Collaborative Filtering with Personalized Skylines
Ilaria Bartolini, Zhenjie Zhang, and Dimitris Papadias

Abstract — Collaborative filtering (CF) systems exploit previous ratings and similarity in user behavior to recommend the top-k objects/records which are potentially most interesting to the user assuming a single score per object. However, in various applications a record (e.g., hotel) may be rated on several attributes (value, service etc), in which case simply returning the ones with the highest overall scores fails to capture the individual attribute characteristics and to accommodate different selection criteria. In order to enhance the flexibility of CF, we propose Collaborative Filtering Skyline (CFS), a general framework that combines the advantages of CF with those of the skyline operator. CFS generates a personalized skyline for each user based on scores of other users with similar behavior. The personalized skyline includes objects that are good on certain aspects, and eliminates the ones that are not interesting on any attribute combination. Although the integration of Skylines and CF has several attractive properties, it also involves rather expensive computations. We face this challenge through a comprehensive set of algorithms and optimizations that reduce the cost of generating personalized Skylines. In addition to exact skyline processing, we develop an approximate method that provides error guarantees. Finally, we propose the top-k personalized skyline, where the user specifies the required output cardinality.

Accepted at TKDE, December 2009

Index Terms — Skyline, Collaborative Filtering

1 INTRODUCTION

Collaborative filtering (CF) [1] is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. Popular CF systems include those of Amazon and Netflix, for recommending books and movies, respectively. Such systems maintain a database of scores entered by users for records/objects (books, movies) that they have rated. Given an active user \( u_i \) looking for an interesting object, these systems usually take two steps: (i) retrieve users who have similar rating patterns with \( u_i \); (ii) utilize their scores to return the top-k records that are potentially most interesting to \( u_i \). Conventional CF assumes a single score per object. However, in various applications a record may involve several attributes. As our running example, we use Trip Advisor (www.tripadvisor.com), a site that maintains hotel reviews written by travelers. Each review rates a hotel on features such as Service, Cleanliness, and Value (the score is an integer between 1 and 5).

The existence of multiple attributes induces the need to distinguish the concepts of scoring patterns and selection criteria. For instance, if two users \( u_m \) and \( u_n \) have visited the same set of hotels and have given identical scores on all dimensions, their scoring patterns are indistinguishable. On the other hand, they may have different selection criteria; e.g., service may be very important to business traveler \( u_m \), whereas \( u_n \) is more interested in good value for selecting a hotel for her/his vacation. A typical CF system cannot differentiate between the two users, and based on their identical scoring patterns would likely make the same recommendations to both. To overcome this problem, the system could ask each user for an explicit preference function that weighs all attributes according to her/his choice criteria, and produces a single score per hotel. Such a function would set apart \( u_m \) and \( u_n \), but would also incur information loss due to the replacement of individual ratings (on each dimension) with a single value. For instance, two (overall) scores (by two distinct users) for a hotel may be the same, even though the ratings on every attribute are rather different. Furthermore, in practice casual users may not have a clear idea about the relative importance of the various attributes. Even if they do, it may be difficult to express it using a mathematical formula. Finally, their selection criteria may change over time depending on the purpose of the travel (e.g., business or vacation).

Motivated by the above observations, we apply the concept of Skylines to collaborative filtering. A record (in our example, a hotel) \( r_i \) dominates another \( r_j \) (\( r_i \succ r_j \)), if and only if \( r_i \) is not worse than \( r_j \) on any dimension, and it is better than \( r_j \) on at least one attribute. This implies that \( r_i \) is preferable to \( r_j \) according to any preference function which is monotone on all attributes. The skyline contains all non-dominated records. Continuing the running example, as-
assume that the system maintains the average rating for each hotel on every attribute. A traveler could only select hotels that belong to the skyline (according to the attributes of her/his choice). The rest can be eliminated, since for each hotel that is not in the skyline, there is at least another, which is equal or better on all aspects, independently of the preference function. In other words, the skyline allows the clients to make their own choices, by including hotels that are good on certain aspects and removing the ones that are not interesting on any attribute combination.

So far, our example assumes a single skyline computed using the average scores per attribute. However, replacing the distinct scores with a single average per dimension contradicts the principles of CF because it does not take into account the individual user characteristics and their similarities. To solve this problem, we propose collaborative filtering skyline (CFS), a general framework that generates a personalized skyline, Skyl, for each active user ut based on scores of other users with similar scoring patterns. Let si,m be the score of user ut for record (e.g., hotel) ri, and rj is a vector of values, each corresponding to an attribute of ri (e.g., value, service, etc). We say that a tuple ri dominates another rj with respect to an active user ut (and denote it as ri ≻ rj) if there is a large number1 of pairs s1,m ≻ s1,n, especially if those scores originate from users u1,m that are similar to each other and to ut. The personalized skyline Skyl of ut contains all records that are not dominated. A formal definition appears in Section 3.

Consider the scenario of Figure 1 in the context of Trip Advisor. There are four tuples r1-r4 (hotels) represented by different shapes, involving two attributes (value, service). These records are rated by three users u1, u2, and u3. Each record instance corresponds to a user rating; e.g., s1,1, s1,2, s1,3 are scores of r1, whereas s4,2 and s4,3 are scores of r4. We assume that higher scores on attributes are more preferable. Users u1 and u2 have given analogous scores to hotels r1 and r3 (see s1,1, s1,2, and s3,1, s3,2). Furthermore, u2 has also rated highly r2 (see s2,2) on service. Since u1 and u2 have similar rating patterns, r2 probably would rank high in the preferences of u1, as well, and should be included in Skyl. Note that although s2,2 is dominated by s4,3 given by user u3, u1 and u3 do not have similar preferences; they have rated a single record (r1) in common, and their scores (s1,1, s1,3) are rather different. Instead, the opinion of u2 is much more valuable to u1, and consequently the personalized skyline of u1 should contain r2 rather than r4.

Similar examples can be constructed for other domains including film (resp. real estate) with ratings on entertainment value, image and sound quality etc. (resp. space, quality of neighborhood, proximity to schools etc.). Our experimental evaluation demonstrates that indeed recommendations made by CFS are rated higher by travelers (after they visited the hotels) than those made by a typical CF algorithm. However, similar to conventional CF, CFS involves expensive computations, necessitating efficient indexing and query processing techniques.

We address these challenges through the following contributions: (i) we develop algorithms and optimization techniques for exact personalized skyline computation, (ii) we present methods for approximate skylines that significantly reduce the cost in large datasets without compromising effectiveness, (iii) we propose top-k personalized skylines, which restrict the skyline cardinality to a user-specified number. The rest of the paper is organized as follows. Section 2 overviews related work. Section 3 introduces the CFS framework. Section 4 describes exact, and Section 5 approximate skyline computation, respectively. Section 6 deals with the top-k personalized skyline. Section 7 evaluates the proposed techniques using real and synthetic datasets. Section 8 concludes the paper.

2 Background
Section 2.1 surveys background on skylines. Section 2.2 overviews collaborative filtering and related systems. In addition to previous work, we discuss its differences with respect to the proposed approach.

2.1 Skyline Processing
We assume records with d (≥2) attributes, each taking values from a totally ordered domain. Accordingly, a record can be represented as a point in the d-dimensional space (in the sequel we use the terms record, point and object interchangeably). The skyline contains the best points according to any function that is monotonic on each attribute. Conversely, for each skyline record r there is such a function that would assign it the highest score. These attractive properties of skylines have led to their application in various domains including multi-objective optimization [40], maximum vectors [21], and the contour problem [25]. Börzsönyi et al. [5] introduced the skyline operator to the database literature and proposed two disk-based algorithms for large datasets. The first, called D&C (for divide and conquer) divides the dataset into partitions that fit in memory, computes the partial skyline in every partition, and generates the final skyline by merging the partial ones. The second algorithm, called BNL, applies the concept of block-nested loops. SFS [10] improves BNL by sorting the data. Other variants of BNL include LESS [14] and SalSa [3]. All these methods do not use any indexing and, usually, they have to scan the entire dataset before reporting any skyline point. Another set of algorithms utilizes conventional or multi-dimensional indexes to speed up query processing and progressively report skyline points. Such methods include Bitmap, Index [42], NN [20] and BBS [29].

Figure 1 Example of personalized skyline

1 The number of pairs s1,m ≻ s1,n depends on a user-defined threshold that controls the skyline cardinality.
In addition to conventional databases, skyline processing has been studied in other scenarios. For instance, Morse et al. [27] use spatial access methods to maintain the skyline in streams with explicit deletions. Efficient skyline maintenance has also been the focus of [22]. In distributed environments, several methods (e.g., [18]) query independent subsystems, each in charge of a specific attribute, and compute the skylines using the partial results. In the data mining context, Wong et al. [44] identify the combinations of attributes that lead to the inclusion of a record in the skyline. The sky-cube [46] consists of the skylines in all possible subspaces. The compressed sky-cube [45] supports efficient updates. Subsky [43] aims at computing the skyline on particular subspaces. Techniques to reduce the number of skyline points are discussed in [9]. Chan et al. [8] focus on skyline evaluation for attributes with partially-ordered domains, whereas Morse et al. [28] consider low-cardinality domains. A dynamic skyline changes the co-ordinate system according to a user-specified point [29]. The reverse skyline [13] retrieves those objects, whose dynamic skyline contains a given query point. Given a set of points Q, the spatial skyline retrieves the objects that are the nearest neighbors of any point in Q [39].

All the above techniques consider that each tuple has a single representation in the system. On the other hand, in CFS a record is associated with multiple scores. The only work similar to ours on this aspect is that on probabilistic skylines [31], which assumes that a record has several instances. Let \( s_{i,m} \) be an instance of \( r_i \), and \( s_{j,n} \) an instance of record \( r_j \). \( S_i \) denotes the set of instances of \( r_i \) (resp. for \( S_j \)). There are in total \( |S_i| \cdot |S_j| \) pairs \((s_{i,m}, s_{j,n})\). In this model, a record \( r_i \) dominates another \( r_j \) with probability \( Pr[r_i \succ r_j] \) which is equal to the ratio of all pairs such that \( s_{i,m} \succ s_{j,n} \) over \( |S_i| \cdot |S_j| \). Given a probability threshold \( p \), the p-skyline contains the subset of records whose probability to be dominated does not exceed \( p \).

Figure 2 shows a simplified version of the example introduced in Figure 1, where the user component (i.e., the grey-scale color information) has been eliminated. Each record instance corresponds to a rating; e.g., grey-square color information. Each instance of a record \( r_i \) is represented by a circle, and the value of the score of each record is denoted by a diamond. The size of the diamond indicates the value of the score. For example, Fig. 2(a) shows the skyline for the user component of the example in Fig. 1. If a user \( u_i \) has rated both \( r_i \) and \( r_j \), their scores \( s_{i,m} \) and \( s_{j,n} \) are used to determine the dominance of \( r_i \) over \( r_j \). The dominance relation is computed using the ratings of the active user on records similar to \( r_i \). If \( p \leq 1/6 \), neither \( r_i \) nor \( r_j \) are in the p-skyline because they are both dominated (by each other). Pei et al. [31] propose a bottom-up and a top-down algorithm for efficiently computing probabilistic skylines. Both algorithms utilize heuristics to avoid dominance checks between all possible instance pairs. Lian and Chen [24] extend these techniques to reverse skyline processing.

The probabilistic skyline model was not aimed at CF, and it has limited expressive power for such applications. Specifically, the system outputs a single skyline for all users, instead of the personalized skylines of CFS. As explained in the example of Figure 1, since \( u_1 \) and \( u_2 \) have similar rating patterns, \( r_2 \) probably would rank high in the preferences of \( u_1 \) as well. However, according to Figure 2, \( r_2 \) has a low chance to be included in the skyline because its single instance \( s_{2,2} \) is dominated by \( s_{4,2} \) this is counter-intuitive because \( s_{4,2} \) is of little importance for \( u_1 \). Furthermore, the query processing algorithms of [31] are inapplicable to CFS because they prune using minimum bounding rectangles without differentiating between scores of distinct users. On the contrary, CFS necessitates the inspection of the individual scores (and the corresponding users who input them) for the similarity computations.
users. GroupLens [34] and Ringo [38] were among the first systems to propose CF algorithms for automatic predictions. Several recommendation systems (e.g., Syskill & Webert [30], Fab [2], Filterbot [16], P-Tango [11], Yoda [37]) have been applied in information retrieval and information filtering. CF systems are also used by several companies, including Amazon and Netflix. It is worth noting that Netflix established a competition to beat the prediction accuracy of its own CF method, which attracted several thousand participants.

Moreover, CF has been investigated in machine learning as a classification problem, by applying various techniques including inductive learning [4], neural and Bayesian networks [30, 6], and, more recently, probabilistic models, such as personality diagnosis [33] and probabilistic clustering [23]. Surveys on various recommendation approaches and CF techniques can be found in [35, 32, 1]. Herlocker et al. [17] review key issues in evaluating recommender systems, such as the user tasks, the types of analysis and datasets, the metrics for measuring predictions effectiveness, etc. An open framework for comparing CF algorithms is presented in [12].

Similar to user-based CF systems, we utilize similarity between the active user $u_i$ and the other users. However, whereas existing systems aim at suggesting the top- $k$ records assuming a single score per object, CFS maintains the personalized skylines of the most interesting records based on multiple attributes. Unlike content-based systems, we do not assume a set of well-defined features used to determine an objective similarity measure between each pair of records. Instead, each user rates each record subjectively. CFS permits the distinction of scoring patterns and selection criteria, as discussed in the introduction, i.e., two users are given diverse choices even if their scoring patterns are identical, which is not possible in conventional CF. Furthermore, CFS enhances flexibility by eliminating the need for a scoring function (to assign weights to different attributes).

### 3 CFS FRAMEWORK

In this section, we provide the dominance and similarity definitions in CFS, and outline the general framework. Table I summarizes the frequently used symbols. Let $R$ be the set of records and $U$ the set of users in the system. The score $s_{i,m}$ of a user $u_i \in U$ on a record $r_i \in R$ is a vector of values $[s_{i,1}, \ldots, s_{i,d}]$, each corresponding to a rating on a dimension of $r_i$. Without loss of generality, we assume that higher scores on attributes are more relevant. It follows that, $s_{i,m}$ dominates $s_{j,n}$ ($s_{i,m} > s_{j,n}$), if the rating of $r_i$ by $u_m$ is not lower than that of $r_j$ by $u_n$ on any dimension, and it is higher on at least one attribute.

The personalized skyline $Sky_w$ of an active user $u_i$ contains all records that are not dominated according to the following definition.

**Definition 3.1 (Personalized Dominance):** A tuple $r_i$ dominates another $r_j$ with respect to an active user $u_i$ ($r_i \succ r_j$) iff

$$\sum_{m,n} w_{m,n}[s_{i,m} > s_{j,n}] \geq \theta_i$$

where:

$$[s_{i,m} > s_{j,n}] = \begin{cases} 1 & \text{if } s_{i,m} \text{ and } s_{j,n} \text{ are not null and } s_{i,m} > s_{j,n} \\ 0 & \text{otherwise} \end{cases}$$

$S_i$ denotes the set of ratings for $r_i$, and $|S_i|$ is its cardinality. The product $|S_i| \cdot |S|$ normalizes the value of personalized dominance in the range $[0,1]$. $\theta$ is a user-defined threshold that controls the skyline cardinality; a value close to 0 leads to a small $Sky_w$ because most records are dominated. Each dominance pair $(s_{i,m} > s_{j,n})$ has a weight $w_{m,n}$ in the range $[0,1]$, which is proportional to the pairwise similarities $\sigma(u_{m,i}, u_{j,n})$, and $\sigma(u_{m,i})$:

$$w_{m,n} = sf(\sigma(u_{m,i}, u_{j,n}), \sigma(u_{m,i}))$$

The function $sf(.)$ should be monotonically increasing with the pairwise similarities. An intuitive implementation of $sf(.)$ is the average function, but other choices are applicable. CFS can accommodate alternative pairwise similarity measures proposed in the CF literature. Here, we use the Pearson Correlation coefficient [1], shown in Equation 3.4.

$$corr(u_{m,i}, u_{j,n}) = \frac{\sum_{k=1}^{d} (s_{m,k} - \bar{s}_m)(s_{n,k} - \bar{s}_n)}{\sqrt{\sum_{k=1}^{d} (s_{m,k} - \bar{s}_m)^2} \sqrt{\sum_{k=1}^{d} (s_{n,k} - \bar{s}_n)^2}}$$

Let $R_m$ (resp. $R_n$) be the set of records rated by $u_m$ (resp. $u_n$). The correlation $corr(u_{m,i}, u_{j,n})$ between $u_m$ and $u_n$ is computed on the records $R_m \cap R_n$ that both have reviewed; $\bar{s}_m$ and $\bar{s}_n$.

---

2 User-independent similarity measures based on term occurrences are natural for document retrieval. On the other hand, CF applications often involve inherently subjective recommendations (e.g., for hotels, films or books).

3 For simplicity, we assume that each score $s_{i,m}$ contains a rating for every attribute. If some attributes are not rated in $s_{i,m}$, we can apply the dominance definitions of [19] for incomplete data.

---

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>record set, $R={r}$</td>
</tr>
<tr>
<td>$U$</td>
<td>user set, $U={u}$</td>
</tr>
<tr>
<td>$s_{i,m}$</td>
<td>score on $r_i$ given by $u_m (s_{i,1}, \ldots, s_{i,d})$</td>
</tr>
<tr>
<td>$w_{m,n}$</td>
<td>weight of $(s_{i,m} &gt; s_{j,n})$ with respect to $w$</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>similarity between users $u_m$ and $u_n$</td>
</tr>
<tr>
<td>$S_i$</td>
<td>score set for record $r_i$</td>
</tr>
<tr>
<td>$R_m$</td>
<td>records reviewed by user $u_m$</td>
</tr>
<tr>
<td>$S_{i,m} &gt; S_{j,n}$</td>
<td>dominance relationship between two scores</td>
</tr>
<tr>
<td>$r_i \succ r_j$</td>
<td>personalized dominance between $r_i$ and $r_j$ wrt $w$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>dominance threshold for user $u$</td>
</tr>
<tr>
<td>$Sky_w$</td>
<td>skyline set for $w$</td>
</tr>
</tbody>
</table>

---

Table I: Frequent Symbols
denote the average scores of \( u_m \) and \( u_n \) on all records of \( R_m \) and \( R_n \), respectively. Intuitively, two users have high correlation, if for most records \( r_i \in R_m \cap R_n \), they have both rated \( r_i \) above or below their averages. Since the correlation is a value between -1 and 1, we apply Equation 3.5 to normalize similarity in the range \([0,1]\). 

\[
\sigma(u_m, u_n) = \frac{1 + corr(u_m, u_n)}{2} \tag{3.5}
\]

Note that Equation 3.5 is just one of several alternatives for user similarity. Another option would be to define \( \sigma(u_m, u_n) = corr(u_m, u_n) \) only considering positively correlated users, and setting the similarity of negatively correlated users to 0. CFS utilizes three hash tables for user similarity computation and maintenance: (i) given \( r_i \) and \( u_m \), the user table \( UT \) retrieves \( s_{i,m} \); (ii) given \( r_i \), the record table \( RT \) retrieves the set of users who have rated \( r_i \); (iii) given a pair \((u_m, u_n)\) of user ids, the similarity table \( ST \) retrieves \( \sigma(u_m, u_n) \). Depending on the problem size, these indexes can be maintained in main memory, or be disk-based.

Before presenting algorithms for CFS, we discuss some properties of personalized skylines. First note that given a threshold \( \theta \), it is possible that both \( r_i \succ r_j \) and \( r_j 
\succ r_i \) are simultaneously true. For example, in the figure above, if \( \theta = 1/6 \) and all weights are equal to 1, we have \( r_i \succ r_j \) and \( r_j \succ r_i \) for every user (i.e., the records are dominated by each other and, therefore, they are not in \( Sky_i \)). On the other hand, if \( \theta = 1/3 \) only \( r_i \succ r_j \) is true, while if \( \theta > 1/3 \) none of the dominance relationships holds. Second, personalized dominance is not transitive, i.e., \( r_i \ntriangleleft r_j \) and \( r_j \ntriangleleft r_k \) does not necessarily imply that \( r_i \ntriangleleft r_k \) if the weights \( w_{m,n} \) of score pairs \((s_{i,m}, s_{k,n})\) are low.

These properties suggest that any exact algorithm for computing personalized skylines should exhaustively consider all pairs of records: it is not possible to ignore a record during processing even if it is dominated because that particular record may be needed to exclude another record from the personalized skyline through another personalized dominance relationship. Thus, optimizations proposed in traditional skyline algorithms, where transitivity holds, cannot be applied to our scenario. Instead, in the following, we present specialized algorithms for personalized skyline computation.

### 4 Exact Skyline Computation

Given an active user \( u \) and a threshold \( \theta \), the personalized skyline \( Sky_i \) contains all records that are not dominated by Definition 3.1. Section 4.1 presents the basic algorithm, and Sections 4.2 proposes optimizations to speed-up query processing. Section 4.3 discusses an alternative algorithm for personalized skyline computation.

#### 4.1 Basic Algorithm

Figure 3 illustrates the basic functionality of Exact Skyline Computation (ESC) for an active user \( u \). The input of the algorithm is threshold \( \theta \). Initially, every record \( r_i \) is a candidate for \( Sky_i \) and compared against every other record \( r_j \). The variable \( Sum \) is used to store the weighted sum of each pair \((s_{i,m}, s_{j,n})\) such that \( s_{i,m} \succ s_{j,n} \). If \( Sum \) exceeds \( \theta \), \( r_i \) is dominated by \( r_j \) and, therefore, it is excluded from the skyline. The set of users who have rated each record (Lines 5 and 6) is obtained through the record table \( RT \). The scores of these users (Line 7), and their similarities (Line 8) are retrieved through the user \( UT \) and similarity \( ST \) tables, respectively.

<table>
<thead>
<tr>
<th>Basic ESC (user ( u ), threshold ( \theta )) // computation of ( Sky_i ) for ( u )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( Sky_i = \emptyset )</td>
</tr>
<tr>
<td>2. for each record ( r_i ) in ( R )</td>
</tr>
<tr>
<td>3. for each record ( r_j \neq r_i )</td>
</tr>
<tr>
<td>4. ( Sum = 0 )</td>
</tr>
<tr>
<td>5. for each user ( u_m ) who has rated ( r_i )</td>
</tr>
<tr>
<td>6. for each user ( u_n ) who has rated ( r_j )</td>
</tr>
<tr>
<td>7. if ( s_{i,m} \succ s_{j,n} )</td>
</tr>
<tr>
<td>8. ( Sum = Sum + w_{m,n} \cdot</td>
</tr>
<tr>
<td>9. if ( Sum \geq \theta )</td>
</tr>
<tr>
<td>10. skip ( r_j ); goto 2</td>
</tr>
<tr>
<td>11. insert ( r_i ) into ( Sky_i ); ( r_i ) is not dominated</td>
</tr>
<tr>
<td>12. report ( Sky_i )</td>
</tr>
</tbody>
</table>

#### Figure 3 Basic algorithm for Sky_i computation

Assuming that locating an entry in a hash table takes constant time\(^5\), the worst case expected cost of the algorithm is \( O(|R|^2 \cdot |S_{AVG}|^2) \), since it has to consider all pairs \((|R|^2)\) of records and for each pair to retrieve all scores \((|S_{AVG}|)\) is the average number of ratings per record). Compared to conventional skylines, personalized skyline computation is inherently more expensive because of the multiple scores (i.e., instances) per record. For instance, the block nested loop algorithm for conventional skylines [5], which compares all pairs of records (similar to basic ESC), has cost \( O(|R|^3) \). Furthermore, pruning heuristics (e.g., based on minimum bounding rectangles [29, 31]) that eliminate records/instances collectively are inapplicable because CFS needs to consider the weights of individual score pairs. On the other hand, for several applications the high cost is compensated by the fact the personalized skylines do not have to be constantly updated as new scores enter the system. Instead, it could suffice to execute the proposed algorithms and optimizations on a daily or weekly basis (e.g., recommend a set of movies for the night, or the books of the week).

For generality, we assume that the personalized skyline is computed over all attributes of every record. However, the proposed techniques can be adapted to accommodate selection conditions and subsets of attributes. In the first case (e.g., hotels should be in a given city), only records satisfying the input conditions are considered in Lines 2 and 3. In the second case (e.g., take into account only the service and value attributes), the dominance check in Line 7 considers just those dimensions. Depending on the application, the personalized skylines can be computed upon request, or pre-computed during periods of low workloads (e.g., at night), or when the number of incoming scores exceeds a threshold (e.g., after 1000 new scores have been received). Furthermore, given an incoming \( s_{i,n} \), the CFS system can exclude \( r_i \) from \( Sky_i \) (e.g., a sub-

\(^5\) In general, hash indexes do not provide performance guarantees, although in practice they incur constant retrieval cost.
scribe of Amazon is not likely to be interested in a book that she has already read), or not (in Trip Advisor, a hotel remains interesting after the client has rated it).

4.2 Optimizations

In this section, we propose three optimizations to speed-up ESC: pre-pruning, score pre-ordering and record pre-ordering. Pre-pruning is a pre-processing technique, which generates two set of records: C contains objects that are in every personalized skyline, and N contains objects that cannot be in any skyline. Records of both C and N are excluded from consideration in Line 2 of Figure 3, reducing the cost of individual skylines (all elements of C are simply appended to Sky\(_i\) of every user \(u_i\)). Pre-pruning is based on (i) the monotonic property of \(sf(\cdot)\) in Equation 3.3, and (ii) the observation that during the computation of a personalized skyline, the only factor that depends on the active user \(u_i\) is \(w_{\sigma,n}\) (Line 8 of basic ESC). Assuming that \(sf(\cdot)\) is the average function we have:

\[
\begin{align*}
lbw_{\sigma,n} &= (\sigma u_i u_i) + lbw_{\sigma,n} + lbw_{\sigma_n}/3 \leq \frac{\sigma u_i u_i + lbw_{\sigma,n} + lbw_{\sigma_n}}{3} \\
w_{\sigma,n} &= (\sigma u_i u_i) + w_{\sigma,n} + w_{\sigma_n}/3 \leq \frac{\sigma u_i u_i + w_{\sigma,n} + w_{\sigma_n}}{3}
\end{align*}
\]

(4.1)

Given \(u_n\) and \(u_m\), \(lbw_{\sigma,n}\) is a lower bound of \(w_{\sigma,n}\) for any possible user \(u_i\); \(lbw_{\sigma_n}\) (resp. \(lbw_{\sigma}\)) denotes the minimum similarity of \(u_m\) (resp. \(u_n\)) to every other user. Similarly, \(ubw_{\sigma,n}\) is an upper bound for \(w_{\sigma,n}\) and \(ubw_{\sigma_n}\) are the maximum similarities of \(u_n\) and \(u_m\). The values of \(lbw_{\sigma_n}\) and \(ubw_{\sigma_n}\) can be stored (and maintained) with the profile of \(u_m\); alternatively, they can be set to 0 and 1, respectively, reducing, however, the effectiveness of pruning.

Figure 4 illustrates the pseudo-code for pre-pruning using the above bounds. Similar to Figure 3, the algorithm considers all pairs of records and for each pair it retrieves all scores \(s_{i,m}\), \(s_{j,n}\). Variables \(SumN\) and \(SumC\) store the aggregate weights for pairs \(s_{i,m} > s_{i,m}\) using the lower and upper bounds, respectively. When \(SumN\) exceeds \(\theta\), \(r_i\) is inserted into N because it cannot be in the skyline of any user \(u_i\). If \(r_j\) is dominated by \(r_i\) even if the lower bound \(lbw_{\sigma,n}\) is used instead of \(w_{\sigma,n}\). On the other hand, if all records \(r_i\) have been exhausted and \(SumC < \theta\), \(r_i\) is inserted into C. \(r_j\) cannot be dominated for any user \(u_i\) even if the upper bound \(ubw_{\sigma,n}\) is used instead of \(w_{\sigma,n}\). Since the lists C and N depend on the threshold \(\theta\), the algorithm must be repeated for all values of \(\theta\) that are commonly used by individual users. This overhead is not significant because (i) the cost of pre-pruning is the same as that of computing a single skyline \((O(|R| \cdot |SAVG| \cdot \log(|SAVG|)))\), and (ii) it is amortized over all personalized skyline queries that involve the same threshold.

In the worst case, the basic ESC algorithm requires the iteration over all scores \(s_{i,m}\) (Lines 5-7 in Figure 3) for each \(s_{i,m}\). Score pre-ordering avoids considering scores \(s_{i,m}\) that cannot dominate \(s_{i,m}\). Specifically, the set \(S_i\) of scores on each record \(r_i\) is sorted in descending order of the maximum attribute value. Using this order, the scores \(s_{i,m}\) with higher probability to dominate \(s_{i,m}\) are visited first. Once a score \(s_{i,m}\) with maximum attribute equal to, or smaller than, the minimum attribute of \(s_{i,m}\) is reached, the inner iteration over \(s_{i,m}\) stops. For instance, given that the current \(s_{i,m}=(2,3)\) (assuming 2 attributes) and \(s_{i,m}=(3,4)\), there can be no subsequent score in the sorted \(S_i\) such that \(s_{i,m} > s_{i,m}\). The cost of score pre-ordering is \(O(|R| \cdot |SAVG| \cdot \log(|SAVG|))\) because it involves sorting the scores of each record. Similar to the other optimizations, the cost is amortized over all skyline queries.

Pre-pruning (threshold \(\theta\))

1. \(C = \emptyset\) // records that are in all personalized skylines
2. \(N = \emptyset\) // records that are not in any personalized skyline
3. for each record \(r_i\) in \(R\)
4. for each record \(r_j\) not in \(N\)
5. \(SumC=0, SumN=0\)
6. for each user \(u_i\) who has rated \(r_i\)
7. for each user \(u_i\) who has rated \(r_j\)
8. if \(s_{i,m} > s_{i,m}\)
9. \(SumN=SumN+lbw_{\sigma,n}\)
10. \(SumC=SumC+ubw_{\sigma,n}\)
11. if \(SumN \geq \theta\)
12. insert \(r_i\) into \(N\); goto 3
13. if \(SumC < \theta\)
14. insert \(r_i\) into \(C\); goto 3
15. return \(C\) and \(N\)

Figure 4 Pre-pruning algorithm

A record \(r_i\) can be pruned from Sky\(_i\) if we can find some record \(r_j\) such that \(r_j > r_i\). Thus, it is crucial to devise an order, where those records more likely to dominate \(r_i\) are considered early. An intuitive ordering is motivated by the observation that, if a record \(r_i\) has better overall ratings than \(r_j\), \(r_i\) is more likely to dominate \(r_j\) than \(r_j\). Based on this observation, \(r_i\) is ordered before \(r_j\) if the sum of the average ratings on all dimensions of \(r_i\) is larger than that of \(r_j\). The cost of record pre-ordering is \(O(|R| \cdot |SAVG| + |R| \cdot \log(|R|))\) because it involves computing the average rating for each record \((|R| \cdot |SAVG|)\), and then sorting all records \((|R| \cdot \log(|R|))\).

4.3 Two-Scans ESC (2S-ESC)

Recall from Section 3 that the personalized dominance is not symmetric and transitive; thus, any exact algorithm for computing personalized skylines should exhaustively consider all pairs of records. In the following, we propose a two-scans paradigm that aims at avoiding the exhaustive comparison of all record pairs. 2S-ESC performs two nested loops\(^6\), where the inner loop iterates only over potential skyline records, as summarized in Figure 5. Specifically, the first loop (Lines 2–11), inserts into Sky\(_i\) each record \(r_i\) that is not dominated by another record already in Sky\(_i\). Compared to the basic ESC algorithm, the number of records in line 3 is small. However, after this pass Sky\(_i\) may contain false positives, which are removed during the second nested loop (Lines 12 to 20). Note that, due to the absence of transitivity, Line 13 needs to consider dominance with respect to all records in \(R\) (and not just those in Sky\(_i\)).

The efficiency of 2S-ESC depends on the number of false positives produced by the first scan. Assuming that the cardinality of Sky\(_i\) after the first scan is \(|R'\)| (including

\(^6\) A similar idea for computation of \(k\)-dominant skylines is used in [9].
false positives), the cost of the first scan is \( O(|R'| \cdot |R| \cdot |S_{AVG}|^2) \), since each record is compared only with the elements of \( Sky_j \). The second scan verifies the candidates in \( Sky_j \) by comparing against all records in \( R \), with cost also \( O(|R'| \cdot |R| \cdot |S_{AVG}|^2) \) time. Consequently, the complexity of the complete algorithm is \( O(|R'| \cdot |R| \cdot |S_{AVG}|^2) \). The three optimizations of Section 4.2 also apply to 2S-ESC. Specifically, pre-pruning eliminates records that cannot participate in the skyline from both scans, and record/score preordering can speed-up each scan.

2S-ESC (user \( u_o \), threshold \( \Theta \))
1. \( Sky_j = \emptyset \)
2. for each record \( r_j \) in \( R \)
3. for each record \( r_j \) in \( Sky_j \)
   ... lines 4-11 identical to corresponding lines in Figure 3
   ...
12. for each record \( r_j \) in \( Sky_j \)
13. for each record \( r_j \) in \( R \)
14. \( \text{Sum} = 0 \)
15. for each user \( u_o \), who has rated \( r_j \)
   16. for each user \( u_o \), who has rated \( r_j \)
   17. if \( S_{u_o} > s_{u_o} \)
   18. \( \text{Sum} = \text{Sum} + w^j_{u_o, /|S_i| \cdot |S_j|} \)
   19. if \( \text{Sum} \geq \Theta \)
20. Remove \( r_j \) from \( Sky_j \); goto 12
21. report \( Sky_j \)

Figure 5 Two-scans paradigm for exact \( Sky_j \) computation

5 APPROXIMATE SKYLINE COMPUTATION

In this section, we introduce Approximate Skyline Computation (ASC) in CFS. As we show experimentally, ASC leads to minimal loss of effectiveness, but significant gain of efficiency. The main difference with respect to ESC lies in the dominance test between two records \( r_i \) and \( r_j \). Instead of iterating over all pairs of scores in \( S_i \) and \( S_j \), ASC utilizes samples of size \( N \). We show that ASC provides error guarantees related to \( N \).

Equation 5.1 splits the dominance relationship of Definition 3.1 into two parts, of which only the second one depends on the active user \( u_i \):

\[
\sum_{m,n} w_{m,n}(s_{m,n} > s_{j,m}) = \frac{|S_i| \cdot |S_j|}{3 |S_i| |S_j|} \sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i))[s_{m,n} > s_{j,m}] \geq \Theta' \quad \text{if} \quad \frac{\sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i))}{3 |S_i| |S_j|} \geq \Theta', \tag{5.3}
\]

where \( w_{m,n} \) is given by Equation 3.3 and the scoring function \( sf(.) \) is the average function. By combining Equation 5.1 and Definition 3.1, record \( r_j \) dominates \( r_i \) with respect to \( u_i \) (\( r_i \neq r_j \)), if the following condition is satisfied:

\[
\sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i))[s_{m,n} > s_{j,m}] \geq \Theta' \quad \text{if} \quad \frac{\sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i))}{3 |S_i| |S_j|} \geq \Theta', \tag{5.4}
\]

If we divide both sides of Equation 5.2 by \( \frac{\sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i))}{3 |S_i| |S_j|} \), the condition can be further transformed into the following form:

\[
\sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i))[s_{m,n} > s_{j,m}] \geq \Theta', \quad \text{where} \quad \Theta' = \frac{\Theta}{3 |S_i| |S_j|} \sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i)) \tag{5.5}
\]

The new threshold \( \Theta' \) can be pre-computed, in linear time to the number of users, at a pre-processing step. The left-hand side of Inequality 5.3 is the expectation of some binary variable. Based on sampling theory [26], this expectation can be approximated by the average of the samples over the score pairs, provided that every \([s_{i,m} > s_{j,n}]\) is sampled with probability:

\[
\frac{\sigma(u_m, u_i) + \sigma(u_n, u_i)}{3 |S_i| |S_j|} \sum_{m,n} (\sigma(u_m, u_i) + \sigma(u_n, u_i)) \tag{5.6}
\]

According to the Chernoff bound [26], the average over the samples is an \( \varepsilon \)-approximation with confidence \( \delta \), if the number of samples on the pairs \(|S_i| \cdot |S_j|\) is at least:

\[
N = 2 \ln(1/\delta) / \varepsilon^2 \Theta'
\]

To efficiently implement the score pair sampling, we exploit the following observations. First, it is easy to verify that:

\[
|S_i| \cdot \sigma(u_m, u_i) + \sum_n \sigma(u_n, u_i) \leq |S_i| \cdot \sigma(u_m, u_i) + \sum_n \sigma(u_n, u_i) \tag{5.7}
\]

Second, given \( u_m \), the probability of selecting \( u_n \) is:

\[
\frac{\sigma(u_m, u_i) + \sum_n \sigma(u_n, u_i)}{|S_i| \cdot \sigma(u_m, u_i) + \sum_n \sigma(u_n, u_i) + \sum_n \sigma(u_n, u_i)} \tag{5.8}
\]

Based on Equation 5.7, we select \( u_n \) (given \( u_m \)) as follows. We generate a random number \( rnd \) between 0 and 1. If
$rn < S_i \cdot \sigma(u_a, u_i) / (S_i \cdot \sigma(u_a, u_i) + \sum_{n \neq a} \sigma(u_a, u_n))$, $u_a$ is chosen uniformly with probability $1 / |S_i|$. Otherwise, $u_a$ is chosen with probability $\sigma(u_a, u_i) / \sum_{n \neq a} \sigma(u_a, u_n)$. The advantage of the above scheme is that sampling can be performed efficiently (in linear time to $|S_i|$ and $|S_j|$) by pre-computing $\sum_{n \neq a} \sigma(u_a, u_n)$ for every $u_a$.

Figure 6 summarizes ASC and the sampling process. Similar to ESC, the algorithm iterates over all record pairs $r_i$ and $r_j$. However, for every $(r_i, r_j)$ only a sample (of size $N$) of the scores in $S_i$ and $S_j$ is used to determine dominance. For each sampled pair $(s_{i,n}, s_{j,m})$, the variable $\text{Sum}$ is incremented by 1, when $s_{i,n} \succ s_{j,m}$. After the sampling process terminates, if the average $\text{Sum}/N$ is larger than the new threshold $\theta$, $r_j$ is expected to dominate $r_i$ with high confidence. If no record can approximately dominate $r_i$, $r_j$ is inserted into the personal skyline set $\text{Sky}_i$. The complexity of ASC depends on the number of samples created for every record pair $(r_i, r_j)$. If $\theta$ is high, the necessary number of samples is small, and vice versa. The approximate algorithm is expected to be significantly more efficient than exact skyline computation when $N << |S_i| \cdot |S_j|$ for all record pairs. The pre-pruning optimization of Section 4.2 can be applied to speed up ASC. Furthermore, the two-scan execution paradigm of Section 4.3 can also be extended to ASC.

**ASC (user $u_a$, threshold $\theta$)** // computation of approximate $\text{Sky}_i$

1. $\text{Sky}_i = \emptyset$
2. for each $r_i$ in $R$
3. for each record $r_j \neq r_i$
4. $\text{Sum} = 0$
5. let $\theta'$ be the new threshold computed by Equation 5.3 and $N$ be the number of samples: $N = 2 \sqrt{\frac{1}{\delta}} / |\theta'|$
6. for each sample
7. select user $u_a$ with probability $\frac{|S_i| \cdot \sigma(u_a, u_i) / (|S_i| \cdot \sigma(u_a, u_i) + \sum_{n \neq a} \sigma(u_a, u_n))}{\text{rand}}$
8. generate random number $rn \in [0, 1]$
9. if $rn < \frac{|S_i| \cdot \sigma(u_a, u_i) / (|S_i| \cdot \sigma(u_a, u_i) + \sum_{n \neq a} \sigma(u_a, u_n))}{\text{rand}}$
10. select user $u_i$ with probability $1 / |S_j|$
11. else
12. select user $u_i$ with probability $\frac{\sigma(u_a, u_i) / \sum_{n \neq a} \sigma(u_a, u_n)}{\text{rand}}$
13. if $s_{i,n} \succ s_{j,m}$
14. $\text{Sum} = \text{Sum} + 1$
15. if $\text{Sum}/N \geq \theta'$
16. discard $r_i$
17. insert $r_i$ into $\text{Sky}_i$ // $r_i$ is not dominated
18. report $\text{Sky}_i$

**Figure 6** Approximate skyline computation

### 6 TOP-K PERSONALIZED SKYLINE

According to Definition 3.1, each active user $u$ has to provide the dominance threshold $\theta$ that determines the cardinality of her/his personalized skyline $\text{Sky}_u$. The proper setting of $\theta$ may be counter-intuitive, especially for the casual user. An option for eliminating the threshold parameter is the top-$k$ personalized skyline, which contains the $k$ least dominated records. Specifically, recall that based on Definition 3.1, the cardinality of the personalized skyline decreases for lower values of $\theta$; in the extreme case that $\theta = 0$, the skyline is empty because each record is dominated. The top-$k$ personalized skyline corresponds to the minimum value of $\theta$ that generates $k$ records.

**Exact Top-k Skyline (ETKS)** extends the ESC algorithm for top-$k$ computation. Initially, Pre-processing($k$), shown in Figure 7 estimates a threshold value $\theta$. Pre-processing uses the upper bounds (i.e., $ub_{w_{mn}}$) on the dominance weights of records as derived in Equation 4.1. $USkyP$ maintains the set of records with $k$-minimal upper bounds of dominance weights ($Sum_U$ in the pseudo-code) in $R$. For each record in $USkyP$, the algorithm stores the pair $(r_i, Sum_U)$. Initially, $\theta$ is set to 1 and then gradually decreases (Lines 12-16). The final $\theta$ is returned as the upper bound on the threshold $\theta$ for ETKS. ETKS is similar to ESC, except that (i) it only keeps in $\text{Sky}_u$ the $k$-least dominated records, and (ii) it continuously updates the value of $\theta$ as more skyline records are discovered (similar to Pre-processing). **Approximate Top-k Skyline Computation (ATKS)** is derived by applying the approximation strategy of Section 5; i.e., instead of iterating over all pairs of scores, ATKS utilizes samples of size $N$. The complexity of ETKS and ATKS are exactly the same as that of ESC and ASC, since both algorithms have to compare every pair of records in worst case. All optimizations can be easily extended to ETKS and ATKS. However, the two-scan paradigm is inapplicable because we cannot determine the result cardinality in the first scan, which leads to the exact top-$k$ results after the pruning in the second loop.

**Figure 7** Pre-processing for top-$k$ skyline computation

### 7 EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness and the efficiency of CFS. All programs are compiled with GCC 3.4.3 in Linux, and executed on an IBM server, with Xeon 3.0GHz CPU and 4 GB main memory. Section 7.1 presents the datasets used in our experiments. Section 7.2 measures the effectiveness of CFS compared to conventional collaborative filtering. Section 7.3 evaluates the efficiency of the proposed algorithms and optimizations.
7.1 Datasets

The main challenge for evaluating the effectiveness of CFS regards the absence of datasets with a sufficient number of real ratings on multiple attributes to obtain meaningful similarity information. As discussed in Section 1, Trip Advisor [41], contains for each hotel, an overall rating (an integer between 1 and 5), and four numerical ratings (from 1 to 5) for rooms, service, cleanliness, and value. The attributes rooms, service and cleanliness are positively correlated, while value is negatively correlated with respect to the other attributes (often, hotels rated highly in several attributes are expensive). Some information about the users is also recorded, including email address, nationality and city of origin. However, the original dataset is too sparse (i.e., most users have reviewed a single hotel), rendering the similarity comparisons meaningless. To overcome this problem, we merge reviewers into groups based on their city of origin, so that users in the same group are regarded as a single virtual user. After this clean-up step, we obtain a dataset with 345 virtual users, 50 records (hotels), and 997 reviews, i.e., on the average, each hotel is associated with roughly 20 reviews. If multiple users from the same group review the same hotel, the average scores on the attributes are used as the group score over this hotel. Although our partitioning process may cluster together users with dissimilar tastes, each derived “virtual user” roughly represents cultural habits/preferences of people from the same city. Other aggregation modalities could also be considered, e.g., group user reviews using their nationality.

In addition, we use a real estate dataset [28] that contains 160K house records. Each house record is represented by an 8-dimensional vector, \( x_i = (x_i[1],..., x_i[8]) \), where the attributes include the number of bedrooms, number of bathrooms etc. We normalize the values on each dimension between 0 and 1, according to the minimal and maximal values on each attribute. Since the original dataset does not contain ratings, we create virtual users. In particular, we define the score \( s_i,m[h] \) for record \( r_i \) on dimension \( h \) as:

\[
s_i,m[h] = (x_i[h])^\alpha \quad \text{ (7.1)}
\]

where the positive number \( a_{m,h} \) is the user preference parameter on attribute \( h \). Examples of preference functions are depicted in Figure 8a. When \( a_{m,h} = 1 \), the function is a straight line defined by points \((0,0), (1,1)\). When \( a_{m,h} > 1 \), the curve is always below the line, implying that the user is critical on this attribute. On the other hand, if \( a_{m,h} < 1 \), the user may give a high score even when the attribute value is low. To simulate different virtual users \( u_m \), the parameters \( a_{m,h} \) are generated as follows. For each \( a_{m,h} \), a random number \( r \) is drawn from a standard Gaussian distribution with mean \( m=0 \) and variance \( \text{var}=1 \). Given the random number \( r \), we set \( a_{m,h} = e^{rn} \). Figure 8b shows the distributions of the ratio: user rating/original attribute value.

Note that the average rating is about the same as the original attribute value (i.e., most of the ratings are generated around the original average value), implying that the generated and the original data are consistent.

7.2 Effectiveness

The goal of this set of experiments is to demonstrate that CFS is indeed useful in practice. However, there is no other CFS competitor in the literature. Moreover, the personalized skylines produced by CFS and the rankings generated by CF are not directly comparable quantities. To solve this problem we assume a CF algorithm which predicts the ratings on each dimension individually, and estimates the total score as the average of such ratings on all dimensions. The predicted score \( s^{\star}_{i,j} \) for active user \( u_i \) on \( r_j \) on each dimension is computed according to the Pearson Correlation coefficient [1]:

\[
s^{\star}_{i,j} = s_f + \frac{ \sum_{m \in U_i} \text{corr}(u_i,u_m)(s_{i,m} - \overline{s})}{ \sum_{m \in U_i} \text{corr}(u_i,u_m)} \quad \text{ (7.1)}
\]

where \( \overline{s} \) is the average rating of user \( u_i \) and \( U_i \) denotes the set of users that are similar to \( u_i \) (we set the cardinality of \( U_i \) to 10 users for all experiments.). The intuition behind Equation 7.1 is that \( s^{\star}_{i,j} > s_f \), if several users have rated \( r_j \)
above their averages (i.e., $s_{ij} > \overline{s}_{m}$), especially if those users are similar\(^7\) to $u_i$. Ideally, $s_{ij}$ should be equal to the actual overall score $s_{ij}$. CF recommends to $u_i$ the set $\text{Top}_k$ of records with the highest predicted scores. Let $\text{NTop}_l = R - \text{Top}_k$ be the set of non-recommended objects. $\overline{\text{Top}}_k$ (or $\overline{\text{NTop}}_l$) signifies the average actual score of $u_i$ to (non) recommended records. The gain $G$ is the score difference between recommended and non-recommended objects:

$$G = \overline{\text{Top}}_k \cdot \text{NTop}_l$$  \hspace{1cm} (7.2)

A large positive value of $G$ indicates that the recommendations of CF are of high quality. On the other hand, a small, or negative, value signifies low effectiveness. Similarly for CFS, $\overline{\text{Sky}}_{l}$ (or $\overline{\text{NSky}}_{l}$) denotes the average overall score of $u_i$ for (non) skyline records. In this case, the gain is defined according to Equation 7.3:

$$G = \overline{\text{Sky}}_{l} - \overline{\text{NSky}}_{l}$$  \hspace{1cm} (7.3)

Intuitively, a high gain implies that records in the personalized skyline of $u_i$ are indeed preferable for the user as demonstrated by her/his actual overall rating. We compare exact (ESC) and approximate (ASC) CFS against CF using the Trip Advisor and House datasets. For fairness, given the threshold $\theta_i$, we set the CF parameter $k$ so that $k$ is the closest integer to the average skyline cardinality $|\text{Sky}_{l}|$, i.e., the output of CFS and CF has (almost) the same cardinality. Equations 7.2 and 7.3 take into account only records whose overall score exists in the data.

Figure 9a presents the gain as a function of the skyline threshold. Figure 9b shows the average skyline cardinality (and value of $k$ used in CF) for the tested threshold values on Trip Advisor. The reported results correspond to the average values after performing the experiment for ten users. Note in Figure 9a that the three techniques initially (when $\theta = 0.1$ and the skyline contains only 2 hotels) provide comparable gains. However, as the threshold increases, both CFS techniques significantly outperform CF. Even when $\theta = 0.3$, and the skyline contains most of the (50) hotels, ESC and ASC yield a positive gain. Note that in some cases ASC is better than ESC because the randomness introduced by the sampling process may benefit some preferable records (which could be dominated if all scores were considered).

Figure 10 repeats the experiment on the House dataset. ESC is the most effective method, followed by ASC. Both techniques outperform CF with a maximum gain difference of around 0.3 when $\theta = 0.033$ (see Figure 10a). Note that compared to Figure 9, the threshold values are lower due to the higher skyline cardinalities of House. For example, when $\theta$ ranges from 0.005 to 0.007, the skyline includes between around 30 and 70 records. This is because House contains 5,274 records as opposed to 50 for Trip Advisor.

In conclusion, this set of experiments indicates that CFS indeed captures the user selection criteria better than CF in the presence of ratings on multiple attributes. Consider, for instance, two scores $s_{ij} = (2,2,5,2)$ and $s_{ij} = (3,3,3,3)$ from $u_i$ on the four attributes of records $r_i$ and $r_j$. Although $r_i$ has a higher average, $u_i$ may give a better overall score to $r_i$ because $s_i$ can be lower than the third attribute. The personalized skylines provide flexibility by allowing the users to make their own choices.

### 7.3 Efficiency

Next, we evaluate the efficiency of CFS with respect to CF when recommending a specified number of records (as in most recommender systems), Figure 11 illustrates the gain $G$ using the top-$k$ CFS algorithms and varying $k$ between 5 and 25. ETKS and ATKS again outperform CF consistently. Note that there is no exact correspondence between the results in Figure 11 and those in Figures 9 and 10 because CFS queries with fixed threshold return results with different cardinalities for different users, while for top-$k$ queries the cardinality is fixed for all users.

In summary, this set of experiments indicates that CFS indeed captures the user selection criteria better than CF in the presence of ratings on multiple attributes.
with top-$k$ personalized skylines. Table III illustrates the default values and ranges for the experimental parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold $\theta$</td>
<td>0.1</td>
<td>0.05, 0.075, 0.1, 0.125, 0.15</td>
</tr>
<tr>
<td>Error $\epsilon$</td>
<td>0.3</td>
<td>0.1, 0.2, 0.3, 0.4, 0.5</td>
</tr>
<tr>
<td>Confidence $\delta$</td>
<td>0.3</td>
<td>0.1, 0.2, 0.3, 0.4, 0.5</td>
</tr>
</tbody>
</table>

### 7.3.1 ESC vs. ASC

Initially, we use the most optimized version of each method, and later evaluate the effect of individual optimizations. Specifically, we apply all optimizations of Section 4.2 (pre-pruning, score and record pre-ordering) for both ESC and ASC. For ASC we also use the two-scan implementation (2S-ASC), whereas for ESC we omit it because (as we show in Figure 15a) it is not effective. Figure 12 compares ESC and ASC as a function of the user cardinality, after setting the parameters of Tables II and III to their default values. Given that the score cardinality is fixed ($|S| = 100K$), more users imply a smaller number of scores per user. Consequently, the set of common records rated by a user pair decreases and so does their similarity. Thus, dominance relationships become more difficult to establish with increasing $|S|$, and the personalized skylines grow accordingly. This is reflected in the cost of ESC. On the other hand, ASC is not affected by the skyline cardinality, and its cost is rather insensitive to $|S|$ since the sampling rate in ASC is independent to the number of users. Figure 12b investigates the effect of the record cardinality. The cost of ESC decreases when more records participate in the computation of personalized skylines due to the positive effect of the basic optimizations for larger datasets ($|S|$ is again fixed to 100K). Similarly, the CPU time of ASC slowly decreases.

Figure 13a shows the CPU cost versus the score cardinality. Each record is rated by more users, leading to the linear increase in the CPU time of ESC. On the other hand, the cost of record comparison in ASC only depends on the sampling rate. Figure 13b indicates that the overhead of both methods grows with the dimensionality because the skyline cardinality increases fast with $d$, forcing more comparisons during record verification (a skyline record is compared with every other tuple, whereas some non-skyline records are eliminated early). This is particularly true for dimensionality values larger than 4.

Next we evaluate the efficiency of alternative implementations of ESC and ASC. Specifically, 2S-ESC$_\text{opt}$ denotes two-scans ESC with all optimizations of Section 4.2, and ESC$_\text{basic}$ (resp. ASC$_\text{basic}$) denotes the basic algorithm of Figure 3 with (resp. without) these optimizations. Figure 14a illustrates the CPU time for exact skyline computation as a function of the user cardinality. The best method is ESC$_\text{opt}$, implying that the two-scans paradigm is not effective for exact skyline computation due to the large number of false positives introduced by the first scan. Figure 14b repeats the experiment for approximate skyline computation, where we use the same notation for the different versions as their exact counterparts. 2S-ASC$_\text{opt}$ achieves noticeable improvement over ASC$_\text{opt}$ because after the first scan there are relatively fewer records in the buffer $\text{Sky}_d$ compared to the exact solution.
Figure 16 measures the effect of optimizations as a function of the score cardinality. The results are consistent with those of Figure 15: ESC\textsubscript{opt} is the best choice for exact computation, whereas 2S-ASC\textsubscript{opt} is the winner for approximation computation. Both algorithms scale well as the number of scores increases.

Finally, we analyze the impact of the sampling parameters of ASC. Figure 17a (resp. 17b) illustrates the overhead as a function of the error parameter $\varepsilon$ (resp. confidence parameter $\delta$). As expected, the CPU cost of ASC is proportional to both $1/\varepsilon$ and $\ln(1/\delta)$.

### 7.3.2 ETSK vs. ATKS

In this section, we evaluate the efficiency of top-$k$ algorithms for CFS using the parameters of Tables II and III. Both ETSK and ATKS are implemented as discussed in Section 6 and use all optimizations of Section 4.2. The default value of $k$ is 20. Figure 18a presents the impact of the user cardinality on the efficiency of ETSK and ATSKS. Similar to Figure 12a, the CPU time of ETSK grows with $|U|$, while ATKS is insensitive to $|U|$. According to Figure 18b, the CPU time of ETSK decreases as the record cardinality grows because there are fewer scores per record.

FIGURE 16 Alternative implementations: CPU time versus score cardinality

(a) Exact skylines

(b) Approximate skylines

FIGURE 17 ASC efficiency vs. error and confidence

(a) CPU time vs. $\varepsilon$

(b) CPU time vs. $\delta$

FIGURE 18 ETSK, ATKS efficiency vs. user and record cardinality

(a) CPU time vs. $|U|$

(b) CPU time vs. $|R|$

FIGURE 19 ETSK, ATKS efficiency vs. score cardinality and dimensionality

(a) CPU time vs. $|S|$

(b) CPU time vs. $d$

FIGURE 20 ETSK, ATKS efficiency vs. $k$ and distribution

Next, we evaluate the effect of optimizations. Since the two-scans paradigm is not applicable on top-$k$ algorithms, in the following we only consider the four scenarios ETKS\textsubscript{opt}, ETKS\textsubscript{basic}, ATKS\textsubscript{opt} and ATKS\textsubscript{basic}, where the optimized versions apply pre-pruning, score and record pre-ordering. Figure 21a (resp. 21b) illustrates the cost as a function of $|U|$ for the exact (resp. approximate) solution. ETKS\textsubscript{opt} and ATKS\textsubscript{opt} outperform ETKS\textsubscript{basic} and ATKS\textsubscript{basic}, respectively, by at least one order of magnitude. Similar performance gains are observed when varying the number of records, scores and dimensions.

FIGURE 21 Alternative implementations: CPU time vs. user cardinality

(a) Exact skylines

(b) Approximate skylines

In summary, pre-pruning, score and record pre-ordering decrease significantly the cost of skyline computation under all settings (exact/approximate, threshold/top-$k$). On the other hand, the two-scan paradigm is beneficial only for approximate skylines under the conventional (i.e., threshold) model.
8 Conclusions

This paper proposes collaborative filtering skyline (CFS), a general framework that generates a personalized skyline for each active user based on scores of other users with similar scoring patterns. The personalized skyline includes objects that are good on certain aspects, and eliminates the ones that are not interesting on any attribute combination. CFS permits the distinction of scoring patterns and selection criteria, i.e., two users are given diverse choices even if their scoring patterns are identical, which is not possible in conventional collaborative filtering. We first develop an algorithm and several optimizations for exact skyline computation. Then, we propose an approximate solution, based on sampling, that provides confidence guarantees. Furthermore, we present top-k algorithms for personalized skyline, which contains the k least dominated records. Finally, we evaluate the effectiveness and efficiency of our methods through extensive experiments.

References


Ilaria Bartolini is an Assistant Professor with the DEIS department of the University of Bologna (Italy). She graduated in Computer Science (1997) and received a Ph.D. in Electronic and Computer Engineering (2002) from the University of Bologna. In 1998, she spent six months at the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam as a junior researcher. In 2004, she was a visiting researcher for three months at the New Jersey Institute of Technology (NJIT) in Newark. From January 2008 to April 2008, she was visiting the Hong Kong University of Science and Technology (HKUST). Her current research mainly focuses on learning of user preferences, similarity and preference query processing in large databases, collaborative filtering, and retrieval and browsing of multimedia data collections. Ilaria Bartolini has published about 30 papers in major international journals (including IEEE TPAMI, ACM TODS, DKE, KAIS, and MTAP) and conferences (including VLDB, ICDE, PKDD, and CIKM). She served in the program committee of several international conferences and workshops. She is a member of ACM SIGMOD and IEEE.

Zhenjie Zhang received his B.S. from Department of Computer Science and Engineering, Fudan University in 2004. He is currently a Ph.D. candidate at the School of Computing, National University of Singapore. He was a visiting student of the Hong Kong University of Science and Technology in 2008, and a visiting scholar of AT&T Shannon Lab in 2009. His research interests cover a variety of topics including clustering analysis, skyline query processing, non-metric indexing and game theory. He serves as a PC member in VLDB 2010 and KDD 2010.

Dimitris Papadias is a Professor of Computer Science and Engineering, Hong Kong University of Science and Technology. Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the National Center for Geographic Information and Analysis (NCGIA, Maine), the University of California at San Diego, the Technical University of Vienna, the National Technical University of Athens, Queen’s University (Canada), and University of Patras (Greece). He has published extensively and been involved in the program committees of all major Database Conferences, including SIGMOD, VLDB and ICDE. He is an associate editor of the VLDB Journal, the IEEE Transactions on Knowledge and Data Engineering, and on the editorial advisory board of Information Systems.