the full version of the paper [6].

Proposition 1. For objects/users x, y, and an attribute A_k, let $x.I_k$, $y.I_k$ denote the intervals associated with the value of x, y on A_k. Then the following hold:
(1) $|x.A_k| = \|x.I_k\|$
(2) $|x.A_k \cap y.A_k| = \|x.I_k \cap y.I_k\|$
(3) $|x.A_k \cup y.A_k| = \|x.I_k\| + \|y.I_k\| - \|x.I_k \cap y.I_k\|$

3.2 Algorithm Description

This section introduces the *Index-based* MCP (IND) algorithm. The key ideas of IND are: (1) apply the hierarchy transformation, previously described, and index the resulting intervals, and (2) define upper bounds for the matching degrees of a group of objects, so as to guide the search and quickly prune unpromising objects.

We assume that the set of objects $\mathcal{O}$ and the set of users $\mathcal{U}$ are transformed so that each attribute A_k value is an interval I_k. Therefore, each object (and user) defines a (hyper-)rectangle on the d-dimensional cartesian product of the numerical domains, i.e., $[0, |A_1|) \times \cdots \times [0, |A_d|)$.

Figure 2 depicts the transformation of the objects and users shown in Table 1 for the hierarchies in Figure 1. For instance, object o_1 is represented as the rectangle $[9, 10) \times [1, 2)$ in the "Engine"×"Body" plane. Similarly, user u_1 is represented as two intervals, $[0, 4)$, $[0, 3)$, on the transformed "Engine", "Body" axes, respectively.

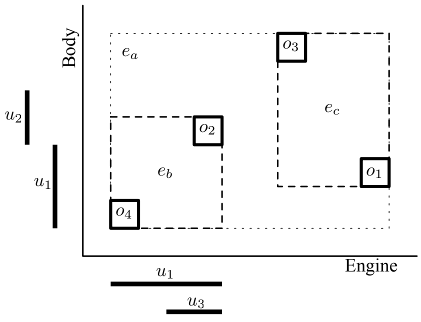


Fig. 2. Transformed objects and users

The IND algorithm indexes the set of objects in this d-dimensional numerical space. In particular, IND employs an R*-Tree $\mathcal{T}$ [5], which is well suited to index rectangles. Each $\mathcal{T}$ node corresponds to a disk page, and contains a number of entries. Each entry e_i comprises (1) a pointer $e_i.ptr$, and (2) a Minimum Bounding Rectangle (MBR) $e_i.mbr$.

A leaf entry e_i corresponds to an object o_i, its pointer $o_i.ptr$ is *null*, and $e_i.mbr$ is the rectangle defined by the intervals of o_i. A non-leaf entry e_i corresponds to a child node N_i, its pointer $e_i.ptr$ contains the address of N_i, and $e_i.mbr$ is the MBR of (i.e., the tightest rectangle that encloses) the MBRs of the entries within N_i.

Due to the enclosing property of MBRs, the following holds. The MBR of an entry e_i encloses all objects that are stored at the leaf nodes within the $\mathcal{T}$ subtree rooted at node N_i. It is often helpful to associate an entry e_i with all the objects it encloses, and thus treat e_i as a group of objects.

Consider a $\mathcal{T}$ entry e_i and a user $u_j \in \mathcal{U}$. Given only the information within entry e_i, i.e., its MBR, and not the contents, i.e., its enclosing objects, at the subtree rooted at N_i, it is impossible to compute the matching vectors for the objects within this subtree. However, it is possible to derive an *upper bound* for the matching degrees of any of these objects.

We define the *maximum matching degree* $M_i^j.A_k$ of entry e_i on user u_j w.r.t. specified attribute A_k as the highest attainable matching degree of any object that may reside within $e_i.mbr$. To do this we first need a way to compute lower and upper bounds on unions and intersections of a user interval with an MBR.

Proposition 2. Fix an attribute A_k. Consider an object/user x, and let I_x, denote the interval associated with its value on A_k. Also, consider another object/user y whose interval I_y on A_k is contained within a range R_y. Given an interval I, $\delta(I)$ returns 0 if I is empty, and 1 otherwise. Then the following hold:

(1) $1 \ \leq |y.A_k| \leq \|R_y\|$
(2) $\delta(I_x \cap R_y) \ \leq |x.A_k \cap y.A_k| \leq \|I_x \cap R_y\|$
(3) $\|I_x\| + 1 - \delta(I_x \cap R_y) \ \leq |x.A_k \cup y.A_k| \leq \|I_x\| + \|R_y\| - \delta(I_x \cap R_y)$

Then, defining the maximum matching degree reduces to appropriately selecting the lower/upper bounds for the specific matching function used. For example, consider the case of the Jaccard coefficient, $\frac{|o_i.A_k \cap u_j.A_k|}{|o_i.A_k \cup u_j.A_k|}$. Assume e_i is a non-leaf entry, and let $e_i.R_k$ denote the range of the MBR on the A_k attribute. We also assume that $u_j.I_k$ and $e_i.R_k$ overlap. Then, we define $M_i^j.A_k = \frac{\|e_i.R_k \cap u_j.I_k\|}{\|u_j.I_k\|}$, where we have used the upper bound for the intersection in the enumerator and the lower bound for the union in the denominator, according to Proposition 2. For an indifferent to the user attribute A_k, we define $M_i^j.A_k = 1$. Now, assume that e_i is a leaf entry, that corresponds to object o_i. Then the maximum matching degree $M_i^j.A_k$ is equal to the matching degree $m_i^j.A_k$ of o_i to u_j w.r.t. A_k.

Computing maximum matching degrees for other metrics is straightforward. In any case, the next proposition shows that an appropriately defined maximum matching degree is an upper bound to the matching degrees of all objects enclosed in entry e_i.

Proposition 3. The maximum matching degree $M_i^j.A_k$ of entry e_i on user u_j w.r.t. specified attribute A_k is an upper bound to the highest matching degree among all objects in the group that e_i defines.

In analogy to the matching vector, the *maximum matching vector* M_i^j of entry e_i on user u_j is defined as a d-dimensional vector whose k-th coordinate is the maximum matching degree $M_i^j.A_k$. Moreover, the norm of the maximum matching vector is $\|M_i^j\| = \sum_{A_k \in \mathcal{A}} M_i^j.A_k$.

Next, consider a $\mathcal{T}$ entry e_i and the entire set of users $\mathcal{U}$. We define the *score* of an entry e_i as $score(e_i) = \sum_{u_j \in \mathcal{U}} \|M_i^j\|$. This score quantifies how well the enclosed objects of e_i match against all users' preferences. Clearly, the highest the score, the more likely that e_i contains objects that are good matches to users.

Algorithm 2 presents the pseudocode for IND. The algorithm maintains two data structures: a heap H which stores $\mathcal{T}$ entries sorted by their score, and a list CM of collectively maximal objects discovered so far. Initially the list CM is empty (line 1), and the root node of the R*-Tree is read (line 2). The score of each root entry is computed and all entries are inserted in H (line 3). Then, the following process (loop in line 4) is repeated as long as H has entries.

The H entry with the highest score, say e_x, is popped (line 5). If e_x is a non-leaf entry (line 6), it is *expanded*, which means that the node N_x identified by $e_x.ptr$ is read (line 7). For each child entry e_i of N_x (line 8), its maximum matching degree M_i^j with respect to every user $u_j \in \mathcal{U}$ is computed (lines 10–11). Then, the list CM is scanned (loop in line 12). If there exists an object o_a in CM such that (1) for each user u_j, the matching vector m_a^j of o_a is better than M_i^j, and (2) there exists a user u_k so that the matching vector m_a^k of o_a is strictly better than M_i^k, then entry e_i is discarded (lines 13–15). It is straightforward to see (from Proposition 3) that if this condition holds, e_i cannot contain any object that is in the collectively maximal objects, which guarantees IND' correctness. When the condition described does not hold (line 16), the score of e_i is computed and e_i is inserted in H (line 17).

Algorithm 2: IND

```
    Input: R*-Tree $\mathcal{T}$, users $\mathcal{U}$
    Output: $CM$ the collectively maximal
    Variables: $H$ a heap with $\mathcal{T}$ entries sorted by $score()$
 1  $CM \leftarrow \varnothing$
 2  read $\mathcal{T}$ root node
 3  insert in $H$ the root entries
 4  while $H$ is not empty do
 5      $e_x \leftarrow$ pop $H$
 6      if $e_x$ is non-leaf then
 7          $N_x \leftarrow$ read node $e_x.ptr$
 8          foreach $e_i \in N_x$ do
 9              $pruned \leftarrow false$
10              foreach $u_j \in \mathcal{U}$ do
11                  compute $M_i^j$
12              foreach $o_a \in CM$ do
13                  if $\forall A_j : m_a^j \succeq M_i^j \wedge \exists A_k : m_a^k \succ M_i^k$ then
14                      $pruned \leftarrow true$
15              if not $pruned$ then
16                  insert $e_i$ in $H$
17      else
18          $o_x \leftarrow e_x$
19          $result \leftarrow true$
20          foreach $o_a \in CM$ do
21              if $o_a \succ o_x$ then
22                  $result \leftarrow false$
23          if $result$ then
24              insert $o_x$ in $CM$
```

Now, consider the case that e_x is a leaf entry (line 18), corresponding to object o_x (line 19). The list CM is scanned (loop in line 21). If there exists an object that is collectively preferred over o_x (line 22), it is discarded. Otherwise (line 25–26), o_x is inserted in CM.

The algorithm terminates when H is empty (loop in line 4), at which time the list CM contains the collectively maximal objects.

IND performs in the worst case $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ comparisons, and computes matching degrees on the fly at a cost of $O(|\mathcal{O}| \cdot |\mathcal{U}|)$. Overall, IND takes $O(|\mathcal{O}|^2 \cdot |\mathcal{U}| \cdot d)$ time, the same as BSL. However, in practice IND is more than an order of magnitude faster than BSL (see Section 6).

4 The *p*-MCP Problem

As the number of users increases, it becomes more likely that the users express very different and conflicting preferences. Hence, it becomes difficult to find a pair of objects such that the users unanimously agree that one is worst than the other. Ultimately, the number of maximally preferred objects increases. This means that the answer to an MCP problem with a large set of users becomes less meaningful.

The root cause of this problem is that we require unanimity in deciding whether an object is collectively preferred by the set of users. The following definition relaxes this requirement. An object o_a is *p-collectively preferred* over o_b, denoted as $o_a \succ_p o_b$, iff there exist a subset $\mathcal{U}_p \subseteq \mathcal{U}$ of at least $\frac{p}{100} \cdot |\mathcal{U}|$ users such that for each user $u_i \in \mathcal{U}_p$ o_a is preferred over o_b, and there exists a user $u_j \in \mathcal{U}_p$ for which o_a is strictly preferred over o_b. In other words, we require only $p\%$ of the users votes to decide whether an object is universally preferred. Similarly, the *p-collectively maximal objects* of $\mathcal{O}$ with respect to users $\mathcal{U}$, is defined as the set of objects in $\mathcal{O}$ for which there exists no other object that is p-collectively preferred over them. The above definitions give rise to the p-MCP problem.

Problem 2. [p-MCP] Given a set of objects $\mathcal{O}$ and a set of users $\mathcal{U}$ defined over a set of categorical attributes $\mathcal{A}$, the *p-Multiple Categorical Preference* (p-MCP) problem is to find the p-collectively maximal objects of $\mathcal{O}$ with respect to $\mathcal{U}$.

If an object o_1 is collectively preferred over o_2 is also p-collectively preferred for any p. Any object that is p-collectively maximal is also collectively maximal for any p. Therefore, the answere to the p-MCP problem is a subset of the answer to the corresponding MCP. Furthermore, the following transitivity relation holds for three objects o_1, o_2, o_3: if $o_1 \succ o_2$ and $o_2 \succ_p o_3$, then $o_1 \succ_p o_3$. This implies that if an object is not p-collectively maximal, then there must exist a collectively maximal object that is p-collectively preferred over it. These observations are similar to those for the k-dominance notion [10].

Based on this observation, we propose a baseline algorithm for the p-MCP problem, based on BSL (Section 2.2). The p-BSL algorithm first computes the collectively maximal objects applying BSL. Then among the results, it determines the objects for which there exists no other object that is p-collectively preferred over them, and reports them. This refinement is implemented using a block-nested loop procedure. Additionally, we propose an extension of the IND algorithm (Section 3.2) for the p-MCP problem, termed p-IND. Pseudocodes and details can be found in [6].

5 Related Work

Recommendation systems. There exist several techniques to specify preferences on objects. The *quantitative* preferences, e.g., [2], assign a numeric score to attribute values, signifying importance. There also exist *qualitative* preferences, e.g., [16], which are relatively specified using binary relationships. This work assumes the case of boolean quantitative preferences, where some attribute values are preferred, while others are indifferent.

The general goal of *recommendation systems* [7,26] is to identify those objects that are most aligned to a user's preferences. Typically, these systems provide a *ranking* of the objects by *aggregating* user preferences; e.g., [18,2,16,26].

Recently, several methods for *group recommendations* are proposed [15]. These methods, recommend items to a group of users, trying to satisfy all the group members [24,21]. The existing methods are classified into two approaches. In the first, the preferences of each group member are combined to create a virtual user; the recommendations to the group are proposed w.r.t. to the virtual user. In the second, individual recommendations for each member is computed; the recommendations of all members are merged into a single recommendation.

Several methods to combine different ranked lists are presented in the IR literature. There the data fusion problem is defined. Given a set of ranked lists of documents returned by different search engines, construct a single ranked list combining the individual rankings; e.g., [3,13].

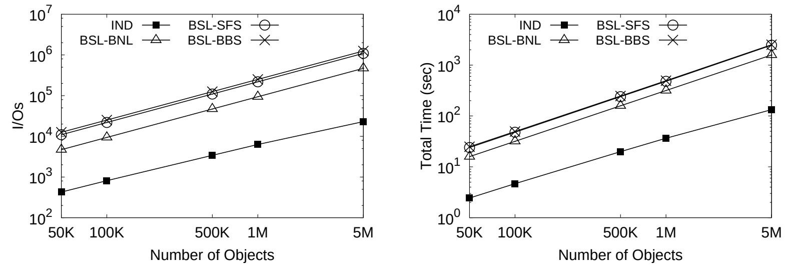
Pareto-based aggregation. The work of [8] rekindled interest in the problem of finding the maximal objects and re-introduces it as the skyline operator. An object is dominated if there exists another object before it according to the partial order enforced by the Pareto-based aggregation. The maximal objects are referred to as the skyline. The authors propose several external memory algorithms. The most well-known method is Block Nested Loops (BNL) [8], which checks each point for dominance against the entire dataset. Sort-based skyline algorithms (i.e., SFS [12], LESS [14], and SaLSa [4]) attempt to reduce the number of dominance checks by sorting the input data first. In other approaches, multidimensional indexes are used to guide the search and prune large parts of the space. The most well-known algorithm in this category is BBS [23] which uses an R-Tree. Specific algorithms are proposed to efficiently compute the skyline over partially ordered domains [9,28,25,30], metric spaces [11], or non-metric spaces [22].

Several lines of research attempt to address the issue that the size of skyline cannot be controlled, by introducing new concepts and/or ranking the skyline (see [20] for a survey). [29] ranks tuples based on the number of records they dominate, [10] relaxes the notion of dominance, and [19,27] find the k most representative skylines.

6 Experimental Analysis

6.1 Setting

Datasets. We use two datasets in our experimental evaluation. The first is *Synthetic*, where objects and users are synthetically generated. All attributes have the same hierarchy, a binary tree of height $\log |A|$, and thus all attributes have the same number of leaf hierarchy nodes $|A|$. To obtain the set of objects, we fix a level, ℓ_o (where $\ell_o = 1$ corresponds to the leaves), in all attribute hierarchies. Then, we randomly select nodes from this level to obtain the objects' attribute value. The number of objects is denoted as $|\mathcal{O}|$, while the number of attributes for each object is denoted as d. Similarly, to obtain the set of users, we fix a level, ℓ_u, in all hierarchies. We further make the assumption that a user specifies preferences for only one of the attributes. Thus, for each user we randomly select an attribute, and set its value to a randomly picked hierarchy node at level ℓ_u. The number of users is denoted as $|\mathcal{U}|$.



(a) I/O Operations

(b) Total Time

Fig. 3. Synthetic Data: Varying $|\mathcal{O}|$

The second dataset is *Cars*, where we use as objects $|\mathcal{O}| = 4261$ car descriptions retrieved from the Web[1]. We consider two categorical attributes, Body and Engine, having 3 and 4 levels, and 10 and 35 leaf hierarchy nodes, respectively. The user preferences are selected based on car rankings[2]. In particular, for the Body attribute, we use the most popular cars list to derive a subset of desirable nodes at the leaf level. For the Engine attribute, we use the most fuel efficient cars list to derive a subset of desirable nodes at the second hierarchy level. As in Synthetic, a user specifies preference on a single attribute, and thus randomly selects an attribute, and picks its value among the desirable subsets.

Table 3. Parameters (Synthetic)

Symbol	**Values** [Default]
$\|\mathcal{O}\|$	50K, 100K, [500K], 1M, 5M
d	2, 3, [4], 5, 6
$\|\mathcal{U}\|$	2, 4, [8], 16, 32
$\log\|A\|$	4, 6, [8], 10, 12
ℓ_o	[1], 2, 3, 4, 5
ℓ_u	[2], 3, 4, 5, 6

Synthetic Data Parameters. Table 3 lists the parameters that we vary and the range of values examined for Synthetic. To segregate the effect of each parameter, we perform six experiments, and in each we vary a single parameter from the table, while we set the remaining ones to their default values.

Methods. For the MCP problem, we implement IND (Section 3) and three flavors of the BSL algorithm (Section 2.2), denoted BSL-BNL, BSL-SFS, and BSL-BBS, which use the skyline algorithms BNL [8], SFS [12], BBS [23], respectively. Similarly, for the p-MCP problem (Section 4), we implement the respective extensions of all algorithms (IND and BSL variants), distinguished by a p prefix. All algorithms were written in C++, compiled with gcc, and experiments were performed on a 2GHz CPU.

Evaluation Metrics. To guage efficiency of all algorithms, we measure: (1) the number of disk I/O operations; (2) the number of dominance checks; and (3) the total execution time, measured in secs. Due to the space limitation, in most cases we present only the total time graphs; all the graphs can be found in [6].

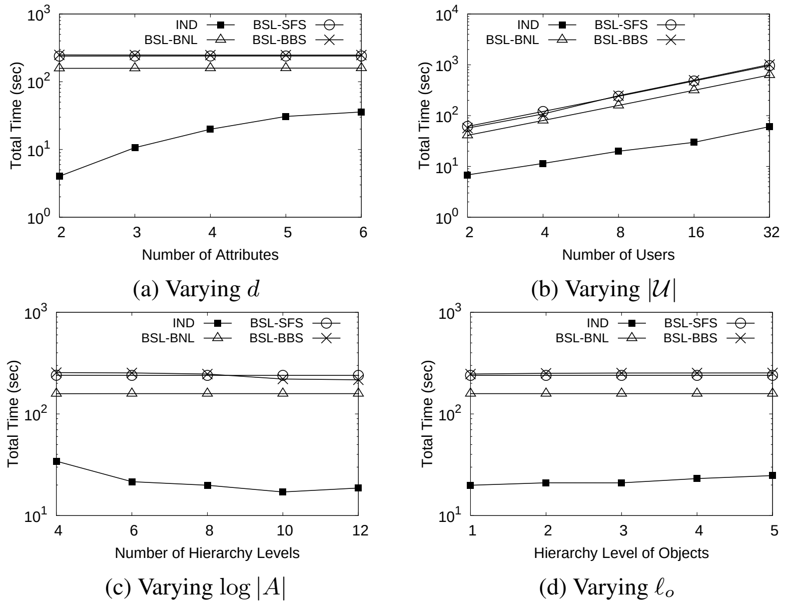
6.2 Efficiency of MCP algorithms

Results on Synthetic Data

Varying the number of objects. In the first experiment, we study performance with respect to the objects' set cardinality $|\mathcal{O}|$. Particularly, we vary the number of objects

[1] http://www.carlik.com

[2] http://www.edmunds.com/car-reviews/



(a) Varying d

(b) Varying $|\mathcal{U}|$

(c) Varying $\log |A|$

(d) Varying ℓ_o

Fig. 4. Synthetic Data: Varying several parameters

from 50K up to 5M and measure the number of I/Os, the number of dominance checks, and the total processing time.

When the number of objects increases, the performance of all methods deteriorates. The number of I/Os (Figure 3a) performed by IND is much less than the BSL variants, the reason being BSL needs to construct a file containing matching degrees. Moreover, the SFS and BBS variants have to preprocess this file, to sort it and build the R-Tree, respectively. Hence, BSL-BNL requires the fewest I/Os among the BSL variants.

All methods require roughly the same number of dominance checks [6]. IND performs fewer checks, while BSL-BNL the most. Compared to the other BSL variants, BSL-BNL performs more checks because, unlike the others, computes the skyline over an unsorted file. IND performs as well as BSL-SFS and BSL-BBS, which have the easiest task. Overall, however, Figure 3b shows that IND is more than an order of magnitude faster than the BSL variants.

Varying the number of attributes. Figure 4a investigates the effect as we increase the number of attributes d from 2 up to 6. Figure 4a shows that the total time of IND increases with d, but it is still significantly smaller (more than 4 times) than the BSL methods even for $d = 6$.

Varying the number of users. In the next experiment, we vary the users' set cardinality $|\mathcal{U}|$ from 2 up to 32; results are depicted in Figure 4b. The performance of all methods deteriorates with $|\mathcal{U}|$. Figure 4b shows that IND is more than an order of magnitude faster than all the BSL variants, among which BSL-BNL is the fastest.

Varying the hierarchy height. In this experiment, we vary the hierarchy height $\log |A|$ from 4 up to 12 levels. Figure 4c illustrates the results. All methods are largely unaffected by this parameter. Overall, IND is more than an order of magnitude faster than all BSL variants.

Varying the objects level. Figure 4d depicts the results of varying the level ℓ_o from which we draw the objects' values. The performance of all methods is not significantly affected by ℓ_o.
Varying the users level. In this experiment, we vary he level ℓ_u from which we draw the users' preference values. The total time of IND takes its highest value of $\ell_u = 6$, as the number of required dominance checks increases sharply for this setting. Still IND is around 3 times faster than BSL-BNL [6].

Results on Real Data

In this experiment, we have $|\mathcal{O}| = 4261$ cars and we vary the number of users $|\mathcal{U}|$ from 2 up to 32. Figure 5 presents the results. As the number of users increase, the performance of the BSL variants worsens, while that of IND is slightly affected. This pattern is in analogy to the case of the Synthetic dataset (Figure 4). For more than 4 users, IND outperforms the BSL methods by at least an order of magnitude.

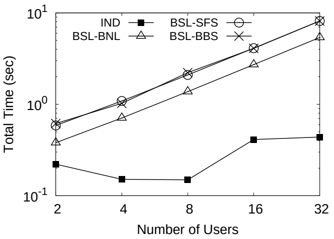
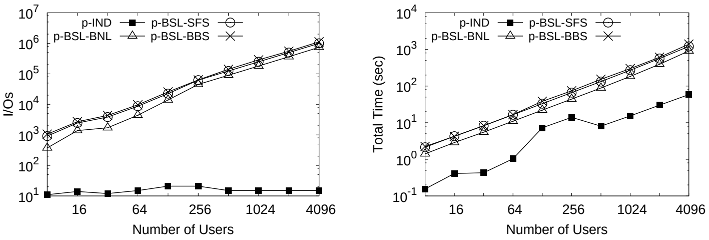


Fig. 5. Cars: varying $|\mathcal{U}|$

6.3 Efficiency of *p*-MCP algorithms

In this section, we evaluate the p-MCP algorithms using the Car dataset; as before, we measure the number of I/Os, dominance checks and the total time.
Varying the number of users. Figure 6 shows the effect of varying the number of users from 8 up to 4096, while $p = 30\%$. The required number of I/Os operations increases with $|\mathcal{U}|$ for both methods, as Figure 6a shows; the rate of increase for p-BSL-BNL is much higher. Overall, Figure 6b shows that p-IND constantly outperforms p-BSL-BNL and is up to 1 order of magnitude faster.
Varying parameter p. We increase the parameter p from 10% up to 50%. The performance of all methods (in terms of I/Os and total time) remains unaffected by p, and thus we omit the relevant figures in the interest of space.



(a) I/O Operations (b) Total Time

Fig. 6. p-MCP, Cars: varying $|\mathcal{U}|$

References

1. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *SIGMOD*, 1989.

2. R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD*, 2000.
3. J. A. Aslam and M. H. Montague. Models for metasearch. In *SIGIR*, 2001.
4. I. Bartolini, P. Ciaccia, and M. Patella. Efficient Sort-based Skyline Evaluation, TODS, 33(4), 2008.
5. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
6. N. Bikakis, K. Benouaret, D. Sacharidis. Reconciling multiple categorical preferences with double pareto-based aggregation. Technical Report 2013, `http://www.dblab.ntua.gr/~bikakis/MCP.pdf`
7. J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowl.-Based Syst.*, 46, 2013.
8. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
9. C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, 2005.
10. C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.
11. L. Chen and X. Lian. Efficient processing of metric skyline queries. *TKDE*, 21(3), 2009.
12. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.
13. C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.
14. P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDBJ*, 16(1), 2007.
15. A. Jameson and B. Smyth. Recommendation to groups. In *The Adaptive Web*, 2007.
16. W. Kießling. Foundations of preferences in database systems. In *VLDB*, 2002.
17. H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4), 1975.
18. M. Lacroix and P. Lavency.Preferences:Putting more knowledge into queries.In *VLDB*,1987.
19. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.
20. C. Lofi and W.-T. Balke. On skyline queries and how to choose from pareto sets. In *Advanced Query Processing (1)*. 2013.
21. E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast group recommendations by applying user clustering. In *ER*, pages 126–140, 2012.
22. D. P, P. M. Deshpande, D. Majumdar, and R. Krishnapuram. Efficient skyline retrieval with arbitrary similarity measures. In *EDBT*, 2009.
23. D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1), 2005.
24. S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Space efficiency in group recommendation. *VLDB J.*, 19(6), 2010.
25. D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically sorted skylines for partially ordered domains. In *ICDE*, 2009.
26. K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3), 2011.
27. Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.
28. R. C.-W. Wong, A. W.-C. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu. Efficient skyline querying with variable user preferences on nominal attributes. *VLDB*, 1(1), 2008.
29. M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *VLDB*, 2007.
30. S. Zhang, N. Mamoulis, B. Kao, and D. W.-L. Cheung. Efficient skyline evaluation over partially ordered domains. *VLDB*, 3(1), 2010.