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_l looking for an interesting object, these systems usually take two steps: (i) retrieve users who have similar rating patterns with u_l; (ii) utilize their scores to return the top-*k* records that are potentially most interesting to u_l. 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-

- *I. Bartolini is with DEIS, Alma Mater Studiorum - Università di Bologna, Bologna, Italy. Email: i.bartolini@unibo.it*
- *Z. Zhang is with the Department of Computer Science, National University of Singapore. Email: zhenjie@comp.nus.edu.sg*
- *D. Papadias is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. Email: dimitris@cse.ust.hk*

sume 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*, Sky_l, for each active user u_l based on scores of other users with similar scoring patterns. Let $s_{i,m}$ be the score of user u_m for record (e.g., hotel) r_i; $s_{i,m}$ is a vector of values, each corresponding to an attribute of r_i (e.g., value, service, etc). We say that a tuple r_i dominates another r_j with respect to an active user u_l (and denote it as $r_i \succ_l r_j$), if there is a large number[1] of pairs $s_{i,m} \succ s_{j,n}$, especially if those scores originate from users u_m, u_n that are similar to each other and to u_l. The personalized skyline Sky_l of u_l 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 r_1-r_4 (hotels) represented by different shapes, involving two attributes (*value*, *service*). These records are rated by three users u_1-u_3. Each record instance corresponds to a user rating; e.g., $s_{1,1}$, $s_{1,2}$, $s_{1,3}$ are scores of r_1, whereas $s_{4,2}$ and $s_{4,3}$ are scores of r_4. We assume that higher scores on attributes are more preferable. Users u_1 and u_2 have given analogous scores to hotels r_1 and r_3 (see $s_{1,1}$, $s_{1,2}$ and $s_{3,1}$, $s_{3,2}$). Furthermore, u_2 has also rated highly r_2 (see $s_{2,2}$) on service. Since u_1 and u_2 have similar rating patterns, r_2 probably would rank high in the preferences of u_1 as well, and should be included in Sky_1. Note that although $s_{2,2}$ is dominated by $s_{4,3}$ given by user u_3, u_1 and u_3 do not have similar preferences: they have rated a single record (r_1) in common, and their scores ($s_{1,1}$, $s_{1,3}$) are rather different. Instead, the opinion of u_2 is much more valuable to u_1, and consequently the personalized skyline of u_1 should contain r_2 rather than r_4.

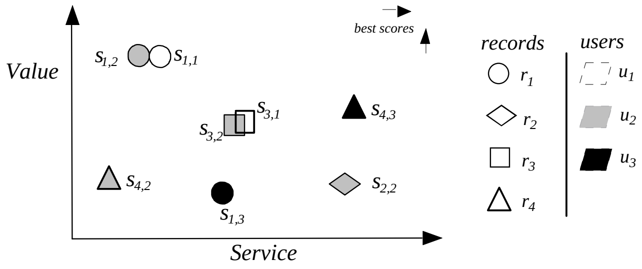


Figure 1 Example of personalized skyline

[1] The number of pairs $s_{i,m} \succ s_{j,n}$ depends on a user-defined threshold that controls the skyline cardinality.

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].

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., $s_{1,1}$, $s_{1,2}$, $s_{1,3}$ are scores of r_1, whereas $s_{4,2}$ and $s_{4,3}$ are scores of r_4. $Pr[r_1 \succ r_4]$=1/3 since there are two pairs $(s_{1,1}, s_{4,2})$, $(s_{1,2}, s_{4,2})$ out of the possible six, where $r_1 \succ r_4$. Conversely, $Pr[r_4 \succ r_1]$=1/6 because there is a single pair $(s_{4,3}, s_{1,3})$ such that $r_4 \succ r_1$. If $p \leq 1/6$, neither r_1 nor r_4 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 $s_{4,3}$; this is counter-intuitive because $s_{4,3}$ 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.

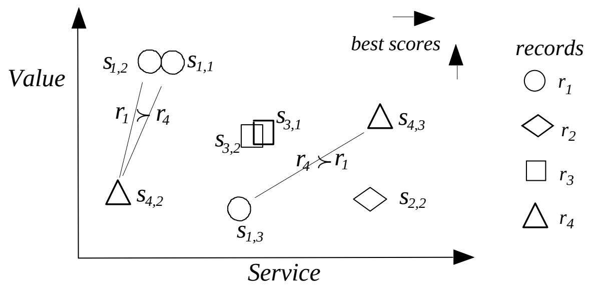


Figure 2 Example of probabilistic skyline

2.2 Collaborative Filtering and Recommendation Systems

Let R be a set of records and S be a set of scores on the tuples of R, submitted by a set of users U. Given an *active* user $u_l \in U$, CF can be formulated as the problem of predicting the score $s_{i,l}$ of u_l for each record $r_i \in R$ that s/he has not rated yet. Depending on the estimated scores, CF recommends to u_l the k records with the highest rating. Existing systems can be classified in two broad categories [1]: *user-based* and *item-based*. User-based approaches maintain the pairwise similarities of all users computed on their rating patterns. In order to estimate $s_{i,l}$, they exploit the scores $s_{i,m}$ of each user u_m who is similar to u_l. Item-based approaches maintain the pairwise similarities of all records, e.g., two tuples that have received the same scores from each user that has rated both are very similar. Then, $s_{i,l}$ is predicted using the scores $s_{j,l}$ of the active user, on records r_j that are similar to r_i. Common similarity measures include the *Pearson Correlation Coefficient* [1], *Mean Squared Difference* [38], and *Vector Space Similarity* [6].

Content-based techniques [2] maintain the pairwise similarities of all records, which depend solely on their features. For example, two documents may be considered identical if they contain the same terms. Then, $s_{i,l}$ is predicted using the ratings of the active user on records similar to r_i. Note that these techniques do not fall in the CF framework because the scores are not considered either (i) for computing the similarity between two records, as in item-based approaches, or (ii) for computing the similarity between users as in user-based methods. *Hybrid* techniques [7] combine CF and content-based solutions. One approach implements collaborative and content-based methods independently and combines their prediction. A second alternative incorporates some content-based (resp. CF) characteristics into a CF (resp. content-based) system.

Regarding concrete systems, *Grundy* proposes *stereotypes* as a mechanism for modeling similarity in book recommendations [36]. *Tapestry* [15] requires each user to manually specify her/his similarity with respect to other 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_l 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[2]. 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 $u_m \in U$ on a record $r_i \in R$ is a vector of values[3] ($s_{i,m}[1],\ldots,s_{i,m}[d]$), each corresponding to a rating on a dimension of r_i. Without loss of generality, we assume that higher scores on attributes are more preferable. It follows that, $s_{i,m}$ dominates $s_{j,n}$ ($s_{i,m} \succ s_{j,n}$), 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.

[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 I FREQUENT SYMBOLS

Symbol	Definition
R	record set, $R=\{r_i\}$
U	user set, $U=\{u_m\}$
$s_{i,m}$	score on r_i given by u_m ($s_{i,m}[1],\ldots,s_{i,m}[d]$)
$w^l_{m,n}$	weight of ($s_{i,m} \succ s_{j,n}$) with respect to u_l
$\sigma(u_m, u_n)$	similarity between users u_m and u_n
S_i	score set for record r_i
R_m	records reviewed by user u_m
$s_{i,m} \succ s_{j,n}$	dominance relationship between two scores
$r_i \overset{l}{\succ} r_j$	personalized dominance between r_i and r_j wrt u_l
θ_l	dominance threshold for user u_l
Sky_l	skyline set for u_l

The personalized skyline Sky_l of an active user u_l contains all records that are not dominated according to the following definition[4].

Definition 3.1 (Personalized Dominance): A tuple r_i dominates another r_j with respect to an active user u_l ($r_i \overset{l}{\succ} r_j$) iff

$$\frac{\sum_{m,n} w^l_{m,n}[s_{i,m} \succ s_{j,n}]}{|S_i| \cdot |S_j|} \geq \theta_l \qquad (3.1)$$

where:

$$[s_{i,m} \succ s_{j,n}] = \begin{cases} 1 & \textit{if } s_{i,m} \textit{ and } s_{j,n} \textit{ are not null and } s_{i,m} \succ s_{j,n} \\ 0 & \textit{otherwise} \end{cases} \qquad (3.2)$$

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

$$w^l_{m,n} = sf(\sigma(u_m,u_n), \sigma(u_m,u_l), \sigma(u_n,u_l)) \qquad (3.3)$$

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.

$$corr(u_m,u_n) = \begin{cases} \dfrac{\left\| \sum_{r_i \in R_m \cap R_n} (s_{i,m} - \bar{S}_m)(s_{i,n} - \bar{S}_n) \right\|}{\sqrt{\left\| \sum_{r_i \in R_m \cap R_n} (s_{i,m} - \bar{S}_m)^2 \right\| \cdot \left\| \sum_{r_i \in R_m \cap R_n} (s_{i,n} - \bar{S}_n)^2 \right\|}} & \textit{if } |R_m \cap R_n| > 0 \\ 0 & \textit{otherwise} \end{cases} \qquad (3.4)$$

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

[4] Probabilistic dominance [31] is a special case of Definition 3.1, where the weight of all instance pairs is 1 and there is no concept of user similarity.

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 θ_l, it is possible that both $r_i \not\succ r_j$ and $r_j \not\succ r_i$ are simultaneously true. For instance, in the example of Figure 1, if $\theta_l = 1/6$ and all weights are equal to 1, we have $r_1 \succ r_4$ and $r_4 \succ r_1$ for every user (i.e., the records are dominated by each other and, therefore, they are not in Sky_l). On the other hand, if $\theta_l = 1/3$ only $r_1 \succ r_4$ is true, while if $\theta_l > 1/3$ none of the dominance relationships holds. Second, personalized dominance is not transitive, i.e., $r_i \not\succ r_j$ and $r_j \not\succ r_k$ does not necessarily imply that $r_i \not\succ r_k$, if the weights $w^l_{m,n}$ 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_l and a threshold θ_l, the personalized skyline Sky_l 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_l. The input of the algorithm is threshold θ_l. Initially, every record r_i is a candidate for Sky_l 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 $s_{j,n} \succ s_{i,m}$. If *Sum* exceeds θ_l, 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.

```
Basic ESC (user $u_l$, threshold $\theta_l$) // computation of $Sky_l$ for $u_l$
1.  $Sky_l = \varnothing$
2.  for each record $r_i$ in $R$
3.      for each record $r_j \neq r_i$
4.          $Sum=0$
5.          for each user $u_m$ who has rated $r_i$
6.              for each user $u_n$ who has rated $r_j$
7.                  if $s_{j,n} \succ s_{i,m}$
8.                      $Sum=Sum + w^l_{m,n} / |S_i|\cdot|S_j|$
9.                  if $Sum \geq \theta_l$
10.                     skip $r_i$ ; goto 2
11.         insert $r_i$ into $Sky_l$ // $r_i$ is not dominated
12.     report $Sky_l$
```
Figure 3 Basic algorithm for Sky_l 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 $O(|R|^2)$. 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 $s_{i,l}$, the CFS system can exclude r_i from Sky_l (e.g., a sub-

[5] In general, hash indexes do not provide performance guarantees, although in practice they incur constant retrieval cost.

scriber of Amazon is not likely to be interested in a book that s/he 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_l of every user u_l). 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_l is $w^l_{m,n}$ (Line 8 of basic ESC). Assuming that $sf(\cdot)$ is the average function we have:

$$\begin{aligned} lbw_{m,n} &= (\sigma(u_m,u_n)+ lb\sigma_m+ lb\sigma_n)/3 \leq \\ w^l_{m,n} &= (\sigma(u_m,u_n)+\sigma(u_m,u_l)+\sigma(u_n,u_l))/3 \leq \\ ubw_{m,n} &= (\sigma(u_m,u_n)+ ub\sigma_m+ ub\sigma_n)/3 \end{aligned} \quad (4.1)$$

Given u_m and u_n, $lbw_{m,n}$ is a lower bound of $w^l_{m,n}$ for any possible user u_l; $lb\sigma_m$ (resp. $lb\sigma_n$) denotes the minimum similarity of u_m (resp. u_n) to every other user. Similarly, $ubw_{m,n}$ is an upper bound for $w^l_{m,n}$, and $ub\sigma_m$, $ub\sigma_n$ are the maximum similarities of u_m and u_n. The values of $lb\sigma_m$, $ub\sigma_m$ 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 $s_{j,n} \succ s_{i,m}$ using the lower and upper bounds, respectively. When *SumN* exceeds θ, r_i is inserted into N because it cannot be in the skyline of any user u_l; r_i is dominated by r_j, even if the lower bound $lbw_{m,n}$ is used instead of $w^l_{m,n}$. On the other hand, if all records r_j have been exhausted and $SumC < \theta$, r_l is inserted into C; r_i cannot be dominated for any user u_l, even if the upper bound $ubw_{m,n}$ is used instead of $w^l_{m,n}$. Since the lists C and N depend on the threshold θ, the algorithm must be repeated for all values of θ 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 ($O(|R|^2\cdot|S_{AVG}|^2)$), 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 $s_{j,n}$ (Lines 5-7 in Figure 3) for each $s_{i,m}$. *Score pre-ordering* avoids considering scores $s_{j,n}$ that cannot dominate $s_{i,m}$. Specifically, the set S_j of scores on each record r_j is sorted in descending order of the maximum attribute value. Using this order, the scores $s_{j,n}$ with higher probability to dominate $s_{i,m}$ are visited first. Once a score $s_{j,n}$ with maximum attribute equal to, or smaller than, the minimum attribute of $s_{i,m}$ is reached, the inner iteration over $s_{j,n}$ stops. For instance, given that the current $s_{j,n}$=(2,3) (assuming 2 attributes) and $s_{i,m}$=(3,4), there can be no subsequent score in the sorted S_j such that $s_{j,n} \succ s_{i,m}$. The cost of score pre-ordering is $O(|R|\cdot|S_{AVG}|\cdot log(|S_{AVG}|))$ 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 θ)
1.  C = ∅ // records that are in all personalized skylines
2.  N = ∅ // records that are not in any personalized skyline
3.  for each record $r_i$ in R
4.      for each record $r_j \neq r_i$
5.          SumC=0, SumN=0
6.          for each user $u_m$ who has rated $r_i$
7.              for each user $u_n$ who has rated $r_j$
8.                  if $s_{j,n} \succ s_{i,m}$
9.                      SumN=SumN+ $lbw_{m,n}$
10.                     SumC=SumC + $ubw_{m,n}$
11.                     if SumN ≥ θ
12.                         insert $r_i$ into N; goto 3
13.     if SumC < θ
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_l if we can find some record r_j such that $r_j \succ_l r_i$. 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_j has better overall ratings than r'_j, r_j is more likely to dominate r_i than r'_j. Based on this observation, r_j is ordered before r'_j if the sum of the average ratings on all dimensions of r_j is larger than that of r'_j. The cost of *record pre-ordering* is $O(|R|\cdot|S_{AVG}| + |R|\cdot log(|R|))$ because it involves computing the average rating for each record ($|R|\cdot|S_{AVG}|$), 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_l each record r_i that is not dominated by another record already in Sky_l. Compared to the basic ESC algorithm, the number of records in line 3 is small. However, after this pass Sky_l 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_l).

The efficiency of 2S-ESC depends on the number of false positives produced by the first scan. Assuming that the cardinality of Sky_l 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_l. The second scan verifies the candidates in Sky_l 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 $u_l$, threshold $\theta_l$)
1.  $Sky_l = \varnothing$
2.  for each record $r_i$ in $R$
3.      for each record $r_j$ in $Sky_l$
...
lines 4-11 identical to corresponding lines in Figure 3
...
12. for each record $r_i$ in $Sky_l$
13.     for each record $r_j$ in $R$
14.         $Sum=0$
15.         for each user $u_m$ who has rated $r_i$
16.             for each user $u_n$ who has rated $r_j$
17.                 if $s_{j,n} \succ s_{i,m}$
18.                     $Sum=Sum + w^l_{m,n} / |S_i|\cdot|S_j|$
19.                     if $Sum \geq \theta_l$
20.                         Remove $r_i$ from $Sky_l$; goto 12
21. report $Sky_l$
```
Figure 5 Two-scans paradigm for exact Sky_l 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_l:

$$\frac{\sum_{m,n} w^l_{m,n}[s_{i,m} \succ s_{j,n}]}{|S_i|\cdot|S_j|} = \frac{\sum_{m,n}\sigma(u_m,u_n)[s_{i,m} \succ s_{j,n}]}{3\cdot|S_i|\cdot|S_j|} + \frac{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))[s_{i,m} \succ s_{j,n}]}{3\cdot|S_i|\cdot|S_j|} \tag{5.1}$$

where $w^l_{m,n}$ 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_i dominates r_j with respect to u_l ($r_i \succ^l r_j$), if the following condition is satisfied:

$$\frac{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))[s_{i,m} \succ s_{j,n}]}{3\cdot|S_i|\cdot|S_j|} \geq \theta_l - \frac{\sum_{m,n}\sigma(u_m,u_n)[s_{i,m} \succ s_{j,n}]}{3\cdot|S_i|\cdot|S_j|} \tag{5.2}$$

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

$$\frac{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))[s_{i,m} \succ s_{j,n}]}{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))} \geq \theta'_l, \quad where$$
$$\theta'_l = \left(\theta_l - \frac{\sum_{m,n}\sigma(u_m,u_n)[s_{i,m} \succ s_{j,n}]}{3\cdot|S_i|\cdot|S_j|}\right)\cdot\left(\frac{3\cdot|S_i|\cdot|S_j|}{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))}\right) \tag{5.3}$$

The new threshold θ'_l 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 $[s_{i,m} \succ s_{j,n}]$ is sampled with probability:

$$\frac{\sigma(u_m,u_l)+\sigma(u_n,u_l)}{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))} \tag{5.4}$$

According to the *Chernoff* bound [26], the average over the samples is an ε-approximation with confidence δ, if the number of samples on the pairs $|S_i|\times|S_j|$ is at least:

$N = 2\ln(1/\delta)/\varepsilon^2\theta'_l$

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

$$\sum_n(\sigma(u_m,u_l)+\sigma(u_n,u_l)) = |S_j|\cdot\sigma(u_m,u_l)+\sum_n\sigma(u_n,u_l) \tag{5.5}$$

Therefore, the probability of choosing any score pair involving u_m can be calculated by the following equation:

$$\frac{|S_j|\cdot\sigma(u_m,u_l)+\sum_n\sigma(u_n,u_l)}{\sum_{m,n}(\sigma(u_m,u_l)+\sigma(u_n,u_l))} \tag{5.6}$$

Second, given u_m, the probability of selecting u_n is:

$$\frac{\sigma(u_m,u_l)+\sigma(u_n,u_l)}{|S_j|\cdot\sigma(u_m,u_l)+\sum_n\sigma(u_n,u_l)} = \frac{\sigma(u_m,u_l)}{|S_j|\cdot\sigma(u_m,u_l)}\cdot\frac{|S_j|\cdot\sigma(u_m,u_l)}{|S_j|\cdot\sigma(u_m,u_l)+\sum_n\sigma(u_n,u_l)} + \frac{\sigma(u_n,u_l)}{\sum_n\sigma(u_n,u_l)}\cdot\frac{\sum_n\sigma(u_n,u_l)}{|S_j|\cdot\sigma(u_m,u_l)+\sum_n\sigma(u_n,u_l)} \tag{5.7}$$

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

$rnd < |S_j|\cdot\sigma(u_m,u_l) \,/\, (|S_j|\cdot\sigma(u_m,u_l)+\sum_{m,n}\sigma(u_n,u_l))$, u_n is chosen uniformly with probability $1/|S_j|$. Otherwise, u_n is chosen with probability $\sigma(u_n,u_l) \,/\, \sum_n\sigma(u_n,u_l)$. 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 $\sum_n\sigma(u_n,u_l)$ for every u_n.

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 ($s_{i,m}$, $s_{j,n}$), the variable *Sum* is incremented by 1, when $s_{j,n} \succ s_{i,m}$. After the sampling process terminates, if the average *Sum/N* is larger than the new threshold θ'_l, r_j is expected to dominate r_i with high confidence. If no record can approximately dominate r_i, r_i is inserted into the personal skyline set Sky_l. The complexity of ASC depends on the number of samples created for every record pair (r_i, r_j). If θ'_l is high, the necessary number of samples is small, and vice versa. Therefore, 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_l, threshold θ_l) // computation of approximate Sky_l
1.  $Sky_l=\varnothing$
2.  for each $r_i$ in $R$
3.    for each record $r_j \neq r_i$
4.      $Sum = 0$
5.      let $\theta'_l$ be the new threshold computed by Equation 5.3 and
        $N$ be the number of samples: $N = 2\sqrt{1/\delta}\,/\,\varepsilon^2\theta'_l$
6.      for each sample
7.        select user $u_m$ with probability
              $|S_j|\cdot\sigma(u_m,u_l)/(|S_j|\cdot\sigma(u_m,u_l)+\sum_{m,n}\sigma(u_n,u_l))$
8.        generate random number $rnd \in [0,1]$
9.        if $rnd < |S_j|\cdot\sigma(u_m,u_l)\,/\,(|S_j|\cdot\sigma(u_m,u_l)+\sum_{m,n}\sigma(u_n,u_l))$
10.           select user $u_n$ with probability $1/|S_j|$
11.       else
12.           select user $u_n$ with probability $\sigma(u_n,u_l)/\sum_n\sigma(u_n,u_l)$
13.       if $s_{j,n} \succ s_{i,m}$
14.           $Sum = Sum+1$
15.       if $Sum/N \geq \theta'_l$
16.           discard $r_i$
17.   insert $r_i$ into $Sky_l$ // $r_i$ is not dominated
18.  report $Sky_l$
```
Figure 6 Approximate skyline computation

6 TOP-K PERSONALIZED SKYLINE

According to Definition 3.1, each active user u_l has to provide the dominance threshold θ_l that determines the cardinality of her/his personalized skyline Sky_l. The proper setting of θ_l 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 θ_l; in the extreme case that θ_l=0, the skyline is empty because each record is dominated. The top-*k* personalized skyline corresponds to the minimum value of θ_l 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 θ_U. *Pre-processing* uses the upper bounds (i.e., $ubw_{m,n}$) 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 (*SumU* in the pseudo-code) in *R*. For each record in *USkyP*, the algorithm stores the pair (r_i,*SumU*). Initially, θ_U is set to 1 and then gradually decreases (Lines 12-16). The final θ_U is returned as the upper bound on the threshold θ_l for ETKS. ETKS is similar to ESC, except that (i) it only keeps in Sky_l the *k*-least dominated records, and (ii) it continuously updates the value of θ_l 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.

```
Pre-processing (k)
1.  $USkyP = \varnothing$
2.  $\theta_U = 1$ // upper bound threshold
3.  for each record $r_i$ in $R$
4.    for each record $r_j \neq r_i$
5.      $SumU=0$
6.      for each user $u_m$ who has rated $r_i$
7.        for each user $u_n$ who has rated $r_j$
8.          if $s_{j,n} \succ s_{i,m}$
9.            $SumU=SumU + ubw_{m,n}$
10.           if $SumU > \theta_U$
11.             skip $r_i$; goto 3
12.     if $SumU \leq \theta_U$
13.       insert ($r_i$,$SumU$) into $USkyP$
14.     if $USkyP$ has more than $k$ records
15.       remove ($r_i$,$SumU$) with the maximal $SumU$ from $USkyP$
16.       set $\theta_U$ to the max $SumU$ in the remaining records
17. return $\theta_U$
```
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.

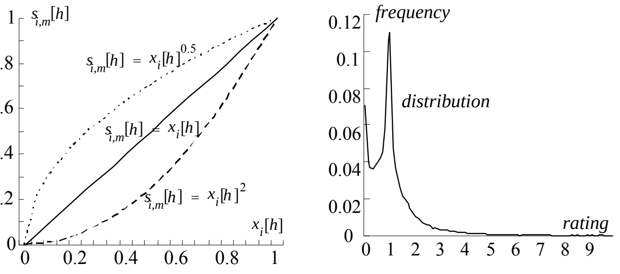
7.1 Datasets

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 r_i 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 $s_{i,m}[h]$ of user u_m for record r_i on dimension h as:

$$s_{i,m}[h] = (x_i[h])^{a_{m,h}} \tag{7.1}$$

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 rn is drawn from a standard *Gaussian* distribution with mean m=0 and variance var=1. Given the random number rn, 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.



(a) Rating generation functions (b) Distribution on the ratings
Figure 8 Rating generation functions and rating distribution

A user rates a record r_i with probability p, i.e., the expected number of scores on r_i is $|U|\cdot p$. We also generate the overall rating of each virtual user on the reviewed houses as the median rating on all dimensions. After the above transformations, we obtain the semi-real dataset, named *House*, with 1,000 virtual users, 5,274 records (houses), and 64,206 reviews, i.e., on the average, each house (resp. user) is associated with roughly 12 (resp. 64) reviews.

For the efficiency experiments, in order to be able to adjust the number of users $|U|$, records $|R|$, scores $|S|$, and data dimensionality d, we use totally synthetic data sets. Table II summarizes the parameters involved in synthetic data generation, as well as their default values and ranges. In each experiment, we vary one parameter, while setting the remaining ones to their default values. The independent, correlated and anti-correlated distributions are common in the skyline literature (e.g., [29, 31]).

TABLE II PARAMETERS FOR SYNTHETIC DATASETS

Parameter	Default	Range
User cardinality $\|U\|$	5,000	3,4,5,6,7 (×10³)
Record cardinality $\|R\|$	50,000	30,40,50,60,70 (×10³)
Score cardinality $\|S\|$	100,000	50, 75, 100, 125, 150 (×10³)
Dimensionality d	4	2, 3, 4, 5, 6
Distribution	Independent	Correlated, Independent, Anticorrelated

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 $s^*_{i,l}$ for active user u_l on r_i on each dimension is computed according to the *Pearson Correlation* coefficient [1]:

$$s^*_{i,l} = \overline{s_l} + \frac{\sum_{m \in U_l} corr(u_l, u_m)(s_{i,m} - \overline{s_m})}{\sum_{m \in U_l} |corr(u_l, u_m)|} \tag{7.1}$$

where $\overline{s_l}$ is the average rating of user u_l and U_l denotes the set of users that are similar to u_l (we set the cardinality of U_l to 10 users for all experiments.). The intuition behind Equation 7.1 is that $s^*_{i,l} > \overline{s_l}$, if several users have rated r_i

above their averages (i.e., $s_{i,m} > \overline{s_m}$), especially if those users are similar[7] to u_l. Ideally, $s^*_{i,l}$ should be equal to the actual overall score $s_{i,l}$. CF recommends to u_l the set Top_l of records with the highest predicted scores. Let $NTop_l = R-Top_l$ be the set of non-recommended objects. $\overline{Top_l}$ ($\overline{NTop_l}$) signifies the average *actual* score of u_l to (non) recommended records. The *gain G* is the score difference between recommended and non-recommended objects:

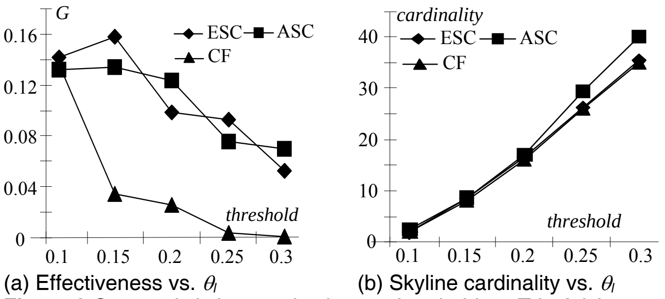
$$G = \overline{Top_l} - \overline{NTop_l} \tag{7.2}$$

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, $\overline{Sky_l}$ ($\overline{NSky_l}$) denotes the average overall score of u_l for (non) skyline records. In this case, the gain is defined according to Equation 7.3:

$$G = \overline{Sky_l} - \overline{NSky_l} \tag{7.3}$$

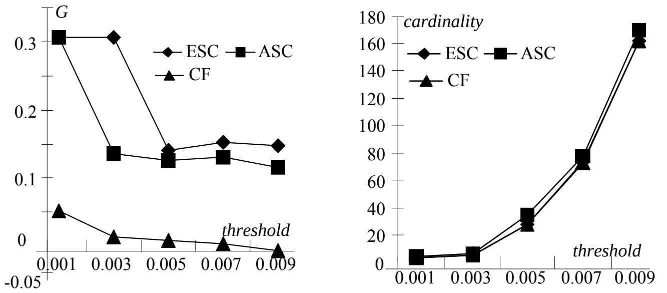
Intuitively, a high gain implies that records in the personalized skyline of u_l are indeed preferable for the user as demonstrated by her/his actual overall rating. We compare exact (ESC) and approximate (ASC) CFS against CF using the *Trip Advisor* and *House* datasets. For fairness, given the threshold θ_l, we set the CF parameter k so that k is the closest integer to the average skyline cardinality $|Sky_l|$, 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 θ_l = 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 θ_l = 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).



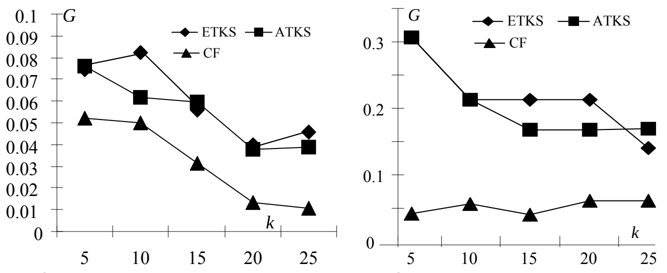
(a) Effectiveness vs. θ_l (b) Skyline cardinality vs. θ_l
Figure 9 Gain and skyline cardinality vs. threshold on *Trip Advisor*

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 θ_l = 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 θ_l 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*.



(a) Effectiveness vs. θ_l (b) cardinality vs. θ_l
Figure 10 Gain and skyline cardinality vs. threshold on *House*

To evaluate the effectiveness 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.



(a) Gain vs. k on *Trip Advisor* (b) Gain vs. k on *House*
Figure 11 Effectiveness vs. k on (a) *Trip Advisor* and (b) *House*

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 $s_{i,l}$ = (2,2,5,2) and $s_{j,l}$ = (3,3,3,3) from u_l on the four attributes of records r_i and r_j. Although r_j has a higher average, u_l may give a better overall score to r_i because s/he may value more 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 the number of users $|U|$, records $|R|$, scores $|S|$, and data dimensionality d. We compare exact and approximate skyline computation using the synthetic data. In each experiment we report the average CPU time for a personalized skyline computation. Section 7.3.1 focuses on CFS using the threshold value, whereas Section 7.3.2 deals

[7] We use the similarity measure of Equation 3.4.

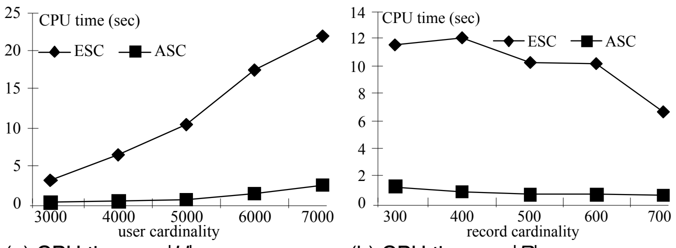
with top-*k* personalized skylines. Table III illustrates the default values and ranges for the experimental parameters.

TABLE III PARAMETERS FOR EFFICIENCY EXPERIMENTS

Parameter	Default	Range
Threshold θ_l	0.1	0.05, 0.075, 0.1, 0.125, 0.15
Error ε	0.3	0.1, 0.2, 0.3, 0.4, 0.5
Confidence δ	0.3	0.1, 0.2, 0.3, 0.4, 0.5

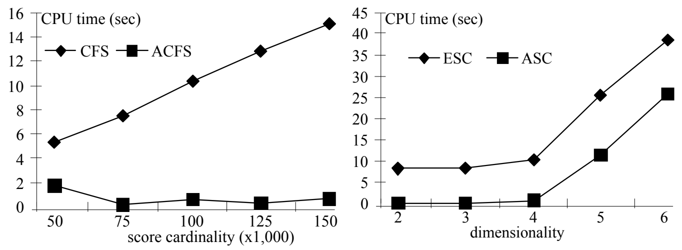
7.3.1 ESC vs. ASC

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 12a 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 ($|S| = 100$K), 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 $|U|$, 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 $|U|$ 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.



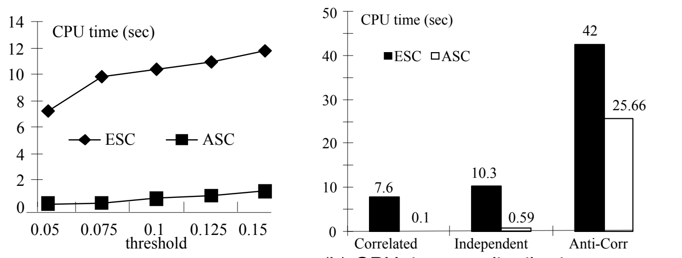
(a) CPU time vs. |*U*| (b) CPU time vs. |*R*|
Figure 12 ESC, ASC efficiency vs. user and record cardinality

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.



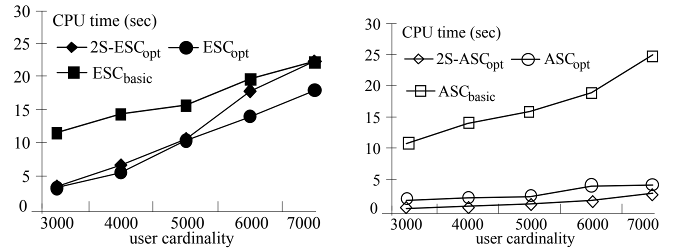
(a) CPU time vs. |*S*| (b) CPU time vs. *d*
Figure 13 ES, ASC efficiency vs. score and dimension cardinality

Figure 14a studies the impact of the threshold parameter on the efficiency of skyline computation. Recall that a high value of θ_l leads to large skylines, and the cost of ESC increases because fewer records are eliminated. On the other hand, the effect on ASC is not as serious because a higher threshold reduces the sampling rate (recall from Section 5 that the sampling rate is inversely proportional to the threshold). Figure 14b investigates correlated, independent, and anti-correlated distributions. Anti-correlated datasets are the most expensive to process because they entail the largest skylines. On the other hand, correlated datasets have small skylines, and incur the lowest cost.



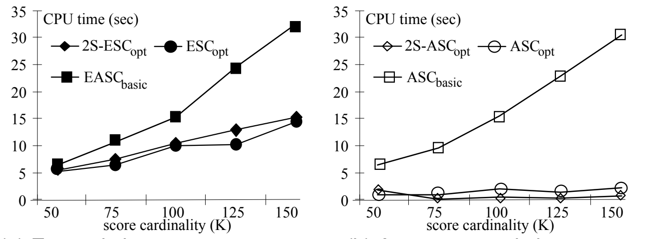
(a) CPU time vs. θ_l (b) CPU time vs. distribution
Figure 14 ESC, ASC efficiency vs. threshold and distribution

Next we evaluate the efficiency of alternative implementations of ESC and ASC. Specifically, 2S-ESC$_{opt}$ denotes two-scans ESC with all optimizations of Section 4.2, and ESC$_{opt}$ (resp. ESC$_{basic}$) denotes the basic algorithm of Figure 3 with (resp. without) these optimizations. Figure 15a illustrates the CPU time for exact skyline computation as a function of the user cardinality. The best method is ESC$_{opt}$, 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 15b repeats the experiment for approximate skyline computation, where we use the same notation for the different versions as their exact counterparts. 2S-ASC$_{opt}$ achieves noticeable improvement over ASC$_{opt}$ because after the first scan there are relatively fewer records in the buffer *Sky$_l$* compared to the exact solution.



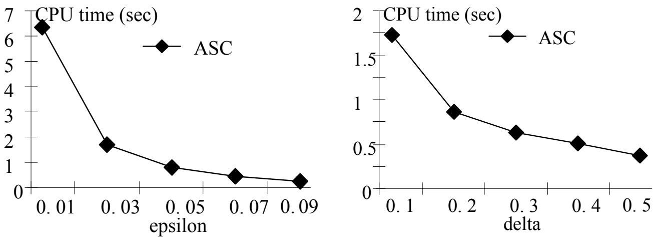
(a) Exact skylines (b) Approximate skylines
Figure 15 Alternative implementations: CPU time vs. user cardinality

Figure 16 measures the effect of optimizations as a function of the score cardinality. The results are consistent with those of Figure 15: ESC_{opt} is the best choice for exact computation, whereas $2S\text{-}ASC_{opt}$ is the winner for approximation computation. Both algorithms scale well as the number of scores increases.



(a) Exact skylines (b) Approximate skylines
Figure 16 Alternative implementations: CPU time versus score cardinality

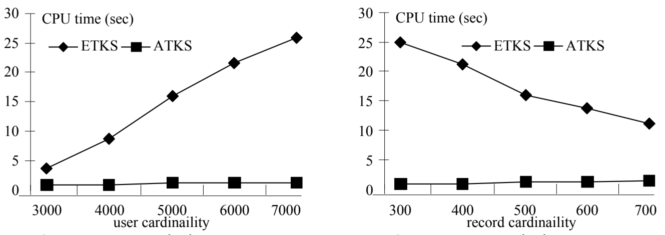
Finally, we analyze the impact of the sampling parameters of ASC. Figure 17a (resp. 17b) illustrates the overhead as a function of the error parameter ε (resp. confidence parameter δ). As expected, the CPU cost of ASC is proportional to both $1/\varepsilon^2$ and $\ln(1/\delta)$.



(a) CPU time vs. ε (b) CPU time vs. δ
Figure 17 ASC efficiency vs. error and confidence

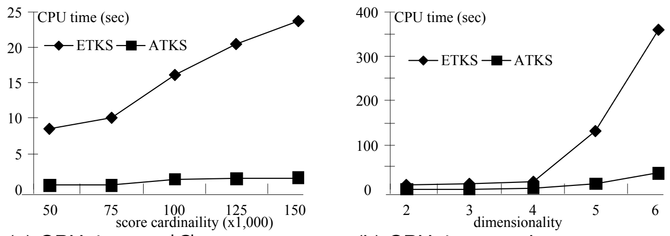
7.3.2 ETSK vs. ATSK

In this section, we evaluate the efficiency of top-k algorithms for CFS using again the parameters of Tables II and III. Both ETKS 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 ETKS and ATKS. Similar to Figure 12a, the CPU time of ETKS grows with $|U|$, while ATKS is insensitive to $|U|$. According to Figure 18b, the CPU time of ETKS decreases as the record cardinality grows because there are fewer scores per record.



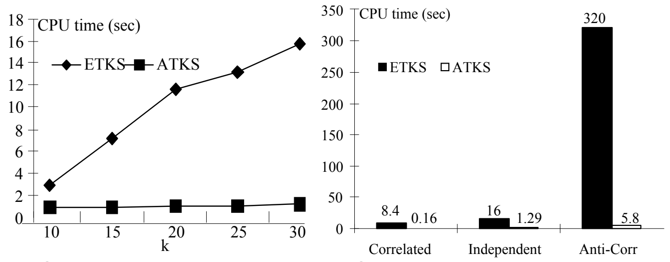
(a) CPU time vs. $|U|$ (b) CPU time vs. $|R|$
Figure 18 ETKS, ATKS efficiency vs. user and record cardinality

Figures 19a and 19b present the CPU time with respect to the score cardinality and the dimensionality, respectively. The results are consistent with those in Figure 13, and ATKS has a great advantage over ETKS, especially when more scores (resp. higher values of d) are considered.



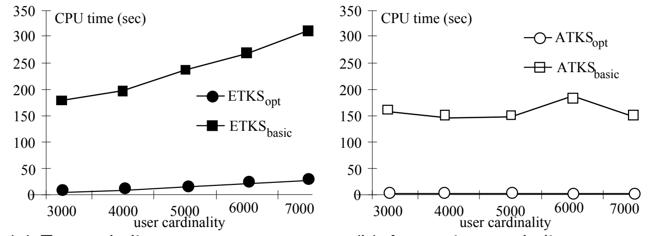
(a) CPU time vs. $|S|$ (b) CPU time vs. d
Figure 19 ETKS, ATKS efficiency vs. score cardinality and dimensionality

Figure 20a analyzes the effect of k, which controls the number of records returned to the users. As expected, the performance of ETKS degrades with k. On the other hand, the impact of k on ATKS is negligible because its cost is dominated by the sampling probability computation, which is independent of k. Figure 20b summarizes the experimental results with three different distributions. Note that for anti-correlated data ATKS reduces the computation cost by almost two orders of magnitude.



(a) CPU time vs. k (b) CPU time vs. distribution
Figure 20 ETKS, 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 $ETKS_{opt}$, $ETKS_{basic}$, $ATKS_{opt}$ and $ATKS_{basic}$, 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. $ETKS_{opt}$ and $ATKS_{opt}$ outperform $ETKS_{basic}$ and $ATKS_{basic}$, respectively, by at least one order of magnitude. Similar performance gains are observed when varying the number of records, scores and dimensions.



(a) Exact skylines (b) Approximate skylines
Figure 21 Alternative implementations: CPU time vs. user cardinality

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

1. Adomavicius, G., Tuzhilin, A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *TKDE*, 17(6): 734-749, 2005.
2. Balabanovic, M., Shoham, Y. Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM*, 40(3):66-72, 1997.
3. Bartolini, I., Ciaccia, P., Patella, M. Efficient Sort-based Skyline Evaluation. *TODS*, 33(4):31.1-31.49, 2008.
4. Basu, C., Hirsh, H., Cohen, W. W. Recommendation as Classification: Using Social and Content-based Information in Recommendation. *AAAI*, 1998.
5. Börzsönyi, S., Kossmann, D., Stocker, K. The Skyline Operator. *ICDE*, 2001.
6. Breese, J. S., Heckerman, D., Kadie, C. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *UAI*, 1998.
7. Burke, R. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction Archive*, 12(4): 331-370, 2002.
8. Chan, C. Y., Eng, P.-K., Tan, K.-L. Stratified Computation of Skylines with Partially-Ordered Domains. *SIGMOD*, 2005.
9. Chan, C.Y., Jagadish, H., Tan, K.-L., Tung, A., Zhang, Z. Finding *k*-dominant Skylines in High Dimensional Space. *SIGMOD*, 2006.
10. Chomicki, J., Godfrey, P., Gryz, J., Liang, D. Skyline with Presorting. *ICDE*, 2003.
11. Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin, M. Combining Content-based and Collaborative Filters in an Online Newspaper. *ACM SIGIR Workshop on Recommender Systems*, 1999.
12. Cosley, D., Lawrence, S., Pennock, D. M. REFEREE: An Open Framework for Practical Testing of Recommender Systems using ResearchIndex. *VLDB*, 2002.
13. Dellis, E., Seeger, B. Efficient Computation of Reverse Skyline Queries. *VLDB*, 2007.
14. Godfrey, P., Shipley, R., Gryz, J. Maximal Vector Computation in Large Data Sets. *VLDB*, 2005.
15. Goldberg, D., Nichols, D., Oki, B. M., Terry, D. Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, 35(12):61-70, 1992.
16. Good, N., Schafer, J. B., Konstant, J. A., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J. Combining Collaborative Filtering with Personal Agents for Better Recommendations. *AAAI*, 1999.
17. Herlocker, J. L., Konstan, J. A., Terveen, L. G., Riedl, J. T. Evaluating Collaborative Filtering Recommender Systems. *TOIS*, 22(1):5-53, 2004.
18. Huang, Z., Jensen, C. S., Lu, H., Ooi, B. C. Skyline Queries Against Mobile Lightweight Devices in MANETs. *ICDE*, 2006.
19. Khalefa, M., Mokbel, M., Levandoski, J. Skyline Query Processing for Incomplete Data. *ICDE*, 2008.
20. Kossmann, D., Ramsak, F., Rost, S. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. *VLDB*, 2002.
21. Kung, H., Luccio, F., Preparata, F. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4): 469-476, 1975.
22. Lee, K., Zheng, B., Li, H., Lee, W. Approaching the Skyline in Z Order. *VLDB*, 2007.
23. Lee, W. S. Collaborative Learning for Recommender Systems. *ICML*, 2001.
24. Lian X., Chen, L. Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases. *SIGMOD*, 2008.
25. McLain D. Drawing Contours from Arbitrary Data Points. *Computer Journal*, 17(4), 1974.
26. Mitzenmacher, M., Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge Press, 2005.
27. Morse, M., Patel, J., Grosky, W. Efficient Continuous Skyline Computation. *ICDE*, 2006.
28. Morse, M., Patel, J., Jagadish, H. Efficient Skyline Computation over Low-Cardinality Domains. *VLDB*, 2007.
29. Papadias, D., Tao, Y., Fu, G., Seeger, B. Progressive Skyline Computation in Database Systems. *TODS*, 30(1): 41-82, 2005.
30. Pazzani, M., Billsus, D. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27:313-331, 1997.
31. Pei, J., Jiang, B., Lin, X., Yuan, Y. Probabilistic Skyline on Uncertain Data. *VLDB*, 2007.
32. Pennock, D. M., Horvitz, E., Giles, C. L. Social Choice Theory and Recommender systems: Analysis of the Axiomatic Foundations of Collaborative filtering. *AAAI*, 2000.
33. Pennock, D. M., Horvitz, E., Lawrence, S., Giles, C. L. Collaborative Filtering by Personality Diagnosis: A Hybrid Memory and Model-based Approach. *AAAI*, 2000.
34. Resnick, P., Iakovou, N., Sushak, M., Bergstrom, P., Riedl, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *CSCW*, 1994.
35. Resnick, P., Varian, H. R. Recommender Systems. *Communications of the ACM*, 40(3): 56-58, 1997.
36. Rich, E. User Modelling via Stereotypes. *Cognitive Science*, 3(4):329-354, 1979.
37. Shahabi, C., Banaei-Kashani, F., Chen, Y., McLeod, D. Yoda: An Accurate and Scalable Web-based Recommendation System. *COOPIS*, 2001.

38. Shardanand, U., Maes, P. Social Information Filtering: Algorithms for Automating 'Word of Mouth'. *CHI*, 1995.
39. Sharifzadeh, M., Shahabi, C. The Spatial Skyline Queries. *VLDB*, 2006.
40. Steuer, R. *Multiple Criteria Optimization*. Wiley, New York, 1986.
41. Talwar, A., Jurca, R., Faltings, B. Understanding User Behavior in Online Feedback Reporting. *ACM Conference on Electronic Commerce*, 2007.
42. Tan, K.-L., Eng, P.-K., Ooi, B. C. Efficient Progressive Skyline Computation. *VLDB*, 2001.
43. Tao, Y., Xiao, X., Pei, J. SUBSKY: Efficient Computation of Skylines in Subspaces. *ICDE*, 2006.
44. Wong, R. C.-W., Pei, J., Fu, A. W.-C., Wang, K. Mining Favorable Facets. *KDD*, 2007.
45. Xia, T., Zhang, D. Refreshing the Sky: The Compressed Skycube with Efficient Support for Frequent Updates. *SIGMOD*, 2006.
46. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J. X., Zhang, Q. Efficient Computation of the Skyline Cube. *VLDB*, 2005.

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 Shanon 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.