This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier’s archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright
An efficient skyline framework for matchmaking applications

Hyuck Han a, Hyungsoo Jung b,*, Hyeonsang Eom a, H.Y. Yeom a

a School of Computer Science and Engineering, Seoul National University, Seoul 151-742, South Korea
b School of Information Technologies, University of Sydney, NSW 2006, Australia

ABSTRACT

In this article, we present a skyline-based matchmaking framework. The current method of carrying out the matchmaking procedure identifies items based on users’ specifications. We rethink matchmaking procedures in such a way that they can find items that can satisfy a specific computing demand from a user and recommend a collection of better candidates among the identified items. This endows a user with the right of choice on deciding the best-possible items.

We approach the recommendation from the perspective of skyline computation and present an efficient skyline algorithm that gathers interesting item candidates efficiently. To devise an efficient sequential skyline algorithm, we adopt (i) lattice-based indexing using a lattice composition technique and (ii) an optimized dominance-check algorithm. Moreover, we parallelize the algorithm using breadth-first-search (BFS). Our extensive experimental results show that our algorithm outperforms current state-of-the-art algorithms, and the speedup factor of the parallelized algorithm is near-linear.

1. Introduction

In the era of web services, a user typically specifies requirements for interesting items and submits the specification through a matchmaking web service. Then, the matchmaking web service finds appropriate and available candidate items, and returns them to the user. In this context, we have an interesting question about “appropriate” items: which items are the best among thousands or millions of candidates to a user? We believe that candidates that have at least a single comparative advantage over other candidates are more appropriate. However, it is a non-trivial job to find the most appropriate candidates among millions of items.

Example 1 (Item recommendation & skyline computation). Table 1 shows a data set from a hotel reservation service. Each hotel has amenities such as a workout center and a parking lot represented as boolean attributes. The hotel also has price, star rating, and distance from the city center. From the table, users want candidates of hotels that satisfy users’ requirements ($80 \leq \text{Price} \leq 110$ and $0.5 \text{ mile} \leq \text{Distance from City Hall} \leq 1.2 \text{ mile}$). Hawaii Inn is excluded in results since it does not meet users’ requirements. Although both Honolulu Inn and Kalawao Motel satisfy users’ needs, the user prefers Honolulu Inn to Kalawao Motel with the assumption that users prefer hotels with more stars and amenities, lower price, and shorter distance. On the other hand, Hawaii Inn and Maui Motel can be recommended since both hotels are not worse than (not dominated by) other items.

According to the example, recommending better items involves the same work as finding the maximals in a set of vectors (Godfrey et al., 2005), and the maximal vector problem is known as the skyline computation in the database society. The skyline operator that Börzsönyi et al. (2001) introduced is used to find all better tuples instead of vectors that satisfy users’ needs. Thus, the skyline operator is a novel solution for the item recommendation for matchmaking procedures from a theoretical viewpoint.

Definition 1 (Skyline operator Börzsönyi et al. (2001)). Given a set of data points, the skyline operator returns a set of points that are not dominated by other points. A point $p_i$ is said to dominate another point $p_j$ if $p_i$ is better than $p_j$ in at least one dimension and not worse than $p_j$ in other dimensions.

The hotel skyline in Example 1 is composed of Honolulu Inn, Maui Motel and Kauai Sleep. The skyline for the hotel reservation is computed over a data set with multiple low-cardinality attributes (workout center, parking lot, and star rating) and multiple unrestricted attributes (price and distance). In this paper, we focus on the skyline computation over multiple low-cardinality and unrestricted attributes that are common in real-world applications. Morse et al. (2007) addressed the...
low-cardinality characteristics of such applications, and proposed an efficient skyline algorithm called the Lattice Skyline (LS) algorithm. However, the LS algorithm cannot process the skyline computation over data with more than two (multiple) unrestricted attributes.

In the earlier version of this paper (Han et al., 2009), we proposed an efficient sequential algorithm to process skyline computation over data with multiple low-cardinality and unrestricted attributes in real-world applications. The basic idea of the algorithm is to (i) transform multiple low-cardinality domains into a single lattice structure, (ii) construct a set of R-trees (Guttman, 1984) according to the lattice structure, and (iii) compute skylines from the R-trees with additional optimized techniques. We call our algorithm F ast i tem R recommendation SKY line (FIR-SKY).1

In this paper, we extend our algorithm to process skyline queries with users’ criteria (e.g., $80 \leq Price \leq 110$). Then, by using BFS instead of topological sorting, we improve the algorithm to a parallel algorithm, which was not addressed in our previous work. The main contributions of this paper are as follows:

- We extend FIR-SKY to process conditional skyline queries that specify users’ criteria.
- We also improve our sequential algorithm to process skyline queries in a parallel manner. The parallelized algorithm is also progressive.2
- We show that our algorithm outperforms current state-of-the-art algorithms through a performance evaluation, and the speedup factor of the parallelized algorithm is near linear.

The rest of the paper is organized as follows. Section 2 reviews the related work in further detail. Section 3 explains the details of the FIR-SKY algorithm and improvements of FIR-SKY. Section 4 evaluates our algorithms in comparison to current state-of-the-art algorithms. Finally, Section 5 concludes the paper.

2. Related work

A considerable amount of work has been published on the topic of algorithms that compute skyline queries. The previous work can be grouped into two categories, namely non-index-based (e.g., block nested loop Börszönyi et al., 2001, divide and conquer Börszönyi et al., 2001) and sort-based skyline algorithms Bartolini et al., 2008; Godfrey et al., 2005; Chomicki et al., 2003) and index-based (e.g., the B-tree-based scheme Börszönyi et al., 2001, R-tree based schemes Börszönyi et al., 2001; Papadias et al., 2005, bitmap Tan et al., 2001, and index Tan et al., 2001). Typically, the index-based approaches outperform the non-index-based approaches. Index-based approaches can also guarantee to return answers progressively without scanning whole data sets. This property is called “progressiveness” (Tan et al., 2001).

As the number of attributes increases, the number of skyline points generally increases, and this might lead to a burden of matchmaking applications. To mitigate this problem, several techniques to identify representative skyline points such as the skyline frequency (Chan et al., 2006b), x-dominant skyline (Chan et al., 2006a) and strong skyline points (Zhang et al., 2005) have been proposed. Techniques to evaluate skylines in subspace of a given space (i.e., subsets of attributes) are proposed in Yuan et al. (2005) and Pei et al. (2005, 2006), and they consider skylines in multiple subspaces simultaneously to identify better skyline items. In Lee et al. (2008), the MaxPrune framework is proposed to minimize the number of preference elicitation steps for skyline computations over categorical attributes. Thus, owing to minimum preference elicitation from users the framework has a great advantage when the amount of users’ preference is not adequate. In Lofi et al. (2010), the trade-off or compromise concept for skyline computation is first proposed. For example, users are willing to pay an additional $1000 for a car without an air conditioner to install a new air-conditioning. To support the trade-off concept, several algorithms including new indexing and dominance-check algorithms are devised. Lattice Skyline (LS) (Morse et al., 2007) transforms multiple low-cardinality domains into a lattice structure and computes the skyline based on the structure of the lattice. However, the LS algorithm can only process a skyline query with only one unrestricted attribute.

In our previous work (Han et al., 2009), we proposed an efficient sequential skyline algorithm for data with multiple low-cardinality and unrestricted attributes. In Jung et al. (2010), we extended our skyline algorithm to process skyline queries for partially-ordered attributes which generally have more complex structures than low-cardinality attributes. Additionally, the iMinMax-based (Yu et al., 2004) dominance-check algorithm is also proposed. In this paper, we extend our work (Han et al., 2009) to process skyline queries with additional users’ conditions and devise a parallel skyline algorithm to exploit recent popular CPU architectures such as SMP and multi-core processors.

Skyline computation can be also applied to applications of different contexts. In Lee and Teng (2007), skyline computation is used to process multi-criteria rating for recommendation systems. In Personal Preference Search Service (Abel et al., 2007), skyline computation with partially-ordered attributes is used to search the best learning resources. However, no specific skyline algorithm is proposed in either research work. In Kodama et al. (2008), two spatial skyline algorithms based on the nearest neighbor search are proposed for mobile environments. The algorithms use distance information, categorical attributes, and unrestricted attributes to return the best candidates considering users’ current position.

Branch-and-Bound Skyline (BBS) (Papadias et al., 2005), which is based on the R-tree (Guttman, 1984), is IO-optimal and outperforms other proposed algorithms (Kossmann et al., 2002; Börszönyi et al., 2001). Per Algorithm 1, BBS traverses the R-tree

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Example data set.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hotel name</strong></td>
<td><strong>Parking available</strong></td>
</tr>
<tr>
<td>Hawaii Inn</td>
<td>F</td>
</tr>
<tr>
<td>Honolulu Inn</td>
<td>F</td>
</tr>
<tr>
<td>Maui Motel</td>
<td>F</td>
</tr>
<tr>
<td>Kauai Sleep</td>
<td>T</td>
</tr>
<tr>
<td>Kalawao Motel</td>
<td>F</td>
</tr>
</tbody>
</table>

1 In the preliminary version of this paper, our algorithm was called F ast i tem SKY line (FIS-SKY). In this article, we rename our algorithm in order to represent the contents of the paper clearly.
2 Progressiveness means that interesting candidates are returned immediately once they are identified. Thus, progressiveness is a very important property for applications that need a fast initial response.
Fig. 1. An example of BBS (Papadias et al., 2005): (a) R-tree and (b) heap contents \((\text{entry id, } \text{L1 distance} )\).

Algorithm 1. Algorithm BBS.

**Algorithm BBS**

**Input** \(R\) is an R-tree.

**Output** Set of skyline points.

1: Initialize heap \(H\) to empty;
2: Insert all entries in the root node of \(R\) into heap \(H\);
3: \(\text{while } (H \text{ is not empty}) \text{ do} \)
4: Remove top entry \(e\) from \(H\);
5: \(\text{if } (e \text{ is an internal entry}) \text{ then} \)
6: \(\text{if } (e \text{ is not dominated by any entry in } S) \text{ then} \)
7: \(\text{for each child entry } e_i \text{ of } e \text{ do} \)
8: \(\text{if } (e_i \text{ is not dominated by any entry in } S) \text{ then} \)
9: \(\text{Insert } e_i \text{ into } H; \)
10: \(\text{endfor} \)
11: \(\text{else} \)
12: \(S = \text{UpdateSkylines}(e, S); \)
13: \(\text{endwhile} \)
14: \(\text{return } S; \)

**Algorithm UpdateSkylines**

**Input** \(e\) is a data point in some leaf node of an R-tree.

**Output** An updated set \(S\).

1: \(\text{for each } p \in S \text{ do} \)
2: \(\text{if } (e \text{ is dominated by } p) \text{ then} \)
3: \(\text{return } S; \)
4: \(\text{endfor} \)
5: \(\text{Insert } e \text{ into } S; \)
6: \(\text{return } S; \)

There is also a lot of research work that focuses on matchmaking engines for Web Services (OASIS, 2010; Graham, 2001a, 2001b) and Grid Services (Globus Alliance, 2005). The ANSAware Trader (Consortium, 2010) and the CORBA/ODP Trading Service (ISO, 1995; OMG, 2000) can be regarded as an advanced form of a directory. Consumers in the virtual marketplace (Hoffner et al., 2000; Facciorusso et al., 2003) provide a
description of what they want and select candidates from providers. These works do not deal with skyline computation as item recommendation. If such matchmaking applications include skyline computation, it would be better since they have item recommendation as an advanced feature.

In this paper, we focus on skyline computation for data with multiple unrestricted and low-cardinality attributes, which has not been addressed before. To this end, we devise a new skyline algorithm (FIR-SKY) to process skyline queries over such data by extending BBS.

3. FIR-SKY algorithm

Throughout this paper, and without loss of generality, we consider the skyline with the min operator for all attributes. This means that smaller values dominate larger ones. To this end, we map the value \( T \) and \( F \) into 0 and 1 in the case of boolean attributes. In the case of a numeric attribute, if larger values are preferable to smaller ones, we simply transform each value of the attribute into the maximum value of the attribute—the original value. Otherwise, we do not need any transformation.

We first explain the lattice-based indexing technique. Then, we describe the FIR-SKY algorithm that computes the skyline from all tuples, and we explain the way that the FIR-SKY algorithm processes skyline queries with user-specified conditions. Finally, we extend the FIR-SKY algorithm to a parallelized algorithm.

3.1. Lattice-based indexing technique

Before indexing data, we first preprocess all low-cardinality attributes using a lattice. If a relation \( R_i \) has \( n \) attributes (\( n_u \) unrestricted attributes and \( n_l \) low-cardinality attributes), low-cardinality attributes of the relation \( R_i \) are modeled by \( A_1, A_2, \ldots, A_n \).

The unified domain of the lattice LV is the Cartesian-product space of all low-cardinality attributes. Thus, the number of R-trees that is required when maintaining \( n_l \) low-cardinality attributes is the size of \( LV, N_LV \). Our stratification technique does not lead to redundant tuples across R-trees because the lattice value of each tuple is unique. Our method, thanks to the disjointness property of our stratification technique, does not consume more amounts of space than the BBS algorithm, the current state-of-the-art algorithm. Rather, the total space consumed by a set of R-trees in FIR-SKY is less than that of an R-tree in BBS because the dimension of each R-tree in FIR-SKY is not \( n_l \) but \( n_u \).

3.2. FIR-SKY: BFS-based-skyline-computation

Our algorithm is based on a hybrid data structure, a lattice where each node has values from multiple low-cardinality domains and a set of R-trees. All the R-trees have the same unrestricted attributes. Since the lattice structure (Fig. 2(b)) is a kind of directed acyclic graph (DAG), we use a breadth-first search (BFS) (Fig. 3) to determine the visit order of R-trees when data points are extracted from the R-trees for skyline computation.

The BFS order of a DAG is a linear ordering of its nodes, and this linear ordering makes our algorithm progressive.

Property 1. BFS visits each node before all nodes to which it has edges if a given DAG has the lattice structure.

By the definition of BFS, BFS treats neighbors of a vertex \( v_i \) as successors of \( v_i \), where neighbors are pointed by outgoing edges of \( v_i \). BFS begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes until all connected nodes are visited. From Figs. 2(b) and 3, we can verify this property.

Lemma 1. An intermediate skyline point \( p_i \) that is found in the RT, cannot be dominated by any data point in RT, where \( i \) precedes \( j \) in the BFS order.
Algorithm 2. Algorithm Dominance-Check-Over-Relevant-Sets.

Data Structures

\(LV\): lattice values from low-cardinality domains

Algorithm Dominance-Check-Over-Relevant-Sets \((e, \mathcal{S}, v)\)

Input \(e\) is an entry or a data point of an R-tree.

\(\mathcal{S}\) is a set of disjoint groups of intermediate skyline points

\(v\) is the lattice value of \(e\)

Output true if \(e\) is dominated by any intermediate skyline points

false if \(e\) is incomparable to all current skyline points

\(S = \{S_u | u \in LV\}\) is a set of disjoint groups of intermediate skyline points

1: \(RS = \{S_u | u \in LV, S_u \in \mathcal{S}, u = v\ or u\ ancestor\ of\ v\} \)
2: for each \(RS \in RS\) do
3: \quad for each \(p \in RS\) do
4: \quad \quad if \((e\ is\ dominated\ by\ p)\ then\ \//n_r\text{-dimensional comparison
5: \quad \quad return true;
6: \quad endfor
7: endfor
8: return false;

Algorithm 2 describes “Dominance-Check-Over-Relevant-Sets”. It derives relevant skyline sets from all intermediate skyline points (line 1), and then it returns true if any skyline point dominates the entry of an R-tree (line 2–7). Otherwise, it returns false (line 8).

After determining the selection order of the R-trees by using the BFS, FIR-SKY computes a skyline set using the FIR-BBS algorithm. In each stage, FIR-SKY selects an R-tree \(RT_i\) in the BFS order, and performs the skyline computation for each \(RT_i\). By Lemma 1, the order for each \(RT_i\) enables intermediate skyline points from each \(RT_i\) to be returned to a user immediately.

Algorithm 3 sketches the FIR-SKY algorithm. The Scheduler determines the BFS order of all lattice values, and the order represents the selection order of R-trees. The \(i\)-th run of FIR-BBS with \(RT_i\) traverses the R-tree \(RT_i\), then searches for the nearest internal entry or data point(s) that are not dominated by any points of current skyline. If such points are found, they are inserted into the MinHeap \(H\). If the top entry point of the MinHeap is not dominated by any other point of the current skyline set, the top entry point is inserted into the set, and the set is updated. FIR-BBS and UpdateSkylines use the Dominance-Check-Over-Relevant-Sets in order to check whether an internal entry or a data point is dominated by other points of the current skyline set, and this minimizes the comparison time.
Theorem 1. The algorithm FIR-SKY achieves the IO-optimality and progressiveness in skyline computation.\footnote{The IO-optimality means that the skyline query processing requires only one scan of whole tuples.}

Algorithm 3. Algorithm FIR-SKY.

Data Structures

\[ LV : \text{lattice values from low-cardinality domains} \]

Algorithm FIR-SKY \((R_T)\)

Input \( R_T = \{ RT_v \forall v \in LV \} \) is a set of R-trees

Output \( A \) set of skyline points

\[ S = \{ S_v \forall v \in LV \} \text{ is a set of disjoint groups of intermediate skyline points} \]

1: \( I, LO = \text{Scheduler}(RT) \);
2: \( \text{for } i = 1 \text{ to } N_{LV} \) do
3: \( S = \text{FIR-BBS}(I[i], S, LO(i)) \);
4: \text{endfor}
5: \text{return } S

Algorithm Scheduler \((RT)\)

Input \( RT = \{ RT_v \forall v \in LV \} \) is a set of R-trees

Output \( I, LO : \text{a sorted array of R-trees and lattice values via BFS} \)

1: \( LO = \text{BFS}(LV) \);
2: \( \text{for } i = 1 \text{ to } N_{LV} \) do
3: \( I[i] = RT_v, \text{ where } v = LO[i] \in LV \)
4: \text{endfor}
5: \text{return } I, LO

Algorithm FIR-BBS \((RT_v, S, v)\)

Input \( RT_v \) is an R-tree for \( v \in LV \)

Output \( S \) is a set of R-trees and performs the FIR-BBS algorithm.

\[ \text{Process } \{ 0,1 \} \text{ for skyline values } \{ 0,1 \text{,} \} \text{, where } B_v = LO(1) \text{ in Fig. 4(a). Under the condition, the scheduling order is } \{ 0,1,0 \} \rightarrow \{ 0,1,1 \} \rightarrow \{ 1,1,1 \}. \text{ Algorithm 4 gives description of our algorithm for conditional skyline queries with both do not-care and qualifiable conditions.} \]
skyline query over the R-trees of the same node depth in a parallel manner.

**Property 2.** If any two nodes of a given lattice structure have the same depth in the BFS tree of the lattice, both are incomparable to each other.

Let us verify this property by contradiction. That is, we can assume that there exist two nodes of the lattice structure with the same depth in the BFS tree but one of the two nodes is dominated by the other. In a lattice structure, all nodes that have the same depth of the BFS tree share at least one of the same parent nodes. Since the two nodes are siblings, they are not reachable from each other. This means that the two are not dominated by each other, and the contradiction is broken.

**Algorithm 4.** FIR-SKY algorithm for conditional skyline queries.

**Data Structures**

*LV* : lattice values from low-cardinality domains

**Algorithm FIR-SKY**(*RT*)

**Input**

*RT* = {*RT* | v ∈ *LV*} is a set of R-trees

**Output**

A set of skyline points

*S* = {*S* | v ∈ *LV*} is a set of disjoint groups of intermediate skyline points

1: Γ, LO = Scheduler(*RT*);
2: for i = 1 to *N* do
3:  
4:  
5:  
6:  
7:  
8:  
9:  
10: endfor
11: return *S*

**Algorithm Scheduler**(*RT*)

**Input**

*RT* = {*RT* | v ∈ *LV*} is a set of R-trees

**Output**

Γ, LO: a sorted array of R-trees and lattice values via BFS

Scheduler is the same as the Scheduler algorithm in Algorithm 3, except that

(i) in the case of do not-care conditions on unrestricted attributes, the BFS function computes the order excluding the attributes with do not-care conditions, and

(ii) in the case of qualifiable conditions on unrestricted attributes, the BFS function computes the order by using only the attributes values that satisfy qualifiable conditions.

**Algorithm FIR-BBS**(*RT*, *S*, v)

**Input**

*RT* is an R-tree for v ∈ *LV*

*S* is a set of disjoint groups of intermediate skyline points

v is the lattice id of the current iteration

**Output**

*S* is an updated skyline

FIR-BBS is the same as the FIR-BBS algorithm in Algorithm 3, except that

(i) in the case of do not-care conditions on unrestricted attributes, the algorithm computes L1 distance excluding unrestricted attributes, and

(ii) in the case of qualifiable conditions on unrestricted attributes, the algorithm pushes an (internal or leaf) entry that satisfies qualifiable conditions into the MinHeap.

**Algorithm UpdateSkylines**(*e*, *S*, v)

**Input**

*e* is a data point in some leaf node of an R-tree.

*S* is a set of disjoint groups of intermediate skyline points

v is the lattice id of *e*
Output an updated skyline set $S$.

**UpdateSkylines** is the same as the UpdateSkylines algorithm in Algorithm 3.

**Algorithm Dominance-Check-Over-Relevant-Sets**($e,S,v$)

Input $e$ is an entry or a data point of an R-tree.
$S$ is a set of disjoint groups of intermediate skyline points
$v$ is the lattice value of $e$

Output **true** if $e$ is dominated by any intermediate skyline points
**false** if $e$ is incomparable to all current skyline points

**Dominance-Check-Over-Relevant-Sets** is the same as the Dominance-Check-Over-Relevant-Sets algorithm in Algorithm 2, except that
(i) in the case of do not-care conditions on unrestricted attributes, the algorithm examines an entry of an R-tree excluding unrestricted attributes, and
(ii) in the case of qualifiable conditions on unrestricted attributes, the algorithm checks whether an (internal or leaf) entry satisfies qualifiable conditions as well as whether the entry is dominated.

From Fig. 6(b), we can see that this property is correct. For example, the node depth of $(2,20)$ and $(3,10)$ is 2 and the both nodes are incomparable to each other.

From this property, parallel invocation of the **FIR-BBS** procedure in Algorithm 3 is allowed.

**Lemma 2.** Any data point in $RT_i$ cannot be dominated by any data point in $RT_j$, where $i$ and $j$ nodes of a given lattice structure have the same depth in the BFS tree.

**Proof.** We prove the lemma by contradiction. Assume that an intermediate skyline point $p_x \in RT_i$ is dominated by a data point $p_y \in RT_j$. If $p_x$ is dominated by $p_y$, $i$ is dominated by $j$. By Property 2, this dominance implies that the depth of $i$ node is different from that of $j$ node in the BFS tree. This contradicts the assumption, and we then complete the proof.

**Example 4.** In the scheduling order of Fig. 3, we can see that the skyline computation on $RT_{(1,20)}$ and $RT_{(2,10)}$ is parallelizable since $(1,20)$ is incomparable to $(2,10)$. And, the depth of the node $(1,20)$ in Fig. 6 is the same as that of the node $(2,10)$.

**Algorithm 5.** Parallel **FIR-SKY** algorithm.

**Data Structures**

$LV$: lattice values from low-cardinality domains

$HT$ is the height of the BFS tree.
Algorithm FIR–SKY($RT$)

Input $RT = \{RT_i | v \in LV\}$ is a set of R-trees
Output A set of skyline points $S = \{S_i | v \in LV\}$ is a set of disjoint groups of intermediate skyline points

1: $\Gamma, LO = \text{Scheduler}(RT)$;
2: for $i = 1$ to $HT$ do
3: for all $j$ in parallel do
4: $S = \text{FIR–BBS}(\Gamma, LO, j, S, LO(j))$;
5: endfor
6: endfor
7: return $S$

Algorithm Scheduler($RT$)

Input $RT = \{RT_i | v \in LV\}$ is a set of R-trees
Output $\Gamma, LO$: a sorted array of stratified R-trees and lattice values

1: $LO = \text{BFS\_with\_classification}(LV)$;
2: for $i = 1$ to $N_{LV}$ do
3: $ht_i = \text{the depth of LO}[i]$
4: Append $RT_i$ to $\Gamma[ht_i]$
5: endfor
6: return $\Gamma, LO$

FIR–BBS and Algorithm UpdateSkylines are the same as those in Algorithm 3.

Based on Lemma 2, we modify the Scheduler and the FIR-SKY procedures in Algorithm 3 to execute the FIR-BBS procedure in parallel. Algorithm 5 sketches the parallel FIR-SKY algorithm. The Scheduler stratifies the BFS tree according to the depth of each node. Fig. 6 (c) shows an example of such classification. The FIR–SKY invokes the FIR–BBS procedure to be executed concurrently per each stratum. Note that the total number of strata is the same as the height of the BFS tree (see Fig. 6).

Theorem 2. The parallel FIR–SKY algorithm also achieves the IO-optimality and progressiveness in skyline computation.

Proof. By Lemmas 1 and 2, there does not exist an intermediate skyline point that can be removed from existing skyline sets. This guarantees the progressiveness of the parallel FIR-SKY algorithm. Since the parallel FIR–SKY also executes exact-one-invocation of FIR–BBS on each R-tree, the parallel FIR-SKY does not read the same node twice from the disk. □

4. Performance evaluation

In this section, we give performance results detailing the efficiency and progressiveness of our algorithms. Section 4.1 describes the experimental environment. Sections 4.2 and 4.3 explain the details of experimental results. Section 4.4 discusses the usability of our algorithms.

4.1. Experimental environment

The experiments were performed on a workstation with two quad-core processors and 8 GB of main memory, which was running the SUSE Linux operating system. We used synthetic data sets described in Börzsönyi et al. (2001). We have compared our algorithm to BBS and SaLSa (for Sort and Limit Skyline algorithm). BBS and SaLSa are known as the best index-based solution (Papadias et al., 2005) and the best sort-based algorithm (Bartolini et al., 2008), respectively. BBS uses an R-tree to index the input data. When skyline computation is performed over the R-tree index, BBS finds the nearest neighbor to origin, prunes search space and repeats find the nearest neighbor in unpruned space. SaLSa is a skyline algorithm that pre-sorts the input data in order to effectively limit the number of tuples to be read and compared. SaLSa reads one record from the pre-sorted input data, verifies if the record can be included in the skyline set. All algorithms used in the experiments have been implemented based on the Java version of the spatial index library (Hadjieleftheriou, 2010) and Apache Axis (Apache Web Services Project, 2010). Table 2 shows the parameters and values used in the experiments. It is noted that throughout this section an unrestricted attribute and low-cardinality attribute is abbreviated to $U$ and $L$, respectively.

In unrestricted attributes, values whose domain is (0.0, 1000.0] were generated by our synthetic data generator described in Börzsönyi et al. (2001). We conducted the experiments by varying the number of data, the number of attributes, and the structure of a lattice. It is noted that due to similar behaviors we skip results of conditional skyline queries and correlated distribution cases.

4.2. IO and computation efficiency

We evaluate the performance of FIR-SKY, measuring IO and computation time by varying the number of data points. We then break the measured time down into IO and computation time. From Figs. 7 and 8, we can see that FIR-SKY outperforms BBS and SaLSa. Table 3 shows the number of skyline in both experiments. From the table, a larger number of data points generates more skyline points and this leads to more computation time.

The dimension of R-tree in BBS is $n_u$, while that in FIR-SKY is $n_{ar}$. Thus, the difference of the dimensionality between BBS and FIR-SKY is 4 or 5 in cases of Figs. 7 and 8. The lattice-based indexing method mitigates the effect of “the curse of dimensionality” problem, and leads to a lower height of R-trees in FIR-SKY than BBS. The lower dimension and height of R-trees in FIR-SKY incur lower IO overhead than BBS. FIR-SKY improves IO performance by 10 and 12 times in 4 and 5 unrestricted attributes cases, respectively.

Figs. 7 and 8 also show that FIR-SKY has shorter computations time than BBS and SaLSa. FIR-SKY uses Dominance-Check-Over-Relevant-Sets, and Dominance-Check-Over-Relevant-Sets is based on comparisons to only relevant skyline points and $n_{ar}$-dimensional comparisons. The two main advantages of Dominance-Check-Over-Relevant-Sets lead to the superiority of FIR-SKY. Due to superior computation efficiency, FIR-SKY computes skylines 7 and 4 times (15 and 8 times) faster than BBS (SaLSa) in 4 and 5

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td># of unrestricted attributes</td>
<td>2–6</td>
</tr>
<tr>
<td># of low-cardinality attributes</td>
<td>4–5</td>
</tr>
<tr>
<td>Data size ($\times 10^k$)</td>
<td>1–128</td>
</tr>
<tr>
<td>Lattice size ($N_L$)</td>
<td>625, 1024</td>
</tr>
<tr>
<td>Attribute correlation</td>
<td>Uniform, anti-correlated</td>
</tr>
<tr>
<td>Disk page size and Node size of an R-tree</td>
<td>4KB</td>
</tr>
</tbody>
</table>

Table 2
Experimental parameters and values.

---

5 The sorting function of SaLSa is $\min C$ (minimum coordinate sort) since in Bartolini et al. (2008) the $\min C$ sorting function is known to outperform other sorting functions.

6 In this article, IO time means time for R-tree or any index operations. However, SaLSa does not have any index structure. Thus, we do not present IO time for SaLSa. In our implementation for experiments, sorted data for SaLSa reside in main memory.
unrestricted attributes cases in uniform distribution cases, respectively. In anti-correlated cases, FIR-SKY processes skyline queries 7 times (36 times) faster than BBS (SaLSa).

From the observations, we can conclude that FIR-SKY computes the skyline much faster than BBS and SaLSa. Moreover, FIR-SKY is a more scalable algorithm than BBS and SaLSa since FIR-SKY shows stable performance even though the number of tuples increases. Note that FIR-SKY behaves similarly on processing conditional skyline queries and processing skyline queries over data sets with other distributions.

Discussion: Some might doubt the scalability of our approach; a set of R-trees in FIR-SKY appears to consume huge amounts of space. Our stratification technique does not lead to redundant tuples across trees because of the disjointedness property of the stratification. Rather, our approach does not store values of low-cardinality attributes since each R-tree in FIR-SKY is built for each value of lattice values. The total space consumed by a set of R-trees in FIR-SKY, therefore, is less than that of an R-tree in BBS.

In summary, FIR-SKY improves the IO and computation efficiency through the lattice-based indexing and optimized dominance check technique. The lattice-based indexing technique achieves IO efficiency since values of low-cardinality domains are not stored and this mitigates the effect of “the curse of dimensionality” problem. The optimized dominance check technique improves the computation efficiency greatly because the dominance check operation involves relevant intermediate skyline points rather than all intermediate skyline points.

4.3. Other experiments

Effects of increasing low-cardinality attributes: Figs. 9 and 10 show IO and computation time of FIR-SKY when the number of low-cardinality attributes increases. As the number of low-cardinality attributes increases, the probability that a point will not be dominated by existing skyline points also increases. From the analysis in Buchta (1989), we can verify this from theoretical viewpoints. Thus, the increase of low-cardinality attributes leads to more skyline points as shown in Table 4. More skyline points mean more invocations of dominance check operations, and our optimized technique enables FIR-SKY to finish the skyline computation faster than BBS and SaLSa.

As the difference of the dimensionality between BBS and FIR-SKY increases, the difference in IO time between BBS and FIR-SKY increases. From Figs. 9(a), (b), and 10(a), we can see a big difference in IO time between BBS and FIR-SKY in 4 and 5 unrestricted attributes cases (13 and 18 times better than BBS).

Table 3

The size of skyline: (A) case of 4U & 4L, (B) case of 5U & 4L, and (C) case of 2U & 4L.

<table>
<thead>
<tr>
<th># of data points (× 10k)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1020</td>
<td>1436</td>
<td>2046</td>
<td>2838</td>
<td>3500</td>
<td>4511</td>
<td>5162</td>
<td>6111</td>
</tr>
<tr>
<td>B</td>
<td>1910</td>
<td>2691</td>
<td>3885</td>
<td>5450</td>
<td>7157</td>
<td>9778</td>
<td>12,328</td>
<td>16,440</td>
</tr>
<tr>
<td>C</td>
<td>1161</td>
<td>1684</td>
<td>2210</td>
<td>3011</td>
<td>4263</td>
<td>6211</td>
<td>8211</td>
<td>12,225</td>
</tr>
</tbody>
</table>

Fig. 7. Results of 4U & 4L and 5U & 4L cases (seconds, uniform distribution): (a) IO time (4U & 4L); (b) IO time (5U & 4L); (c) computation time (4U & 4L); and (d) computation time (5U & 4L).

Fig. 8. Results of 2U & 4L (seconds, anti-correlated): (a) IO time and (b) computation time.
and this difference is much bigger than that in Figs. 7(a), (b), and 8(a). From Figs. 9(c) and (d), we can see that FIR-SKY computes skylines 10 and 11 times (25 and 21 times) faster than BBS (SaLSa) in 4 and 5 unrestricted attributes cases, respectively. In the anti-correlated case, FIR-SKY processes skyline queries 18 times (73 times) faster than BBS (SaLSa).

Table 5 shows pre-processing time of FIR-SKY. Since the pre-processing procedure in FIR-SKY is only BFS, the pre-processing time of FIR-SKY lies within a small range.

Effects of increasing unrestricted attributes: Figs. 11 and 12, and Table 6 show the results when the number of unrestricted attributes increases. A large number of unrestricted attributes increases the size of a skyline set, due to the high probability that a data point will become a skyline point. In both cases of 4 and 5 low-cardinality attributes, the difference in IO time between FIR-SKY and BBS gets larger (16–20 times in 4 and 5 low-cardinality attributes) than that in the results of the previous section. From Figs. 11(c) and (d), we can see that FIR-SKY computes skylines 4.5 and 6.7 times (7 and 10 times) faster than BBS (SaLSa) in 4 and 5 low-cardinality attributes cases, respectively. In the anti-correlated case, FIR-SKY processes skyline queries 16 times (21 times) faster than BBS (SaLSa).

It is noted that due to similar behaviors we report only results of uniform distribution in following experiments.

Progressiveness: Fig. 13 shows how fast each algorithm returns all answers progressively. To this end, we recorded the time it took to output each portion of the answers. In BBS and SaLSa, the time slope rises more steeply than that of FIR-SKY because of less efficient computation. In contrast, FIR-SKY shows more efficient behavior due to the optimized dominance check. With this observation, we can validate the efficiency of FIR-SKY in progressive skyline computation.

Effect of optimized dominance check: Fig. 14 shows the computation time of FIR-SKY with and without the optimized dominance check operation. FIR-SKY+ and FIR-SKY- represent FIR-SKY implementations with and without “Dominance-Check-Over-Relevant-Sets” algorithm. In both figures, FIR-SKY+ outperforms FIR-SKY- by 9 and 12 times since FIR-SKY- have less pruning effect than FIR-SKY+. These results validate the excellence of
Effect of parallelization: For this experiment, we adopt the master-worker style to implement the parallelized FIR-SKY algorithm. In each trial, a fixed number of threads are created beforehand. For a given depth in the BFS tree, all nodes with the same depth are inserted into an atomic queue on which operations are atomic. Each thread consumes a queue item and executes skyline computation over the corresponding R-tree.
All threads iteratively perform such procedure until the queue is empty. When the queue is empty and all threads are suspended, all nodes with the following depth are inserted into the queue and the above procedure is repeated.

Fig. 15 shows the computation time of FIR-SKY with increasing the number of threads. From the figure, we can see that our parallelization technique based on the BBS with stratification is very effective. In the “8 threads” case, the speedup factor (\(=\frac{T_s}{T_p}\)) is 7.9 (nearly 8). Generally, when the number of low-cardinality domains increases, the number of lattice values per a stratum increases, and this enables parallel invocation of FIR-BBS procedures. From this observation, even heavy skyline queries can be performed efficiently by parallel FIR-SKY.

4.4. Discussion

It is certain that as the number of attributes or data items increases the skyline size increases to the point of burdening item recommendation applications. In this sense, skyline computation might be not appropriate for item recommendation due to a large set of skyline points. To mitigate this problem (reduce the skyline set), various research work is performed to identify more important skyline points. For example, the skyline frequency (Chan et al., 2006b), \(k\)-dominant skyline (Chan et al., 2006a), and strong skyline (Zhang et al., 2005) algorithms identify more important skyline points by counting in how many subspace skylines each skyline item is present or by evaluating skyline queries in subsets of attributes. In this sense, our work has a shortcoming since our algorithms cannot catch the most relevant skyline points by using various metrics. In fact, our work does not focus on recommending the most relevant ones of skyline points to the user.

In Papadias et al. (2005), BBS supports \(k\)-dominating skyline query, which retrieves the \(k\) points that dominate the largest number of other points. The basic idea of \(k\)-dominating skyline computation is to (i) first compute the skyline by using BBS and (ii) apply a query point in the R-tree for each skyline point \(p\) and count the number of points falling inside the dominance region of \(p\). It is noted that R-tree is well suitable for such counting operations. We slightly modify the above procedure so that FIR-SKY can process \(k\)-dominating skyline queries. For the first step, FIR-SKY computes all skyline points. Then, all descendant R-trees are identified based on the lattice value of a given skyline point \(p\), and the counting procedure is repeatedly performed on the identified R-trees. For example, counting operations are executed over \(RT_{1,20}\), \(RT_{2,20}\), and \(RT_{3,20}\) for a skyline point from \(RT_{1,20}\) in Fig. 2 (b). Thus, FIR-SKY can also support the \(k\)-dominating query.

Like the \(k\)-dominating query, FIR-SKY can help other skyline algorithms such as \(k\)-dominant skyline and subspace skyline algorithms which are generally computationally intensive. In Yuan et al. (2005), if \(\mathcal{U} \subset \mathcal{V}\) for two different subsets of attributes, the skyline of \(\mathcal{U}\) is included in the skyline of \(\mathcal{V}\) with the assumption that any two data points do not have same value on the same dimension. FIR-SKY first computes all skyline points for all attributes. Then, subspace skyline frameworks compute more important points among the identified skyline points instead of all points. This reduces computation overhead of subspace skyline algorithms owing to small input data. It is noted that the above procedure is not valid if the number of attributes is extremely large to the extent that all points are skyline points.

To increase users’ satisfaction, it is important to support various orderings of low-cardinality attributes. In comparison with BBS and SaLSa, FIR-SKY has an advantage when order relationship of low-cardinality attributes changes. For example, let us assume that at first “blue” and “red” are mapped into 0 and 1, respectively. This means that a user prefer blue items to red items. Then, the user’s preference is changed inversely later. In such cases, FIR-SKY only re-constructs the lattice structure to reflect the changed relationship and rest of skyline computation can be processed without any changes. However, all data items should be re-indexed and re-sorted in BBS and SaLSa, and this leads to large overhead.

We evaluate this property by conducting a simple experiment. For this experiment, we changed the order relation of a given low-cardinality attribute randomly during skyline computation. Since the BBS and SaLSa algorithms cannot adapt to such change without rebuilding whole R-tree and resorting all data items, they are allowed to compute a skyline set only with the previous
structure. In contrast to BBS and SaLSa, FIR-SKY can compute a skyline set simply by re-computing the BFS scheduling order for affected values. We ran FIR-SKY three times, and the sizes of skyline sets in all trials were 4524, 4619, and 4567. Additionally, the counts of the skyline points that BBS and SaLSa could not detect were 295, 317, and 128, respectively.

Similarly, FIR-SKY can support personalized preference of low-cardinality attributes for skyline computation in that FIR-SKY constructs the lattice structure based on the preference of each user. However, it is non-trivial to support personalized preference in BBS and SaLSa. We believe that these properties are important since user’s preference may change many times and users’ preference may differ from other’s preference in the item recommendation context.

Based on our observation, FIR-SKY has both advantages and disadvantages in terms of user satisfaction. FIR-SKY cannot limit skyline points by using various metrics, and this might lead to burden of item recommendation applications. However, FIR-SKY can support the F-dominating skyline query. Moreover, FIR-SKY supports various users’ preferences without large computational overhead.

5. Conclusion

We have presented the FIR-SKY algorithm for item matchmaking applications. In this study, items are modeled on data with multiple low-cardinality and unrestricted attributes. To compute a skyline query efficiently, FIR-SKY adopts lattice-based indexing, and uses BFS to exploit the data characteristics. The lattice-based indexing method incurs low IO overhead, and the BFS scheduling guarantees progressiveness. By the indexing and the scheduling, we design an optimized dominance-check (Dominance-Check-Over-Relevant-Sets) that enables the FIR-SKY algorithm to have much better performance than state-of-the-art skyline algorithms. Then, we extend our algorithm to process conditional queries and improve the algorithm to execute FIR-BBS in parallel.

Acknowledgement

This work was supported by the National Research Foundation (NRF) grant funded by the Korea government (MEST) (No. 2010-0014387), The ICT at Seoul National University provided research facilities for this study.

References

Buchta C. On the average number of maxima in a set of vectors. Inf Process Let 1985;33(2).

Glooscore Alliance. GT Information Services: Monitoring & Discovery System (MDS); 2005.
Graham S. The role of private UDDI nodes in web services, part 1: six species of UDDI. IBM Emerging Internet Technologies; 2001a.
Grall G. The role of services, part 2: nodes and operator nodes. IBM Emerging Internet Technologies; 2001b.
ISO. Open distributed processing reference model; 1995.
OMG. CORBA trading object service; 2000.