Knowl Inf Syst
DOI 10.1007/s10115-012-0489-6

REGULAR PAPER

Recommendations for two-way selections using skyline view queries

Jian Chen · Jin Huang · Bin Jiang · Jian Pei · Jian Yin

Received: 28 November 2010 / Revised: 3 December 2011 / Accepted: 6 March 2012

Abstract We study a practical and novel problem of making recommendations between two parties such as applicants and job positions. We model the competent choices of each party using skylines. In order to make recommendations in various scenarios, we propose a series of skyline view queries. To make recommendations, we often need to answer skyline view queries for many entries in one or two parties in batch, such as for many applicants versus many jobs. However, the existing skyline computation algorithms focus on answering a single skyline query at a time and do not consider sharing computation when answering skyline view queries for many members in one party or both parties. To tackle the batch recommendation problem, we develop several efficient algorithms to process skyline view queries in batch. The experiment results demonstrate that our algorithms significantly outperform the state-of-the-art methods.

Keywords Mutual recommendation · Skyline query · Multi-objective optimization · Stable matching

J. Chen
South China University of Technology, Guangzhou, China

J. Huang (✉)
South China Normal University, Guangzhou 510631, China
e-mail: jinhuang@scnu.edu.cn

B. Jiang
Facebook Inc., Menlo Park, CA, USA

J. Pei
Simon Fraser University, Burnaby, Canada

J. Yin
Sun Yat-sen University, Guangzhou, China

Published online: 25 March 2012

1 Introduction

It has been well recognized that skyline queries are important in many multi-criteria decision-making applications. For example, consider a scenario where an employer screens applicants for a job position. Suppose each applicant has two attributes: the experience level and the qualification level, the higher the better. An applicant a dominates another applicant b, that is, a is a better applicant than b in the context of this example, if a is better than b in at least one aspect and is not worse than b in the other aspect. Among a set of given applicants, the ones not dominated by any others in the set form the skyline. In other words, the skyline applicants are all possible trade-offs between experience and qualification that are superior to other applicants.

There are many studies on efficient skyline computation, as will be briefly reviewed in Sect. 3. Existing studies focus on computing the skyline(s) for a given data set in either the full space or subspaces. In real applications when selections involve more than one party, skyline analysis can be much more complicated, which motivates our study.

Consider a headhunter making recommendations between job applicants and job positions. Given a set of applicants in Table 1 and a set of positions in Table 2, suppose the higher the experience level and qualification level, the better an applicant, and the higher the salary level and the benefit level, the more attractive a job. In Table 1, each applicant indicates the jobs interesting to her/him. Symmetrically, each job has some requirement on applicants. In Table 2, each job has a list of applicants who satisfy the requirement of the job. Those applicants are qualified for the job.

A natural question from an applicant is to find the skyline jobs that the applicant is qualified. We call this the *reciprocal skyline* of the applicant, which represents the best choices for the applicant. For example, Ada in Table 1 is qualified for jobs J1, J2, and J3. J2 dominates J3, since J2 is better than J3 in both salary and benefit. Thus, Ada's reciprocal skyline consists of J1 and J2, though Ada is not currently interested in J1. This information may help Ada to adjust her interest. Symmetrically, we can also provide a job the reciprocal skyline of applicants that contains the best applicants who are interested in the job.

An applicant may also want to know which jobs regard the applicant a skyline applicant. We can model such an information need by an *inverse skyline query*. For example, Cathy in Table 1 is interested in and also qualified for jobs J1, J2, and J3. Unfortunately, Cathy's inverse skyline is empty since Cathy is not in the skyline of any jobs. Knowing her inverse skyline, Cathy may adjust her expectation and job hunting strategy properly, since she may

Table 1 A set of applicants

Applicant	Experience	Qualification	Interest
Ada	4	3	J2, J3
Bob	2	4	J1, J3
Cathy	3	1	J1, J2, J3
Dan	1	2	J2, J3

Table 2 A set of job positions

Job	Salary	Benefit	Interest
J1	80K	1	Ada, Bob, Cathy
J2	50K	3	Ada, Cathy, Dan
J3	40K	2	Ada, Bob, Cathy, Dan

not be the first choice of any jobs. Symmetrically, we can also provide a job the inverse skyline of applicants that contains the applicants who view the job a skyline one.

Both reciprocal skyline queries and inverse skyline queries are the examples of a series of skyline view queries to be discussed in this paper. As illustrated, various skyline view queries may be asked in the course of recommendations for two-way selections. Although those queries can be answered for one applicant or one job individually by adapting some existing skyline computation algorithms, a headhunter often has to answer such queries in batch for all or a large number of applicants and/or jobs. For example, online job search engines, such as mon\discretionary-ster.com, have millions of job posts and registered applicants. Even more, thousands of new job posts and applicants are added everyday. The search engines have to make recommendations for new jobs and applicants as well as update the recommendations for existing jobs and applicants. Such systems need to process millions of skyline view queries periodically. The existing skyline computation algorithms only consider computing a single skyline at a time and thus are not optimized for answering multiple queries in batch. Can we answer a batch of skyline view queries in a more efficient and more scalable way than processing them one by one? It is far from trivial to develop efficient computation sharing strategies when computing many skylines at the same time.

The scenario of matching applicants and jobs is just one example of recommendations between two parties towards two-way selections. Recommendations between two parties for two-way selections are needed in many applications. Online dating services recommend appropriate dates for men and women, who may specify requirements on various factors such as age, location, and income. Online advertisement service applications, such as Google AdSense, make mutual recommendations between web pages and advertisers. Skyline view queries are highly useful in recommendations between two parties for two-way selections. However, to the best of our knowledge, skyline queries for recommendations in two-way selection have not been studied systematically.

In this paper, we identify a series of skyline view queries useful for recommendations between two parties and develop several novel algorithms to answer the skyline view queries in batch efficiently. An experimental study indicates that our batch algorithms significantly outperform answering queries individually using the existing methods.

The rest of the paper is organized as follows. In Sect. 2, we formulate 8 types of skyline view queries. We review the related work in Sect. 3. Section 4 develops efficient algorithms for answering the 8 types of skyline view queries. We report an extensive empirical study in Sect. 5. Section 6 concludes the paper.

2 Skyline view queries

In this section, we first recall the preliminaries of skylines. Then, we formally define 8 skyline view queries.

2.1 Preliminaries

Consider a set of points S in a multidimensional space $\mathcal{D} = (D_1, \ldots, D_d)$. To keep our discussion simple, we assume that the domain of each attribute $D_i (1 \leq i \leq d)$ is numeric. Our discussion can be generalized to attributes of other types where partial orders are defined.

Consider two points $x, y \in S$. x is said to *dominate* y, denoted by $x \succ y$, if for every attribute D_i, $x.D_i \geq y.D_i$, and there exists one attribute $D_j (1 \leq j \leq d)$ such that $x.D_j > y.D_j$.

Table 3 A summary of frequently used notions

Notion	Meaning
$V(A)$	The view of A, $\{J \in \mathcal{J} \mid A.P(J) = true\}$
$IV(A)$	The inverse view of A, $\{J \in \mathcal{J} \mid J.P(A) = true\}$
$SV(A)$	The skyline view of A, $Sky(V(A))$
$iSky(A)$	The inverse skyline of A, $\{J \mid A \in SV(J)\}$
$rSky(A)$	The reciprocal skyline of A, $Sky(IV(A))$
$MV(A)$	The mutual view of A, $\{J \mid J \in V(A) \wedge A \in V(J)\}$
$SMV(A)$	The skyline mutual view of A, $Sky(MV(A))$
$SIS(A)$	The skyline of inverse skyline of A, $Sky(iSky(A))$

A point $x \in S$ is a *skyline point* in S if there does not exist another point $x' \in S$ such that $x' \succ x$. The *skyline* of S, denoted by $Sky(S)$, is the set of skyline points in S.

A *predicate* P defined in space $\mathcal{D}$ specifies a region in either $\mathcal{D}$ or a subspace of $\mathcal{D}$. A point x is said to *satisfy* P, denoted by $P(x) = true$, if x falls in the region specified by P. We also overload symbol P to denote the region specified by the predicate of the same name. In this paper, we focus on simple predicates whose regions are rectangles. To be concrete, a predicate specifies the maximum and/or minimum values on some attributes. For example, an applicant may specify the predicate of his job interests as Salary $>$50K, and a job position may specify the predicate of the applicant requirements as Experience $>$ 3 and Qualification $\in [2, 3]$.

2.2 Skyline view queries

In this paper, we study skyline view queries involving two parties. To make our discussion easy to follow, we use "applicants" and "jobs" as the aliases of the two parties in the question. If one prefers being formal, we can always replace "applicants" and "jobs" by "Party 1" and "Party 2," respectively, in the mathematical statements.

Given an applicant data set $\mathcal{A}$ and a job data set $\mathcal{J}$, we study the problem of recommendations between the two data sets.

In practice, it may not be easy to obtain the explicit and complete interest of an applicant or a job. Instead, an applicant may specify the minimum and/or maximum requirements on attributes such as salary and benefit. Hence, we model the interest of an applicant $A \in \mathcal{A}$ as a predicate $A.P$ in the space of the job data set $\mathcal{J}$. Symmetrically, the qualification of a job $J \in \mathcal{J}$ is also modeled as a predicate $J.P$ in the space of the applicants $\mathcal{A}$. Applicant A is interested in job J if J satisfies the predicate of A, that is, $A.P(J) = true$. Job J is interested in applicant A, that is, the applicant qualifies job J, if A satisfies the predicate of J, that is, $J.P(A) = true$.

In the rest of this section, we only define the skyline view queries for applicants. Their counterparts for jobs can be obtained symmetrically. Table 3 summaries the 8 types of skyline view queries. To help understand the queries, Table 4 shows the results of these queries on the data sets in Tables 1 and 2.

Definition 1 (*View*) The **view** of an applicant $A \in \mathcal{A}$ consists of all jobs in $\mathcal{J}$ satisfying the predicate of A, that is, $V(A) = \{J \in \mathcal{J} \mid A.P(J) = true\}$.

Symmetrically, we define the view of a job J with predicate $J.P$ as $V(J) = \{A \in \mathcal{A} \mid J.P(A) = true\}$.

For example, in Table 1, Ada's view includes J2 and J3.

It is also important for an applicant to know the jobs that she/he is qualified. This can be modeled by the *inverse view* of the applicant.

Definition 2 (*Inverse view*) The **inverse view** of an applicant A is the set of jobs in $\mathcal{J}$ that are interested in A. That is, $IV(A) = \{J \in \mathcal{J} | J.P(A) = true\}$.

If a job J is in the inverse view of applicant A then A is in the view of J, and vice versa. Therefore, $IV(A) = \{J \in \mathcal{J} | A \in V(J)\}$. Similarly, the inverse view of a job $J \in \mathcal{J}$ is $IV(J) = \{A \in \mathcal{A} | A.P(J) = true\} = \{A \in \mathcal{A} | J \in V(A)\}$.

For an applicant A, $V(A)$ includes all jobs that satisfy A's predicate. Among these jobs, the ones in the skyline of $V(A)$ are more attractive to A since they are not dominated by any other jobs in $V(A)$.

Definition 3 (*Skyline view*) For an applicant $A \in \mathcal{A}$, the **skyline view** of A is the skyline of the jobs that A is interested in, that is, the skyline of A's view denoted by $SV(A) = Sky(V(A))$.

Symmetrically, the skyline view of a job $J \in \mathcal{A}$ is $SV(J) = Sky(V(J))$.

An applicant A may like to know the jobs J whose skyline view $SV(J)$ contains A. Those jobs are the ones A has good promise.

Definition 4 (*Inverse skyline*) For an applicant $A \in \mathcal{A}$, the **inverse skyline** of A is the set of jobs whose skyline views contain A, denoted by $iSky(A) = \{J | A \in SV(J)\}$.

Jobs in the inverse view $IV(A)$ of A should be recommended to A, since A is qualified those jobs. Moreover, A should pay more attention to the jobs in the skyline of $IV(A)$ because they are the best jobs A can get. We model this set of jobs as the reciprocal skyline of A.

Definition 5 (*Reciprocal skyline*) For an applicant $A \in \mathcal{A}$, the **reciprocal skyline** of A is the skyline of the set of jobs that A is qualified, denoted by $rSky(A) = Sky(\{IV(A)\})$.

An important task for headhunters is to match applicants and jobs. When an applicant and a job satisfy the requirement of each other, they have mutual interest to each other. For an applicant, the set of jobs which the applicant is interested in and is qualified form the applicant's *mutual view*.

Definition 6 (*Mutual view*) For an applicant $A \in \mathcal{A}$, the **mutual view** of A is $MV(A) = \{J | J \in V(A) \wedge A \in V(J)\}$.

Furthermore, Ada may be interested in the skyline in her mutual view, captured by her *skyline mutual view*, which consists of the best jobs which she is qualified and interested in.

Definition 7 (*Skyline mutual view*) For an applicant $A \in \mathcal{A}$, the **skyline mutual view** of A is $SMV(A) = Sky(MV(A))$.

The inverse skyline of an applicant cares about the dominance relationships among applicants. For each job in an applicant's inverse skyline, he is a good candidate (i.e., does not dominated by any other applicant). However, the jobs in his inverse skyline may be dominated by other jobs, and this dominance relationship has not been exploreed in inverse skylines. Therefore, we consider the skyline of an applicant's inverse skyline and call it the *skyline of inverse skyline*. The skyline of inverse skyline consists of the best jobs that consider the applicant as one of their best candidates. It takes into account the preferences of both the applicants and the jobs.

Definition 8 (*Skyline of inverse skyline*) For an applicant $A \in \mathcal{A}$, the **skyline of inverse skyline** of A is $SIS(A) = Sky(iSky(A))$.

It is easy to see that a view query is a multidimensional range query where the predicate specifies the search region. An inverse view query is a stabbing query. The inverse view of an applicant consists of all jobs whose predicate regions contain the applicant in the space of applicants. The other 6 types of queries are essentially skyline queries on a subset of the original data set.

Straightforwardly, we can employee an existing range query algorithm, a stabbing query algorithm and a skyline algorithm to answer those queries. In two-way selection recommendation systems, recommendations may be made to many or all members of one or both parties in batch. For example, a headhunter may want to make recommendations to all applicants and jobs. Apparently, when answering a query for all individuals, much computation may be redundant. For example, some jobs are compared again and again when computing skyline views of different applicants. Therefore, in Sect. 4, we develop algorithms to answer each type of queries for *all* applicants/jobs in batch mode in order to share as much computation as possible.

2.3 Properties

Following with the definitions of the skyline view queries, we have several interesting properties about the queries.

Property 1 For any $A \in \mathcal{A}$, $SV(A) \subseteq V(A)$, $iSky(A) \subseteq IV(A)$, $SMV(A) \subseteq MV(A)$, and $SIS(A) \subseteq iSky(A)$.

Property 2 For any $A \in \mathcal{A}$, $MV(A) = V(A) \cap IV(A)$.

We observe that every job in the inverse skyline view of A is also in the inverse view of A.

Property 3 For any applicant $A \in \mathcal{A}$, $iSky(A) \subseteq IV(A)$.

Proof 1 According to Definition 4, for any job $J \in iSky(A)$, $A \in Sky(V(J))$. Thus, $A \in V(J)$. That is, $J \in IV(A)$. Therefore, $iSky(A) \subseteq IV(A)$. $\square$

Figures 1 and 2 show the derivative relationships and the containment relationships among the 8 types of skyline view queries according to the above properties. It is easy to verify that the above properties hold for the example shown in Table 4.

3 Related work

Our study is highly related to the existing work on skyline computation and the stable matching problem.

3.1 Skyline computation on single data set

Since Börzsönyi et al. [1] introduced the skyline operator into the database community, there has been a great amount of research dedicated to skyline query processing as well as extensions and applications of skyline in preference-based data analysis.

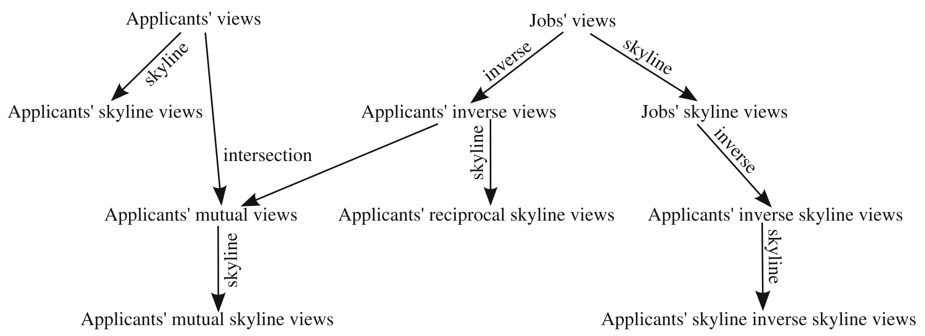

Applicants' views
skyline
Applicants' skyline views
intersection
Jobs' views
inverse
skyline
Applicants' inverse views
Jobs' skyline views
skyline
inverse
Applicants' mutual views
Applicants' reciprocal skyline views
Applicants' inverse skyline views
skyline
skyline
Applicants' mutual skyline views
Applicants' skyline inverse skyline views

Fig. 1 The derivative relationships of the 8 types of skyline view queries

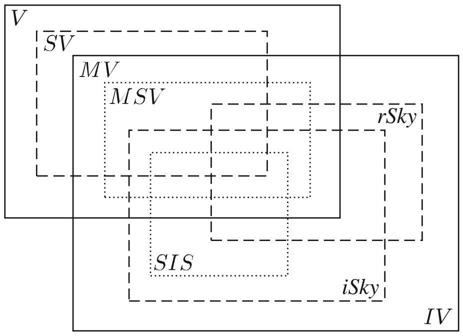


Fig. 2 The containment relationships among the 8 types of skyline view queries

Table 4 The results of the 8 skyline view queries on data sets in Table 1 and 2

Query on applicants	Ada	Bob	Cathy	Dan
V	J2, J3	J1, J3	J1, J2, J3	J2, J3
IV	J1, J2, J3	J1, J3	J1, J2, J3	J2, J3
SV	J2	J1, J3	J1, J2	J2
$iSky$	J1, J2, J3	J1, J3	∅	∅
MV	J2, J3	J1, J3	J1, J2, J3	J2, J3
SMV	J2	J1, J3	J1, J2	J2
$rSky$	J1, J2	J1, J3	J1, J2	J2
SIS	J1, J2	J1, J3	∅	∅
Query on jobs	J1	J2	J3	
V	Ada, Bob, Cathy	Ada, Cathy, Dan	Ada, Bob, Cathy, Dan	
IV	Bob, Cathy	Ada, Cathy, Dan	Ada, Bob, Cathy, Dan	
SV	Ada, Bob	Ada	Ada, Bob	
$iSky$	Bob, Cathy	Ada, Cathy, Dan	Bob	
MV	Bob, Cathy	Ada, Cathy, Dan	Ada, Bob, Cathy, Dan	
SMV	Bob, Cathy	Ada	Ada, Bob	
$rSky$	Bob, Cathy	Ada	Ada, Bob	
SIS	Bob, Cathy	Ada	Bob	

The algorithms of skyline computation fall into two categories: the non-indexed algorithms and the index-based algorithms. The non-indexed algorithms do not assume any index on the data. Such algorithms include the *Divide and Conquer algorithm* and the *Block Nested Loops algorithm* by Börzsönyi et al. [1], the *Sort Filter Skyline algorithm* by Chomicki et al. [2], and the *Linear Elimination Sort for Skyline algorithm* by Godfrey et al. [3].

Different from the non-indexed algorithms which have to scan the whole data set at least once, index-based algorithms take advantage of various pre-built indexes on data sets, so that they only access a portion of a data set to compute the skyline. The *Bitmap algorithm* and the *Index algorithm* by Tan et al. [4], the *Nearest Neighbor algorithm* by Kossmann et al. [5], and the *Branch and Bound Skyline algorithm* by Papadias et al. [6,7] belong to this category.

Particularly, R-trees [8] have been used extensively in index-based algorithms [9]. When computing the skyline of a set of objects, the R-tree index provides a possibility to filter out a subset of objects without examining them one by one. Because the maximum corner of the MBB of a node in the R-tree dominates all objects contained in the MBB, if the maximum corner is dominated by some other object, then none of the objects contained in the MBB can be in the skyline.

The *Branch and Bound Skyline algorithm* (*BBS* for short) [6,7] is an R-tree based skyline algorithm and is one of the fastest algorithms for computing skyline in practice. Given a set of points indexed by an R-tree, BBS traverses the R-tree to compute the skyline. BBS maintains a *min-heap* H built against the minimum distance to the origin of the data space $mindist$ of every entry (node). The algorithm proceeds iteratively.

At the beginning, entries at the root are inserted into H. In each iteration, the top element e of H is processed. If e is fully dominated (i.e., the minimum corner of $e.MBB$ is dominated) by a known skyline point, then e is discarded. Otherwise, if e is a data point, then it is output as a skyline point; if not, e is discarded and those entries at e (that is, the children of e) which are not fully dominated by any known skyline point are inserted into H. An in-memory R-tree on known skyline points is maintained in order to facilitate examining the dominance relationship. The algorithm terminates when H is empty.

BBS ensures that each output point is in the skyline. Moreover, BBS is I/O optimal.

In this paper, we assume that every data set is indexed by an R-tree [8] and use BBS as the baseline in our experiments.

Recently, there is much research studying skylines in multidimensional subspaces. Pei et al. [10,11] and Yuan et al. [12] proposed a skyline cube data structure that completely pre-computes the skylines of all possible subspaces for a given data set. Xia et al. [13] addressed the incremental maintenance of skyline cubes. Tao et al. [14] developed the SUBSKY algorithm to answer subspace skyline queries efficiently in any subspaces. To tackle the problem of skylines in high dimensional spaces, Chan et al. [15] relaxed the definition of dominance to k-dominance and proposed k-dominant skylines.

There are also numerous studies on skyline computation in different environments, such as skyline query processing over data streams [16–20] and time series [21], computing skylines in distributed systems [22,23] and on mobile lightweight devices such as MANETs [24], spatial skyline queries [25], skyline computation in partially ordered domains [26], skyline queries in metric space [27], skyline computation on uncertain data [28,29], using skylines to mine user preferences and make recommendations [30,31].

To the best of our knowledge, we are the first to study using skyline queries for recommendation in two-way selections. Although the proposed skyline view queries can be answered by existing methods individually, those methods only consider answering a single query at a time. It is far from trivial to develop efficient algorithms for answering skyline view queries in batch.

3.2 Skyline computation involving two data sets

Most of the previous work only considers computing skylines (or their variants) on a single data set, whereas our skyline view queries deal with two distinct data sets. Recently, Dellis et al. [32] studied the concept of reverse skyline and bichromatic reverse skyline. They consider a set of products described by multiple features and a set of customers, each of whom specifies his/her favorite product features as a data point in the product space. Given a product q, the reverse skyline consists of the customers whose dynamic skyline contains q. The dynamic skyline of a customer preference p corresponds to a transformed data space where p becomes the origin and all other products are represented by their distance vectors to p. Lian et al. [29] studied the bichromatic probabilistic reverse skyline queries over uncertain data. Wu et al. [33] proposed several heuristics to enhancing the performance.

Our skyline view queries are fundamentally different to bichromatic skylines and reverse skylines.

From the application point of view, bichromatic skylines and reverse skylines care about only the preference on one party. Customers have interests and preferences in products, but products cannot choose customers. However, our skyline views queries consider two-way mutual selections between two parties and both parties have preferences in each other.

Given this fundamental difference, our problems are modeled differently to bichromatic skylines and reverse skylines. Bichromatic and reverse skylines compute the skyline of the data set of one party in a transformed space according to the query point, and they consider all points in computation. On the other hand, our skyline view queries essentially compute the skyline of a subset of one party of the data set according to the constraints specified by the query, and we do not use space transformation.

As a result, the query answering techniques are quite different. The algorithms for bichromatic and reverse skylines focus on reducing the overhead of data space transformation in order to answer one query. However, our algorithms aim at minimizing repetitive computation when answering multiple queries in batch.

Nevertheless, we pointed out that the inverse skylines, and the reverse skylines are equivalent to each other for special input data as follows.

– For inverse skylines, for each applicant, choose his/her predicate such that all jobs in the job data set satisfy this predicate.
– For reverse skylines, for each customer, choose his/her preference to be the origin point in the product space such that no space transformation is needed to answer a reverse skyline query.

In the above two special cases, the inverse skyline of a job is the same as the reverse skyline of a product. In fact, the inverse skyline and the reverse skyline of a job/product are equal to the set of applicants/customers, if this job/product is in the skyline of the set of jobs/products. Otherwise, the inverse skyline and the reverse skyline of this job/product are an empty set. In our experiments, we experimentally compare our algorithms for inverse skylines and the state-of-the-art algorithms for reverse skylines [33].

3.3 The stable matching problem

Making recommendations between two parties is related to the classical *stable matching problem* [34] (also known as *the stable marriage problem*). An applicant and a job can be matched only if they satisfy the requirements from each other. In a matching between all applicants and all jobs, assume applicant A is paired with job J, and applicant A' is paired with job J'. If A and J' satisfy the requirements from each other, and A prefers J' to J and J' prefers A to A', then the pair A and J' is called a dissatisfied pair. A matching is said to be stable if there is no dissatisfied pair.

In the original version of the stable marriage problem [34], the applicant set and the job set have the same cardinality (denoted by n). The preference of each applicant on jobs is a complete ranked list (i.e., including all jobs in the job set) without tie. The preference of each job on applicants is also such a ranked list. Gale and Shapley [34] showed that there always exists a stable matching and it can be found in $O(n^2)$ time.

Numerous variants of the stable marriage problem are studied, such as the college admission problem where a college can accept more than one student [34], the roommate assignment problem where two parties are from the same set [35], the stable matching problem with incomplete preference lists and/or with ties [36–38], etc. In general, the preference can be a partial order. In this case, the problem is proved NP-hard and hard to be approximated [38]. In this paper, the dominance relationship is employed to model the preference, which in fact defines a partial order.

Although the stable marriage problem and our problem share similar motivations, they are very different in terms of the goal and the computational issues. The stable marriage focuses on the global picture and tries to find an assignment such that every one in both parties is satisfied. Each individual does not have the ability to choose after the assignment is made. However, instead of giving the matching pairs straightforwardly, our paper addresses a series of flexible queries for mutual decision-making parties. Our problem cares about the recommendations for individuals. We provide the results of skyline view queries as recommendations which are good candidates based on each individual's interest. Individuals can then make decisions and choices from our skyline view queries without processing the original large data set.

4 Query answering algorithms

In this section, we develop algorithms to answer the 8 skyline view queries in batch. In particular, we design efficient algorithms for reciprocal skyline queries and inverse skyline queries in Sects. 4.2 and 4.3, respectively. Other queries can be reduced to these two queries as discussed in Sect. 4.1. We assume that both the applicant data set and the job data set are indexed by R-trees. We only describe the algorithms for queries on applicants since the same algorithms can be applied to answer queries for jobs symmetrically.

4.1 Query reduction

According to the definition of view queries and that of inverse view queries, with linear time complexity, we can derive the inverse views of all applicants from the views of all jobs. Thus, an inverse view query can be answered efficiently using the view query answering algorithm. This property also holds between skyline view queries and inverse skyline view queries. Besides, a mutual view can be obtained by intersecting the corresponding view and the corresponding inverse view.

Example 1 (*Query Reduction*) Consider the example in Table 4. By scanning the view of jobs J1 which is {Ada, Bob, Cathy}, we can assign J1 to the inverse views of Ada, Bob, and Cathy. Similarly, we assign J2 to the inverse views of Ada, Cathy, and Dan, and J3 to the inverse views of Ada, Bob, Cathy, and Dan. Thus, we obtain the inverse views of Ada, Bob, Cathy, and Dan as shown in the table.

Since the view and the inverse view of Ada are {J2, J3} and {J1, J2, J3}, respectively, whose intersection is exactly the mutual view of Ada, {J2, J3}. It is easy to verify this property for other applicants.

Similarly, by intersecting the corresponding skyline view and the corresponding inverse skyline view, we get a skyline mutual view.

Therefore, we can reduce the 8 types of skyline view queries into 5 types of queries—view queries, inverse skyline queries, reciprocal skyline queries, skyline mutual view queries, and skyline of inverse skyline queries.

Using R-trees, a view query can be answered by a simple R-tree traverse. When computing the views of all applicants in batch, we first insert the predicate regions of all applicants into an R-tree, referred as the *applicant predicate R-tree*. Then, an R-tree join algorithm [39] on the R-tree of the job data set and the applicant predicate R-tree can compute all views efficiently.

Sections 4.2, 4.3, and 4.4, respectively, discuss how to compute inverse skylines, reciprocal skylines, skyline mutual views, and skyline of inverse skylines efficiently.

4.2 Answering inverse skyline queries

To process inverse skyline queries in batch and avoid redundant comparisons, we systematically make use of the dominance relation between applicants and the dominance relation between the maximum corners of the views of jobs.

4.2.1 Algorithm framework

Our algorithm framework explores Property 3 of inverse skylines.

Algorithm 1 The inverse skyline algorithm.

```
Input: $\mathcal{A}$, $\mathcal{J}$, and $IV(A)$ for all $A \in \mathcal{A}$;
Output: $iSky(A)$ for all $A \in \mathcal{A}$;
Description:
1: for all $A \in \mathcal{A}$ do
2:    $iSky(A) = IV(A)$;
3: end for
4: call Algorithm 3 to refine $iSky(A)$ for all $A$ using dominance relationships among applicants;
5: call Algorithm 4 to refine $iSky(A)$ for all $A$ using relationships among jobs' predicates;
6: return $iSky(A)$ for all $A \in \mathcal{A}$;
```

Using Property 3, we can compute the inverse skyline of an applicant from her/his inverse view instead of from the whole data set. Algorithm 1 shows the framework based on Property 3.

To begin with, we initialize the candidate set of $iSky(A)$ for each applicant A to $IV(A)$ (lines 1–3). Then, in Sect. 4.2.2, we discuss our techniques to refine the inverse skylines of all applicants using the dominance relation between applicants (line 4). We prove in Lemmas 1 and 2 that the inverse skyline of an applicant A is her/his inverse view excluding the inverse skylines of the applicants who dominate A. In Sect. 4.2.4, we make use of the dominance relation between the maximum corners of the views of jobs (line 5). We prove in Lemmas 3 and 4 that if a job J_1 is not in the inverse skyline of an applicant A, then any job J_2 is not in the inverse skyline of A if the maximum corner of J_2's view dominates the maximum corner of J_1's view. We describe the details in the rest of this subsection.

4.2.2 Dominance relationships among applicants

First, we explore a property of the dominance relation among applicants.

Lemma 1 (Dominance relation among applicants) *Given two applicants A_1 and A_2, if $A_1 \succ A_2$, then for any $J \in iSky(A_1)$, $J \notin iSky(A_2)$.*

Proof 2 If $A_2 \notin V(J)$, then $A_2 \notin SV(J)$. Otherwise, if $A_2 \in V(J)$, since $A_1 \in V(J)$ and $A_1 \succ A_2$, then we also have $A_2 \notin SV(J)$. Thus, $J \notin iSky(A_2)$. □

By Lemma 1, we define the dominating set for each applicant.

Definition 9 (*Dominating set*) For an applicant A, the **dominating set** of A is the set of applicants dominating A, denoted by $DS(A) = \{A' \in \mathcal{A} | A' \succ A\}$.

We have the following result based on the dominance relation among applicants.

Lemma 2 (Inverse skyline)

$$iSky(A) = IV(A) \setminus \bigcup_{A' \in DS(A)} iSky(A'). \tag{1}$$

Proof 3 Combining Property 3 and Lemma 1, we have

$$iSky(A) \subseteq IV(A) \setminus \bigcup_{A' \in DS(A)} iSky(A'). \tag{2}$$

Suppose there exists a job

$$J \in IV(A) \setminus \bigcup_{A' \in DS(A)} iSky(A')$$

and $J \notin iSky(A)$. Because $A \in V(J)$, there must be another applicant $A'' \in SV(J)$ such that $A'' \succ A$. Therefore, $J \in iSky(A'')$ and $A'' \in DS(A)$. This contradicts to the assumption. Thus,

$$IV(A) \setminus \bigcup_{A' \in DS(A)} iSky(A') \subseteq iSky(A). \tag{3}$$

Equation (1) follows with Eqs. (2) and (3). □

Example 2 (*Inverse skylines*) In our running example (Tables 1 and 2), Ada's inverse skyline is {J1, J2, J3}. Since Ada dominates Cathy, none of J1, J2, and J3 is in Cathy's inverse skyline view, though all of them are in Cathy's inverse view.

Using Lemma 1, because $DS(Cathy) = \{Ada\}$, $iSky(Cathy) = IV(Cathy) \setminus iSky(Ada) = \{J1, J2, J3\} \setminus \{J1, J2, J3\} = \emptyset$.

Using Eq. (1) directly, however, is not efficient to compute the inverse skylines of all applicants iteratively, because the complexity of finding out the exact dominating sets of all applicants is $O(|\mathcal{A}|^2)$, where $|\mathcal{A}|$ is the cardinality of the applicant data set. To tackle the problem, we introduce a *layer structure* to approximate the dominating sets effectively. Similar data structures are adopted in some previous work, such as the layer structure in [40], the dominance graph in [41] and DADA cube [42].

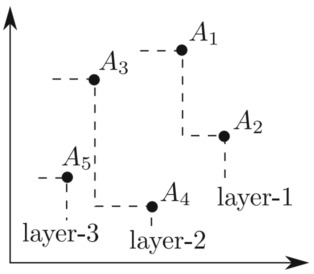


Fig. 3 An example of the layer structure (larger values are preferred)

4.2.3 The layer structure

We divide the applicants into layers. The first layer of the applicant data set is the skyline among all applicants. The second layer contains those applicants only dominated by some other applicants at the first layer. The rest layers can be defined inductively.

Definition 10 (*Layers*) An applicant $A \in \mathcal{A}$ is at **layer-**1 if $A \in Sky(\mathcal{A})$. An applicant A is at **layer-**k ($k > 1$) if A is not at the 1-st, ..., the $(k-1)$-th layers and A is not dominated by any applicant except for those at the 1-st, ..., the $(k-1)$-th layers.

Example 3 (*The layer structure*) Figure 3 shows an example of the layer structure. A_1 and A_2 are at layer-1. A_3 (dominated by A_1) and A_4 (dominated by A_2) are at layer-2. A_5 is at layer-3 since it is dominated by A_3.

In the layer structure, we only maintain the dominance relation between applicants in adjacent layers. In the example shown in Fig. 3, we only keep records of $A_1 \succ A_3$, $A_1 \succ A_4$, $A_2 \succ A_4$, and $A_3 \succ A_5$. $A_1 \succ A_5$ can be found transitively through $A_1 \succ A_3$ and $A_3 \succ A_5$.

The dominance relation between applicants in adjacent layers may not keep all dominance relations. For example, $A_2 \succ A_5$ is missing. This is the tradeoff we make so that we can compute the layer structure in time less than $O(|\mathcal{A}|^2)$. As we will show, using the layer structure and some other information from the job data set, it is still sufficient to answer inverse skyline queries correctly.

We describe our layer structure construction algorithm in Algorithm 2. To build the layer structure, for each applicant A, we define the *key* of an applicant as the sum of its values on all attributes, that is, $A.key = \sum_{i=1}^{d} A.D_i$ (lines 1–3). Then, we sort all applicants in the key descending order (line 4). This is motivated by the sort-filter-skyline algorithm [2]. The sorted list of applicants has a nice property: for applicants A_1 and A_2 such that $A_1 \succ A_2$, A_1 precedes A_2 in the sorted list. Using this property, we scan the sorted list once (lines 6–9) and for each applicant in the sorted order, we determine the layer it belongs to.

The first applicant has the maximum key value and is assigned to layer-1 (line 5). We compare the second applicant with the first one. If the second one is dominated, then it is assigned to layer-2. Otherwise, it is assigned to layer-1.

Generally, when we process an applicant A (lines 8–25), suppose at the time there already exist h layers. We compare A with the applicants currently at layer-$\lfloor \frac{h}{2} \rfloor$. Then,

- if A is dominated by an applicant at that layer, then A must be at some layer higher than $\lfloor \frac{h}{2} \rfloor$;
- otherwise, A is neither dominated by nor dominates, any applicant at that layer. Then, A must be at that layer or some lower layer.

Algorithm 2 The algorithm of building the layer structure.

```
Description:
Input: the applicant data set $\mathcal{A}$;
Output: the layer structure $LA$;
Description:
1: for all $A \in \mathcal{A}$ do
2:   $A.key = \sum_{i=1}^{d} A.D_i$;
3: end for
4: sort all applicants in $\mathcal{A}$ according to their $key$ values in descending order into a sorted list $L$;
5: assign the first applicant in $A$ to layer-1 of $LA$;
6: the number of layers we have so far $h = 1$;
7: while $A$ is not the last applicant in $L$ do
8:   let $A$ be next applicant in $L$;
9:   $low = 1, high = h$;
10:  while $low \leq end$ do
11:    $middle = \lfloor (low + high)/2 \rfloor$;
12:    if $A$ is dominated by an applicant at layer-$middle$ then
13:      $low = middle + 1$;
14:    else
15:      $high = middle - 1$;
16:    end if
17:  end while
18:  if $low == middle$ then
19:    insert $A$ to layer-$middle$;
20:  else
21:    if $middle == h$ then
22:      create a new layer-$(h + 1)$ in $LA$; $h = h + 1$;
23:    end if
24:    insert $A$ to layer-$(middle + 1)$;
25:  end if
26: end while
27: return $LA$;
```

This comparison operation (line 12) can be done efficiently if we sort all applicants at a layer in the descending order of their values on the first attribute (in fact, any attribute works here). Then, instead of comparing with all applicants at this layer, A is compared with only those applicants that have a larger value on the first attribute than A. We conduct this binary search recursively until A is assigned to a layer. After that, we find all applicants dominating A at the layers just below the layer of A. Example 4 shows an example of Algorithm 2.

Example 4 (*Building layer structure*) In Fig. 3, assume the 5 applicants have values $A_1 = (4, 5)$, $A_2 = (5, 3)$, $A_3 = (2, 4)$, $A_4 = (3, 1)$, and $A_5 = (1, 2)$, respectively, on the horizontal and vertical dimensions. By soring them in the key descending order, we have the list A_1, A_2, A_3, A_4, A_5. Then, we determine their layers one by one in the sorted order.

First, A_1 is at layer-1. A_2 is not dominated by A_1, so A_2 is also at layer-1. Then, A_3 is dominated by A_1, so A_3 is not at layer-1, and we create layer-2 and add A_3 into it. Now the number of layers $h = 2$. Then, for A_4, we check layer-1 (since $\frac{h}{2} = 1$), and find that A_4 is dominated by A_1, so A_4 should be at layers higher than layer-1. Recursively, we compare A_4 with applicants at layer-2. A_4 is not dominated by any applicant at layer-2, so A_4 is at layer-2. Finally, for A_5, it is dominated by A_1 at layer-1, then dominated by A_3 at layer-2, so we put it at a new layer-3.

Algorithm 3 shows the procedure of refining the inverse skylines of applicants using the layer structure. We iteratively process applicants layer-by-layer. The applicants at layer-1 are

Algorithm 3 Refining using dominance relationships among applicants.

Description:

```
1: build the layer structure $LA$ of $\mathcal{A}$;
2: for all layer $l$ of $LA$ iterating from layer-2 to the highest one do
3:   for all applicant $A$ at layer $l$ do
4:     for all applicant $A'$ at the layer just below layer $l$ do
5:       if $A' \succ A$ then
6:         $iSky(A) = iSky(A) \setminus iSky(A')$; /* Lemma 1 */
7:       end if
8:     end for
9:   end for
10: end for
```

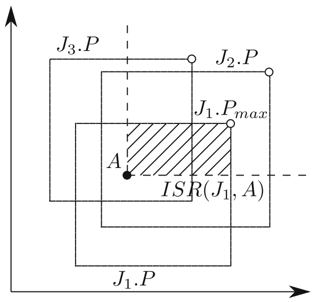


Fig. 4 An example of the inverse skyline region

not dominated by any other applicants. Therefore, their inverse skylines are the same as their inverse views. For an applicant A at layer-$l (l > 1)$, we find every applicant A' at layer-$(l-1)$ such that $A' \succ A$ and remove jobs in $iSky(A')$ from $iSky(A)$ according to Lemma 1 (line 6).

Using Algorithm 3, the candidate set of every applicant's inverse skyline is significantly reduced. However, the layer structure only captures a subset of the dominating set of an applicant. As shown in Fig. 3, $A_2 \succ A_5$ is missing. Therefore, we need to explore some further refining methods using the dominance relation among predicates of jobs.

4.2.4 Dominance relation among predicates of jobs

Let us show that whether a job J is in the inverse skyline of an applicant A can be determined by simply checking whether there is any other applicant in a specific region referred by the *inverse skyline region* of J with respect to A.

Definition 11 (*Inverse skyline region*) Denote $J.P_{\max}$ the maximum (top right) corner of $J.P$, the predicate of J. $ISR(J, A)$, the **inverse skyline region** of J with respect to A, is the region with A as the minimum (bottom left) corner and $J.P_{\max}$ as the maximum corner.

Example 5 (*The inverse skyline region*) Suppose for an applicant A the inverse view $IV(A) = \{J_1, J_2, J_3\}$. The predicates of all jobs in $IV(A)$ are shown in Fig. 4. For a job $J_i \in IV(A)$ $(i = 1, 2, 3)$, A is inside the predicate of J_i. The shaded region in Fig. 4 is $ISR(J_1, A)$.

The inverse skyline region can be used to verify whether a job belongs to the inverse skyline of an applicant, as indicated by the following result.

Lemma 3 (Inverse skyline region) *Given an applicant A and a job $J \in IV(A)$, $J \in iSky(A)$ if and only if there does not exist another applicant $A' \neq A$ such that $A' \in ISR(J, A)$.*

Proof 4 $J \in iSky(A)$ if and only if $A \in SV(J)$. Because $J \in IV(A)$, $A \in V(J)$. Hence, A is in the skyline of $V(J)$ if and only if there exists no applicant in region $ISR(J, A)$. □

Lemma 3 provides a simple method to compute the inverse skyline of an applicant A. If the inverse skyline region $ISR(J, A)$ of a job J with respect to an applicant A contains some other applicants, then $J \notin iSky(A)$.

Lemma 4 (Dominance between predicates of jobs) *Given an applicant A and two jobs $J_1, J_2 \in IV(A)$, let $J_1.P_{\max}$ and $J_2.P_{\max}$ be the maximum corners of the predicates of J_1 and J_2, respectively. If $J_1 \notin iSky(A)$ and $J_2.P_{\max} \succ J_1.P_{\max}$, then $J_2 \notin iSky(A)$.*

Proof 5 According to Definition 11, $ISR(J_1, A)$ and $ISR(J_2, A)$ share the same minimum corner A. For their maximum corners $J_1.P_{\max}$ and $J_2.P_{\max}$, respectively, if $J_2.P_{\max} \succ J_1.P_{\max}$, then $ISR(J_2, A)$ contains $ISR(J_1, A)$. If $J_1 \notin iSky(A)$, there must be an applicant in $ISR(J_1, A)$, thus in $ISR(J_2, A)$. Therefore, $J_2 \notin iSky(A)$. □

Example 6 (*Using Lemma 4*) In Fig. 4, $ISR(J_2, A)$ contains $ISR(J_1, A)$. Thus, we immediately know that $J_2 \notin iSky(A)$.

Lemma 4 transforms the containment relation among inverse skyline regions into the dominance relation among the maximum corners of predicates. Similar to the layer structure on applicants, we build another layer structure on the maximum corners of the predicates of all jobs so that we can apply Lemma 4 iteratively (pseudocode in Algorithm 4).

Algorithm 4 Refining using relations among predicates of jobs.

```
Description:
1: build the layer structure $LJ$ on the maximum corners of the predicates of all jobs in $\mathcal{J}$;
2: for all applicant $A \in \mathcal{A}$ not at layer-1 of $LA$ do
3:   organize the maximum corners of jobs in $iSky(A)$ as layer structure $LJ(A)$ according to $LJ$;
4:   for all layer $l$ of $LJ(A)$ iterating from the highest layer to layer-1 do
5:     for all $J.P_{max}$ at layer $l$ do
6:       for all $J'.P_{max}$ at the layer just above layer $l$ do
7:         if $J.P_{max} \succ J'.P_{max}$ and $J' \notin iSky(A)$ then
8:           $iSky(A) = iSky(A) \setminus \{J\}$; /* Lemma 4 */
9:           goto line 15;
10:        end if
11:      end for
12:      if $ISR(J, A)$ contains other applicants then
13:        $iSky(A) = iSky(A) \setminus \{J\}$; /* Lemma 3 */
14:      end if
15:    end for
16:  end for
17: end for
```

First, we build the layer structure LJ on the maximum corners of the predicates of all jobs in $\mathcal{J}$. Applicants at layer-1 do not need any refinement. For each applicant A not at layer-1, we retrieve the jobs in the current candidate set of $iSky(A)$ and organize the maximum corners of their predicates in a layer structure, denoted by $LJ(A)$, which is simply a subset of LJ since $iSky(A) \subseteq \mathcal{J}$. Similar to Algorithm 3, we iteratively examine the jobs in $iSky(A)$ and apply Lemma 4 to further refine $iSky(A)$ (lines 5–11). If Lemma 4 does not determine that J is not in $iSky(A)$, Lines 12-14 conduct a final check of the candidate jobs in $iSky(A)$ according to Lemma 3. This can be done by a range query on the R-tree of the applicant data set.

4.2.5 Performance analysis

The time complexity of the layer construction algorithm (Algorithm 2) depends on the input. In the best case where every applicant dominates the one immediate after in the sorted list according to their *key* values, the algorithm acts exactly as a binary search for each applicant, so it takes $O(|\mathcal{A}| \log |\mathcal{A}|)$ time in total. In the worst case where applicants do not dominate each other at all and every applicant is at the first layer, the algorithms takes $O(|\mathcal{A}|^2)$ time.

Using the layer structure, it is expected that the inverse skyline of an applicant A can be refined significantly by subtracting the inverse skylines of the applicants that dominate A (Algorithm 3). A subtraction operation can be done in linear time with respect to the size of the inverse skylines involved if we organize the inverse skylines as sorted lists with a global order on jobs (e.g., the IDs of jobs).

It is expected that Algorithm 3 can refine the inverse skyline significantly. However, in the worst case where Algorithm 3 fails to refine the inverse skyline, Algorithm 4 can be still fast if Lemma 4 can successfully remove a job from consideration. Otherwise, we have to do range queries as specified in Lemma 3 (Algorithm 4), and we assume a range query takes time $O(\log |\mathcal{A}|)$ using an R-tree on data set $\mathcal{A}$.

4.3 Answering reciprocal skyline queries

In this section, we develop an R-tree based algorithm to efficiently compute the reciprocal skylines for all applicants from their inverse views.

The reciprocal skyline of an applicant A is the skyline in the inverse view of A. Intuitively, for another applicant A' close to A in the applicant space, that is, they have similar value on every attribute, A' may have similar inverse view as A. Hence, the skyline computation on their inverse views can be shared.

Taking advantage of an R-tree, applicants close to each other are often clustered into one leaf node. This gives us the opportunities to develop a computation sharing strategy. We first show how to share the computation at the leaf level of the R-tree and then extend the method to other levels.

4.3.1 Sharing computation at the leaf level

Let N be a leaf node in the R-tree of the applicant data set. We also use N to denote the set of applicants stored at node N. Denote by $iIV(N)$ the intersection of the inverse views of the applicants in N, that is, $iIV(N) = \bigcap_{A \in N} IV(A)$. $iIV(N)$ is the set of common jobs that all applicants in N are qualified. For each applicant $A \in N$, let $eIV(A) = IV(A) \setminus iIV(N)$ be the set of jobs that A is qualified but not all other applicants in N are qualified. Because the applicants in a leaf node have similar inverse views, $eIV(A)$ is often a small set.

The equation below computes the reciprocal skyline.

$$\begin{aligned} rSky(A) &= Sky(IV(A)) \\ &= Sky\Big(Sky(iIV(N)) \cup Sky(eIV(A))\Big) \end{aligned} \tag{4}$$

We pre-compute $iIV(N)$ and $eIV(A)$ for each applicant A in N. According to Eq. (4), we only need to compute the skyline of $iIV(N)$ once for all applicants. Computing the skyline of $eIV(A)$ is easy since $eIV(A)$ only contains a small number of jobs. Furthermore, both $Sky(iIV(N))$ and $Sky(eIV(A))$ are small thus the final skyline can be computed efficiently.

4.3.2 Sharing computation at other levels

When computing the skyline of $iIV(N)$ of a non-leaf node N, the method in Sect. 4.3.1 can be applied recursively.

Let M be the parent node of N. We also use M to denote the set of child nodes in M. For each sibling node $N' \in M$ of N, $iIV(N')$ is relatively similar to $iIV(N)$ due to the property of the R-tree. Therefore, we compute the intersection of $iIV(N)$ of every child node $N \in M$, denoted by $iIV(M) = \bigcap_{N \in M} iIV(N)$. We also compute $eIV(N) = iIV(N) \setminus iIV(M)$. Then, Eq. (4) can be extended to

$$Sky(iIV(N)) = Sky\Big(Sky(iIV(M)) \cup Sky(eIV(N))\Big). \tag{5}$$

Recursively, we apply this procedure at every level of the R-tree. Algorithm 5 shows the details. We first compute $eIV(A)$, $iIV(N)$ and $eIV(N)$ for all applicants and nodes in the R-tree on the applicant data set from bottom up (lines 1–14). Then, we traverse the R-tree in a breath-first manner to compute $Sky(iIV(N))$ for each node and $rSky(A)$ for each applicant using Eqs. (5) and (4), respectively (lines 15–22).

Algorithm 5 The reciprocal skyline algorithm.

```
Input: the R-tree $\mathcal{R}$ of $\mathcal{A}$, and $IV(A)$ for all $A \in \mathcal{A}$;
Output: $rSky(A)$ for all $A \in \mathcal{A}$;
Description:
1: for all node $N$ on the leaf level of the R-tree of $\mathcal{A}$ do
2:   $iIV(N) = \bigcap_{A \in N} IV(A)$;
3:   for all applicant $A \in N$ do
4:     $eIV(A) = IV(A) \setminus iIV(N)$;
5:   end for
6: end for
7: for all level $l$ of $\mathcal{R}$ from the bottom up except the leaf level do
8:   for all node $M$ at level $l$ do
9:     $iIV(M) = \bigcap_{N \in M} iIV(M)$;
10:    for all child node $N \in M$ do
11:      $eIV(N) = iIV(N) \setminus iIV(M)$;
12:    end for
13:  end for
14: end for
15: for all level $l$ of $\mathcal{R}$ from the top down do
16:   for all node $N$ on level $l$ do
17:     compute $Sky(iIV(N))$ by Equation (5);
18:   end for
19: end for
20: for all applicant $A \in \mathcal{A}$ do
21:   compute $rSky(A)$ by Equation (4);
22: end for
23: return $rSky(A)$ for all $A \in \mathcal{A}$;
```

4.3.3 Performance analysis

Let $|IV|$ denote the average size of the inverse views of applicants. Let the time to compute the reciprocal skyline of an applicant using an existing skyline computation algorithm be $O(s(|IV|))$, which in the worst case takes time $O(|IV|^2)$. Using a straightforward algorithm to compute the reciprocal skyline of each applicant separately, the complexity is

$O(|\mathcal{A}|s(|IV|))$. Thus, in the worst case, the complexity of the straightforward algorithm is $O(|\mathcal{A}||IV|^2)$.

In our algorithm, it takes time $O(|\mathcal{A}||IV|)$ to pre-compute the intersection iIV for every tree node. After that, we expect to share a lot of skyline computation since many applicants have similar inverse views, though the worst case complexity remains the same as the straightforward method.

4.4 Answering skyline mutual view queries and skyline of inverse skyline queries

It is easy to see that the sharing strategy used for computing reciprocal skylines is also applicable to skyline mutual views and skyline of inverse skylines, since a mutual view and an inverse skyline view are subsets of the corresponding inverse view. Thus, with minor revision, Algorithm 5 can also answer skyline mutual view queries and skyline of inverse skyline queries. The only difference is that we replace inverse skyline views by the mutual views or the inverse skyline views for the input, then the algorithm will produce skyline mutual views or skyline of inverse skylines instead of reciprocal skylines.

5 Experiments

In this section, we empirically evaluate the running time of the two algorithms developed in Sect. 4, the inverse skyline algorithm (*iSky* for short) and the reciprocal skyline algorithm (*rSky* for short). We assume all algorithms are in-memory, thus the I/O cost is not considered. The algorithms were implemented in C++ and compiled by Microsoft Visual Studio 2008. All experiments were conducted on a laptop computer with an Intel Core Duo 1.67 GHz CPU and 2 GB main memory running Windows Vista Ultimate.

To the best of our knowledge, no real data sets of two party recommendation applications have been made public. Thus, we had to use synthetic data sets in our experiments. The synthetic applicant data sets and job data sets used in our experiments were generated in the same way. Each (applicant or job) data set contains n objects in a d-dimensional space, where the cardinality n varies from 25,000 to 100,000 and the dimensionality varies from 2 to 5. By default, $n = 50{,}000$ and $d = 3$. The domain of every attribute is [0,1]. We show the results on data sets with 2 distributions, namely, anti-correlated distribution and independent distribution, generated by the benchmark generator developed by Börzsönyi et al. [1]. Limited by space, we do not show the results on the correlated data sets, which are similar to the results on the other two distributions.

The view (predicate) of an applicant is represented by a rectangle. We first generate the centers of the views of all applicants, then bound a rectangle around each center. The view centers in every attribute follow a normal distribution with mean 0.5 and standard deviation σ. When σ is small, the rectangles become heavily overlapped, hence, every applicant has a similar view. On the other hand, when σ is large, the views are diverse. In our experiments, σ takes values $0.1,\ 0.3,\ 0, 5$, or 0.7, and 0.3 is used as the default value. We also control the view sizes (i.e., the number of jobs in an applicant's view) such that they follow a normal distribution with mean s and standard deviation σ'. s varies from 100 to 700 with 300 as the default. Experiment results are not sensitive to different values of this standard deviation σ'. We omit the details due to space limit. In all experiments, we set $\sigma' = s/3$. The views of jobs are generated similarly.

Table 5 summaries experiment parameters and their default values. Limited by space, here we only report the results on inverse skyline queries and reciprocal queries. As indicated in

Table 5 Experiment parameters

Parameter	Values (bold font for default values)
Cardinality n	25,000, **50,000**, 75,000, 100,000
Dimensionality d	2, **3**, 4, 5
View size s	100, **300**, 500, 700
View standard deviation σ	0.1, **0.3**, 0.5, 0.7

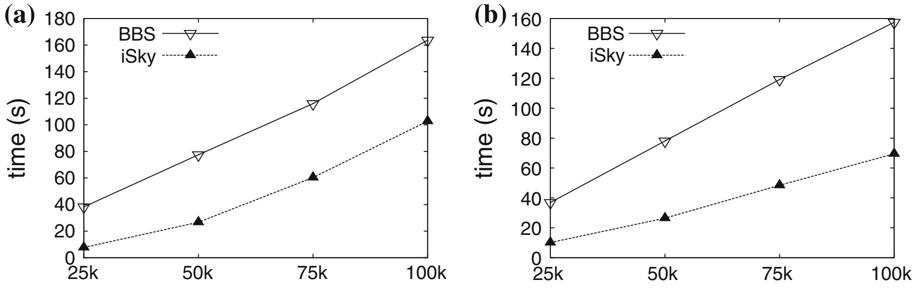


Fig. 5 Effect of cardinality on iSky **a** anti-correlated, **b** independent

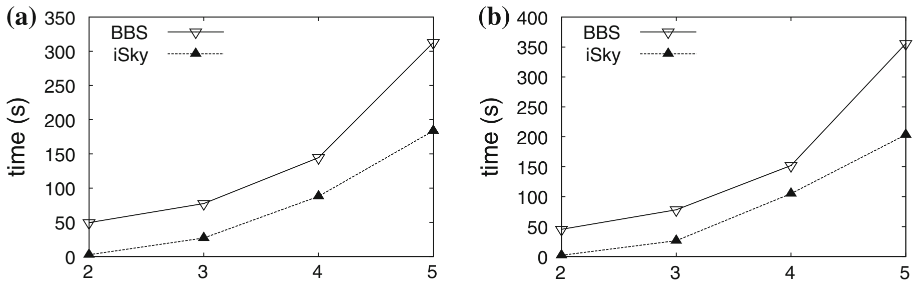


Fig. 6 Effect of dimensionality on iSky **a** anti-correlated, **b** independent

Sect. 4.1, the other skyline view queries can be answered either by straightforwardly using the R-tree indexes or by slightly adapting the algorithms for inverse skyline queries and reciprocal queries.

5.1 Answering inverse skyline queries

In this subsection, we evaluate our inverse skyline algorithm (*iSky* for short).

The skyline view query of a job is to compute the skyline of applicants within a given range. This is called a *constraint skyline query* by Papadias et al. [7]. They developed the branch-and-bound algorithm (BBS for short) that is the best existing method for computing a constraint skyline. Hence, we use *iSky* to compute the inverse skylines of all applicants, and compare *iSky* with a method that employs BBS to compute the skyline view of all jobs one by one. The BBS implementation used in our experiments was provided by the authors of [7]. Note that, for a fair comparison, both algorithms run in memory and the I/O cost is not considered.

Figures 5, 6, 7, and 8 compare *iSky* and BBS with respect to cardinality, dimensionality, view size, and view overlap, respectively. It is clear to see that *iSky* outperforms BBS in every

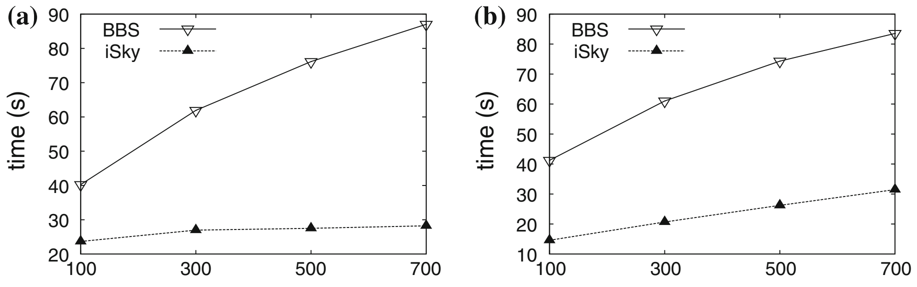


Fig. 7 Effect of view size on iSky **a** anti-correlated, **b** independent

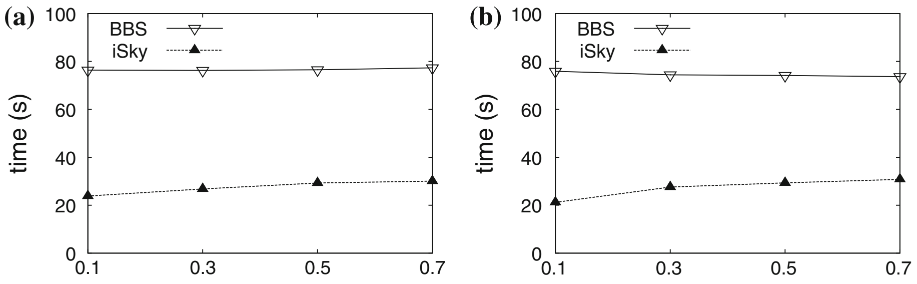


Fig. 8 Effect of view overlap on iSky **a** anti-correlated, **b** independent

experiment. Especially, when we increase the view size (Fig. 7), *iSky* shows a big advantage over BBS. The major reason of the big difference in efficiency is that it is costly to compute the skyline of each view separately when the views are large. Our pruning techniques in *iSky* avoid redundant comparison and computation.

iSky and BBS have similar trends as the experiment parameters vary. The running time of both methods increases linearly when the cardinality increases from 25,000 to 100,000 and increases exponentially when the dimensionality increases from 2 to 5.

Figure 8 shows that BBS is not sensitive to the overlap of views, since it computes the skyline of each view independently. However, when the standard deviation of the view distribution increases from 0.1 to 0.7, the overlap of views decreases, thus the running time of *iSky* increases. However, the increase is very moderate.

As mentioned in Sect. 3.2, our inverse skyline and the reverse skylines [29,32,33] are equivalent in special cases where the view of every applicant contains all jobs and the preference of every customer is the origin in the product space. Figure 9 compares the runtime of our *iSky* algorithm and the state-of-the-art reverse skyline algorithm *BRS* [33] in the special cases. Clearly, our *iSky* is faster than *BRS* by 2 orders of magnitude. The reason is that our *iSky* runs in batch and shares computations when answering multiple inverse skyline queries.

In order to show the effectiveness of our sharing strategies for batch processing, Fig. 10 investigates the performance of *iSky* on clustered data sets. We generate data sets consisting of 100 clusters. For each data set, we first generate 100 points in anti-correlated or independent distribution, each of which serves as the center of a cluster. Then, for each cluster, we generate 250 points following a Gaussian distribution with mean as the cluster center and variance δ varying from 0.1 to 0.5. Figure 10a, b show the plots of such data sets with δ being 0.1 and 0.5, respectively. We see that the smaller δ the more clustered the data set.

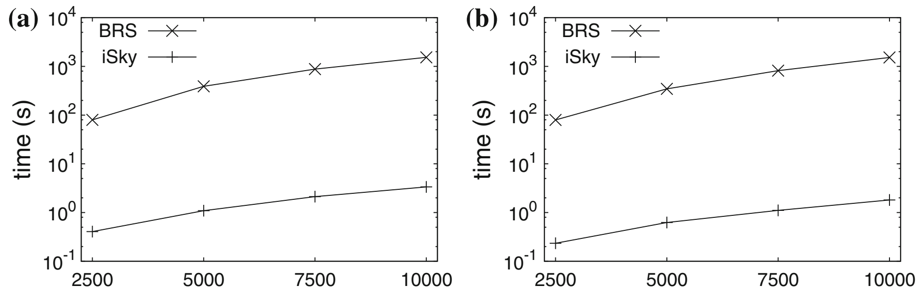


Fig. 9 Comparison between *iSky* and *BRS* with various cardinalities **a** anti-correlated, **b** independent

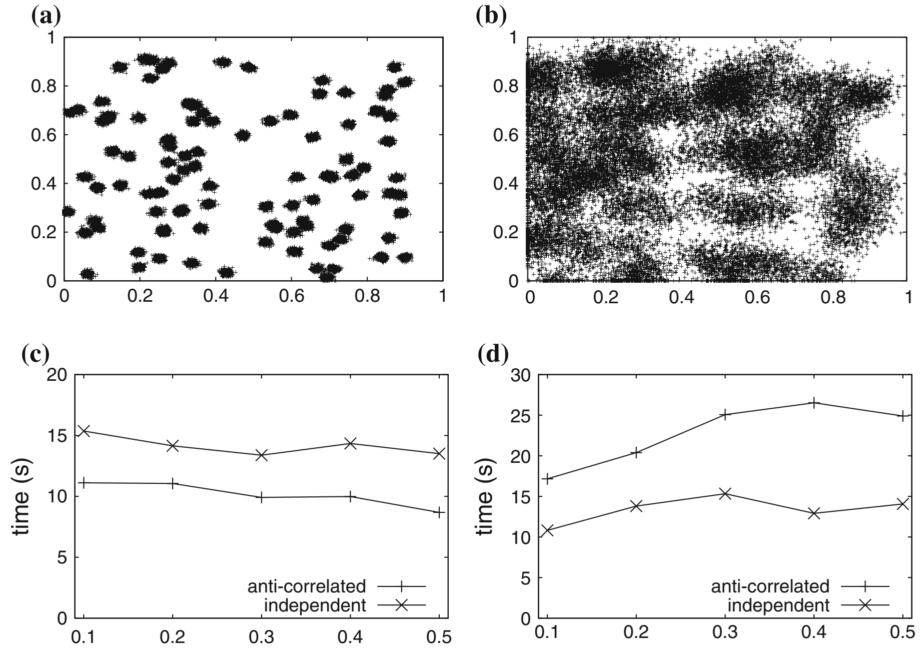


Fig. 10 Effect of data clustering **a** data with $\delta = 0.1$, **b** data with $\delta = 0.5$, **c** iSky, **d** rSky

Figure 10c shows the runtime of *iSky* as δ increases. We observe that *iSky* works better on sparse data sets, and the sharing strategy is slightly less effective on clustered data sets.

5.2 Answering reciprocal skyline queries

A reciprocal skyline query can be computed using the sort-filter-skyline (SFS) algorithm [2]. A straightforward method of computing the reciprocal skylines of all applicants is to apply the sort-filter-skyline algorithm individually for every applicant. We compare our reciprocal skyline algorithm (*rSky* for short) with this non-sharing simple method (referred by NS) as the baseline. Please note that the Linear Elimination Sort for Skyline (LESS) algorithm [3] is the best non-index skyline algorithm in the average case because it has less I/O cost than SFS. The advantage of LESS is that it pushes the skyline computation into external sorting.

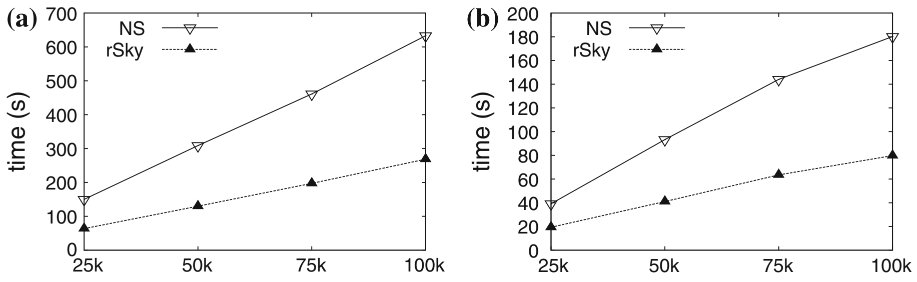


Fig. 11 Effect of cardinality on rSky **a** anti-correlated, **b** independent

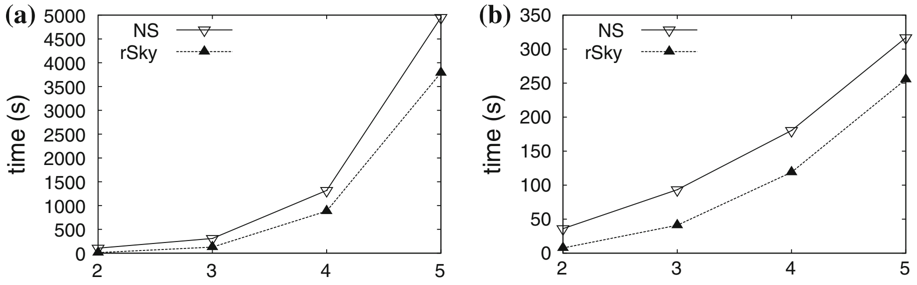


Fig. 12 Effect of dimensionality on rSky **a** anti-correlated, **b** independent

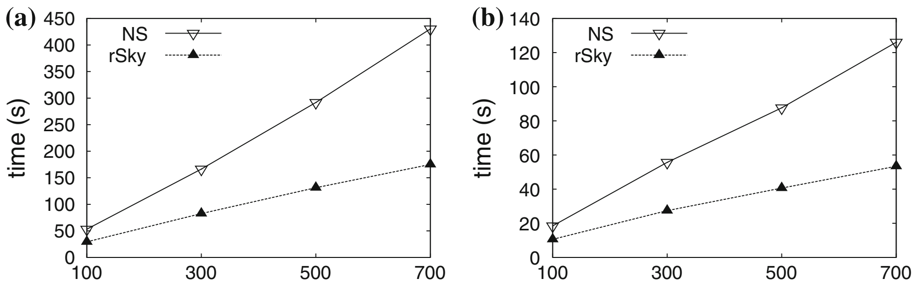


Fig. 13 Effect of view size on rSky **a** anti-correlated, **b** independent

However, the two methods have similar performance if they run in-memory, since the saving is limited when external sorting is not used.

We compare *rSky* and NS in Figs. 11, 12, 13, and 14 as we vary cardinality, dimensionality, view size, and view overlap, respectively. Again, our batch method *rSky* always has a better performance than NS. The running time of both methods rises linearly as the cardinality goes up from 25,000 to 100,000, and rises exponentially as the dimensionality increases from 2 to 5.

Figure 13 shows that the runtime of *rSky* increases more slowly than that of NS as the view size increases. It demonstrates the benefit of the sharing strategy used in *rSky*.

Figure 14a shows that the running time of both algorithms on anti-correlated data sets decreases when the overlap of views decreases (that is, the standard deviation of the view distribution increases from 0.1 to 0.7). When the views are less overlapping, the inverse views become smaller. Hence, the cost of computing the skyline of an inverse view decreases.

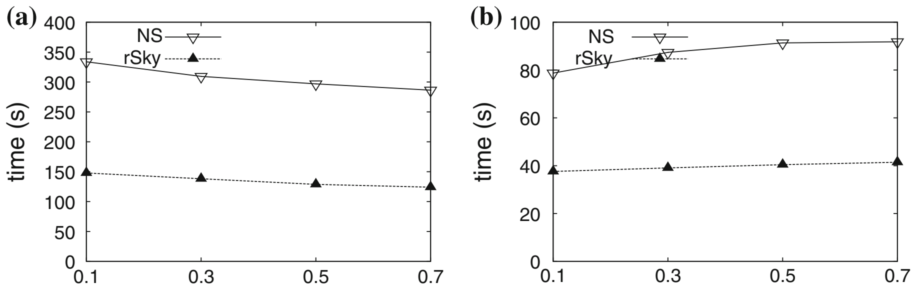


Fig. 14 Effect of view overlap on rSky **a** anti-correlated, **b** independent

This phenomenon is more obvious on anti-correlated data sets since the cost of computing skyline on anti-correlated data sets is much more significant than that on the data sets of the other distributions.

We also show the performance of *rSky* on clustered data sets in Fig. 10d. According to our *rSky* algorithm, the more similar two applicants, the more computation of their *rSky* can be shared. Thus, we see that *rSky* has better performance on more clustered data sets, that is, data sets with smaller δ.

6 Conclusions

In this paper, we studied the problem of making recommendations between two parties in two-way selections. We proposed a series of skyline view queries to make recommendations to all applicants and jobs, and developed several efficient algorithms to answer these queries in batch. The experiment results demonstrate that our algorithms significantly outperform the state-of-the-art methods.

For future work, it is interesting to consider more complex queries using skylines for recommendations between two parties. Moreover, it is interesting to consider more than two parties where parties collaborate and compete simultaneously.

Acknowledgments We are grateful to the anonymous reviewers for their very useful comments and suggestions. Part of this work was done when Jian Chen and Jin Huang visited Simon Fraser University. The work was supported in part by the Fundamental Research Funds for the Central Universities, SCUT(Grant No. 2012ZZ0088), the National Natural Science Foundation of China (61033010), Natural Science Foundation of Guangdong Province (S2011020001182), the Science and Technology Planning Project of Guangdong Province, China (Grant No. 2011A091000036), an NSERC Discovery grant, an NSERC Discovery Accelerator Supplement grant, and a BCFRST Foundation NRAS Endowment Research Team Program grant. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. Börzsönyi S, Kossmann D, Stocker K (2001) The skyline operator. In: Proceedings of the 17th international conference on data engineering. IEEE Computer Society, Washington, DC, USA, pp 421–430
2. Chomicki J, Godfrey P, Gryz J, Liang D (March 2003) Skyline with pre-sorting. In: Proceedings 2003 international conference data engineering (ICDE'03). Bangalore, India, p 717
3. Godfrey P, Shipley R, Gryz J (2005) Maximal vector computation in large data sets. In: VLDB '05: proceedings of the 31st international conference on very large data bases, pp 229–240. VLDB Endowment

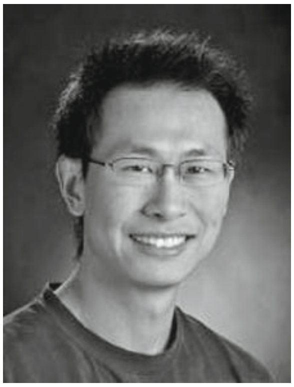
4. Tan K-L, Eng P-K, Ooi BC (2001) Efficient progressive skyline computation. In: VLDB '01: proceedings of the 27th international conference on very large data bases. Morgan Kaufmann Publishers, San Francisco, CA, USA, pp 301–310
5. Kossmann D, Ramsak F, Rost S (2002) Shooting stars in the sky: an online algorithm for skyline queries. In: VLDB '02: proceedings of the 28th international conference on very large data Bases, pp 275–286. VLDB Endowment
6. Papadias D, Tao Y, Fu G, Seeger B (2003) An optimal and progressive algorithm for skyline queries. In: SIGMOD '03: proceedings of the 2003 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 467–478
7. Papadias D, Tao Y, Fu G, Seeger B (2005) Progressive skyline computation in database systems. ACM Trans Database Syst 30(1):41–82
8. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: SIGMOD '84: proceedings of the 1984 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 47–57
9. Chen H, Liu J, Furuse K, Yu JX, Ohbo N (2011) Indexing expensive functions for efficient multi-dimensional similarity search. Knowl Inf Syst 27(2):165–192
10. Pei J, Jin W, Ester M, Tao Y (August 2005) Catching the best views in skyline: a semantic approach. In: Proceedings of the 31th international conference on very large data bases (VLDB'05). Trondheim, Norway
11. Pei J, Fu AWC, Lin X, Wang H (April 2007) Computing compressed skyline cubes efficiently. In: Proceedings of the 23nd international conference on data engineering (ICDE'07). IEEE, Istanbul, Turkey
12. Yuan Y, Lin X, Liu Q, Wang W, Yu JX, Zhang Q (2005) Efficient computation of the skyline cube. In: VLDB '05: proceedings of the 31st international conference on very large data bases, pp 241–252. VLDB Endowment
13. Xia Tian, Zhang Donghui (2006) Refreshing the sky: the compressed skycube with efficient support for frequent updates. In: SIGMOD '06: proceedings of the 2006 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 491–502
14. Tao Y, Xiao X, Pei J (2006) Subsky: efficient computation of skylines in subspaces. In: ICDE '06: proceedings of the 22nd international conference on data engineering. IEEE Computer Society, Washington, DC, USA, p 65
15. Chan C-Y, Jagadish HV, Tan K-L, Tung AKH, Zhenjie Z (2006) Finding k-dominant skylines in high dimensional space. In: SIGMOD '06: proceedings of the 2006 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 503–514
16. Lin X, Yuan Y, Wang W, Lu H (2005) Stabbing the sky: efficient skyline computation over sliding windows. In ICDE '05: proceedings of the 21st international conference on data engineering. IEEE Computer Society, Washington, DC, USA, pp 502–513
17. Tao Y, Papadias D (2006) Maintaining sliding window skylines on data streams. IEEE Trans Knowl Data Eng 18(3):377–391
18. Michael M, Patel JM, Grosky WI (2006) Efficient continuous skyline computation. In: ICDE '06: proceedings of the 22nd international conference on data engineering. IEEE Computer Society, Washington, DC, USA, p 108
19. Sun S, Huang Z, Zhong H, Dai D, Liu H, Li J (2010) Efficient monitoring of skyline queries over distributed data streams. Knowl Inf Syst 25(3):575–606
20. Huang Z, Sun S-L, Wang W (2010) Efficient mining of skyline objects in subspaces over data streams. Knowl Inf Syst 22(2):159–183
21. Jiang B, Pei J (2009) Online interval skyline queries on time series. In: ICDE '09: proceedings of the 2009 IEEE international conference on data engineering. IEEE Computer Society, Washington, DC, USA, pp 1036–1047
22. Balke W-T, Gntzer U, Zheng JX (2004) Efficient distributed skylining for web information systems. In: IN EDBT, pp 256–273
23. Wu P, Zhang C, Feng Y, Zhao BY, Agrawal D, Abbadi AEl (2006) Parallelizing skyline queries for scalable distribution. In: In EDBT06, pp 112–130
24. Huang Z, Jensen CS, Lu H, Ooi BC (2006) Skyline queries against mobile lightweight devices in manets. In: ICDE '06: proceedings of the 22nd international conference on data engineering, IEEE Computer Society, Washington, DC, USA, p 66
25. Sharifzadeh M, Shahabi C (2006) The spatial skyline queries. In: VLDB '06: proceedings of the 32nd international conference on very large data bases, pp 751–762. VLDB Endowment
26. Chan C-Y, Eng P-K, Tan K-L (2005) Stratified computation of skylines with partially-ordered domains. In: SIGMOD '05: proceedings of the 2005 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 203–214

27. Chen L, Lian X (2008) Dynamic skyline queries in metric spaces. In: EDBT '08: proceedings of the 11th international conference on extending database technology. ACM, New York, NY, USA, pp 333–343
28. Pei J, Jiang B, Lin X, Yuan Y (2007) Probabilistic skylines on uncertain data. In: VLDB '07: proceedings of the 33rd international conference on very large data bases, pp 15–26. VLDB Endowment
29. Lian X, Chen L (2008) Monochromatic and bichromatic reverse skyline search over uncertain databases. In: SIGMOD '08: proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 213–226
30. Wong RC-W, Pei J, Fu AW-C, Wang K (2007) Mining favorable facets. In: KDD '07: proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, New York, NY, USA, pp 804–813
31. Jiang B, Pei J, Lin X, Cheung DW, Han J (2008) Mining preferences from superior and inferior examples. In: KDD '08: proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, New York, NY, USA, pp 390–398
32. Dellis E, Seeger B (2007) Efficient computation of reverse skyline queries. In: VLDB '07: proceedings of the 33rd international conference on very large data bases, pp 291–302. VLDB Endowment
33. Wu X, Tao Y, Wong RC-W, Ding L, Yu JX (2009) Finding the influence set through skylines. In: EDBT '09: proceedings of the 12th international conference on extending database technology. ACM, New York, NY, USA, pp 1030–1041
34. Gale D, Shapley LS (1962) College admissions and the stability of marriage. In: American Mathematical Monthly, pp 9–14
35. Gusfield D (1988) The structure of the stable roommate problem: efficient representation and enumeration of all stable assignments. SIAM J Comput 17(4):742–769
36. Iwama K, Miyazaki S, Manlove D, Morita Y (1999) Stable marriage with incomplete lists and ties. In: ICAL '99: proceedings of the 26th international colloquium on automata, languages and programming. Springer, London, UK, pp 443–452
37. Manlove DF (2002) The structure of stable marriage with indifference. Discret Appl Math 122(1-3): 167–181
38. Manlove DF, Irving RW, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1-2):261–279
39. Brinkhoff T, Kriegel H-P, Seeger B (1993) Efficient processing of spatial joins using r-trees. In: SIGMOD '93: proceedings of the 1993 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 237–246
40. Pei J, Jiang B, Lin X, Yuan Y (2007) Probabilistic skylines on uncertain data. In: Proceedings of the 33rd international conference on very large data bases, VLDB '07, pp 15–26. VLDB Endowment
41. Zou L, Chen L (2008) Dominant graph: an efficient indexing structure to answer top-k queries. In: ICDE '08: proceedings of the 2008 IEEE 24th international conference on data engineering. IEEE Computer Society, Washington, DC, USA, pp 536–545
42. Li C, Ooi BC, Tung AKH, Wang S (2006) Dada: a data cube for dominant relationship analysis. In: SIGMOD '06: proceedings of the 2006 ACM SIGMOD international conference on management of data. ACM, New York, NY, USA, pp 659–670

Author Biographies

Jian Chen received her B.S. and Ph.D. degrees, both in Computer Science, from Sun Yat-Sen University, China, in 2000 and 2005 respectively. She is currently an associate professor and director of the Data Mining Group in SSE (SCUT). Her research interests include database, data mining, social networks and their related applications. She has served as a PC member for international conferences such as PAKDD and CIKM.

Jin Huang received his M.E. and Ph.D. degrees, both in Computer Science, from Sun Yat-Sen University, China, in 2004 and 2010 respectively. Currently, he is postdoctoral researcher of South China Normal University, China. His current research interests cover database, data mining and information retrieval.



Bin Jiang is a Research Scientist at Facebook. Bin has published in premier academic journals and conferences. He served as a reviewer for TKDE and in the program committees of several international conferences such as SIGKDD and ICDM. Bin holds a Ph.D. degree in Computer Science from Simon Fraser University, Canada and received his B.Sc. and M.Sc. degrees from Peking University, China and University of New South Wales, Australia, respectively.

Jian Pei is a Professor at the School of Computing Science, Simon Fraser University, Canada. He is interested in researching, developing, and deploying effective and efficient data analysis techniques for novel data intensive applications, including data mining, Web search, data warehousing and online analytic processing, database systems, and their applications in social networks and media, health-informatics, business and bioinformatics. His research has been extensively supported in part by governmental funding agencies and industry partners. He is also active in developing industry relations and collaboration, transferring technologies developed in his group to industry applications, and developing proof-of-concept prototypes. Since 2000, he has published 1 textbook, 2 monographs and over 170 research papers in refereed journals and conferences, which have been cited thousands of times. He has served in the organization committees and the program committees of over 160 international conferences and workshops. He is the associate editor-in-chief of IEEE Transactions of Knowledge and Data Engineering (TKDE), and an associate editor or editorial board member of the premier academic journals in his fields. He is an ACM Distinguished Speaker, and a senior member of ACM and IEEE. He is the recipient of several prestigious awards.

Jian Yin received the B.S., M.S., and Ph.D. degrees from Wuhan University, China, in 1989, 1991, and 1994, respectively, all in computer science. He joined Sun Yat-Sen University in July 1994 and now he is a professor of Information Science and Technology School. He has published more than 100 refereed journal and conference papers. His current research interests are in the areas of Data Mining, Artificial Intelligence, and Machine Learning. He is a senior member of China Computer Federation.