Continuous Monitoring of Pareto Frontiers on Partially Ordered Attributes for Many Users

Afroza Sultana
University of Texas at Arlington
Arlington, Texas
afroza.sultana@mavs.uta.edu

Chengkai Li
University of Texas at Arlington
Arlington, Texas
cli@uta.edu

ABSTRACT

We study the problem of *continuous object dissemination*—given a large number of users and continuously arriving new objects, deliver an object to all users who prefer the object. Many real world applications analyze users' preferences for effective object dissemination. For continuously arriving objects, timely finding users who prefer a new object is challenging. In this paper, we consider an append-only table of objects with multiple attributes and users' preferences on individual attributes are modeled as *strict partial orders*. An object is preferred by a user if it belongs to the *Pareto frontier* with respect to the user's partial orders. Users' preferences can be similar. Exploiting shared computation across similar preferences of different users, we design algorithms to find *target users* of a new object. In order to find users of similar preferences, we study the novel problem of clustering users' preferences that are represented as partial orders. We also present an approximate solution of the problem of finding target users which is more efficient than the exact one while ensuring sufficient accuracy. Furthermore, we extend the algorithms to operate under the semantics of sliding window. We present the results from comprehensive experiments for evaluating the efficiency and effectiveness of the proposed techniques.

1 INTRODUCTION

Many applications serve users better by disseminating objects to the users according to their preferences. User preferences can be modeled via a variety of means including *collaborative filtering* [19], *top-k ranking* [7, 8], *skyline* [2], and general *preference queries* [5, 12]. In various scenarios, users' preferences stand or only change occasionally, while the objects keep coming continuously. Such scenarios warrant the need for a capability of continuous monitoring of preferred objects. While previous studies have made notable contributions on continuous evaluation of skyline [14, 28] and top-k queries [29], we note that two important considerations are missing from prior works:

- *Many users*: There may be a large number of users and the users may have similar preferences. Prior studies focus on the query needs of one user and thus their algorithmic solutions can only be applied separately on individual users. A solution can potentially attain significant query performance gain by leveraging users' *common preferences*.
- *Partially ordered attributes*: Prior works focus on top-k and skyline queries. In multi-objective optimization, a more general concept than skyline is *Pareto frontier*. Consider a table of objects with a set of attributes. An object is *Pareto-optimal* (i.e., it belongs to the Pareto frontier) if and only if it is not dominated by any other object [1, 13]. Object y dominates x if and only if y is better than or equal to x on every attribute and is better on at least one attribute. In defining the *better-than* relations, most studies on skyline queries assume a total order on the ordinal or numeric values of an attribute, except for [17, 30] which consider strict partial orders. The psychological nature of human's preferences determines that it is not always natural to enforce a total order. Oftentimes real-world preferences can only be modeled as strict partial orders [5, 12, 17].

Consider the following motivating applications which monitor Pareto frontiers on partially ordered attributes for many users.

- Social network content and news delivery: It is often impossible and unnecessary for a user to keep up with the plethora of updates (e.g., news feeds in Facebook) from their social circles. When a new item is posted, if the item is Pareto-optimal with respect to a user, it can be displayed above other updates in the user's view. Similar ideas can be adopted by mass media to ensure their news reaches the right audience. User preferences can be modeled on content creator, topic, location, and so on. Enforcing total orders on such attributes is both cumbersome and unnatural.
- Publication alerts: Bibliography servers such as PubMed and Google Scholar can notify users about newly published articles matching their preferences on venues and keywords. Such attributes do not welcome total orders either.
- Product recommendation: When a new product becomes available, a retailer can notify customers who may be interested. It can distill customers' preferences on product specifications (e.g., brand, display and memory for laptops) from profiles, past transactions and website browsing logs. Example 1.1 discusses this application more concretely.

Example 1.1. Consider an inventory of laptops in Table 1 and customers' preferences on individual product attributes (display, brand and CPU) modeled as strict partial orders in Table 2. For an attribute, the corresponding strict partial order is depicted as a directed acyclic graph (DAG), more specifically a Hasse diagram. Given two values x and y in the attribute's domain, the existence of a path from x to y in the DAG implies that x is preferred to y. With respect to customer c_1 and attribute brand, the path from *Lenovo* to *Toshiba* implies that c_1 prefers *Lenovo* to *Toshiba*. There is no path between *Toshiba* and *Samsung*, which indicates c_1 is indifferent between the two brands.

The strict partial orders on various attributes together represent a customer's preferences on objects. For instance, c_1 prefers o_2=⟨14, *Apple*, *dual*⟩ to o_1=⟨12, *Apple*, *single*⟩, since they prefer 13−15.9 to 10−12.9 on display and *dual* to *single* on CPU. With regard to o_1 and o_3=⟨15, *Samsung*, *dual*⟩, c_1 does not prefer one over the other because, though they prefer 13−15.9 to 10−12.9 and *dual* to *single*, they prefer *Apple* to *Samsung* on brand.

According to the data in Tables 1 and 2, if the existing products are o_1 to o_{14} (ignore o_{15} and o_{16} for now), the Pareto frontiers of c_1 and c_2 are $\{o_2\}$ and $\{o_2, o_3\}$, respectively. Suppose o_{15}=⟨16.5, *Lenovo*, *quad*⟩ just becomes available. For c_1, o_{15} does not belong

	display	brand	CPU
o_1	12	*Apple*	*single*
o_2	14	*Apple*	*dual*
o_3	15	*Samsung*	*dual*
o_4	19	*Toshiba*	*dual*
o_5	9	*Samsung*	*quad*
o_6	11.5	*Sony*	*single*
o_7	9.5	*Lenovo*	*quad*
o_8	12.5	*Apple*	*dual*
o_9	19.5	*Sony*	*single*
o_{10}	9.5	*Lenovo*	*triple*
o_{11}	9	*Toshiba*	*triple*
o_{12}	8.5	*Samsung*	*triple*
o_{13}	14.5	*Sony*	*dual*
o_{14}	17	*Sony*	*single*
o_{15}	16.5	*Lenovo*	*quad*
o_{16}	16	*Toshiba*	*single*

Table 1: Product table.

	display	brand	CPU
c_1	13−15.9 → 10−12.9 → 16−18.9, 19−*up* → 9.9−*under*	*Apple* → *Lenovo* → *Sony* → *Toshiba*, *Samsung*	*dual* → *triple*, *quad* → *single*
c_2	13−15.9 → 10−12.9, 16−18.9 → 19−*up* → 9.9−*under*	*Lenovo* → *Apple*, *Samsung* → *Toshiba* → *Sony*	*quad* → *triple* → *dual* → *single*
U	13−15.9 → 10−12.9, 16−18.9 → 19−*up* → 9.9−*under*	*Apple*, *Lenovo* → *Toshiba*, *Sony*, *Samsung*	*dual*, *triple*, *quad* → *single*
$\widehat{U}$	13−15.9 → 10−12.9 → 16−18.9 → 19−*up* → 9.9−*under*	*Apple*, *Lenovo* → *Sony*, *Samsung* → *Toshiba*	*dual*, *quad* → *triple* → *single*

Table 2: User preferences. U={c_1,c_2}.

to the Pareto frontier. It is dominated by o_2, because c_1 prefers 14-inch display over 16.5-inch, *Apple* over *Lenovo*, and *dual*-core CPU over *quad*-core CPU. However, o_{15} is a Pareto-optimal object for c_2 since it is not dominated by any other object according to c_2's preferences. It is thus recommended to c_2, and the Pareto frontier of c_2 is updated to $\{o_2, o_3, o_{15}\}$. △

This paper **formulates the problem of *continuous monitoring of Pareto frontiers***: given a large number of users and continuously arriving new objects, for each newly arrived object, discover all users for whom the object is Pareto-optimal. Users' preferences are modeled as strict partial orders, one for each attribute domain of the objects.

It is key to devise an efficient approach to this problem. The value of a Pareto-optimal object diminishes quickly; the earlier it is found to be worth recommendation, the better. For instance, a status update in a social network keeps getting less relevant since the moment it is posted; a customer's need for a product may be fulfilled by a less preferred choice, if an even better option was not shown to the customer in time.

A simple, brute-force approach is to, given a newly arrived object, compute for every user if the object belongs to the Pareto

Apple, *Toshiba* → *Lenovo* → *Samsung*	*Apple*, *Toshiba* → *Lenovo* → *Samsung*	*Apple*, *Toshiba* → *Lenovo* → *Samsung*
c_1	c_2	U_1
Samsung → *Lenovo* → *Apple* → *Toshiba*	*Samsung* → *Lenovo* → *Apple* → *Toshiba*	*Samsung* → *Lenovo* → *Apple* → *Toshiba*
c_3	c_4	U_2
Lenovo → *Apple* → *Samsung*	*Lenovo* → *Apple* → *Toshiba*, *Samsung*	*Lenovo* → *Apple* → *Samsung*
c_5	c_6	U_3

Table 3: User preferences with respect to brand. U_1={c_1,c_2}, U_2={c_3,c_4}, U_3={c_5,c_6}.

frontier with respect to the user's preferences. This entails continuous maintenance of Pareto frontier for each and every user. The brute-force approach is subject to a clear drawback—repeated and wasteful maintenance of Pareto frontier for every user.

Sharing computation across users To tackle the aforementioned drawback, we partly resort to sharing computation across users. The challenge lies in the diversity of corresponding partial orders—a Pareto-optimal object with respect to one user may or may not be in the Pareto frontier for another user. Nonetheless, users have common preferences. In Table 2, both c_1 and c_2 prefer $13-15.9$ inch display the most. Both prefer *Apple* and *Lenovo* to *Toshiba* and *Sony*, and they both prefer *single*-core CPU the least. In Table 2, U is a *virtual user* whose partial orders depict the common preferences of c_1 and c_2. Intuitively, users having similar preferences can be clustered together.

We thus design algorithms to mitigate repetitive computation via sharing computation across similar preferences of users. To intuitively understand the idea, consider two example scenarios. i) If o is dominated by o' with respect to the common preferences of a set of users, then o is disqualified in Pareto-optimality for all users in the set. In Example 1.1, consider o_{16}=⟨16, *Toshiba*, *single*⟩ as the new object. With respect to U, o_{16} is dominated by both o_2=⟨14, *Apple*, *dual*⟩ and o_{15}=⟨16.5, *Lenovo*, *quad*⟩. Therefore, o_{16} belongs to the Pareto frontier of neither c_1 nor c_2. ii) Before the arrival of o_2, obviously o_1=⟨12, *Apple*, *single*⟩ is the only Pareto-optimal object for U, c_1 and c_2. Now consider the entrance of o_2. As o_1 is dominated by o_2 with respect to U, o_1 is replaced by o_2 in the Pareto frontier. This comparison is sufficient to decide that o_1 is dominated by o_2 for both c_1 and c_2.

Clustering users To find users sharing similar preferences, we study the novel problem of clustering strict partial orders, which are used to model the preferences of both users and clusters. We measure the similarity between clusters and users by their common preferences. Such similarity measures factor in the different significance of preferences at various levels of the partial orders. Table 3 depicts six customers' preferences on brand, in which c_4, c_5, and c_6 prefer *Lenovo* to all other brands except that c_4 prefers *Samsung* over *Lenovo*. Consider the objects in Table 1. For both c_5 and c_6, the Pareto frontiers contain $\{o_7, o_{10}, o_{15}\}$, while c_4 has $\{o_3, o_5, o_{12}\}$ as its Pareto frontier. We can say that c_5 and c_6 are more similar than c_4 and c_5 or c_4 and c_6.

Approximation The clustering algorithm may produce clusters that comprise few users, due to diverse preferences. With

small clusters, the shared computation mentioned above may not pay off its overhead. Our response to this challenge is to use approximation. As in many data retrieval scenarios, insisting on exact answers is unnecessary and answers in close vicinity of the exact ones can be just good enough. Specifically, given a set of users, if a sizable subset of the users agree with a preference, the preference can be considered an approximate common preference. This relaxation eases the aforementioned concern regarding small clusters as more approximate common preferences lead to larger clusters. As an example, in Table 2, while c_2 does not share with c_1 the preference of *Apple* over *Samsung*, its preference does not oppose it either. We can consider "*Apple* over *Samsung*" as an *approximate common preference*. A possible set of approximate common preferences of c_1 and c_2 form the strict partial orders in the row for virtual user $\widehat{U}$.

Alive objects Objects can have limited lifetime. The trends in social networks and news media change rapidly. Similarly, in any inventory, products become unavailable over time. In these scenarios users look for *alive* objects only. To meet this real-world requirement, we further extend our algorithms to operate under the semantics of a *sliding window* and thus to disseminate an object only during its lifespan.

In summary, the contributions of this paper are as follows:

- We study the problem of continuous object dissemination and formalize it as finding Pareto-optimal objects regarding partial orders. Given a large number of users and continuously arriving objects, our goal is to swiftly disseminate a newly arrived object to a user if the user's preferences—modeled as strict partial orders on individual attributes—approve the object as Pareto-optimal.
- We devise efficient solutions exploiting shared computation across similar preferences of different users.
- We study the novel challenge of clustering user preferences represented as strict partial orders. Particularly we design similarity measures for such preferences.
- To address performance degradation due to small clusters, we present an approximate similarity measure that achieves high efficiency and accuracy of answers.
- We extend our proposed solutions to deal with Pareto frontier maintenance under sliding window.
- We conduct extensive experiments using simulations on two real datasets (a movie dataset and a publication dataset). The results demonstrate clear strengths of our solutions in comparison with baselines, in terms of execution time and efficacy.

2 RELATED WORK

Pareto-optimality is a subject of extensive investigation. Its study in the computing fields can be dated back to *admissible points* [1] and *maximal vectors* [13]. Börzsönyi et al. [2] introduced the concept of skyline—a special case of Pareto frontier—in which all attributes are numeric and amenable to total orders. Kießling [12] defined preferences as strict partial orders on which preference queries operate. After that, several studies specialized on skyline query evaluation over categorical attributes [3, 17, 18, 30], among which [17, 18, 30] particularly considered query answer maintenance and only [17, 30] allow partial orders on attribute values. Nevertheless, they all consider only one user and none utilizes shared computation across multiple users' partial orders.

Given a set of objects, Wong et al. [25–27] identify the minimum set of preference relations that preclude an object from being in the Pareto frontier. This minimum set is the combination of each possible preference relation with regard to the values of all unique objects in the set. In case of any update in the object set, the minimum disqualifying condition must be recomputed. Hence, it is not designed for continuously arriving objects.

Vlachou et al. [23, 24] and Yu et al. [29] aimed at finding all users who view a given object as one of their top-k favourites, i.e., the results of a reverse top-k query. Dellis et al. [6] studied reverse skyline query—selecting users to whom a given object is in the skyline. These works consider only numeric attributes. There is no clear way to extend them for categorical attributes or even partial orders.

All these studies, while about object dissemination, focused on different aspects of the problem than ours. Particularly, no previous studies on Pareto frontier maintenance have exploited shared computation across users' preferences. Besides, as Sec. 5 shall explain, no prior work studied similarity measures for partial orders or how to cluster partial orders.

3 PROBLEM STATEMENT

Consider a set of users C and a table of objects O that are described by a set of attributes $\mathcal{D}$. For each user $c \in C$, their preference regarding O is represented by strict partial orders. For each attribute $d \in \mathcal{D}$, the strict partial order corresponding to c's preference on d is a binary relation over $dom(d)$—the domain of d, as follows.

Definition 3.1 (Preference Relation and Tuple). Given a user $c \in C$ and an attribute $d \in \mathcal{D}$, the corresponding *preference relation* is denoted $\succ_c^d$. For two attribute values $x, y \in dom(d)$, if (x, y) belongs to $\succ_c^d$ (i.e., $(x, y) \in \succ_c^d$, also denoted $x \succ_c^d y$), it is called a *preference tuple*. It is interpreted as "user c prefers x to y on attribute d". A preference relation is irreflexive ($(x, x) \notin \succ_c^d$) and transitive ($(x, y) \in \succ_c^d \wedge (y, z) \in \succ_c^d \Rightarrow (x, z) \in \succ_c^d$), which together also imply asymmetry ($(x, y) \in \succ_c^d \Rightarrow (y, x) \notin \succ_c^d$). △

Definition 3.2 (Object Dominance). A user c's preferences regarding all attributes induce another strict partial order $\succ_c$ that represents c's preferences on objects. Given two objects $o, o' \in O$, c prefers o' to o if o' is identical or preferred to o on all attributes and o' is preferred to o on at least one attribute. More formally, $o' \succ_c o$ (called o' *dominates* o), if and only if $(\forall d \in \mathcal{D} : o.d = o'.d \vee o'.d \succ_c^d o.d) \wedge (\exists d \in \mathcal{D} : o'.d \succ_c^d o.d)$. If $(\forall d \in \mathcal{D} : o.d = o'.d)$, we say that o and o' are *identical*, denoted as $o = o'$. △

Definition 3.3 (Pareto Frontier). An object o is *Pareto-optimal* with respect to c, if no other object in O dominates it. The set of *Pareto-optimal objects* (i.e., the *Pareto frontier*) in O for c is denoted $\mathcal{P}_c$, i.e., $\mathcal{P}_c = \{o \in O | \nexists o' \in O \text{ s.t. } o' \succ_c o\}$. Note that the concept of skyline points [2] is a specialization of the more general Pareto frontier, in that the preference relations for skyline points are defined as total orders (with ties) instead of general strict partial orders. △

Definition 3.4 (Target Users). Given an object o, the set of all users for whom o belongs to their Pareto frontiers are called the *target users*. The target user set is denoted C_o, i.e., $C_o = \{c \in C | o \in \mathcal{P}_c\}$. △

Example 3.5. Consider Table 1 and Table 2. $O = \{o_1, o_2, \ldots, o_{15}\}$ (ignore o_{16} for now), $C = \{c_1, c_2\}$, and $\mathcal{D} = \{\text{display}, \text{brand}, \text{CPU}\}$. With respect to c_1, (10−12.9, 16−18.9), (*Apple*, *Samsung*) and (*dual*, *triple*) are some of the preference tuples on attributes display, brand and CPU, respectively. Similarly, for c_2, (16−18.9, 19−*up*), (*Toshiba*, *Sony*) and (*triple*, *dual*) are some sample preference tuples.

$\mathcal{P}_{c_1} = \{o_2\}$, since all other objects are dominated by o_2 with respect to c_1. $\mathcal{P}_{c_2} = \{o_2, o_3, o_{15}\}$, as o_2, o_3 and o_{15} dominate $\{o_1,$

o_4, o_6, o_8, o_9, o_{13}}, {o_4, o_6, o_8, o_{13}} and {o_4, o_5, o_7, o_{10}, o_{11}, o_{12}, o_{14}}, respectively. Therefore, $C_{o_2} = \{c_1, c_2\}$ and $C_{o_3} = C_{o_{15}} = \{c_2\}$. Objects other than o_2, o_3, o_{15} do not have target users in C, i.e., $C_o = \phi$, $\forall o \in O - \{o_2, o_3, o_{15}\}$. △

Problem Statement The problem of *continuous monitoring of Pareto frontiers* is, given a set of users C, their preference relations on attributes $\mathcal{D}$, and a set of continuously growing objects O with the latest object o, find C_o—the target users of o.

In this problem setting, we assume a sizable preference relation is available for each user. In reality, we have insufficient information about the preferences of a less active user, i.e., the corresponding partial orders may contain very few preference tuples. In the extreme case, a new user, for whom we have no information regarding their preferences, admits all objects as Pareto-optimal. Such less active users and new users are the subject of the well-known *cold-start* problem in recommendation systems, which is outside of the scope of this work.

4 SHARING COMPUTATION ACROSS USERS

Algorithm Baseline A simple method to our problem will check, for every user, whether a new object belongs to the corresponding Pareto frontier. The pseudo code of this approach, named Baseline, is shown in Alg. 1. Upon the arrival of a new object o, for every user c, it sequentially compares o with the current Pareto-optimal objects in $\mathcal{P}_c$. 1) If o is dominated by any o' or o is identical to o', further comparison with the remaining objects in $\mathcal{P}_c$ is skipped. In the case of o being dominated by o', o is disqualified from being a Pareto-optimal object; if o is identical to o', then o is Pareto-optimal, i.e., it is inserted into $\mathcal{P}_c$. 2) If o dominates any o', o' is discarded from $\mathcal{P}_c$. It can be concluded already that o belongs to $\mathcal{P}_c$, but the comparisons should continue since o may dominate other existing objects in $\mathcal{P}_c$. 3) If o is not dominated by any object in $\mathcal{P}_c$, it becomes an element of $\mathcal{P}_c$. Readers familiar with the literature on skyline queries may have realized that the gist of the algorithm is essentially the basic skyline query algorithm [2]. The crux of its operation is based on an important property, that it suffices to compare new objects with only the Pareto-optimal objects, since any new object dominated by a non Pareto-optimal object must be dominated by some Pareto-optimal objects too.

Algorithm 1: Baseline

Input: C: all users; O: existing objects; o: a new object
Output: C_o: target users of o

```
1 C_o ← ∅;
2 foreach c ∈ C do
3 |   updateParetoFrontier(c, o);
4 return C_o;
```

Procedure: updateParetoFrontier (c, o)

```
1 isPareto ← true;
2 foreach o' ∈ P_c do
3 |   if o ≻_c o' then
4 |   |   P_c ← P_c − {o'}; C_o' ← C_o' − {c};
5 |   else if o' ≻_c o then isPareto ← false;break ;
6 |   else if o'.D = o.D then isPareto ← true;break ;
7 if isPareto then
8 |   P_c ← P_c ∪ {o}; C_o ← C_o ∪ {c};
```

With regard to a user c, the complexity of finding the Pareto frontier among n objects is $O(n^2)$. Alg. 1 needs $O(n^2 \cdot |C|)$ time to compute the Pareto frontiers for all users in C. The drawback of Baseline is it repeatedly applies the same procedure for every user. In terms of computation efficiency, the approach may become particularly unappealing when there are a large number of users and new objects constantly arrive. To counter this drawback, our idea is to share computations across the users that exhibit similar preferences. To this end, our method is simple and intuitive. If several users share a set of preference tuples, it is only necessary to compare two objects once, if they attain the attribute values in the preference tuples. If an object is dominated by another object according to these common preference tuples, it is dominated with respect to all users sharing the same preferences. This idea guarantees to filter out only "true negatives" for these users, and it only needs to further discern "false positives" for each individual user.

Definition 4.1 (Common Preference Tuple and Relation). Given a set of users $U \subseteq C$, an attribute $d \in \mathcal{D}$, and two values $x, y \in dom(d)$, if (x, y) belongs to preference relation $\succ_c^d$ for all $c \in U$, then it is called a *common preference tuple*. The set of common preference tuples of U on attribute d is denoted $\succ_U^d$, i.e., $\succ_U^d = \bigcap_{c \in U} \succ_c^d$. By definition, $\succ_U^d$ also represents a strict partial order (Theorem 4.2, proof omitted). We call it a *common preference relation*. It can be viewed as the preference of a virtual user that is denoted U. △

THEOREM 4.2. *$\succ_U^d$ is a strict partial order.* △

Since, for each d, $\succ_U^d$ is a strict partial order, the set of users' preferences (i.e., the virtual user U's preferences) regarding all attributes in $\mathcal{D}$ induce another strict partial order $\succ_U$ on objects.

Definition 4.3 (Pareto Frontier for U). An object o is *Pareto-optimal* with respect to U if no other object dominates it according to $\succ_U$. The Pareto frontier of O for U is denoted $\mathcal{P}_U$, i.e., $\mathcal{P}_U = \{o \in O | \nexists o' \in O \text{ s.t. } o' \succ_U o\}$. △

Example 4.4. From Table 2, $\succ_{c_1}^{\mathsf{CPU}} = \{$(*dual*, *single*), (*dual*, *quad*), (*dual*, *triple*), (*triple*, *single*), (*quad*, *single*)$\}$ and $\succ_{c_2}^{\mathsf{CPU}} = \{$(*dual*, *single*), (*triple*, *single*), (*quad*, *single*), (*triple*, *dual*), (*quad*, *dual*), (*quad*, *triple*)$\}$. According to Def. 4.1, the common preference relation of c_1 and c_2 is $\succ_{\{c_1, c_2\}}^{\mathsf{CPU}} = \{$(*dual*, *single*), (*triple*, *single*), (*quad*, *single*)$\}$. Similarly we can derive $\succ_{\{c_1, c_2\}}^{\mathsf{display}}$ and $\succ_{\{c_1, c_2\}}^{\mathsf{brand}}$. In Table 2, the three partial orders are depicted in a row labeled as a virtual user U. The Pareto frontier of U is $\mathcal{P}_U = \{o_2, o_3, o_{10}, o_{15}\}$. △

THEOREM 4.5. *Given any set of users U, for all $c \in U$, $\mathcal{P}_U \supseteq \mathcal{P}_c$ and $\overline{\mathcal{P}}_U \subseteq \bigcap_{c \in U} \overline{\mathcal{P}}_c$.* △

Proof: We prove by contradiction. Suppose that there exists $c \in U$ such that $\mathcal{P}_U \not\supseteq \mathcal{P}_c$, which would mean there exists $o \in O$ such that $o \in \mathcal{P}_c$ and $o \notin \mathcal{P}_U$. That implies the existence of an $o' \in O$ such that $o' \succ_U o$ and $o' \not\succ_c o$. However, by Def. 4.1, $o' \succ_U o$ implies $o' \succ_c o$. Therefore, the existence of o' is impossible. This contradiction eventually leads to that $\mathcal{P}_U \supseteq \mathcal{P}_c$. Hence, $\mathcal{P}_U \supseteq \bigcup_{c \in U} \mathcal{P}_c$, which implies $\overline{\mathcal{P}}_U \subseteq \bigcap_{c \in U} \overline{\mathcal{P}}_c$ according to De Morgan's laws.

LEMMA 4.6. *Given any set of users U, for all $c \in U$, $\mathcal{P}_c = \{o \in \mathcal{P}_U | \nexists o' \in \mathcal{P}_U \text{ s.t. } o' \succ_c o\}$.* △

Example 4.7. In Table 2, $\mathcal{P}_U = \{o_2, o_3, o_{10}, o_{15}\}$ and $\mathcal{P}_{c_1} \cup \mathcal{P}_{c_2} = \{o_2, o_3, o_{15}\}$. $\mathcal{P}_U \supseteq \mathcal{P}_{c_1} \cup \mathcal{P}_{c_2}$. Moreover, $\overline{\mathcal{P}}_U = \{o_1, o_4, o_5, o_6, o_7, o_8, o_9, o_{11}, o_{12}, o_{13}, o_{14}\}$ and $\overline{\mathcal{P}}_{c_1} \cap \overline{\mathcal{P}}_{c_2} = \{o_1, o_4, o_5, o_6, o_7, o_8, o_9, o_{10}, o_{11}, o_{12}, o_{13}, o_{14}, o_{15}\}$. $\overline{\mathcal{P}}_U \subseteq \overline{\mathcal{P}}_{c_1} \cap \overline{\mathcal{P}}_{c_2}$. △

Theorem 4.5 suggests an appealing quality of the common preference relations of U. By $\mathcal{P}_U \supseteq \mathcal{P}_c$, the Pareto frontier of U subsumes the Pareto frontier of every user member in U. What it means is that, if we simply compute the Pareto frontier of U, we get to retain all the objects that we eventually look for. Consider $\mathcal{P}_c$ as the ground truth and $\mathcal{P}_U$ as the predictions. The objects that are filtered out ($\overline{\mathcal{P}}_U$) are all "true negatives" and there are no "false negatives". The set $\mathcal{P}_U$ may contain "false positives", which we just need to throw out after further verification, as Lemma 4.6 suggests.

This approach's merit is the potential saving on object comparisons. For a cluster of users, many non Pareto-optimal objects may be filtered out altogether for all the users, without incurring the same comparisons repeatedly for each user.

To capitalize on the above ideas, our method must answer three questions. (1) How to find users sharing similar preferences? (2) For a set of similar users U, how to maintain the corresponding Pareto frontier $\mathcal{P}_U$ based on their common preference relations $\succ_U^d$ for different attributes d? (3) For each user c in U, how to discern the "false positives" in $\mathcal{P}_U$ and thus find $\mathcal{P}_c$. Note that the second and the last challenges need to be addressed for constantly arriving new objects.

For (1), our method is to cluster users based on the similarity between their preference relations. While many clustering methods have been developed for various types of data, none is specialized in clustering partial orders. Our clustering method is discussed in Sec. 5. For (2) and (3), our algorithm takes a *filter-then-verify* approach and is thus named FilterThenVerify, of which the pseudo code is displayed in Alg 2.

Alg. FilterThenVerify Upon the arrival of a new object o, for every cluster U, FilterThenVerify compares o with the current members of $\mathcal{P}_U$ based on the preference relations of the virtual user U. Various actions are taken, depending on the comparison outcomes, as follows:

I) If o dominates any o' in $\mathcal{P}_U$ according to $\succ_U^d$ of all relevant d, o' is removed from $\mathcal{P}_U$ (Line 7 of Procedure updateParetoFrontierU in Alg. 2). For every $c \in C$ such that $o' \in \mathcal{P}_c$, o' is also discarded from $\mathcal{P}_c$ (Line 6 of Procedure updateParetoFrontierU).

II) If o is dominated by any o' in $\mathcal{P}_U$, then o does not occupy the Pareto frontier of any user in U (Theorem 4.5). Further operations involving o are unnecessary (Line 8 of Procedure updateParetoFrontierU).

III) After comparing o with all current objects in $\mathcal{P}_U$, if it is realized that o is not dominated by any o', then o becomes a member of $\mathcal{P}_U$ (Line 9 of updateParetoFrontierU). Furthermore, for each $c \in U$, o is further compared with the members of $\mathcal{P}_c$ based on the preference relations of c, by using Procedure updateParetoFrontier of Alg.1 (Line 6 of Alg.2).

Example 4.8. In this example we explain the execution of FilterThenVerify on Table 1 and Table 2. Suppose users c_1 and c_2 form a cluster U, of which the preference relations are depicted in Table 2. The existing objects are o_1 to o_{14}, and $o_{15} = \langle 16.5'', \textit{Lenovo}, \textit{quad}\rangle$ is the object that just becomes available. Before o_{15} arrives, the Pareto frontier of U is $\mathcal{P}_U = \{o_2, o_3, o_7, o_{10}\}$. The algorithm starts by comparing o_{15} with each element in $\mathcal{P}_U$. As o_{15} dominates $o_7 = \langle 9.5'', \textit{Lenovo}, \textit{quad}\rangle$ according to U's preference relations, o_7 is discarded from $\mathcal{P}_U$. Before o_{15} arrives, o_7 belongs to $\mathcal{P}_{c_2}$ and $C_{o_7}=\{c_2\}$. Hence, o_7 is removed from $\mathcal{P}_{c_2}$ and C_{o_7} becomes empty. o_{15} does not dominate any other object in $\mathcal{P}_U$. It is not dominated by any either. Therefore, it is inserted into $\mathcal{P}_U$.

o_{15} is further compared with the existing members of $\mathcal{P}_{c_1}$ and $\mathcal{P}_{c_2}$. It is dominated by $o_2=\langle 14'', \textit{Apple}, \textit{dual}\rangle$ according to c_1's preference relations. Thus it is not part of $\mathcal{P}_{c_1}$. According to c_2's preferences, o_{15} does not dominate any existing Pareto optional object (except the aforementioned o_7 which by now is already discarded). Therefore $\mathcal{P}_{c_2}$ is not further changed and o_{15} becomes part of $\mathcal{P}_{c_2}$. Overall, $C_{o_{15}}=\{c_2\}$.

Moreover, consider the arrival of $o_{16} = \langle 16'', \textit{Toshiba}, \textit{single}\rangle$ after o_{15}. In the process of comparing o_{16} with $\mathcal{P}_U = \{o_2, o_3, o_{10}, o_{15}\}$, it is realized that o_{16} is dominated by o_2 according to U's preference relations. Therefore, it does not belong to $\mathcal{P}_U$. It is thus unnecessary to further compare o_{16} with $\mathcal{P}_{c_1}$ or $\mathcal{P}_{c_2}$. $C_{o_{16}}=\emptyset$. Thereby, updateParetoFrontierU acts as a sieve to filter out non Pareto-optimal objects such as o_{16}. In this way FilterThenVerify reduces computation cost by avoiding repeated comparisons with such objects. △

Algorithm 2: FilterThenVerify

Input: $U_1, U_2, ..., U_n$: clusters of users; O: existing objects; o: a new object
Output: C_o: target users of o

```
1  C_o ← ∅;
2  foreach U ∈ {U_1, U_2, ..., U_n} do
3      isPareto ← updateParetoFrontierU(U, o);
4      if isPareto then
5          foreach c ∈ U do
6              updateParetoFrontier(c, o); //Algorithm 1
7  return C_o;
```

Procedure: updateParetoFrontierU (U, o)

```
1  isPareto ← true;
2  foreach o' ∈ P_U do
3      if o ≻_U o' then
4          foreach c ∈ U do
5              if o' ∈ P_c then
6                  P_c ← P_c − {o'}; C_o' ← C_o' − {c};
7          P_U ← P_U − {o'};
8      else if o' ≻_U o then isPareto ← false; break ;
9  if isPareto then P_U ← P_U ∪ {o} ;
10 return isPareto;
```

Complexity Analysis of Alg. 2 As we discussed earlier, given a user c, the complexity of finding Pareto frontier among the n objects is $O(n^2)$. Assume k is the number of clusters. With regard to the virtual user for each cluster U, the complexity of finding Pareto frontier $\mathcal{P}_U$ among the n objects is $O(n^2 \cdot k)$ (calling Procedure updateParetoFrontierU in Line 3 of Alg. 2). Assume each $\mathcal{P}_U$ on average has m objects. In Lines 4-6, Alg. 2 finds $\mathcal{P}_c$ from $\mathcal{P}_U$ for each user c in U (recall that $\mathcal{P}_U \supseteq \mathcal{P}_c$). As Lines 4-6 iterate for each cluster (Line 2), the algorithm eventually computes $\mathcal{P}_c$ for each $c \in C$. Therefore, the complexity of finding Pareto frontier $\mathcal{P}_c$ among the m objects is $O(m^2 \cdot |C|)$. Overall, FilterThenVerify needs $O(n^2 \cdot k + m^2 \cdot |C|)$ time to find the target users for all objects. We compare FilterThenVerify and Baseline in terms of time complexity. Apparently $k < |C|$ and $m < n$. Thus, $n^2 \cdot k < n^2 \cdot |C|$ and $m^2 \cdot |C| < n^2 \cdot |C|$.

5 SIMILARITY MEASURES FOR CLUSTERING USER PREFERENCES

This section discusses how to cluster users based on their preference relations. Our focus is on the similarity measures rather than the clustering method. The method we adopt is the conventional hierarchical agglomerative clustering algorithm [9]. At every iteration, the method merges the two most similar clusters. The common preference relation of the merged cluster U on each attribute d, i.e., $\succ^d_U$, is computed. It then calculates the similarity between U and each remaining cluster. Given two clusters U_1 and U_2, their similarity $sim(U_1, U_2)$ is defined as the summation of the similarities between their preference relations on individual attributes, as follows. This resembles the high-level idea of using L_1 norm distance between centroids for measuring inter-cluster similarity in conventional hierarchial clustering.

$$sim(U_1, U_2) = \sum_{d \in \mathcal{D}} sim^d(U_1, U_2) \tag{1}$$

Individual users' and clusters' preference relations on attributes are strict partial orders. No prior work studied clustering approaches or similarity measures for partial orders. Similarity measures commonly used in clustering algorithms assume numeric or categorical attributes. Kamishima et al. [10, 11] and Ukkonen et al. [22] cluster total orders but not partial orders. Given two totally ordered attributes, these works use the comparative ranks of the corresponding values to measure similarity. Clearly, such similarity measures are not applicable for partially ordered attributes.

In this section we propose four different similarity functions for defining $sim^d(U_1, U_2)$.

1) Intersection size This is simply the size of the intersection of $\succ^d_{U_1}$ and $\succ^d_{U_2}$, i.e., the number of common preference tuples of all users in the two clusters U_1 and U_2. It is defined as

$$sim^d_i(U_1, U_2) = |\succ^d_{U_1} \cap \succ^d_{U_2}| \tag{2}$$

Example 5.1. Table 3 shows three clusters U_1 ($\{c_1, c_2\}$), U_2 ($\{c_3, c_4\}$), and U_3 ($\{c_5, c_6\}$) and the common preference relation associated with each cluster on attribute brand. U_1 and U_2 do not share any preference tuple and thus $sim^{\mathsf{brand}}_i(U_1, U_2) = 0$. U_1 and U_3 have (*Apple*, *Samsung*) and (*Lenovo*, *Samsung*) as common preference tuples, i.e., $sim^{\mathsf{brand}}_i(U_1, U_3) = 2$. Similarly, U_2 and U_3 share (*Lenovo*, *Apple*) and (*Lenovo*, *Toshiba*), i.e., $sim^{\mathsf{brand}}_i(U_2, U_3) = 2$. △

2) Jaccard similarity The measure sim_i captures the absolute size of the intersection of two preference relations. It does not take into account their differences. Consider three clusters U_1, U_2 and U_3 such that $sim^d_i(U_1, U_2) = sim^d_i(U_1, U_3)$ (i.e., $|\succ^d_{U_1} \cap \succ^d_{U_2}| = |\succ^d_{U_1} \cap \succ^d_{U_3}|$) and $|\succ^d_{U_1} \cup \succ^d_{U_2}| < |\succ^d_{U_1} \cup \succ^d_{U_3}|$. We can argue that the similarity between U_1 and U_2 should be higher than (instead of equal to) that between U_1 and U_3, because U_1 and U_2 have a larger percentage of common preference tuples than U_1 and U_3. To address this limitation of sim_i, we define the *Jaccard similarity* between two preference relations as their intersection size over their union size, i.e., the ratio of common preference tuples to all preference tuples in the two preference relations. Formally,

$$sim^d_j(U_1, U_2) = \frac{|\succ^d_{U_1} \cap \succ^d_{U_2}|}{|\succ^d_{U_1} \cup \succ^d_{U_2}|} = \frac{sim^d_i(U_1, U_2)}{|\succ^d_{U_1} \cup \succ^d_{U_2}|} \tag{3}$$

Example 5.2. Continue Example 5.1. $\succ^{\mathsf{brand}}_{U_1}$ and $\succ^{\mathsf{brand}}_{U_3}$ have 6 preference tuples in total while $\succ^{\mathsf{brand}}_{U_2}$ and $\succ^{\mathsf{brand}}_{U_3}$ have 7. Thus, $sim^{\mathsf{brand}}_j(U_1, U_3)$=2/6 and $sim^{\mathsf{brand}}_j(U_2, U_3)$=2/7. △

3) Weighted intersection size Intersection size and Jaccard similarity are based on the cardinalities of intersection and union sets of preference relations. In counting the cardinalities, they both treat all preference tuples equal. We argue that this is counter-intuitive. Values at the top of a partial order matter more than those at the bottom, in terms of their impact on which objects belong to the Pareto frontier. Accordingly we introduce *weighted intersection size*, a modified version of intersection size sim_i. In counting the common preference tuples of two preference relations, it assigns a weight to each preference tuple. Formally,

$$sim^d_{wi}(U_1, U_2) = \sum_{(v, v') \in \succ^d_{U_1} \cap \succ^d_{U_2}} \frac{1}{2} \times \left(\frac{1}{\min_{s \in S^d_{U_1}} D(s, v)+1} + \frac{1}{\min_{s \in S^d_{U_2}} D(s, v)+1} \right) \tag{4}$$

In the above equation, with regard to an attribute d, the similarity between two clusters' preference relations is a summation over their common preference tuples. For each common preference tuple (v, v'), it computes the average weight of the better value v with respect to U_1 and U_2, respectively. Given a cluster U, S^d_U is the set of *maximal values* in the partial order $\succ^d_U$ and $D(s, v)$ for each $s \in S^d_U$ is the shortest distance from s to v in $\succ^d_U$. The weight of v in U is the inverse of the minimal distance from any maximal value to v (plus 1, to avoid division by zero). The concept of maximal value is defined as follows.

Definition 5.3 (Maximal Value). With regard to $\succ^d_U$, value $x \in dom(d)$ is a *maximal value* if no other value in $dom(d)$ is preferred over x. The set of maximal values for $\succ^d_U$ is denoted S^d_U. Formally, $S^d_U = \{x \in dom(d) \mid \nexists y \in dom(d)\ s.t.\ (y, x) \in \succ^d_U\}$. △

Example 5.4. Continue Example 5.1. The maximal values in $\succ^{\mathsf{brand}}_{U_1}$, $\succ^{\mathsf{brand}}_{U_2}$ and $\succ^{\mathsf{brand}}_{U_3}$ are $S^{\mathsf{brand}}_{U_1}$=$\{$*Apple*, *Toshiba*$\}$, $S^{\mathsf{brand}}_{U_2}$ = $\{$*Samsung*$\}$ and $S^{\mathsf{brand}}_{U_3}$=$\{$*Lenovo*$\}$, respectively. In the partial order corresponding to $\succ^{\mathsf{brand}}_{U_1}$, the minimal shortest distances to *Apple*, *Lenovo*, *Samsung*, and *Toshiba* from the maximal values $\{$*Apple*, *Toshiba*$\}$ are 0, 1, 1 and 0, respectively. The corresponding weights are 1, 1/2, 1/2 and 1. Similarly, in $\succ^{\mathsf{brand}}_{U_2}$, the weights of *Apple*, *Lenovo*, *Samsung* and *Toshiba* are 1/3, 1/2, 1 and 1/3, respectively. In $\succ^{\mathsf{brand}}_{U_3}$, the corresponding weights are 1/2, 1, 1/3 and 1/2, respectively.

U_1 and U_3 have (*Apple*, *Samsung*) and (*Lenovo*, *Samsung*) as common preference tuples. For the two better-values in these preference tuples—*Apple* and *Lenovo*, the average weights are both 3/4. The similarity $sim^{\mathsf{brand}}_{wi}(U_1, U_3)$=$\frac{1+\frac{1}{2}}{2} + \frac{\frac{1}{2}+1}{2} = \frac{3}{2}$. Similarly, U_2 and U_3 have (*Lenovo*, *Apple*) and (*Lenovo*, *Toshiba*) as common preference tuples. In U_2 and U_3, the average weight of *Lenovo*—the better-value in both common preference tuples—is 3/4. The similarity $sim^{\mathsf{brand}}_{wi}(U_2, U_3)$=$\frac{\frac{1}{2}+1}{2} + \frac{\frac{1}{2}+1}{2} = \frac{3}{2}$. △

4) Weighted Jaccard similarity This measure is a combination of the last two ideas—Jaccard similarity and weighted intersection size. As in Jaccard similarity, *weighted Jaccard similarity* computes the ratio of intersection size to union size. Similar to weighted intersection size, the values in a preference relation are assigned weights corresponding to their minimal shortest distances to the preference relation's maximal values. The measure's definition is as follows.

$$sim^d_{wj}(U_1, U_2) = \sum_{(v,v')\in >^d_{U_1} \cap >^d_{U_2}} \frac{1}{2} \times \left(\frac{1}{\min_{s\in S^d_{U_1}} D(s, v)+1} + \frac{1}{\min_{s\in S^d_{U_2}} D(s, v)+1}\right) \Big/ \sum_{(v,v')\in >^d_{U_1} \cup >^d_{U_2}} \frac{1}{2} \times \left(\frac{1}{\min_{s\in S^d_{U_1}} D(s, v)+1} + \frac{1}{\min_{s\in S^d_{U_2}} D(s, v)+1}\right)$$

$$= sim^d_{wi}(U_1, U_2) \Big/ \left[sim^d_{wi}(U_1, U_2) + \sum_{(v,v')\in >^d_{U_1} -> ^d_{U_2}} \frac{1}{\min_{s\in S^d_{U_1}} D(s, v) + 1} + \sum_{(v,v')\in >^d_{U_2} -> ^d_{U_1}} \frac{1}{\min_{s\in S^d_{U_2}} D(s, v) + 1}\right] \quad (5)$$

Example 5.5. Continue Example 5.4. Now $sim^{\text{brand}}_{wj}(U_1, U_3) = \frac{\frac{3}{2}}{(1+1)+(1+1)+\frac{3}{2}} = \frac{3}{11}$, since $>^d_{U_1} -> ^d_{U_3} = \{(Apple, Lenovo), (Toshiba, Samsung)\}$ and $>^d_{U_3} -> ^d_{U_1} = \{(Lenovo, Apple), (Lenovo, Toshiba)\}$. Similarly, $sim^{\text{brand}}_{wj}(U_2, U_3) = \frac{\frac{3}{2}}{(1+1+1)+(1+\frac{1}{2})+\frac{3}{2}} = \frac{3}{12}$, as $>^d_{U_2} -> ^d_{U_3} = \{(Samsung, Lenovo), (Samsung, Apple), (Samsung, Toshiba)\}$ and $>^d_{U_3} -> ^d_{U_2} = \{(Lenovo, Samsung), (Apple, Samsung)\}$. Note that $sim^{\text{brand}}_{wj}(U_1, U_3) > sim^{\text{brand}}_{wj}(U_2, U_3)$ although $sim^{\text{brand}}_{wi}(U_1, U_3) = sim^{\text{brand}}_{wi}(U_2, U_3)$. △

6 APPROXIMATE USER PREFERENCES

Two conflicting factors have crucial impacts on the effectiveness of FilterThenVerify. One is the size of the common preference relations. The other is the size of the clusters. Specifically, the more preference tuples a cluster's users share, the more objects can be filtered out and thus the less verifications need to be done for individual users. On the contrary, the more users a cluster contains, the more repeated comparisons are avoided for these individual users. There is a clear tradeoff between these two factors, since larger clusters (i.e., more users in each cluster) naturally leads to smaller common preference relations.

Our approach to this challenge is *approximation*. As discussed in Sec. 1, it suffices for many applications to approximately identify target users. In this section, we show that we can find such approximation through a relaxed notion of common preference tuple, namely *approximate common preference tuple*. For a set of users, it allows a preference tuple to be absent from a tolerably small subset. If a sizable subset of the users agree with the preference tuple, it is considered an approximate common preference tuple. This relaxation addresses the aforementioned concern, since more approximate common preferences lead to larger clusters.

6.1 Approximate Common Preference Tuples and Relations

Based on the aforementioned objective, we procedurally construct approximate common preference relations. Before we provide its formal definition, we explain the intuition, as follows. Given a cluster of users, the resulting approximate common preference relation always includes the common preference tuples. The remaining possible preference tuples are considered in descending order of their frequencies, since preference tuples with higher frequencies are shared by more users. A preference tuple is included into the approximate common preference relation only if its reverse tuple is not included. This guarantees asymmetry. Furthermore, when a preference tuple is included, the transitive closure of the updated approximate common preference relation is also included. This guarantees transitivity. Irreflexivity is guaranteed too since this procedure never considers preference tuples in the form of (x, x). These altogether assure the constructed preference relation is a strict partial order. Given an append-only database of objects, a strict partial order ensures that preference query results are independent of the order by which objects are appended to the database. We denote the approximate common preference relation by $\widehat{\succ}^d_U$. It can be viewed as the preference of a virtual user (denoted $\widehat{U}$) on attribute d. Moreover, we denote the Pareto frontier of O for $\widehat{U}$ as $\widehat{\mathcal{P}}_U$.

Definition 6.1 (Approximate Common Preference Tuple and Relation). Given a set of users $U \subseteq \mathcal{C}$, an attribute $d \in \mathcal{D}$ of which $|dom(d)|=m$, consider $A_{1\ldots P_2^m}$ which is an ordered permutation of all possible preference tuples $\{(x, y) \in dom(d) \times dom(d) \mid x \neq y\}$ such that $freq(A_i) \geq freq(A_{i+1})$ for $i \in [1, P_2^m - 1]$, in which $freq(A_i)$ denotes the percentage of users in U whose preference relations contain preference tuple A_i. The *approximate common preference relation* $\widehat{\succ}^d_U$ is defined as R_j in which j is the largest index $i \in [1, P_2^m]$ that satisfies the condition $(|R_i| < \theta_1 \wedge freq(A_i) > \theta_2) \vee freq(A_i) = 1$ where R_i is defined as

$$R_i = \begin{cases} \{A_1\} & \text{if } i = 1 \\ (R_{i-1} \cup \{A_i\})^+ & \text{if } R_{i-1} \cup \{A_i\} \text{ is a strict partial order} \\ R_{i-1} & \text{otherwise} \end{cases}$$

and θ_1 and θ_2 are two given thresholds. θ_1 limits the size of the resulting $\widehat{\succ}^d_U$ while θ_2 excludes infrequent preference tuples from $\widehat{\succ}^d_U$. △

θ_1 and θ_2 regulate the size of $\widehat{\succ}^d_U$. A pair of large θ_1 and small θ_2 allows $\widehat{\succ}^d_U$ to include infrequent preference tuples. In such a case the approximate common preference relation becomes ineffective, since Procedure updateParetoFrontierU in Alg.2 may retain a large number of candidates that must be verified for each $c \in U$. On the other hand, a pair of small θ_1 and large θ_2 may limit $\widehat{\succ}^d_U$ to contain only $\succ^d_U$, in which case the concern regarding small common preference relation remains.

As Def. 6.1 itself is procedural, it naturally corresponds to a greedy algorithm for constructing approximate preference relation $\widehat{\succ}^d_U$. The pseudo code GetApproxPreferenceTuples is in Alg. 3. First, all the common preference tuples are included (Lines 2-3). After that, preference tuples are considered in the order of frequency, as long as the two thresholds are satisfied (Line 4). For each preference tuple in consideration, if it together with all chosen tuples hitherto do not violate the properties of a strict partial order, their transitive closure is included into the approximate preference relation (Lines 6-7).

Example 6.2. We use Figure 1 to explain the execution of GetApproxPreferenceTuples. Figure 1a depicts three users' preference relations on brand. Suppose together these three users form a cluster. Assume $\theta_1 = 7$ and $\theta_2 = 60\%$.

Table 4 shows the frequencies of all possible preference tuples after sorting. For instance, since all users prefer *Apple* to *Toshiba*, the corresponding frequency is 3/3; the frequency of (*Apple*, *Samsung*) is 2/3 as two of these three users prefer *Apple* to *Samsung*. At first GetApproxPreferenceTuples includes the common preference tuple (*Apple*, *Toshiba*) into $\widehat{\succ}^d_U$. It then includes (*Apple*, *Samsung*), (*Lenovo*, *Toshiba*), and (*Toshiba*, *Samsung*) as approximate preference tuples too. Furthermore, upon the addition of (*Toshiba*, *Samsung*), GetApproxPreferenceTuples includes (*Lenovo*, *Samsung*) as well since (*Lenovo*, *Toshiba*) and

Algorithm 3: GetApproxPreferenceTuples

Input: A_i: ordered permutation of all possible preference tuples, defined on $dom(d)$, in descending order of their frequencies among users U, θ_1 and θ_2: thresholds

Output: $\widehat{\succ}_U^d$: approximate common preference relation of U on attribute d

```
for i = 1 to P_2^{|dom(d)|} do
    if freq(A_i) = 1 then
        ≻̂_U^d ← ≻̂_U^d ∪ {A_i}; continue;
    if |≻̂_U^d| ≥ θ_1 or freq(A_i) ≤ θ_2 then
        break;
    if (≻̂_U^d ∪ {A_i}) is a strict partial order then
        ≻̂_U^d ← (≻̂_U^d ∪ {A_i})^+;
return ≻̂_U^d;
```

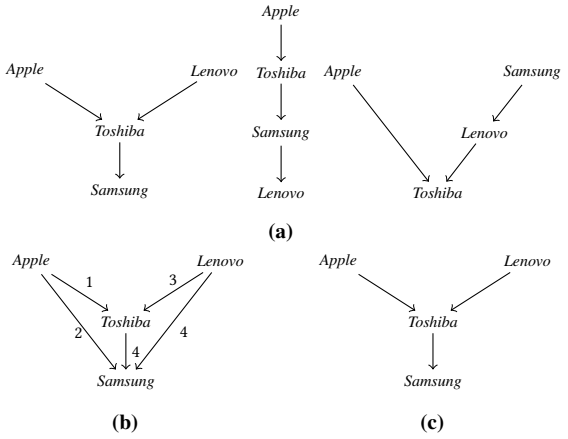


Figure 1: Execution of GetApproxPreferenceTuples. a) Input: the preferences of 3 users w.r.t. brand. b) The sequence of included approximate preference tuples. c) Output: the final Hasse diagram representation of the partial order.

(A,T)	(A,S)	(L,T)	(T,S)	(S,L)	(A,L)	(L,S)	(T,L)	(S,T)	(L,A)	(T,A)	(S,A)
3/3	2/3	2/3	2/3	2/3	1/3	1/3	1/3	1/3	0/3	0/3	0/3

Table 4: All possible preference tuples in order of frequency. (*A*, *L*, *S* and *T* stand for *Apple*, *Lenovo*, *Samsung* and *Toshiba*.)

(*Toshiba*, *Samsung*) transitively induce it. The algorithm then considers (*Samsung*, *Lenovo*), which is disqualified since its reverse tuple (*Lenovo*, *Samsung*) is already included. Otherwise the tuples will not form a strict partial order. The algorithm stops at (*Apple*, *Lenovo*) because its frequency is below the threshold 60%. Fig.1b illustrates the sequence of the included tuples and Fig.1c depicts the output approximate preference relation in the form of a Hasse diagram. △

6.2 False Positives and False Negatives due to Approximation

FilterThenVerify (Alg.2) is extended to use approximate preference tuples and thus we rename it FilterThenVerifyApprox. The algorithm itself remains the same. Procedure updateParetoFrontierU maintains $\widehat{\mathcal{P}}_U$ as the candidate Pareto frontier. The algorithm eventually returns $\widehat{\mathcal{P}}_c$ for each user $c \in U$, in which $\widehat{\mathcal{P}}_c = \{o \in \widehat{\mathcal{P}}_U | \nexists o' \in \widehat{\mathcal{P}}_U \text{ s.t. } o' \succ_c o\}$, i.e., $\widehat{\mathcal{P}}_U \supseteq \widehat{\mathcal{P}}_c$. Thus, $\widehat{C}_o = \{c \in C | o \in \widehat{\mathcal{P}}_c\}$. We use the example below to explain its execution over approximate preference relations.

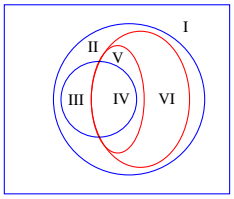


Figure 2: Venn diagram depicting $\mathcal{O}$, $\mathcal{P}_U$, $\widehat{\mathcal{P}}_U$, $\mathcal{P}_c$ and $\widehat{\mathcal{P}}_c$.

Set	Area Covered
$\mathcal{O}$	I,II,III,IV,V,VI
$\mathcal{P}_U$	II,III,IV,V,VI
$\widehat{\mathcal{P}}_U$	IV,V,VI
$\mathcal{P}_c$	III,IV
$\widehat{\mathcal{P}}_c$	IV,V

Table 5: Areas covered by $\mathcal{O}$, $\mathcal{P}_U$, $\widehat{\mathcal{P}}_U$, $\mathcal{P}_c$ and $\widehat{\mathcal{P}}_c$ in Fig.2.

		Exact	
		Pareto frontier	Non Pareto frontier
Approx.	Pareto frontier	IV	V
	Non Pareto frontier	III	I,II,VI

Table 6: Confusion matrix w.r.t. c.

Example 6.3. Reconsider Example 4.8, but use the approximate preference relations associated with virtual user $\widehat{U}$ in Table 2. Upon the arrival of o_{15}, it is compared with the elements in $\widehat{\mathcal{P}}_U = \{o_2, o_7\}$. $\widehat{\mathcal{P}}_U$ becomes $\{o_2, o_{15}\}$ since o_{15} dominates o_7. o_7 is then also removed from $\widehat{\mathcal{P}}_{c_2}$. o_{15} is further compared with $\widehat{\mathcal{P}}_{c_1} = \{o_2\}$ and $\widehat{\mathcal{P}}_{c_2} = \{o_2\}$, which does not lead to any further change. Overall, $\widehat{C}_{o_{15}} = \{c_2\}$. The target users using approximate preference relations remain identical to the exact ones, i.e., no loss of accuracy in this case. △

The rest of this section focuses on the accuracy of FilterThenVerifyApprox. It produces *false positives* if there exists such an o that $o \in \widehat{\mathcal{P}}_c$ but $o \notin \mathcal{P}_c$. It produces *false negatives* if there exists such an o that $o \notin \widehat{\mathcal{P}}_c$ but $o \in \mathcal{P}_c$. Below we present Theorems 6.5 and 6.7 to analyze how $\widehat{\mathcal{P}}_U$ and $\widehat{\mathcal{P}}_c$ relate to $\mathcal{P}_U$ and $\mathcal{P}_c$.

LEMMA 6.4. *Given a set of users U and an attribute d, the common preference relation $\succ_U^d$ and an approximate common preference relation $\widehat{\succ}_U^d$ satisfy the following properties:*

1) *The approximate preference tuples are a superset of the common preference tuples, i.e., $\widehat{\succ}_U^d \supseteq \succ_U^d$.*

2) *If any preference tuple along with its reverse tuple do not belong to the approximate common preference relation, neither of them belongs to the common preference relation either, i.e., $(x,y) \notin \widehat{\succ}_U^d \wedge (y,x) \notin \widehat{\succ}_U^d \Rightarrow (x,y) \notin \succ_U^d \wedge (y,x) \notin \succ_U^d$.* △

THEOREM 6.5. *Given objects $\mathcal{O}$ and users U, the Pareto frontier with regard to approximate common preference relations is a subset of the Pareto frontier with regard to common preference relations, i.e., $\widehat{\mathcal{P}}_U \subseteq \mathcal{P}_U$.* △

Proof: We prove by contradiction. Suppose $\widehat{\mathcal{P}}_U \nsubseteq \mathcal{P}_U$, which would mean there exists $o \in \mathcal{O}$ such that $o \in \widehat{\mathcal{P}}_U$ and $o \notin \mathcal{P}_U$. That leads to the existence of an o' such that $o' \succ_U o$ and $o' \not\succ_{\widehat{U}} o$. However, $o' \succ_U o$ implies $o' \succ_{\widehat{U}} o$ because $\widehat{\succ}_U^d \supseteq \succ_U^d$ for every d (Lemma 6.4). Therefore, the existence of o' is impossible. This contradiction proves that $\widehat{\mathcal{P}}_U \subseteq \mathcal{P}_U$. ■

LEMMA 6.6. *Given any set of users U, for all user $c \in U$, $\widehat{\mathcal{P}}_U \supseteq \widehat{\mathcal{P}}_c$.* △

THEOREM 6.7. *Given any set of users U, for all user $c \in U$, $\widehat{\mathcal{P}}_U \cap \mathcal{P}_c \subseteq \widehat{\mathcal{P}}_c$.* △

Proof: We prove by contradiction. Suppose $\widehat{\mathcal{P}}_U \cap \mathcal{P}_c \nsubseteq \widehat{\mathcal{P}}_c$, which would mean there exists $o \in \mathcal{O}$ such that $o \in \widehat{\mathcal{P}}_U \cap \mathcal{P}_c$ and $o \notin \widehat{\mathcal{P}}_c$. $o \notin \widehat{\mathcal{P}}_c$ implies the existence of an $o' \in \mathcal{O}$ such that $o' \in \widehat{\mathcal{P}}_c$ and $o' \succ_c o$ (since $o \in \widehat{\mathcal{P}}_U \cap \mathcal{P}_c$ and thus $o \in \widehat{\mathcal{P}}_U$ which

means $o' \not\succ_{\widehat{U}} o$). Since $o' \succ_c o$, $o \notin \mathcal{P}_c$ (Def. 3.3) and thus $o \notin \widehat{\mathcal{P}}_U \cap \mathcal{P}_c$. In other words, the existence of o' is impossible. This contradiction proves that $\widehat{\mathcal{P}}_U \cap \mathcal{P}_c \subseteq \widehat{\mathcal{P}}_c$. ∎

Consider a cluster U and a user $c \in U$. The Venn diagram in Fig. 2 shows the effect of approximation through depicting O (rectangle), $\mathcal{P}_U$ (outer blue circle), $\widehat{\mathcal{P}}_U$ (outer red ellipse), $\mathcal{P}_c$ (inner blue circle), and $\widehat{\mathcal{P}}_c$ (inner red ellipse). Besides, Table 5 elaborates the area covered by these sets while Table 6 shows the confusion matrix for c. Note that using approximate common preference relations results in false negatives (III). Mistakenly declaring III as not Pareto-optimal further allows false positives (V) to sneak in.

With these notations in place, we are ready to quantify the accuracy of FilterThenVerifyApprox using standard evaluation measures in information retrieval. Specifically, *precision* is the fraction of objects found by FilterThenVerifyApprox that are truly Pareto-optimal, i.e., $\frac{\sum_{c \in C} \widehat{\mathcal{P}}_c \cap \mathcal{P}_c}{\sum_{c \in C} \widehat{\mathcal{P}}_c}$. *Recall* is the fraction of Pareto-optimal objects that are correctly found by FilterThenVerifyApprox, i.e., $\frac{\sum_{c \in C} \widehat{\mathcal{P}}_c \cap \mathcal{P}_c}{\sum_{c \in C} \mathcal{P}_c}$. With regard to a specific user c, the algorithm's precision, recall and accuracy can be represented using the areas in Fig. 2, as follows.

$$precision = \frac{|\text{ IV }|}{|\text{ IV} \cup \text{V }|} \tag{6}$$

$$recall = \frac{|\text{ IV}|}{|\text{ III} \cup \text{IV }|} \tag{7}$$

$$accuracy = \frac{|\text{ I} \cup \text{II} \cup \text{IV} \cup \text{VI }|}{|\text{ I} \cup \text{II} \cup \text{III} \cup \text{IV} \cup \text{V} \cup \text{VI }|} \tag{8}$$

6.3 Similarity Functions

To make the clustering solution in Sec. 5 compatible with approximate preference relations, we extend the similarity measures, using ideas inspired by the Jaccard similarity for non-negative multidimensional real vectors [4].

1) Jaccard Similarity Consider an attribute d with $|dom(d)| = m$. For each cluster U, construct a vector $\mathbf{U} = (\mathbf{U}(1), \mathbf{U}(2), \ldots, \mathbf{U}(P_2^m))$. For $i \in [1, P_2^m]$, $\mathbf{U}(i)$ represents the frequency of A_i (Definition 6.1) in U. Given two clusters U and V, their *Jaccard similarity* on attribute d is

$$sim_j^d(U, V) = \frac{\sum_i \min(\mathbf{U}(i), \mathbf{V}(i))}{\sum_i \max(\mathbf{U}(i), \mathbf{V}(i))} \tag{9}$$

Example 6.8. Consider U_1 and U_3 in Table 3. Suppose $A(i)$ for $i \in [1, P_2^m]$ are ((*Apple*, *Lenovo*), (*Apple*, *Samsumg*), (*Apple*, *Toshiba*), (*Lenovo*, *Apple*), (*Lenovo*, *Samsung*), (*Lenovo*, *Toshiba*), (*Toshiba*, *Apple*), (*Toshiba*, *Lenovo*), (*Toshiba*, *Samsung*), (*Samsung*, *Apple*), (*Samsung*, *Lenovo*), (*Samsung*, *Toshiba*)). The two vectors are $\mathbf{U_1}$ = (2/2, 2/2, 0/2, 0/2, 2/2, 0/2, 0/2, 1/2, 2/2, 0/2, 0/2, 0/2) and $\mathbf{U_3}$ = (0/2, 2/2, 1/2, 2/2, 2/2, 2/2, 0/2, 0/2, 1/2, 0/2, 0/2, 0/2). For instance, $\mathbf{U_1}$ has 1/2 on the 8^{th}-dimension since only one of the two users' preference relations contains (*Toshiba*, *Lenovo*). Hence, $sim_j^{\text{brand}}(U_1,U_3) = 0.36$. △

2) Weighted Jaccard Similarity This measure, denoted as sim_{wj}^d, extends the namesake measure in Sec. 5 with the idea above. Its definition is the same as Eq. 9 except that a value $\mathbf{U}(i)$ in a vector represents the frequency of A_i in U that takes into consideration the weights explained in Sec. 5. Consider A_i as the preference tuple $(A_i(x), A_i(y))$. This similarity measure is defined as follows.

$$sim_{wj}^d(U, V) = \sum_i (\min(\frac{1}{|U|} \times \sum_{c \in U} \frac{1}{\min_{s \in S_c^d} D(s, A_i(x)) + 1}, \frac{1}{|V|} \times \sum_{c \in V} \frac{1}{\min_{s \in S_c^d} D(s, A_i(x)) + 1})) \Big/ \sum_i (\max(\frac{1}{|U|} \times \sum_{c \in U} \frac{1}{\min_{s \in S_c^d} D(s, A_i(x)) + 1}, \frac{1}{|V|} \times \sum_{c \in V} \frac{1}{\min_{s \in S_c^d} D(s, A_i(x)) + 1})) \tag{10}$$

Example 6.9. In Table 3, in the partial order depicting $\succ_{c_6}^{\text{brand}}$, the distance to *Apple* from the maximal value *Lenovo* is 1, i.e., the weight of *Apple* is 1/2. Since only one of the two users in U_3 has (*Apple*, *Toshiba*) in their preference relation, $\mathbf{U_3}$ has $\frac{\frac{1}{2}+0}{2} = \frac{1}{4}$ on the 3^{rd}-dimension. In this way, we get $\mathbf{U_1}$ = (2/2, 2/2, 0/2, 0/2, 1/2, 0/2, 0/2, 1/2, 2/2, 0/2, 0/2, 0/2) and $\mathbf{U_3}$ = (0/2, 1/2, 1/4, 2/2, 2/2, 2/2, 0/2, 0/2, 1/4, 0/2, 0/2, 0/2). Therefore, $sim_{wj}^{\text{brand}}(U_1,U_3) = 0.19$. △

7 ALIVE OBJECT DISSEMINATION

In Sec. 1, we discussed motivating applications such as social network content dissemination, news delivery and product recommendation. The significance of a particular social network content (e.g. a post in Facebook) or a piece of news diminishes eventually. Similarly, in any inventory, products are consumed and perishable products expire over time. In other words, objects can have limited lifetime. Thus, upon the arrival of a new object, it needs to compete only with the alive objects. To meet this requirement, we extend our problem as continuous monitoring of Pareto frontiers over *alive objects* for many users and formalize it as finding Pareto frontiers over *sliding window*.

Suppose $O = \{o_1, o_2, \ldots, o_N\}$ is a stream of objects, in which the subscript of each object is its timestamp. We consider a sliding window as a sequence of W recent objects. Upon the arrival of an incoming object o_{in}, an object o_{out} expires if $in - out = W$. Specifically, the sliding window contains objects whose timestamps are in $(out, in]$, i.e., an object $o_i \in O$ is alive during $(out, in]$ if $i \in (out, in]$. Given the concept of sliding window, we extend the definition of Pareto frontier in Def. 3.3 and the problem statement in Sec. 3.

Definition 7.1 (Pareto Frontier). An alive object o is Pareto-optimal with respect to c, if no other alive object dominates it. $\mathcal{P}_c = \{o_i \in O | \nexists o_j \in O \text{ s.t. } o_j \succ_c o_i \wedge i, j \in (out, in]\}$. The target users of o_{in} is $C_{o_{in}} = \{c \in C | o_{in} \in \mathcal{P}_c\}$ (Def. 3.4). △

Problem Statement The problem of continuous monitoring of Pareto frontiers over sliding window is, given a set of users C, their preference relations on attributes $\mathcal{D}$, and a stream of objects O with the incoming object o_{in} as well as the outgoing object o_{out}, find $C_{o_{in}}$—the target users of o_{in}.

Algorithms BaselineSW and FilterThenVerifySW We extend Baseline and FilterThenVerify to BaselineSW and FilterThenVerifySW, respectively, to accommodate sliding window. We note that no prior work studied Pareto frontier maintenance with regard to strict partial orders over sliding window. [15, 16, 21] studied skyline maintenance over sliding window, assuming numeric attributes. [18] considered total orders (with ties) on categorical

attributes instead of general partial orders. There is no clear way to extend these works for partially ordered attributes.

Due to space limitations, we leave the detailed pseudo codes and descriptions of BaselineSW and FilterThenVerifySW to the extended version of this paper [20]. Below we highlight the key concepts that dictate the design of these algorithms.

Under the constraint of having a sliding window, an object can be excluded from Pareto frontier forever if it is dominated by any succeeding object. This observation is formalized as Theorem 7.2.

THEOREM 7.2. *Consider a user $c \in C$ and two objects $o_i, o_j \in O$ such that $o_i \prec_c o_j$ and $i < j$. After the arrival of o_j, o_i can never be part of $\mathcal{P}_c$ in its remaining lifetime.* △

Proof: Since $i < j$, o_i expires before o_j and the sliding window always includes o_j if it includes o. Since o_j dominates o_i, o_i will never get into $\mathcal{P}_c$ after the arrival of o_j. ■

By Theorem 7.2, we extend our algorithms to maintain a *Pareto frontier buffer* which stores at most W recent objects that are not dominated by any succeeding object. Clearly, o_{in} is part of the Pareto frontier buffer.

Definition 7.3 (Pareto Frontier Buffer). With regard to user c and the sliding window $(out, in]$, an alive object o belongs to the Pareto frontier buffer if it is not dominated by any succeeding object. The Pareto frontier buffer is $\mathcal{PB}_c = \{o_i \in O | \nexists o_j \in O \text{ s.t. } o_j \succ_c o_i \wedge i, j \in (out, in] \wedge i < j\}$. By definition, $\mathcal{PB}_c \supseteq \mathcal{P}_c$ (Def. 7.1). △

THEOREM 7.4. *Given a set of users U, for all $c \in U$, i) $\mathcal{PB}_U \supseteq \mathcal{P}_U$ and ii) $\mathcal{PB}_U \supseteq \mathcal{PB}_c$.* △

Proof: i) Together Def. 7.1 and 7.3 imply that $\mathcal{PB}_U \supseteq \mathcal{P}_U$.

ii) We prove by contradiction. Suppose that there exists $c \in U$ such that $\mathcal{PB}_U \not\supseteq \mathcal{PB}_c$, which would mean there exists $o \in O$ such that $o \in \mathcal{PB}_c$ and $o \notin \mathcal{PB}_U$. That implies the existence of an $o' \in O$ such that $o' \succ_U o$ and $o' \not\succ_c o$. However, by Def. 4.1, $o' \succ_U o$ implies $o' \succ_c o$. Therefore, the existence of o' is impossible. In conclusion, $\mathcal{PB}_U \supseteq \mathcal{PB}_c$. ■

Note that, BaselineSW needs to maintain an exclusive Pareto frontier buffer for each user ($\mathcal{PB}_c$) while a Pareto frontier buffer per cluster ($\mathcal{PB}_U$) is sufficient for FilterThenVerifySW.

8 EXPERIMENTS

8.1 Experiment Setup

The algorithms were implemented in Java. The maximal heap size of Java Virtual Machine (JVM) was set to 16 GB. The experiments were conducted on a computer with 2.0 GHz Quad Core 2 Duo Xeon CPU running Ubontu 8.10.

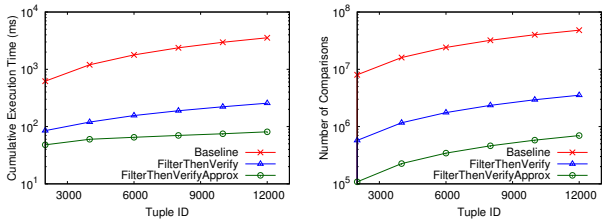
Datasets Currently there exists no publicly available dataset that captures real users' preferences in partial orders. We thus simulated such partial orders using two real datasets of users' preferences.

Movie Dataset We joined the Netflix dataset (netflixprize.com) with data from IMDB (imdb.com). The Netflix dataset contains the ratings (ranging from 0 to 5) given by users to movies. From IMDB we fetched the movies' attribute values, including actors, directors, genres, and writers. In this way, we found the attributes of 12,749 Netflix movies. The goal is to, for each particular movie, identify users who may like it according to their preferences on those attributes. The mapping from our problem formulation to this dataset is the following: (i) O is the set of 12,749 movies. (ii) C is the set of users. It includes the 1,000 most active users based on how many movies they have rated. (iii) $\mathcal{D}$ = {actor, director, genre, writer}. (iv) Given the lack of user preference data, for each attribute, the partial order corresponding to a user's preferences is simulated as follows. For two attribute values, the user's preference is based on the *average rating* and the *count* of movies satisfying these attribute values. More specifically, consider a user c who has rated m movies featuring actor a. Suppose the ratings of these movies are $r_1, r_2, \ldots, r_m$. Given c and a, the average rating is $R_a = \frac{\sum_i r_i}{m}$ and the count is $M_a = m$. Consider another actor b. If $(R_a > R_b \wedge M_a \geq M_b) \vee (R_a \geq R_b \wedge M_a > M_b)$, then $(a, b) \in \succ_c^{\text{actor}}$. Intuitively, if user c watches more movies featuring a than b and gives them higher ratings, our simulation assumes the user prefers a to b.

Publication Dataset We collected from the ACM Digital Library (dl.acm.org) 17,598 publications and their attributes, including affiliations, authors, conferences and topic keywords. The users are the authors themselves. The goal is to notify them about newly published articles. The recommendations are based on the users' preference relations on the attributes. The mapping from our problem formulation to this dataset is the following: (i) O is the set of papers. (ii) C is the set of authors. It includes the 1,000 most prolific authors based on how many publications they have, similar to the 1,000 most active users in the movie dataset. (iii) $\mathcal{D}$ = {affiliation, author, conference, keyword}. The domain of attribute author is the same 1,000 authors in C. (iv) Given a user, the partial order on each attribute is simulated based on their preferences on the attribute values. The preference between two values on affiliation (and similarly author) is based on the *number of collaborations* between the user and the affiliation/author and the *number of citations*. For conference and keyword, the preference between two values is based on *number of publications* and *number of citations*. More specifically, consider a user c and an affiliation (or similarly another author) a. Suppose c has p_a collaborations with a and has cited articles from a q_a times. If $(p_a > p_b \wedge q_a \geq q_b) \vee (p_a \geq p_b \wedge q_a > q_b)$, then $(a, b) \in \succ_c^{\text{affiliation}}$ (or $(a, b) \in \succ_c^{\text{author}}$). With regard to a conference (keyword) x, suppose c has r_x publications associated with x and has cited publications associated with x s_x times. If $(r_x > r_y \wedge s_x \geq s_y) \vee (r_x \geq r_y \wedge s_x > s_y)$, then $(x, y) \in \succ_c^{\text{conference}}$ (or $(x, y) \in \succ_c^{\text{keyword}}$).

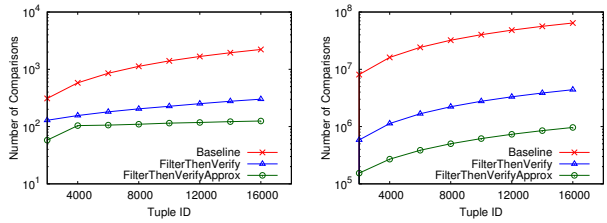
8.2 Baseline, FilterThenVerify, and FilterThenVerifyApprox

We conducted experiments to compare the performance of Baseline, FilterThenVerify and FilterThenVerifyApprox. For FilterThenVerify (resp. FilterThenVerifyApprox), users are clustered by the conventional hierarchical agglomerative clustering algorithm [9] using the similarity functions in Sec. 5 (resp. Sec. 6.3) and, for each cluster, it extracts the common preference relation (resp. approximate common preference relation). The experiments use three parameters which are number of objects ($|O|$), number of attributes (d), and branch cut (h). In hierarchical clustering, the *branch cut h* is a threshold that controls the number of clusters by governing the minimum pairwise similarity that two clusters must satisfy in order to be merged into one cluster. The sequential order of merging clusters is depicted as a tree called *dendrogram*. The branch cut thus controls where to cut the dendrogram. In Example 5.5, the set of clusters are $\{\{c_1, c_2, c_5, c_6\}, \{c_3, c_4\}\}$ for $h \in (0, \frac{3}{11}]$. This is because $sim(U_4, U_2)$=0 where U_2=$\{c_3, c_4\}$ and U_4 is the cluster composed of c_1, c_2, c_5, and c_6.



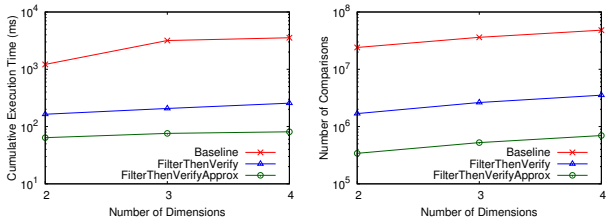
(a) Execution time (b) Object comparisons

Figure 3: Comparison of Baseline, FilterThenVerify and FilterThenVerifyApprox on the movie dataset. Varying $|O|$, $h = 0.55$, $d = 4$.



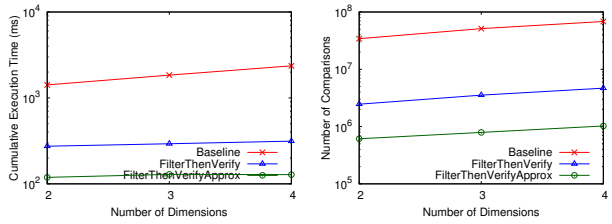
(a) Execution time (b) Object comparisons

Figure 4: Comparison of Baseline, FilterThenVerify and FilterThenVerifyApprox on the publication dataset. Varying $|O|$, $h = 0.55$, $d = 4$.



(a) Execution time (b) Object comparisons

Figure 5: Comparison of Baseline, FilterThenVerify and FilterThenVerifyApprox on the movie dataset. Varying d, $|O| = 12,749$, $h = 0.55$.



(a) Execution time (b) Object comparisons

Figure 6: Comparison of Baseline, FilterThenVerify and FilterThenVerifyApprox on the publication dataset. Varying d, $|O| = 17,598$, $h = 0.55$.

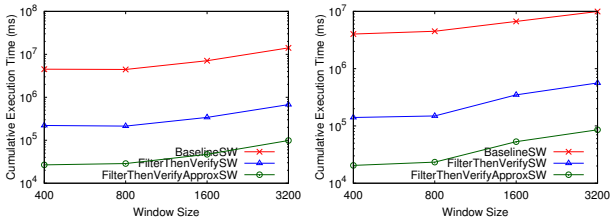
Dataset	$\|O\|$	$h = 0.70$			$h = 0.65$			$h = 0.60$			$h = 0.55$		
		Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
Movie	12, 749	100	95.43	97.67	100	93.93	96.87	99.99	93.28	96.52	99.99	90.46	94.99
Publication	17, 598	100	96.59	98.27	100	95.85	97.88	100	95.54	97.72	100	95.13	97.51

Table 7: The precision, recall and F-measure (in percentage) of FilterThenVerifyApprox. Varying h, d=4.

Fig.3a shows, for each of the three methods on the movie dataset, how its cumulative execution time (by milliseconds, in logarithmic scale) increases while the objects (i.e., movies) are sequentially processed. Fig.4a depicts similar behaviours of these methods on the publication dataset. Fig.3b and Fig.4b, for the two datasets separately, further present the amount of work done by these methods, in terms of number of pairwise object comparisons (in logarithmic scale) for maintaining Pareto frontiers. The figures show that FilterThenVerify and FilterThenVerifyApprox beat Baseline by 1 to 2 orders of magnitude. The reason is as follows. With regard to a user c, Baseline considers all objects as candidate Pareto-optimal objects and compares all pairs. On the contrary, FilterThenVerify eliminates an object o if the corresponding common preference tuples disqualify o. FilterThenVerifyApprox incurs even less comparisons by benefiting from shared computations for clusters of users.

Fig.5a (Fig.6a) shows that the execution time of all these methods increased super-linearly by number of attributes (d). Fig.5b (Fig.6b) further reveals that the number of object comparisons also increases similarly. This is not surprising because more attributes result in larger Pareto frontiers, which makes it necessary for objects to be compared with more existing Pareto-optimal objects.

Table 7 reports the precision, recall and F-measure of FilterThenVerifyApprox on varying h. We can observe that, when h got smaller, the recall slowly decreased. This is expected because smaller h results in larger clusters and potentially more approximate common preference tuples for each cluster. Those approximate common preference tuples cause false negatives—the domination and elimination of objects that are instead in the Pareto frontier under the true common preference tuples, which



(a) Movie dataset (b) Publication dataset

Figure 7: Effect of window size. Varying W, $|O| = 1$M, $h = 0.55$, $d = 4$.

are a subset of the approximate common preference tuples. What can be more surprising is the almost perfect precision under the various h values in Table 7, i.e., almost no false positives were introduced into the results. For a user c, an object o becomes a false positive if every single Pareto optimal object that dominates o becomes a false negative. As long as one of its dominating objects is not mistakenly filtered out, o will not be mistakenly introduced into the Pareto frontier. Therefore, an object is much less likely to become a false positive than a false negative. Overall, under the h values in Table 7, both precision and recall remain high. This may suggest that the thresholds θ_1 and θ_2 (Sec. 6.1) effectively ensure that the approximate common preference relation only includes frequent preference tuples and does not overgrow in size.

8.3 BaselineSW, FilterThenVerifySW, and FilterThenVerifyApproxSW

We further compare the performance of FilterThenVerifySW and FilterThenVerifyApproxSW with BaselineSW. In this regard, we simulated two data streams—movie and publication where O is

Data stream	W	$h = 0.70$			$h = 0.65$			$h = 0.60$			$h = 0.55$		
		Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
Movie	400	100	89.36	94.38	100	87.33	93.24	100	85.94	92.44	100	81.95	90.08
	800	100	87.87	93.54	100	85.78	92.34	100	84.04	91.33	100	80.10	88.95
	1600	100	88.65	93.98	100	86.58	92.81	100	85.01	91.90	100	81.10	89.56
	3200	99.99	94.80	97.33	100	93.08	96.41	100	92.29	95.99	100	88.99	94.17
Publication	400	100	94.58	97.21	100	93.57	96.68	100	92.98	96.36	100	92.06	95.87
	800	100	94.79	97.32	100	93.60	96.70	100	93.01	96.38	100	91.98	95.82
	1600	100	94.62	97.24	100	93.44	96.61	100	92.85	96.29	100	91.81	95.73
	3200	100	96.71	98.33	100	95.98	97.95	100	95.67	97.79	100	95.27	97.58

Table 8: The precision, recall and F-measure (in percentage) of FilterThenVerifyApproxSW. Varying W and h, $|O|$=1M, d=4.

composed of duplicated sequence of the corresponding dataset such that $|O|$=1 million. Following [21], we experimented with windows of size 400, 800, 1,600, and 3,200. Fig.7a shows the cumulative execution times (by milliseconds, in logarithmic scale) of the aforementioned methods on the movie stream. Fig.7a reveals that the cumulative execution times increase super-linearly by W as wider window broadens the size of Pareo frontiers. These figures illustrate that both FilterThenVerifySW and FilterThenVerifyApproxSW outperformed BaselineSW by 1 to 2 orders of magnitude, which concurs with the comparative behaviours of FilterThenVerify, FilterThenVerifyApprox and Baseline. This concurrence is also applicable for the publication stream (Fig.7b). The reason behind the comparative behaviour of Baseline, FilterThenVerify and FilterThenVerifyApprox is also applicable in this case. Moreover, BaselineSW maintains exclusive Pareto buffer for each user ($\mathcal{PB}_c$) while FilterThenVerifySW shares a Pareto buffer across users in a cluster ($\mathcal{PB}_U$). Therefore, in sliding window protocol, the filter-then-verify approach attains the benefit of clustering in a greater extent.

Table 8 demonstrates the precision, recall and F-measure of FilterThenVerifyApproxSW on varying W and h. We can observe that the recall declines slowly by h. Yet h does not have significant impact on the efficacy of FilterThenVerifyApproxSW. Besides, the loss of accuracy is due to false negatives rather than false positives. These behaviors concur with FilterThenVerifyApprox and the reasons behind are same as before. In addition, Table 8 reveals that W does not have noticeable impact on efficacy and FilterThenVerifyApprox remains effective on varying W.

9 CONCLUSION

We studied the problem of continuous object dissemination, which is formalized as finding the users who approve a new object in Pareto-optimality. We designed algorithm for efficient finding of target users based on sharing computation across similar preferences. To recognize users of similar preferences, we studied the novel problem of clustering users where each user's preferences are described as strict partial orders. We also presented an approximate solution of the problem of finding target users, further improving efficiency with tolerable loss of accuracy. Experimental evaluation validated the efficiency and effectiveness of our proposed solutions.

Acknowledgements: The work is partially supported by NSF grant IIS-1719054. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies. Furthermore, we thank Fatma Arslan for her contribution in data collection.

REFERENCES

[1] O. Barndorff-Nielsen and M. Sobel. On the distribution of the number of admissible points in a vector random sample. *Theory of Probability & Its Applications*, 11(2), 1966.
[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
[3] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, 2005.
[4] F. Chierichetti, R. Kumar, S. Pandey, and S. Vassilvitskii. Finding the jaccard median. In *SODA*, 2010.
[5] J. Chomicki. Preference formulas in relational queries. *TODS*, 28(4), 2003.
[6] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, 2007.
[7] R. Fagin. Combining fuzzy information from multiple systems. In *PODS*, 1996.
[8] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
[9] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 3rd edition, 2011.
[10] T. Kamishima and S. Akaho. Clustering orders. In *Discovery Science*, 2003.
[11] T. Kamishima and S. Akaho. Efficient clustering for orders. In *Mining Complex Data*. 2009.
[12] W. Kießling. Foundations of preferences in database systems. In *VLDB*, 2002.
[13] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of ACM*, 22(4), Oct. 1975.
[14] K. C. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in z order. In *VLDB*, 2007.
[15] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, 2005.
[16] M. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. *Information Sciences*, 177(17), 2007.
[17] D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically sorted skylines for partially ordered domains. In *ICDE*, 2009.
[18] N. Sarkas, G. Das, N. Koudas, and A. K. Tung. Categorical skylines for streaming data. In *SIGMOD*, 2008.
[19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.
[20] A. Sultana and C. Li. Continuous monitoring of pareto frontiers on partially ordered attributes for many users. *CoRR*, abs/1709.08312, 2017.
[21] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *TKDE*, 18(3), 2006.
[22] A. Ukkonen. Clustering algorithms for chains. *The Journal of Machine Learning Research*, 12, 2011.
[23] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag. Reverse top-k queries. In *ICDE*, 2010.
[24] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *SIGMOD*, 2013.
[25] 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.
[26] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang. Mining favorable facets. In *SIGKDD*, 2007.
[27] R.-W. Wong, J. Pei, A.-C. Fu, and K. Wang. Online skyline analysis with dynamic preferences on nominal attributes. *TKDE*, 21(1), 2009.
[28] P. Wu, D. Agrawal, O. Egecioglu, and A. El Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In *ICDE*, 2007.
[29] A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-k queries. In *SIGMOD*, 2012.
[30] S. Zhang, N. Mamoulis, D. W. Cheung, and B. Kao. Efficient skyline evaluation over partially ordered domains. *VLDB*, 3(1-2), 2010.