Finding Top-k Preferable Products

Yu Peng, Raymond Chi-Wing Wong, and Qian Wan

Abstract—The importance of dominance and skyline analysis has been well recognized in multicriteria decision-making applications. Most previous studies focus on how to help customers find a set of "best" possible products from a pool of given products. In this paper, we identify an interesting problem, finding top-k preferable products, which has not been studied before. Given a set of products in the existing market, we want to find a set of k "best" possible products such that these new products are not dominated by the products in the existing market. We study two problem instances of finding top-k preferable products. In the first problem instance, we need to set the prices of these products such that the total profit is maximized. We refer such products as top-k profitable products. In the second problem instance, we want to find k products such that these k products can attract the greatest number of customers. We refer these products as top-k products. In both problem instances, a straightforward solution is to enumerate all possible subsets of size k and find the subset which gives the greatest profit (for the first problem instance) or attracts the greatest number of customers (for the second problem instance). However, there are an exponential number of possible subsets. In this paper, we propose solutions to find the top-k profitable products and the top-k popular products efficiently. An extensive performance study using both synthetic and real data sets is reported to verify the effectiveness and efficiency of proposed algorithms.

Index Terms—Skyline, spatial database.

1 Introduction

Dominance analysis is important in many multicriteria decision-making applications.

Example 1 (Skyline). Consider that a customer is looking for a vacation package to Hannover using some travel agencies like Expedia.com and Priceline.com. The customer uses two criteria for choosing a package, namely *price* and *distance-to-beach*, where *price* is the price of a package and *distance-to-beach* is the distance between a hotel in a package and a beach. For two packages p and q, if p is better than q in at least one factor, and is not worse than q in any other factors, then p is said to *dominate* q. Table 1 shows four packages: p_1, p_2, p_3, and p_4. We know that a lower price and a shorter distance-to-beach are more preferable. Thus, p_3 dominates p_4 because p_3 has a lower price and a shorter distance-to-beach than p_4. However, p_3 does not dominate p_1 because p_1 has a lower price than p_3. Similarly, p_1 does not dominate p_3 because p_3 has a shorter distance-to-beach.

A package which is not dominated by any other packages is said to be a *skyline package* or it is in the *skyline*. Recently, skyline analysis has received a lot of interest in the literature [16], [10], [21], [7], [14], [24]. The packages in the skyline are the best possible tradeoffs between the two factors in question. In Example 1, p_3 is in the skyline but p_4 is not.

Consider that a new travel agency wants to start some new packages from a pool of potential packages as shown in Table 2. Table 2 shows three potential packages, namely q_1, q_2, and q_3. In this table, attribute distance-to-beach and attribute cost of each package are given. However, attribute price is to be determined by the agency.

Example 2 (Profitable price with one package). Suppose that we select only one new package, say q_1. What price should we set for package q_1? If we set the price of q_1 to be \$100, since the cost of q_1 is \$100, the *profit* of q_1 is equal to $\$100 - \$100 = \$0$. In other words, we cannot earn any profit. If we set the price to be \$400, although we can earn $\$400 - \$100 = \$300$, this new package q_1 is dominated by p_2 in the existing market. In other words, it is likely that no customer will select q_1 since p_2 is better than q_1. However, if we set the price to be \$300, not only can we earn $\$300 - \$100 = \$200$ but also q_1 is not dominated by any packages in the existing market. We say that \$300 is a *profitable* price of q_1 but \$100 and \$400 are not profitable prices of q_1.

Let us consider another example that we want to select only one new package q_2 (instead of q_1). Similarly, if we set the price to be \$200, the profit is \$0. If we set the price to be \$400, q_2 is dominated by p_2. However, if we set the price to be \$300, we can earn \$100 and q_2 is not dominated by any packages in the existing market. Thus, \$300 is a profitable price of q_2 but \$200 and \$400 are not. □

Unfortunately, how we set the price of a new package may affect how we set the price of another new package.

Example 3 (Profitable price with two packages). Suppose that we are interested in selecting two new packages, says q_1 and q_2, instead of only one new package. From

- *Y. Peng and R.C.-W. Wong are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: {gracepy, raywong}@cse.ust.hk.*
- *Q. Wan is with the Computer Science Department, University of Wisconsin-Madison, Room 3387, 1210 W. Dayton St., Madison, WI 53706. E-mail: qwan@cs.uwm.edu.*

Manuscript received 1 June 2011; revised 28 Nov. 2011; accepted 26 Jan. 2012; published online 2 Mar. 2012.
Recommended for acceptance by S. Abiteboul, C. Koch, K.-L. Tan, and J. Pei.
For information on obtaining reprints of this article, please sende-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2011-06-0314.
Digital Object Identifier no. 10.1109/TKDE.2012.52.

TABLE 1
Packages in the Existing Market

Package	*Distance-to-beach (km)*	*Price*
p_1	*7.0*	*200*
p_2	*4.0*	*350*
p_3	*1.0*	*500*
p_4	*3.0*	*600*

Example 1, if we set both the price of q_1 and the price of q_2 to be $300 *separately*, then we can earn some profits and they are not dominated by any packages in the existing market. However, after we set these prices, the new package q_1 is dominated by another new package q_2. An alternative price setting/assignment is that the prices of q_1 and q_2 are set to $250 and $300, respectively. In this assignment, it is easy to verify that q_1 (q_2) is not dominated by not only any packages in the existing market but also another new package q_2 (q_1). Besides, the profits of q_1 and q_2 are $150 and $100, respectively. The sum of these profits is equal to $250. □

From the above example, we learn that how we set the price of a new package may affect how we set the price of another new package. Let Q be the set of potential new packages. In general, we want to select k packages from Q where k is a positive integer and is an input parameter. For example, k is equal to 2 in Example 3. We denote the set of these selected packages by Q'. Let $F(Q')$ be a utility function on Q' which returns a real number. Different sets for Q' can give different values of F. If the value of F is larger, then the set for Q' is more *preferable*. One example of F is a function which returns the sum of the profits of all packages in Q' as illustrated in Example 3.

In this paper, we study the following problem: given a set P of packages in the existing market and a set Q of potential new packages, we want to select a set Q' of k packages from Q such that $F(Q')$ is maximized and each selected package is not dominated by any packages in the existing market and any selected new packages. We call this problem *finding top-k preferable products*.

Setting different utility functions in this problem gives different problem instances. In this paper, we study two instances of the problem with two different utility functions. The first instance is called *finding top-k profitable products* (TPP) (Section 1.1) where the utility function on Q' is set to the function returning the sum of the profits of all packages in Q'. The second instance is called *finding top-k popular products* (Section 1.2) where the utility function on Q' is set to the function returning the total number of customers who are interested in some packages in Q' when customers' preferences are available.

1.1 Finding Top-k Profitable Products

The first instance is called *finding* t*op-k* p*rofitable* p*roducts*. A naive way for this instance/problem is to enumerate all possible subsets of size k from Q, calculate the sum of the profits of each possible subset, and choose the subset with the greatest sum. However, this approach is not scalable because there are an exponential number of all possible subsets. This motivates us to propose efficient algorithms for problem TPP.

TABLE 2
Potential Packages in the New Travel Agency

Package	*Distance-to-beach (km)*	*Price*	*Cost*
q_1	*5.0*	*?*	*100*
q_2	*4.5*	*?*	*200*
q_3	*0.5*	*?*	*400*

Although how we set the price of a new package may affect how we set the price of another new package and there are an exponential number of possible subsets, interestingly, we propose a *dynamic programming* approach which finds an *optimal* solution when there are two attributes to be considered. But, we show that this problem is NP-hard when there are more than two attributes to be considered. Thus, we propose two greedy algorithms for this problem. One greedy algorithm has a theoretical guarantee on the profit returned while the other greedy algorithm performs well empirically.

Finding top-k profitable products is common in many real-life applications. Other applications include finding profitable laptops in a new laptop company, finding profitable delivery services in a new cargo delivery company and finding profitable e-advertisements in a webpage.

In some cases, data sets are *dynamic* and change from time to time. In this paper, we also study how to find top-k profitable products when data sets change. For example, some new products are launched in the existing market while some products which were present in the existing market become unavailable. Besides, the prices of existing products in the market may change due to various reasons, such as inflation and cost increase.

1.2 Finding Top-k Popular Products

The second instance is called *finding top-k popular products*. In some cases, if we know how many customers are interested in some potential products, we can better find potential products. One well-known application which allows customers to provide their preferences is "Name Your Own Price" developed by "Priceline.com." If customers indicate their preferences on some hotels, "Name Your Own Price" service will return some potential hotels to customers.

Similarly, a naive way is to enumerate all possible subsets of size k from Q, calculate the total number of customers interested in some packages in this subset, and choose the subset with the greatest number of customers. But, it is not scalable. We show that this problem is NP-hard. But, interestingly, we propose a 0.63-approximate algorithm which runs in polynomial time.

Our contributions are summarized as follows: first, to the best of our knowledge, we are the first to study how to find top-k preferable products. Finding top-k preferable products can help the effort of companies to find a subset of products together with their corresponding profitable prices, which cannot be addressed by existing methods. Second, for the first problem of finding top-k profitable products, we propose a dynamic programming approach which can find an optimal solution when there are two attributes to be considered. We show that this problem is NP-hard when there are more than two attributes. Thus, we propose two greedy approaches to solve the problem efficiently. Third, we also propose an incremental approach for the first problem when data sets change over time. Fourth, we show that the second problem of finding top-k popular products is NP-hard and propose a 0.63-approximate algorithm for this problem. Fifth, we present a systematic performance study using both real and synthetic data sets to verify the effectiveness and the efficiency of our proposed approaches. The experimental results show that finding top-k profitable products and top-k popular products is interesting.

The rest of the paper is organized as follows: in Section 2, we formally define the first problem instance. In Sections 3 to 5, we discuss algorithms and issues for the first problem instance. Specifically, in Section 3, we describe an algorithm for finding the sum of the profits given a set of k selected packages. The proposed approaches are presented in Section 4. In Section 5, we give a discussion on the proposed approaches for finding top-k profitable products. In Section 6, we discuss how to find top-k profitable products when data change over time. In Section 7, we address the second problem instance and describe how to find top-k popular products when customer preferences are given. A systematic performance study is reported in Section 8. Related work is given in Section 9. The paper is concluded in Section 10.

2 Problem Definition

The *skyline* of a given data set $\mathcal{D}$ is denoted by $SKY(\mathcal{D})$. We have a set P of m tuples in the existing market, namely $p_1, p_2, \ldots, p_m$. Each tuple p has l attributes, namely $A_1, A_2, \ldots, A_l$. The domain of each attribute is $\mathbb{R}$ where a smaller value is more preferable. The value of attribute A_j for tuple p is given and is denoted by $p.A_j$ where $j \in [1, l]$. In particular, the last attribute A_l represents attribute price and all other attributes represent the attributes other than price. Besides, we have a set Q of n potential new tuples, namely $q_1, q_2, \ldots, q_n$. Similarly, each tuple q has the same l attributes, namely $A_1, A_2, \ldots, A_l$. The value of attribute A_j for tuple q is denoted by $q.A_j$ where $j \in [1, l]$. However, the value of attribute A_l for tuple q is not given and the value of each of the other attributes is given. We assume that no two potential new tuples in Q are identical (i.e., no two tuples in Q have the same attribute values for $A_1, A_2, \ldots, A_{l-1}$). In addition to these l attributes, each tuple q is associated with one additional cost attribute C. The value of attribute C for q is denoted by $q.C$. We assume that for any two tuples in $P \cup Q$, they have at least one attribute value different among the first $l-1$ attributes. This assumption allows us to avoid several complicated, yet uninteresting, "boundary" cases. If this assumption does not hold, the proposed algorithms can be modified accordingly.

In our example, P contains four tuples, namely p_1, p_2, p_3, and p_4 (Table 1), and Q contains three tuples, namely q_1, q_2, and q_3 (Table 2). Attribute A_1 and attribute A_2 are "Distance-to-beach" and "Price," respectively. Attribute C is "Cost."

Let $price_{max}$ be the greatest possible price of a tuple in P. We assume that the price of each tuple in Q should be set to a value at most $price_{max}$. This assumption makes sense since we do not want the price of each package too high compared with all existing packages. Besides, since there are an infinite number of possible values in $\mathbb{R}$, we assume the domain of attribute "Price" (i.e., A_l) is defined to be $D = \{i \cdot \sigma | i$ is a nonnegative integer and $i \cdot \sigma \leq price_{max}\}$ where σ is a real number and a user parameter. If we want to have a finer granularity, we should set σ to a smaller value. This assumption makes sense in real applications where attribute Price involves discrete values instead of continuous values.

2.1 Finding Top-k Profitable Products

A new company is interested in selecting k tuples from Q as the final tuples where k is a positive integer and a user parameter. It wants to maximize the *profit* of this selection. There are many possible subsets containing k tuples from Q. Let us consider one *particular* subset Q'. The price of each of these k tuples (represented by attribute A_l) in Q' is to be assigned with a value in D. Given a tuple q in Q, after we set $q.A_l$ to a value v, the *profit* of q, denoted by $\triangle(q, v)$, is defined to be $v - q.C$.

We define a *price assignment vector* of Q', denoted by $\mathbf{v}$, in form of $(v_1, v_2, \ldots, v_n)$. v_i is said to be ith entry of $\mathbf{v}$. If $q_i \in Q'$ where $i \in [1, n]$, then v_i is assigned with a value in D. Otherwise, v_i is set to 0.

A price assignment vector $\mathbf{v}$ is said to be *feasible* if after we set the price of each $q_i \in Q'$ to v_i, each $q_i \in Q'$ is in the skyline with respect to $P \cup Q'$.

Definition 1 (Profit of selection). *Let Q' be a set of k tuples selected from Q. Let $\mathbf{v}$ be the price assignment vector of Q' in form of $(v_1, v_2, \ldots, v_n)$. The* profit *of Q' with $\mathbf{v}$, denoted by $Profit(Q', \mathbf{v})$, is defined to be $\sum_{q_i \in Q'} \triangle(q_i, v_i)$.*

Definition 2 (Optimal price assignment vector). *Let Q' be a set of k tuples selected from Q. Let $\mathcal{V}$ be a set of all possible feasible price assignment vectors for Q'. The* optimal price assignment vector *of Q' is defined to be the price assignment vector $\mathbf{v}_o$ for Q' such that*

$$Profit(Q', \mathbf{v}_o) = \max_{\mathbf{v}' \in \mathcal{V}} Profit(Q', \mathbf{v}').$$

The optimal profit *of Q', denoted by $Profit_o(Q')$, is defined to be $Profit(Q', \mathbf{v}_o)$ where $\mathbf{v}_o$ is the optimal price assignment vector of Q'.*

In Section 3, we describe an efficient algorithm to find the optimal price assignment vector given a set Q' of k selected tuples.

We just learned that given a *particular* set Q', we can determine the optimal profit of Q'. However, there are many possible subsets of Q containing k tuples. The company wants to find a selection containing k tuples from Q such that the total profit is maximized.

Problem 1 (Finding top-k profitable products). *Let $\mathcal{Q}$ be the set of all possible subsets containing k tuples from Q. We want to select a set Q' of k tuples from Q such that $Profit_o(Q') = \max_{Q'' \in \mathcal{Q}} Profit_o(Q'')$.*

This problem is called *finding* t*op-k* p*rofitable* p*roducts*. Some notations used in this paper are in Table 3.

A naive way for this problem is to enumerate all possible subsets of size k from Q, calculate the optimal profit of each possible subset, and choose the subset with the greatest profit. However, this approach is not scalable because there are an exponential number of all possible subsets. This motivates us to propose efficient algorithms for problem TPP which will be described in Section 4.2.

TABLE 3
Notation Table

Notation	*Description*
P	*a set of tuples in the existing market*
Q	*a set of potential new tuples*
m	*the size of* P
n	*the size of* Q
k	*the total number of tuples in* Q *to be selected*
$q.A_j$	*the* j*-th attribute value of tuple* q
l	*the number of the attributes of tuples in* P
Q'	*the set of top-*k *profitable products*
$Profit(Q', \mathbf{v})$	*total profit of* Q' *when the price vector is* $\mathbf{v}$
$Profit_o(Q')$	*the optimal profit of* Q'
$\gamma(X, q)$	*the set of all tuples in* X *which quasi-dominate* q
$SKY(P)$	*the skyline of* P
σ	*the granularity of attribute price*
$S(i,t)$	*the set* Q' *of size* t *where* $i \in [1,n]$ *and* $t \in [0,k]$ *such that* $Profit_o(Q') = \max_{Q'' \in \mathcal{Q}} Profit_o(Q'')$ *where* $\mathcal{Q}$ *is the set of all possible subsets containing* t *tuples from* $Q(i)$
$\mathbf{v}(i,t)$	*the optimal price assignment vector of set* $S(i,t)$
$T(i,t)$	*the (optimal) profit of set* $S(i,t)$
N	N *is a positive integer where* $N << m+n$

3 Finding Optimal Price Assignment

In this section, we present an algorithm for finding the optimal price assignment called *AOPA* in $O(k(m+n+N))$ time given a set Q' of size k where $N << (m+n)$.

Suppose that Q' is a selection set. Our objective is to find the optimal price assignment vector of Q'. After setting the prices of all tuples in Q' according to this vector, the tuples in Q' are in the skyline with respect to $P \cup Q'$.

Let $X = P \cup Q'$. Given $p \in X$ and $p' \in X$, p is said to *quasidominate* p' if 1) p dominates p' with respect to the first $l-1$ attributes, namely $A_1, A_2, \ldots, A_{l-1}$, or 2) p has the same $l-1$ attribute values as p'. Let $\gamma(X, q_i)$ be a set containing all tuples in X which quasidominate q_i.

The following lemma gives us an intuition of how to design an algorithm to find the optimal price assignment vector of Q':

Lemma 1 ([20]). *Suppose that* $p \in X$ *and* $q_i \in Q'$. *Consider that we are given a price assignment vector of* Q' *equal to* $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ *such that we set the price of each* q_j *(i.e.,* $q_j.A_l$*) in* Q' *to* v_j. *If* p *dominates* q_i, *then* $p \in \gamma(X, q_i)$.

According to the above lemma, we divide the tuples in X into two groups:

- **Group 1 (Outside γ).** Group 1 is the set of all tuples not in $\gamma(X,q)$ (more specifically, all tuples in $X - \gamma(X,q)$). The tuples in this group do not dominate q regardless of any price assignment vector of Q'.
- **Group 2 (Inside γ).** Group 2 is the set of all tuples in $\gamma(X,q)$. For a particular price assignment vector of Q', some tuples in this group may dominate q while for another particular price assignment vector of Q', they may not dominate q.

Our objective is to make sure that each tuple $q \in Q'$ is in the skyline with respect to $X (= P \cup Q')$. That is, each tuple q in Q' is not dominated by any tuple in X. This is our goal. Consider Group 1 (Outside γ). We can achieve the goal because all tuples in this group do not dominate q. Consider Group 2 (Inside γ). It is possible that some tuples in $\gamma(X,q)$ dominate q for a particular price assignment vector. For another price assignment vector, they do not dominate q.

In the above, we learn that if we want to determine the price of q_i in Q' such that q_i is in the skyline, we only need to consider the tuples in $\gamma(X,q)$.

Given a tuple $q \in Q'$, we know that only the tuples in X quasidominating q affect the price of q. Note that the prices of all tuples in P are given and the prices of all tuples in Q' are to be found. Thus, according to the quasidominance relationship, we design a *progressive* algorithm which finds the price of each tuple q in Q' by the following principle:

Principle 1. *Whenever we want to find the price of* q_i *in* Q', *we make sure that the prices of all tuples in* Q' *quasidominating* q_i *have already been determined.*

Next, we need to determine the ordering of processing tuples in Q' which follows the above principle.

We define the following monotonically increasing function f which can determine the ordering. Given a tuple q in Q, function f is defined as $f(q) = \sum_{i=1}^{l-1} q.A_i$.

With this function f, we know the following lemma:

Lemma 2 ([20]). *Suppose* p *and* p' *are in* X. *If* p *quasidominates* p', *then* $f(p)$ *is smaller than or equal to* $f(p')$.

With the above lemma, we can first compute the f values of all tuples in Q'. We sort the tuples in Q' in ascending order of these f values. Then, we determine the price of each tuple q in Q' according to this ordering, which follows Principle 1.

After we obtain the ordering of processing the tuples in Q', we present an algorithm to determine the optimal price assignment vector of Q' incrementally.

Without loss of generality, we assume that $q_1, q_2, \ldots, q_k$ are the tuples in Q' sorted in ascending order of the f values. Let $Q_0 = \emptyset$. Let $Q_i = Q_{i-1} \cup \{q_i\}$ where $i = 1, 2, 3, \ldots, k$.

Lemma 3 ([20]). *Suppose that* $p \in X$ *and* $q_i \in Q'$. *Consider that we are given the optimal price assignment vector of* Q_{i-1} *equal to* $\mathbf{v}_{i-1} = (v_1, v_2, \ldots, v_n)$ *such that we set the price of each* q_j *(i.e.,* $q_j.A_l$*) in* Q_{i-1} *to* v_j. *Suppose that* $\gamma(X, q_i) \neq \emptyset$. *Let* $\mathbf{v}_i$ *be a price assignment vector equal to* $\mathbf{v}_{i-1}$ *except that* i*th entry of* $\mathbf{v}_i$ *is set to* $(\min_{p \in \gamma(X,q_i)} p.A_l) - \sigma$. $\mathbf{v}_i$ *is the optimal price assignment vector of* Q_i.

By Lemma 3, we can derive a progressive algorithm as shown in Algorithm 1. It is easy to verify the following theorem:

Algorithm 1. Algorithm **AOPA**(Q')
Input: A set Q' of tuples in Q
Output: the optimal profit assignment vector .of Q'

```
1: $Q'' \leftarrow \emptyset$
2: $\mathbf{v} \leftarrow (0, 0, ..., 0)$
3: for each $q_i \in Q'$ (which is processed in the sorted
   ordering) do
4:     $\mathbf{v} \leftarrow$ findOptimalIncrementalPrice($q_i, Q'', \mathbf{v}$)(See
       Algorithm 2)
5:     $Q'' \leftarrow Q'' \cup \{q_i\}$
6: return $\mathbf{v}$
```

Theorem 1 ([20]). *Given a set Q' of k tuples from Q, Algorithm 1 returns the optimal price assignment of Q'.*

Implementation and time complexity. We first analyze the time complexity of Algorithm 2 and then the time complexity of Algorithm 1.

In Algorithm 2, the most time-consuming operation is the step of finding all tuples in $P \cup Q'$ which quasidominate q_i (Line 2). In our implementation [20], we have two substeps for this step: 1) we perform a *range query* with range equal to "$A_j \leq q_i.A_j$" for each $j \in [1, l-1]$ on an R*-tree index built on data set $P \cup Q$ (instead of $P \cup Q'$) according to the first $l-1$ attributes, and 2) we perform a postprocessing step of selecting all the tuples in the answer of the above range query which are also in $P \cup Q'$. The cost of a range query (in (1)) is $O(|P|] + |Q| + N)$ where N is the total number of tuples returned in a range query. Typically, N is extremely small compared with $(|P| + |Q|)$. That is, $N << |P| + |Q|$. In the worst case, $N \approx |P| + |Q|$. But, this situation rarely happens. The postprocessing step (in 2) takes $O(N)$ time. Thus, the step of finding all tuples in $P \cup Q'$ which quasidominate q_i takes $O(|P| + |Q| + N)$ time. Thus, it is easy to verify that Algorithm 2 takes $O(|P| + |Q| + N)$ time.

Algorithm 2. Algorithm **findOptimalIncremental Price**$(q_i, Q_{i-1}, \mathbf{v}_{i-1})$
Input: tuple q_i in Q', a set $Q_{i-1} (= \{q_1, q_2, ..., q_{i-1}\})$ and the optimal price assignment vector $\mathbf{v}_{i-1}$ of Q_{i-1}
Output: the optimal price assignment vector $\mathbf{v}_i$ of Q_i

```
1: $\mathbf{v}_i \leftarrow \mathbf{v}_{i-1}$
2: find a set $Y$ containing all tuples in $P \cup Q'$ which
   quasi-dominate $q_i$
3: if $Y \neq \emptyset$ then
4:     $v \leftarrow (\min_{p \in Y} p.A_l) - \sigma$
5: else
6:     $v \leftarrow \infty$
7: set the $i$-th entry in $\mathbf{v}_i$ to $v$
8: return $\mathbf{v}_i$
```

Consider Algorithm 1. Since there are $|Q'|$ iterations (in lines 3-5) and each iteration calls Algorithm 2 (which takes $O(|P| + |Q| + N)$ time), the overall time complexity of Algorithm 1 is $O(|Q'|(|P| + |Q| + N))$. Since $m = |P|, n = |Q|$, and $k = |Q'|$, the time complexity becomes $O(k(m + n + N))$.

4 Algorithm

In [20], we described a dynamic programming approach which finds an optimal solution when $l = 2$. However, when $l > 2$, we show that the problem is NP-hard. Since the problem is NP-hard, we propose two greedy algorithms for this problem when $l \geq 2$. In the following, we give a brief description of the dynamic programming approach when $l = 2$ in Section 4.1. When $l > 2$, we show the hardness of this problem and give two greedy algorithms for this problem in Section 4.2.

4.1 Dynamic Programming Approach

Consider $l = 2$. Without loss of generality, we assume that $q_1, q_2, \ldots, q_n$ are sorted in ascending order of the f values in Q. In this dynamic programming approach, we define a notation of a general problem statement denoted by $X[i, t]$ where $i \in [1, n]$ and $t \in [0, k]$ to denote problem TPP except that we consider the first i tuples in Q instead of all tuples in Q and we select t tuples only instead of k tuples.

With the problem statement notation, our final goal is to find the solution for problem $X[n, k]$ (where n is the size of Q). We solve problem $X[n, k]$ by first solving a number of subproblems $X[i, t]$ where $i \in [1, n]$ and $t \in [0, k]$ and then combining the solutions for the subproblems for the solution of problem $X[n, k]$. Specifically, we solve all subproblems in a particular ordering since solving a subproblem may need the solutions of some other subproblems ordered previously. In the following, we first describe the ordering of processing subproblems. Then, we give a general idea of how to solve a given subproblem $X[i, t]$ based on the solutions of some subproblems ordered previously.

First, we process subproblems in the following ordering. We start with the first iteration. In this iteration, we initialize the solution of problem $X[1, 0]$ and then the solutions of problems $X[2, 0], X[3, 0] \ldots, X[n, 0]$ to $\emptyset$. For the second iteration, we solve problems $X[1, 1], X[2, 1] \ldots, X[n, 1]$. We do this iteratively. In the last iteration (or the $(k+1)$th iteration), we solve problems $X[1, k], X[2, k] \ldots, X[n, k]$. Finally, we obtain the solution for problem $X[n, k]$. Second, we can find the solution of a given subproblem $X[i, t]$ based on the solutions of subproblems $X[i-1, t]$ and $X[i-1, t-1]$ according to whether including q_i in the final selection set of size t gives a greater profit. Details can be found in [20].

4.2 Greedy Algorithms

However, when $l > 2$, we show that the problem is NP-hard as follows:

Theorem 2. *When $l > 2$, problem TPP is NP-hard.*

Since the problem is NP-hard, we propose two greedy algorithms for this problem. As we described in Example 3, the price of a new selected tuple may affect the price of another new selected tuple. We call this phenomenon a *price correlation*. The first version of the greedy algorithm is the algorithm which selects tuples in Q iteratively without considering the price correlation. The first greedy algorithm returns a solution with a theoretical guarantee on the profit. The second version is the algorithm which selects tuples in Q iteratively considering the price correlation. Since the second greedy algorithm considers the price correlation, it returns a greater profit compared with the first version but it takes more time.

4.2.1 Greedy-Based Algorithm I

The first version of the greedy algorithm is the algorithm which selects tuples in Q iteratively without considering the price correlation.

For each tuple q in Q, we first define the optimal profit of the selection set containing q only. We call this profit the *stand-alone profit* of q_i.

Definition 3 (Stand-alone profit). *Given a tuple q in Q, the* stand-alone profit *of q, denoted by $SP(q)$, is defined to $Profit_o(\{q\})$.*

The first version of the greedy algorithm called *Greedy1* is described as follows. Specifically, for each tuple q in Q, we find the stand-alone profit of q. Then, we choose k tuples which have the greatest stand-alone profits.

Although this greedy approach is a heuristical approach, it has theoretical guarantees on the profit returned by the algorithm.

Suppose that O is the *optimal selection set* for problem TPP (i.e., the selection set which has the greatest profit). Note that the optimal profit of O is equal to $Profit_o(O)$. Recall that we want to maximize the profit, due to the heuristical nature of the greedy algorithm, this algorithm may return a selection Q' which has a lower profit (which is equal to $Profit_o(Q')$). That is,

$$Profit_o(Q') \leq Profit_o(O).$$

In the following, we give two theoretical results about the error guarantee on the profit returned by the algorithm. The first result corresponds to an *additive error guarantee* while the second one corresponds to a *multiplicative error guarantee*.

We first show the result about the additive error guarantee.

Theorem 3 ([20]). *Let O be the optimal selection set and Q' be the selection set returned by Greedy1. $Profit_o(O) - \epsilon_{add} \leq Profit_o(Q')$ where $\epsilon_{add} = \frac{k(k-1)}{2}\sigma$.*

Next, we show the result about the multiplicative error guarantee.

Theorem 4 ([20]). *Let O be the optimal selection set and Q' be the selection set returned by Greedy1. Suppose that $Profit_o(Q') > 0$. Let $\Delta = \sum_{q_i \in Q'} SP(q_i)$. Greedy1 is a $(1 - \epsilon_{mult})$-approximate algorithm. That is, $Profit_o(Q') \geq (1 - \epsilon_{mult})Profit_o(O)$ where $\epsilon_{mult} = \frac{k(k-1)\sigma}{2\Delta}$.*

Time complexity. Consider Greedy1. We need to calculate the stand-alone profit of q for each tuple $q \in Q$. This step takes $O(n(m + n + N))$ time. Then, we need to choose the k tuples which have the greatest stand-alone profits, which can be done in $O(k \log n)$ time. Thus, the time complexity of Greedy1 is $O(n(m + n + N) + k \log n)$. Since $k = O(n)$, the complexity becomes $O(n(m + n + N))$.

4.2.2 Greedy Based Algorithm II

In the previous section, we describe the first version of the greedy algorithm which does not consider the price correlation. In this section, we describe the second version of the greedy algorithm called *Greedy2* which selects tuples in Q iteratively considering the price correlation.

The second version of our greedy algorithms is shown in Algorithm 3:

Algorithm 3. Greedy algorithm (Version 2)

```
1: $Q' \leftarrow \emptyset$
2: while $|Q'| \leq k$ do
3:    for each $q_i \in Q$ do
4:        $x_i \leftarrow$ AOPA($Q' \cup \{q_i\}$)
5:    find the tuple $q_i$ in $Q$ such that $q_i$ has the greatest
      value of $x_i$
6:    $Q' \leftarrow Q' \cup \{q_i\}$
7: return $Q'$
```

Time complexity. Consider Greedy2 (Algorithm 3). There are $O(k)$ iterations where statements from lines 3 to 6 correspond to an iteration. Consider an iteration. Statements from lines 3 to 4 take $O(n \cdot k((m + n + N)) = O(nk(m + n + N))$ time. Statements from lines 5 to 6 take $O(n)$ time. Thus, each iteration takes $O(nk(m + n + N) + n) = O(nk(m + n + N))$ time. The overall time complexity of Algorithm 3 is $O(k \cdot nk(m + n + N)) = O(nk^2(m + n + N))$. Note that compared with the time complexity of Greedy1 (i.e., $O(n(m + n + N))$, the time complexity of Greedy2 (Algorithm 3) is higher.

5 Discussion

In problem TPP, after we set the price of each tuple in the selection set Q', we know that each of these tuples is in the skyline with respect to $P \cup Q'$. In other words, after we set the price of each tuple in Q', each of these tuples is *one* of the best choices for the customer to choose (because there may be more than one tuple in the skyline). In order to make sure that each tuple in Q' will be chosen by a customer in the market with a higher probability, we would like to set the price of each of these tuples such that not only each of these tuples is in the skyline but also each of these tuples dominates at least h tuples in the existing market P where h is an input parameter. This problem is called Finding *t*op-k *p*rofitable *p*roducts with the h-dominance constraint. The h-dominance constraint corresponds to that each of these tuples dominates at least h tuples in the existing market P. If we set $h = 0$, then the new problem becomes problem TPP without the h-dominance constraint. The algorithm proposed for the original problem can be adapted easily for the new problem. Details can be found in [20].

6 TPP over Dynamic Data Sets

In previous sections, we discussed how to find top-k profitable products over *static* data sets P and Q. However, in some cases, data sets are *dynamic* and change from time to time. In this section, we study the problem of finding top-k profitable products on *dynamic* data sets. In this paper, there are two kinds of data sets, namely P and Q, which can change over time. In the following, for the interest of space, we focus on studying how to find top-k profitable products when P changes. We do not discuss the case when Q changes because similar conclusions can also be drawn.

We study three kinds of *operations* on P, namely insertion, deletion, and modification. Suppose that o is one of the three operations. After o is executed on P, we obtain a new data set denoted by P_{new}. Specifically, we have the following operations and the corresponding P_{new}:

1. *Insertion*. A tuple p_{new} is inserted into P. Then, $P_{new} = P \cup \{p_{new}\}$.
2. *Deletion*. A tuple p is removed from P. Then, $P_{new} = P - \{p\}$.
3. *Modification*. Some of the attribute values of a tuple p in P are changed and p becomes p'. Then, $P_{new} = (P - \{p\}) \cup \{p'\}$.

Note that a modification operation can be regarded as a sequence of the other two operations (i.e., a deletion operation and then an insertion operation). It is sufficient to describe how we execute the deletion operation and the insertion operation. In the following, for the sake of space, we focus on how we execute the insertion operation only. Details of the deletion operation can be found in [19].

Problem 2 (Dynamic TPP). *Let o be an operation and P_{new} be the resulting set P after o is executed on P. We want to find a set Q'_{new} of top-k profitable products based on P_{new} and Q.*

A straightforward approach is to run one of the algorithms in Section 4.2 on the new data sets P_{new} and Q from *scratch* whenever there is an operation. However, it is very costly because this approach does not make use of some useful results computed before the operation is executed. Instead, we propose an *incremental* algorithm to find a set Q'_{new} of top-k profitable products based on not only the new data sets P_{new} and Q but also the previous result Q' (computed based on the previous data sets P and Q).

Since P changes over time, it is desirable to design an efficient algorithm. In the following, we give an *incremental* version of Greedy1 because Greedy1 is more efficient compared with Greedy2. This incremental version which is based on not only the new data sets P_{new} and Q but also the previous result Q', and returns the same selection set as the original Greedy1 which is based on the new data sets only.

Now, we focus on describing how the insertion operation is executed. Suppose that a new tuple p_{new} is inserted into P and then P_{new} is formed. Before we discuss our incremental algorithm, we first give two lemmas or properties for the insertion operation.

Lemma 4 (P_{new}-based property). *If $p_{new} \notin SKY(P_{new})$, then $Q'_{new} = Q'$.*

Lemma 5 (Q'-based property). *There does not exist $q \in Q'$ such that p_{new} dominates q if and only if $Q'_{new} = Q'$.*

According to Lemmas 4 and 5, we design an incremental algorithm as shown in Algorithm 4:

Algorithm 4. Incremental Algorithm for Insertion

```
1:  // Checking $P_{new}$-Based Property (specified in
    Lemma 4)
2:  if $p_{new} \notin SKY(P_{new})$ then
3:      $Q'_{new} \leftarrow Q'$
4:  else
5:      // Checking $Q'$-Based Property (specified in
        Lemma 5)
6:      if there does not exist $q \in Q'$ such that $p_{new}$ dominates
        $q$ then
7:          $Q'_{new} \leftarrow Q'$
8:      else
9:          // need to calculate $Q'_{new}$ incrementally
10:         for each $q \in Q$ which is quasi-dominated by $p_{new}$ do
11:             re-compute the standalone profit of $q$
12:         $Q'_{new} \leftarrow$ a set of the $k$ tuples in $Q$ which have the
            greatest standalone profits
13: return $Q'_{new}$
```

It is easy to verify the following theorem:

Theorem 5. *Algorithm 4 returns a selection set Q'_{new} which is equal to the selection set returned by Greedy1.*

The complexity of checking the *P_{new}-Based Property* (Lines 2 to 3) is $O(m)$. Otherwise, we can check the *Q'-Based Property* (Lines 6 to 7), which takes $O(k)$ time. If both of the properties are not satisfied, then we need to recompute $SP(q)$ for each $q \in Q$ which is quasidominated by p_{new}. For other q which is not quasidominated by p_{new}, their standalone profits are the same as before. Similar to the time complexity analysis in Section 4.2.1, the step (Lines 10 to 11) takes $O(u \cdot (m + n + N))$ time where u is the total number of tuples in Q quasidominated by p_{new}. Finally, we choose k tuples which have the greatest stand-alone profits. It can be done in $O(k \log n)$ time. So, the total complexity is $O(m + u \cdot (m + n + N) + k \log n)$.

7 Finding Top-k Popular Products

7.1 Problem Definition

In this section, we extend our problem when the preferences of customers are given. The preferences of customers can be collected by conducting surveys where customers can provide their preferences on products by questionnaire. The preferences can also be obtained by extracting customers' preferences from their past histories [7]. Besides, they can also be obtained directly by some online systems such as "Name Your Own Price" where customers can provide their preferable prices directly.

In our running example, a customer would like to choose a package which is not too expensive and the hotel in the package is not too far away from a beach. For example, she/he gives his/her *preference* that the price is at most \$450 and the distance to the beach is at most 4.0 km. \$450 and 4.0 km are said to be the *greatest possible acceptable values* of attribute price and attribute distance-to-beach, respectively.

Formally, each *customer preference* cp is represented by a set of l values, namely $g_1, g_2, \dots, g_l$, where l is the total number of attributes and g_j is the greatest possible acceptable value of attribute A_j for $j \in [1, l]$. If a customer does not have any special preference on a particular attribute A_j, she/he can simply specify the greatest possible acceptable value of attribute A_j to be ∞. In addition, each customer preference cp is associated with a *weight*, denoted by $w(cp)$, denoting the total number of customers who give this customer preference cp. Let CP be a set of customer preferences.

In the following, in order to simplify the discussion, following the spatial database literature [8], [9], [15], [22], [23], we assume that each potential tuple can satisfy as many customer preferences as possible. There are a lot of applications following this assumption. Generating popular laptops is an example since the components in the market for assembling laptops is abundant and we can assume that a laptop can meet as many customer preferences as possible. Finding popular delivery services in a new cargo company is another example where a delivery service can serve a lot of customers. Finally, finding popular cell phone plans in a new phone company is one example where a plan can be subscribed by a lot of customers. Considering the *capacities* of potential packages (i.e., how many units of potential packages which are available) is left as a future work.

Given a tuple q in a final selection set Q', we have to set the price of q. Since we do not want to lose any money, we should set the price of q at least the cost of q. Besides, we want to guarantee that q is in the skyline with respect to $P \cup Q'$ after we set the price of q. Given a tuple $q \in Q$, we define the set of all possible prices of q, denoted by $PS(q)$, which satisfy the above conditions to be $\{v | v \geq q.C$ and $q \in SKY(P \cup \{q\})$ if we set $q.A_l = v\}$.

Note that Q' is the selection set. Suppose that for each tuple $q \in Q'$, we set the price of q to a positive real number $v \in PS(q)$. Given $q \in Q'$ and a customer preference represented by $\{g_1, g_2, \ldots, g_l\}$, q is said to *satisfy cp* if for each $j \in [1, l]$, $q.A_j \leq g_j$.

Given a tuple $q \in Q$ and a value $v \in PS(q)$, the *influence set* of q with respect to v, denoted by $IS(q, v)$, is defined to be the set of customer preferences which are satisfied by q if we set the price of q to v.

Definition 4 (Influence set and influence value). *Given a tuple $q \in Q$, the* influence set *of q, denoted by $IS(q)$, is defined to be $\cup_{v \in PS(q)} IS(q, v)$. Given $q \in Q$, the* influence value *of q, denoted by $IV(q)$, is defined to be $\sum_{cp \in IS(q)} w(cp)$.*

Let Q' be a subset of Q. The $influenceset$ of Q', denoted by $IS(Q')$, is defined to be $\cup_{q \in Q'} IS(q)$. The $influencevalue$ of Q', denoted by $IV(Q')$, is defined to be $\sum_{cp \in IS(Q')} w(cp)$.

Problem 3 (Finding top-k popular products). *Let $\mathcal{Q}$ be the set of all possible subsets containing k tuples from Q. We want to select a set Q' of k tuples from Q such that 1) $IV(Q') = \max_{Q'' \in \mathcal{Q}} IV(Q'')$ and 2) each tuple in Q' is in the skyline with respect to $P \cup Q'$.*

The tuples in the output of the above problem are called *top-k popular products*. Note that in the above problem, setting different values of k gives different influence values of the final selection set. Let IV_i be the optimal influence value of the final selection set of size i. It is easy to verify that IV_i is *monotonically* increasing with i since more customers are interested in the tuples in the final selection set when more tuples are included in the final selection set. It is also easy to see that when the final selection set contains a certain number of tuples, says k_{max}, the influence value keeps unchanged even if the selection set contains more tuples. We define the greatest possible influence value denoted by IV_{max} to be $\max_{i \in [1, |Q|]} IV_i$. We also define k_{max} to be the smallest possible number of tuples in the final selection set such that its influence value is equal to IV_{max}. That is, $k_{max} = \min\{i | IV_i = IV_{max}\}$. In the following, we assume $k \leq k_{max}$. If $k > k_{max}$, then the additional $k - k_{max}$ tuples are redundant for influencing customers.

Theorem 6. *Problem Finding Top-k Popular Products is NP-hard.*

7.2 Algorithm

We propose a greedy approach to find top-k popular products. Before we give the algorithm, we define the concept of "optimal price" as follows. Given a tuple $q \in Q$ and a set $X \subseteq IS(q)$, the *optimal price* of q satisfying tuples in X, denoted by $optPrice(q, X)$, is the greatest possible price $v \in PS(q)$ we can set such that $X \subseteq IS(q, v)$.

The algorithm is shown in Algorithm 5. In line 3 of Algorithm 5, we find $q \in Q$ such that $IV(Q' \cup \{q\})$ is the greatest. In some cases, there are ties. That is, there are at least two tuples q and q' in Q which give the same greatest values of $IV(Q' \cup \{q\}) (= IV(Q' \cup \{q'\}))$. Let Y be the set of tuples q in Q which give the same greatest values of $IV(Q' \cup \{q\})$. In this case, we choose q in Y such that $f(q)$ is the smallest where f is the function defined in Section 3.

Algorithm 5. Algorithm for Finding Top-k Popular Products

```
1: $Q' \leftarrow \emptyset$
2: while $|Q'| < k$ do
3:   find $q \in Q$ such that $IV(Q' \cup \{q\})$ is the greatest
4:   $X \leftarrow IS(Q' \cup \{q\}) - IS(Q')$
5:   set the price of $q$ to be $optPrice(q, X)$
6:   $Q \leftarrow Q - \{q\}$
7:   $Q' \leftarrow Q' \cup \{q\}$
8: return $Q'$
```

Note that there are two criteria in Problem 3. The first criterion is to maximize the influence value of a selection set while the second criterion is to make sure that each tuple in the selection set is in the skyline with respect to $P \cup Q'$. Interestingly, although Algorithm 5 finds q iteratively according to the influence value of a selection set $Q \cup \{q\}$ but not the criterion on whether q can be in the skyline with respect to $P \cup Q'$, it also returns a set Q' such that each tuple in Q' is in the skyline with respect to $P \cup Q'$. This result can be found in the following lemma:

Lemma 6. *Let Q' be the selection set returned by the algorithm for finding top-k popular products (i.e., Algorithm 5). Each tuple in Q' is in the skyline with respect to $P \cup Q'$.*

This algorithm not only can satisfy the second criterion but also can give a theoretical guarantee for the first criterion (even though the problem is NP-hard).

Theorem 7. *The algorithm for finding top-k popular products (i.e., Algorithm 5) is 0.63-approximate. Let Q' be the selection set returned by the algorithm and O be the optimal set (which gives the greatest influence value). We have $IV(Q') \geq 0.63 \cdot IV(O)$.*

8 Empirical Studies

We have conducted extensive experiments on a Pentium IV 2.4 GHz PC with 4 GB memory, on a Linux platform. We implemented all algorithms we proposed, namely *DP*, *GR1*, and *GR2*. *DP* corresponds to our dynamic programming approach [20] while *GR1* and *GR2* correspond to the first version and the second version of the greedy algorithms (Section 4). We also implemented a naive (or brute-force) algorithm described in Section 2. We name it as *BF*. All the programs were implemented in C++. In the following, we consider problem TPP with the h-dominance constraint discussed in Section 5 since it is more general than problem TPP without the h-dominance constraint.

We measured the algorithms with four measurements, namely:

1. *Execution time*. The execution time of an algorithm corresponds to the time it takes to find the final selection.

TABLE 4
Experimental Setting on Synthetic Data Sets

Parameter	*Values*
$\|P\|$ (the size of P)	*0.5M,* **1M**, *1.5M, 2.0M*
$\|Q\|$ (the size of Q)	*0.5M, 1M,* **2M**, *3M*
d (discount rate)	*0.25,* **0.5**, *0.75, 1*
l (the number of the attributes of tuples in P)	*2,* **5**, *10,15, 20*
k (the total number of tuples in Q to be selected)	*10,* **20**, *50, 100*
σ (the granularity of attribute price)	*50,* **100**,*150, 200*
h (the minimum number of tuples that are dominated by a tuple in Q')	*0,* **10**,*20,30*

2. *Preprocessing time*. The preprocessing of an algorithm corresponds to the time it builds an R*-tree index for quasidominance checking.
3. *Memory cost*. The memory cost of an algorithm is the memory occupied by the algorithm.
4. *Profit*. The profit of an algorithm corresponds to the profit returned by the algorithm.

The experiments were conducted over real data sets and synthetic data sets. For the real data sets, same as [20], we obtained real data sets from Priceline.com and Expedia.com. We have 149 packages in P and create 4,787 packages in Q. For each package q in Q, we set $q.C$ to be the price of this package multiplied by a discount rate d where d is a user parameter. Note that although there are values in attribute price in this set Q, we discard all these values in the data sets because our problem is to find these values. Details of this data sets can be found in [20]. For synthetic data sets, we also generated P and Q as anticorrelated data sets [2]. Details of the data sets generation can be found in [20].

In the following, we first give the experimental results for finding top-k profitable products in Section 8.1. Then, in Section 8.2, we present the results when dynamic data are considered. Finally, we show the experimental results for finding top-k popular products in Section 8.3.

8.1 Experimental Result for Finding Top-k Profitable Products

8.1.1 Result over Synthetic Data Sets

In this section, we conducted experiments over both small and large synthetic data sets to study the scalability of *GR1* and *GR2*. We varied $|P|, |Q|, d, l, k, \sigma$, and h in our experiments. The values of each parameter used over large synthetic data sets are given in Table 4 where the default values are in bold. For the sake of space, we show the results when we varied $|Q|$ only in Fig. 1. Other experimental results over small synthetic data sets and large synthetic data sets can be found in [20].

Execution time. Fig. 1a shows the effect on the execution times of all algorithms. *GR2* runs slower than *GR1*. As we discussed in Section 4, the time complexity of *GR2* is higher than that of *GR1*.

Total time. Fig. 1b shows the effect on the total time of each greedy algorithm which corresponds to the sum of the execution time of the greedy algorithm and the execution time of the preprocessing time of the greedy algorithm. We denote the total times of *GR1* and *GR2* by "$GR1 + PREP$" and "$GR2 + PREP$," respectively. It is clear that when $|Q|$ increases, the total times of the algorithms increase.

Memory cost. Fig. 1c shows the effect on the memory cost of the algorithms. Since the memory cost of both *GR1* and *GR2* is the memory occupied by the R*-tree on data sets $P \cup Q$, when $|Q|$ increases, the memory cost increases, as shown in Fig. 1.

Profit. Fig. 1d shows the effect on the profit returned by the algorithms. In most cases, *GR1* and *GR2* give similar profits.

8.1.2 Result over Real Data Sets

We also conducted experiments on real data sets. We varied four factors, namely h, k, d, and σ. For the sake of space, we only show the results with two factors h and k as shown in Figs. 3 and 4, respectively. The default setting configuration is: $k = 150$, $h = 20$, $d = 0.6$, and $\sigma = 50$. The results for real data sets are similar to those for synthetic data sets. Note that there is a big difference in execution times between *GR1* and *GR2* in the *real* data sets but this big difference cannot be found in the *synthetic* data sets. As we described in Section 4.2.2, the time complexity of *GR2* is quadratic with respect to k but the time complexity of *GR1* is not. Thus, when k is larger, then the difference in execution times between *GR1* and *GR2* is larger. Compared with the synthetic data sets where k is set to 10, 20, 50, or 100, in the real data sets, since k is set to a larger value (e.g., 100, 150, 200, and 250), the difference in execution times between *GR1* and *GR2* is larger.

In order to see whether the prices of the top-k profitable packages we found are consistent with the *original* prices of the packages listed in Priceline.com in the market, we conducted experiments by regarding some of the existing products in Pricline.com as new packages in our problem and comparing the prices of the top-k profitable packages we found with their original prices listed in Priceline.com.

We conducted experiments over 100 trials. For each trial, we randomly select 30 percent of the 149 existing packages in Priceline.com to form a new package set Q. In the experiment, we set $k = 5$, $h = 0$, $d = 0.5$, and $\sigma = 50$. The experimental results show that the price of a package we

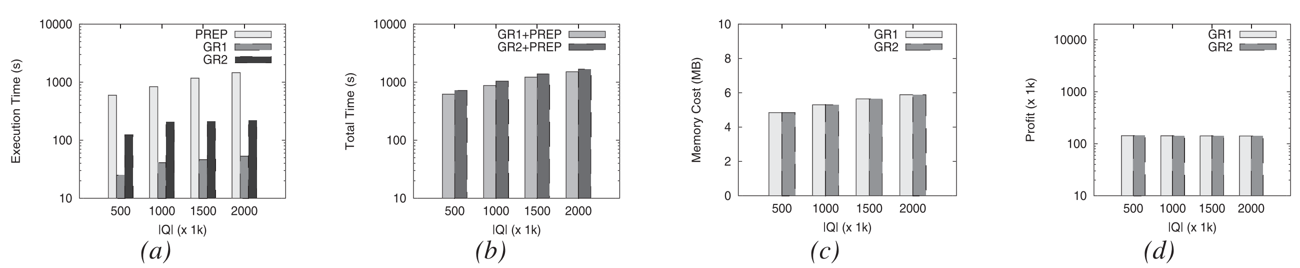
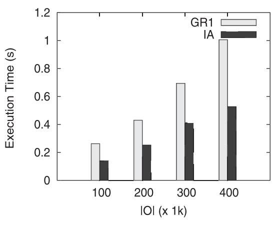


Fig. 1. Effect of $|Q|$ (the number of potential new tuples) on synthetic data sets.

Top	*Found ($)*	*Original ($)*
1	*1014.5*	*1158*
2	*1084.5*	*1087*
3	*800.5*	*844*
4	*898.5*	*965*
5	*655.5*	*577*

(a) Results for one trial



(b) Effect of $|O|$

Fig. 2. Results about top k prices and the effect of $|O|$.

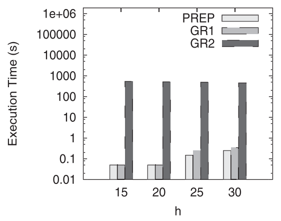
returned is 10.53 percent different from its original price in Priceline.com on average (among all trials). Fig. 2a shows the result for one trial which contain the prices of the top-5 profitable packages we returned and their original prices. In this trial, the average price difference is 7.65 percent.

8.2 Results for TPP over Dynamic Data

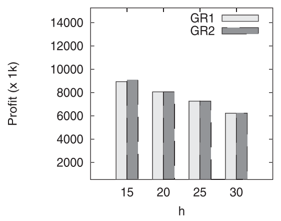
In this section, we want to study the performance of our proposed $i$ncremental $a$lgorithm proposed in Section 6. We denote this algorithm by IA. We compare it with an algorithm which finds top-k profitable products from $scratch$. Since IA is similar to $GR1$ which finds the products from scratch, in this experiment, we choose $GR1$ for comparison. Since IA and $GR1$ have the same preprocessing time, in the following, we do not show $PREP$ in the figure.

We conducted experiments over synthetic data sets and real data sets. The default values for the experiments over these data sets are the same as in Section 8.1.1.

In this dynamic case, we have three types of operations, namely insertion, deletion, and modification. As we discussed before, we focus on the former two operations. In the experiments, we generate operations in this dynamic case as follows. Consider a data sets D. If the operation to be generated is an insertion, we randomly generate a tuple by the method of generating tuples in P and regard it as the tuple to be inserted. If the operation to be generated is a deletion, we randomly pick one of the tuples in D and regard it as the tuple to be removed. In the experiment, for each data sets, we create a batch O of operations. We did three types of batches for experiments. The first type is the batch containing all insertion operations, the second type is the batch containing all deletion operations, and the third type is the batch containing 50 percent insertion operations and 50 percent deletion operations. In our experiments, we varied the size of O (denoted by $|O|$) from 100k to 400k. We evaluate the algorithms with their execution times.

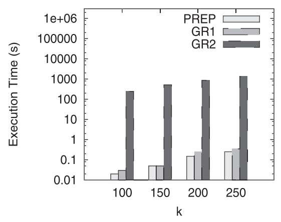


(a)

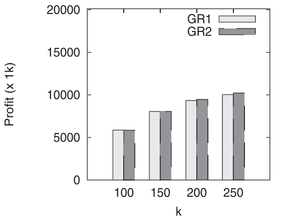


(b)

Fig. 3. Effect of h (the minimum number of tuples dominated by each tuple in the selection set) on a real data sets.



(a)



(b)

Fig. 4. Effect of k (the size of the final selection set) on a real data sets.

For the sake of space, we only show the experimental results over the real data sets as shown in Fig. 2b when the third batch type is considered. The execution times of both algorithms increase with $|O|$. Besides, IA is much more efficient than $GR1$.

8.3 Results for Finding Top-k Popular Products

In this section, we study how our proposed algorithm in Section 7 performs when we want to find top-k popular products. The preprocessing time of this algorithm (which includes the time to build an R*-tree) is denoted by $PREP$ while the main processing time of this algorithm is denoted by GA. We compare it by a $b$rute-$f$orce algorithm which enumerates all possible subsets of size k and finds the subset which gives the greatest influence value. We denote it by BF.

As before, we conducted experiments over the real data sets. The default values of parameters except k and h are the same as in Section 8.1.2. Besides, since running BF is time consuming, we set $k = 3$ in our experiments in order that the execution time of BF is shorter. Since finding top-k popular products does not have any h-dominance constraint described in Section 5, h is set to 0.

In the problem of finding top-k popular products, customer preferences are generated as follows. Suppose that we want to generate a customer preference cp. First, we generate a tuple c by the method of generating a tuple in P. Second, for each attribute A_i of tuple c where $i = 1, 2, \dots, l$, we set g_i to $c_i + r \times C(A_i)$ where r is a positive real number and a user parameter (set to 0.1 by default), and $C(A_i)$ is the cardinality of attribute A_i. We varied the number of customer preferences from 10k (1k) to 40k (4k) where its default value is 10k (1k) in synthetic (real) data sets. We evaluate the algorithms with two measurements, namely the $execution\ time$ of the algorithms and the $influence\ value$ of a selection set returned by the algorithms. We also did experiments to compare the price of a hotel found by our algorithm with its original price. The result is similar to the one shown in Fig. 2a.

Figs. 5 and 6 show the experimental results on a real data sets. In Fig. 5a, GA runs at least 2 orders of magnitude faster than BF. In Fig. 5b, the influence values of the selection sets returned by GA and BF are nearly the same. Similar results can be found in Fig. 6. All the results are consistent with our theoretical result about the 0.63-approximation.

Fig. 7 shows the experiments on large synthetic data sets. Since BF is not scalable, we do not include it in the figure. In the figures, the execution time of GA increases with the size of CP and r.

Summary. In the problem of finding top-k profitable products, although DP finds the optimal solution, it is not

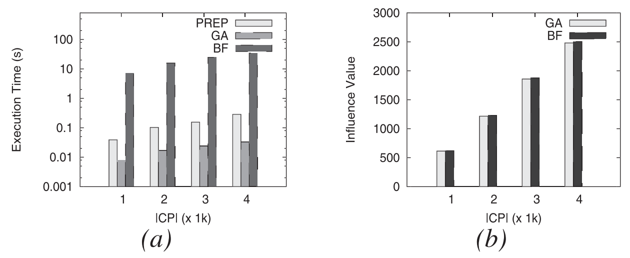


Fig. 5. Effect of $|CP|$ on a real data sets.

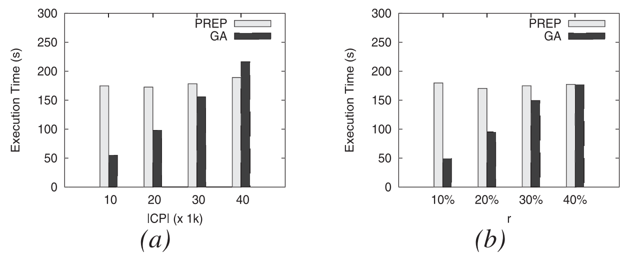


Fig. 7. Effect of $|CP|$ and r on large synthetic data sets.

scalable and is limited to problem TPP when $l = 2$. $GR1$ and $GR2$ are scalable to large data sets. It is shown that they can find a selection set Q' with high profits. In most cases, $GR1$ and $GR2$ return similar profits. However, $GR2$ sometimes gives a higher profit than $GR1$ but the execution time of $GR2$ is greater. In addition, when data sets change, the new incremental algorithm IA performs much more efficient compared with the algorithm which recomputes top-k profitable products from scratch.

In the problem of finding top-k popular products, our proposed greedy algorithm GA performs at least two orders of magnitude faster than a brute-force approach BF in a small data sets. GA is also scalable to large data sets. Besides, the influence values of the selection sets returned by GA and BF are nearly the same.

9 Related Work

Skyline queries have been studied since 1960s in the theory field where skyline points are known as Pareto sets and admissible points [5] or maximal vectors [3]. However, earlier algorithms such as [3], [4] are inefficient when there are many data points in a high-dimensional space. Skyline queries in database was first studied by Borzsonyi [2] in 2001.

After that, various techniques were proposed to accelerate the computation of skyline and its variations. Here, we briefly summarize some of them. Some representative methods include a bitmap method [16], a nearest neighbor (NN) algorithm [10], and a branch-and-bound skylines (BBS) method [13].

Top-K queries about skyline were studied in [13], [12], [17]. Papadias et al. [13] discussed *ranked skyline* and *K-dominating queries*. Given a set of points in d-dimensional space, *ranked skyline* specifies a monotone ranking function, and returns k tuples in the d-dimensional space which have the smallest (or greatest) scores according to an input function. Given a set of points in d-dimensional space, *K-dominating queries* retrieve K points that dominate the greatest number of points. It is similar to the h-dominance constraint introduced in Section 5.

Lin et al. [12], [17] studied *representative skyline queries*. The problem is to select k points among all skyline points according to a predefined objective function. The k points in the output are said to be representative.

Lin et al. [12] were the first to introduce *representative skyline queries*. Lin et al. [12] find a set of k points among all skyline points such that the number of points dominated by this set is maximized. However, the method in [12] cannot be applied in our problem because we consider both the profitability (or popularity) of products and the dominance relation of products, but Lin et al. [12] consider the dominance relation only. Besides, the price of each product is to be found in our problem.

Another definition of representative skyline queries was proposed by Tao et al. [17]. In [17], representative skyline queries are to find k points (or k representative points) among all skyline points such that the sum of the distances between each skyline point and its "closest" representative point is minimized.

All of the above studies are to find k points or tuples given a *single* table where all attribute values of each tuple in the table are *given*. This paper has the following differences. First, we want to find k tuples given *two* tables (one is P and the other is Q) where one of the attribute values of each tuple in one table (Q) is *not* given and is to be found. Second, the concept of *profit* is considered in this paper but not in the above studies.

Although some existing studies focus on profit optimization, since they do not consider the skyline techniques, they are different from this paper. The two representative studies are [1] and [11]. Other related studies about profit optimization can be found in a recent work [11]. Archak et al. [1] use an existing economic model, Rosen's hedonic price model, to find customer preferences from reviews given by customers in the web and then find profitable products. Li et al. [11] propose a regression model to find profitable products. But, since Ghose et al. [1] and [11] do not consider the skyline techniques, some products found by Ghose et al. [1], [11] can dominate some other products and thus these studies cannot guarantee that the products found are in the skyline, which is one of the goals studied in this paper.

The most closely related work is [18]. Given a set P of existing tuples and a number of source tables, Wan et al. [18] find all tuples "generated" from the source tables such that these tuples are in the skyline with respect to the tuples in the existing market. Those tuples are called competitive tuples or products. Note that the set of all competitive tuples generated in [18] can be regarded as set Q described in this

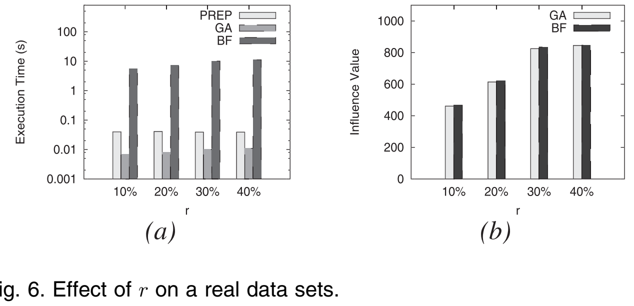


Fig. 6. Effect of r on a real data sets.

paper. However, in [18], too many competitive products are generated. In their experimental studies, there are 10,000 competitive products in a real data sets. In most cases, it is good to choose some of the competitive products instead of all for promotion. One criterion is to maximize the total profit of the selection set which is studied in the first problem of this paper.

A preliminary version of this paper was published in [20]. However, [20] does not consider the dynamic case for finding top-k profitable products and the problem of finding top-k popular products which are studied in this paper.

10 Conclusion

In this paper, we identify and tackle the problem of finding top-k preferable products, which has not been studied before. We study two instances of preferable products, namely profitable products and popular products. We propose methods to find top-k profitable products and top-k popular products efficiently. An extensive performance study using both synthetic and real data sets is reported to verify its effectiveness and efficiency. As future work, we will study other instances of the problem of finding top-k preferable products by setting the utility function to other meaningful objective functions. One promising utility function is the function which returns the sum of the unit profits of the selected products multiplied by the number of customers interested in these products.

Appendix

Proof of Lemmas/Theorems

Proof of Theorem 2. Note that problem TPP (in Problem 1) is a maximization problem. In order to show the NP-hardness of the problem we are studying, we first give a *d*ecision problem for problem *TPP* called *DTPP*: given a nonnegative real number X, does there exist a set Q' of k tuples from Q such that $Profit_o(Q') \geq X$?

The NP-hardness proof can be achieved by transforming an NP-complete problem, the d-coverage problem [19], to the DTPP problem.

d**-coverage problem.** Given a set U of elements, a collection J of sets containing elements in U, a positive integer d and a positive integer t where $1 < d \leq |J|$ and $t \leq |U|$, does there exist a subset $I \subseteq J$ such that $|I| = d$ and $| \cup_{C \in I} C| let$?

Given an instance of the d-coverage problem. We want to construct an instance of the DTPP problem from the above instance. We define two positive real numbers, M and m, where $M >> m$ and $\frac{M}{2} > m$. We construct the instance as follows. We set $\sigma = \frac{M}{2}$, $k = d + |U|$, $l = |U| + |J| + 1$, and $X = d \cdot m + |U| \cdot M - t \cdot \frac{M}{2}$. Next, we set P and Q according to U and J. Specifically, we set P to be the set containing only one tuple p, and Q to be the set containing $|U| + |J|$ tuples.

Note that tuple p is associated with l attributes, namely $A_1, A_2, \ldots, A_l$, where the last attribute A_l corresponds to attribute Price. All l attribute values of p are to be set. Besides, each tuple in Q is also associated with the same l attributes together with an additional cost attribute C where only the first $l - 1$ attribute values and the cost attribute value of each tuple in Q is to be set for the problem instance construction.

Now, we describe in detail how we generate $|U| + |J|$ tuples in Q. Initially, Q is an empty set. Then, for each set S in J, we create a tuple q_S and insert it into Q. There are $|J|$ tuples generated from J and these tuples are called *type I tuples*. Each of these tuples is arbitrarily given a unique *label* which is a positive integer $\in [1, |J|]$. For each element e in U, we create a tuple q_e and insert it into Q. There are $|U|$ tuples generated from U and these tuples are called *type II tuples*. Similarly, each of these tuples is arbitrarily given a unique label which is a positive integer $\in [|J| + 1, |U| + |J|]$. For each set S in J and each element e in S, we say that tuple q_S (of type I) is a *parent* of tuple q_e (of type II).

Next, we set the attribute values of the tuples in P and Q. Consider tuple p in P. For each $j \in [1, l - 1]$, we set $p.A_j$ to m. We set $p.A_l$ to $M + m + \sigma$. Consider tuples in Q. For each $q_i \in Q$, we set $q_i.C$ to M if q_i is a type I tuple and set it to m if q_i is a type II tuple. Then, we set the first $l - 1$ attribute values with the following three major steps. For Step 1, we initialize a variable i to 1. Let the tuple with label equal to i in Q be q_i. We set $q_i.A_i$ to any real number b such that $m < b < M$. Note that $q_i.A_j$ is not set in this step and will be set in later steps for $j \in [i + 1, l - 1]$. For Step 2, we increment the variable i by 1. Tuple q_i is another tuple with label equal to an updated value i. We execute three substeps. *Step 2a:* we set $q_i.A_j$ to be M for $j \in [1, i - 1]$. *Step 2b:* we set $q_i.A_i$ to be any real number b such that $m < b < M$. Similarly, note that $q_i.A_j$ is not set and will be set in later steps for $j \in [i + 1, l - 1]$. *Step 2c:* for $y \in [1, i - 1]$, we set $q_y.A_i$ to m if q_y is a parent of q_i, and we set it to M otherwise. *Step 3:* we repeat Step 2 until i is equal to $|U| + |J|$. After this construction of Q, we know that a tuple q' is a parent of another tuple q if and only if q' quasidominates q.

We have just defined the constructed problem instance for DTPP. Next, we analyze some properties from the final solution of this constructed problem instance. Note that in the final solution of DTPP, we have to set the attribute A_l value of each tuple q in Q'. In the following, we determine how to set the A_l value of each tuple in Q'.

It is easy to verify that for each tuple $q \in Q$, q is quasidominated by tuple p. Now, we consider how to set the A_l value of a tuple q in Q'. Consider two cases: *Case 1:* q is a type I tuple. $q.A_l$ must be set to $M + m$ so that the profit of q is the greatest and q is in the skyline with respect to $P \cup Q'$ (no matter what Q' is). This is because if $q.A_l$ is set to a value greater than $M + m$, then q is dominated by p. Besides, q is not dominated by any other tuples in Q' no matter what the A_l values of these tuples in Q' are set. In Case 1, the profit of q is equal to $(M + m) - M = m$. *Case 2:* q is a type II tuple. $q.A_l$ is set to different values according to two different subcases. *Case 2a:* there does not exist any type I tuple in Q' which quasidominates tuple q. This case is similar to Case 1. $q.A_l$ must be set to $M + m$ so that the profit of q is the greatest and q is in the skyline with respect to $P \cup Q'$ (no matter what Q' is). In Case 2a, the profit of q is equal to $(M + m) - m = M$. *Case 2b:* there exists a type I tuple q'

in Q' which quasidominates tuple q. Note that q' must be a parent of q. Besides, according to Lemma 1, since q' quasidominates q, the value of $q'.A_l$ can be determined before $q.A_l$ is to be set. According to Case 1, $q'.A_l$ is set to $M + m$. On the other hand, $q.A_l$ must be set to $M + m - \sigma$ so that the profit of q is the greatest and q is in the skyline with respect to $P \cup Q'$ (no matter what Q' is). This is because if $q.A_l$ is set to a value greater than $M + m - \sigma$, says $M + m$ (in Case 2a), then q is dominated by q'. Besides, q is not dominated by any other tuples in Q' no matter what the A_l values of the tuples in Q' other than q' are set. In Case 2b, the profit of q is equal to $(M + m - \sigma) - m = M - \sigma = \frac{M}{2}$.

According to the above strategy to set the A_l value of each tuple in Q' and $\frac{M}{2} > m$, we conclude that the profit of each type II tuple in Q' is greater than the profit of each type I tuple in Q' (no matter what Q' is). Since $k = d + |U|$ and k is the total number of tuples in Q', the final selection set Q' must contain all $|U|$ type II tuples and exactly d type I tuples. The total profit of all the type I tuples in Q' is equal to $d \cdot m$. Let f be the total number of type II tuples in Q' each of which is quasidominated by at least one type I tuple in Q'. The total profit of all $|U|$ type II tuples is equal to $(|U| - f) \cdot M + f \cdot \frac{M}{2} = |U| \cdot M - f \cdot \frac{M}{2}$. Thus, the profit of Q', denoted by $Profit_o(Q')$, is $d \cdot m + |U| \cdot M - f \cdot \frac{M}{2}$.

It is easy to verify that there exists a set Q' of k tuples from Q such that $Profit_o(Q') \geq d \cdot m + |U| \cdot M - t \cdot \frac{M}{2}$ (and thus $f \leq t$) if and only if there exists a set $I \subseteq J$ such that $|I| = d$ and $| \cup_{C \in I} C| \leq t$. Since the d-coverage problem is NP-complete, the DTPP problem is NP-hard. □

Proof of Lemma 4. Since $p_{new} \notin SKY(P_{new})$, we have $SKY(P_{new}) = SKY(P)$. It is easy to verify $Q'_{new} = Q'$. □

Proof of Lemma 5. Suppose $Q'_{new} = Q'$. Since Q'_{new} is the answer of the problem, each tuple in Q'_{new} is in $SKY(P_{new} \cup Q'_{new})$. Since $Q'_{new} = Q'$, we deduce that each tuple in Q' is in $SKY(P_{new} \cup Q')$. Note that $p_{new} \in P_{new}$. We conclude that there does not exist $q \in Q'$ such that p_{new} dominates q.

Suppose that there does not exist $q \in Q'$ such that p_{new} dominates q. We want to show that $Q'_{new} = Q'$ by dividing the proof into three parts.

First, we show that each tuple q in Q' is in $SKY(P_{new} \cup Q')$. Before p_{new} is inserted into P, we know that each tuple q in Q' is in $SKY(P \cup Q')$. Consider a tuple q in Q'. We know that no tuples in $P \cup Q'$ dominate q. Since there does not exist $q \in Q'$ such that p_{new} dominates q, we deduce that no tuples in $P \cup Q' \cup \{p_{new}\} (= P_{new} \cup Q')$ dominate q. We conclude that q is in $SKY(P_{new} \cup Q')$.

Second, we show that Q' is the set of k tuples from Q such that $Profit_o(Q') = \max_{Q'' \in \mathcal{Q}} Profit_o(Q'')$ where $\mathcal{Q}$ is the set of all possible subsets containing k tuples from Q. Let $Profit_o(Q', P)$ be the optimal profit of Q' based on data sets Q and P. Thus, $Profit_o(Q') = Profit_o(Q', P)$ before p_{new} is inserted into P while $Profit_o(Q') = Profit_o(Q', P \cup \{p_{new}\})$ after p_{new} is inserted into P. Before p_{new} is inserted into P, we know that for each $Q'' \in \mathcal{Q}$, we have

$$Profit_o(Q', P) \geq Profit_o(Q'', P). \tag{1}$$

Since there does not exist $q \in Q'$ such that p_{new} dominates q, we deduce that

$$Profit_o(Q', P \cup \{p_{new}\}) = Profit_o(Q', P). \tag{2}$$

Besides, for each $Q'' \in \mathcal{Q}$, we know that

$$Profit_o(Q'', P) \geq Profit_o(Q'', P \cup \{p_{new}\}). \tag{3}$$

From (1), (2), and (3), we conclude that for each $Q'' \in \mathcal{Q}$, $Profit_o(Q', P_{new}) \geq Profit_o(Q'', P_{new})$.

Lastly, since each tuple q in Q' is in $SKY(P_{new} \cup Q')$ and Q' is the set of k tuples from Q such that $Profit_o(Q') = \max_{Q'' \in \mathcal{Q}} Profit_o(Q'')$, we conclude that $Q'_{new} = Q'$. □

Proof of Theorem 5. Let $SP_{before}(q)$ be the stand-alone profit of tuple $q \in Q$ calculated before p_{new} is inserted into P. Let $SP_{after}(q)$ be the stand-alone profit of tuple $q \in Q$ calculated after p_{new} is inserted into P. It is easy to verify that for each $q \in Q$, $SP_{after}(q)$ is smaller than or equal to $SP_{before}(q)$. Consider three cases. *Case 1:* $p_{new} \notin SKY(P_{new})$.We know that for each $q \in Q$, $SP_{before}(q) = SP_{after}(q)$. Thus, the k tuples in Q which have the greatest stand-alone price after p_{new} is inserted are exactly the same as the tuples in Q'. Thus, since $Q'_{new} = Q'$, the selection set returned by Algorithm 4 is the selection set returned by Greedy Algorithm (Version 1). *Case 2: There does not exist* $q \in Q'$ *such that* p_{new} *dominates* q.By using the techniques used in the proof of Lemma 5, we also derive that the k tuples in Q which have the greatest stand-alone price after p_{new} is inserted are exactly the same as the tuples in Q'(because for each $q \in Q$, $SP_{after}(q) \leq SP_{before}(q)$ and for each $q \in Q'$, $SP_{after}(q) = SP_{before}(q)$). *Case 3:* $p_{new} \in SKY(P_{new})$ *and there exists* $q \in Q'$ *such that* p_{new} *dominates* q. All the tuples in Q which stand-alone prices are changed are updated in Lines 10-11. The output of Algorithm 4 (i.e., the k tuples in Q which have the greatest (updated) stand-alone price (Line 13)) are the selection set returned by Greedy1. □

Proof of Theorem 6. Note that Problem 3 is a maximization problem. We give a decision problem: given a nonnegative real number X, does there exist a set Q' of k tuples from Q such that 1) $IV(Q') \geq X$ and 2) each tuple in Q' is in the skyline with respect to $P \cup Q'$.

The NP-hardness proof can be achieved by transforming an NP-complete problem, the *maximum coverage* problem, to our decision problem.

Maximum coverage. Given a positive integer d, another positive integer t, a set U of elements, and a collection J of sets each of which is a subset of U, does there exist a set $I \subseteq J$ such that $|I| \leq d$ and $| \cup_{s \in I} s| \geq t$?

Given an instance of the maximum coverage problem. We want to construct an instance of our decision problem from the above instance. Recall that in the proof of Theorem 2, we construct an instance of the DTPP problem from a given instance of the d-coverage problem, by creating tuples of types I and II (for the tuples in Q). Note that these tuples are created with their attribute values for attribute A_i where $i = 1, 2, \ldots, l-1$ and their attribute value for attribute Cost C. In this proof, we construct tuples for Q and customer preferences for CP similarly. Let M be a very large positive real number. For each set S in J, we create a type I tuple u and then create a customer preference cp_S in form of $\{g_1, g_2, \ldots, g_l\}$ by setting g_i to be $u.A_i$ for $i \in [1, l-1]$ and setting g_l to M. All customer preferences generated form set CP. For each customer preference cp in CP, we set $w(cp)$ to 1. For each element e in U, we create a type II tuple u and create a tuple q which is exactly equal to u. All tuples generated from U form set Q. We set P, k, and X to $\emptyset, d$, and t, respectively.

It is easy to see that this transformation can be constructed in polynomial time. It is also easy to verify that when the problem is solved in the transformed decision problem, the original maximum coverage problem is also solved. Since the maximum coverage problem is an NP-complete problem, our decision problem is NP-hard. □

Proof of Lemma 6. We prove by contradiction. Suppose there exists a tuple q in Q' which is not in the skyline with respect to $P \cup Q'$. This means that there exists a tuple in $P \cup Q'$ which dominates q. Consider two cases. *Case 1:* the tuple dominating q is a tuple from P. Since $PS(q) = \{v | v \geq q.C$ and $q \in SKY(P \cup \{q\})$ if we set $q.A_l = v\}$, we know that $q \in SKY(P \cup \{q\})$. Thus, there does not exist any tuple in P dominating q. This leads to a contradiction. *Case 2:* the tuple in $P \cup Q'$ dominating q is a tuple in Q' other than q. Let this tuple be q'. Note that q' dominates q. We have $q'.A_i \leq q.A_i$ for each $i \in [1, l]$. Thus, $IS(q) \subseteq IS(q')$. Let Q_i be the selection set Q' maintained by Algorithm 5 at the end of the ith iteration for $i = 1, 2, \ldots, k$. We define $Q_0 = \emptyset$. Since $IS(q) \subseteq IS(q')$, we deduce that $IV(Q_i \cup \{q\}) \leq IV(Q_i \cup \{q'\})$ for each $i \in [1, k]$. We further consider two subcases. *Case 2(a):* $IV(Q_i \cup \{q\}) < IV(Q_i \cup \{q'\})$. We deduce that q' is selected and inserted into the selection set maintained by Algorithm 5 before q is selected and inserted. Consider the iteration of selecting q', says the jth iteration. We have $Q_j = Q_{j-1} \cup \{q'\}$. We conclude that for the lth iteration where $l \in [j+1, k]$, $IV(Q_l \cup \{q\}) = 0$. This leads to the contradiction that $IV(Q_l \cup \{q\}) > 0$ for each $l \in [1, k]$ (This is because if $k \leq k_{max}$, we have $IV(Q_l \cup \{q\}) > 0$ for each $l \in [1, k]$). *Case 2(b):* $IV(Q_i \cup \{q\}) = IV(Q_i \cup \{q'\})$. Since q' dominates q, we deduce that $f(q') < f(q)$. Thus, q' is selected and inserted into the selection set maintained by Algorithm 5 before q is selected and inserted. We have a similar conclusion as Case 2(a). □

Proof of Theorem 7. We can transform our problem to the maximum coverage problem by mapping each customer preference and the influence set of each tuple in Q for our problem to an element and a set of elements in the maximum coverage problem, respectively. Note that the greedy algorithm for the maximum coverage problem which chooses the set containing the largest number of *uncovered* elements is 0.63-approximate [6]. Since Algorithm 5 follows the same framework as the above greedy algorithm, it is 0.63-approximate. □

ACKNOWLEDGMENTS

The authors are grateful to the anonymous reviewers for their constructive comments on this paper. The research was supported by HKRGC GRF 621309.

REFERENCES

[1] N. Archak, A. Ghose, and P.G. Ipeirotis, "Show Me the Money!: Deriving the Pricing Power of Product Features by Mining Consumer Reviews," *Proc. 13th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '07),* pp. 56-65, 2007.
[2] S. Borzsonyi, D. Kossmann, and K. Stocker, "The Skyline Operator," *Proc. Int'l Conf. Data Eng. (ICDE),* 2001.
[3] J.L. Bently, H.T. Kung, M. Schkolnick, and C.D. Thompson, "On the Average Number of Maxima in a Set of Vectors and Applications," *J. ACM,* vol. 25, no. 4, pp. 536-543, 1978.
[4] J.L. Bently, K.L. Clarkson, and D.B. Levine, "Fast Linear Expected-Time Algorithms for Computing Maxima and Convex Hulls," *Proc. First Ann. ACM-SIAM Symp. Discrete Algorithms (SODA),* 1990.
[5] O. Barndorff-Nielson and M. Sobel, "On the Distribution of the Number of Admissible Points in a Vector Random Sample," *Theory of Probability and Its Application,* vol. 11, no. 2, pp. 249-269, 1966.
[6] D.S. Hockhbaum, "Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems" *Approximation Algorithms for NP-Hard Problems,* PWS Publishing Company, 1997.
[7] B. Jiang, J. Pei, X. Lin, D.W.-L. Cheung, and J. Han, "Mining Preferences from Superior and Inferior Examples," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* 2008.
[8] J.M. Kang, M.F. Mokbel, S. Shekhar, T. Xia, and D. Zhang, "Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors," *Proc. Int'l Conf. Data Eng. (ICDE),* 2007.
[9] F. Korn and S. Muthukrishnan, "Influence Sets Based on Reverse Nearest Neighbor Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2000.
[10] D. Kossmann, F. Ramsak, and S. Rost, "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB),* 2002.
[11] B. Li, A. Ghose, and P.G. Ipeirotis, "Towards a Theory Model for Product Search," *Proc. 20th Int'l Conf. World Wide Web (WWW '11),* pp. 327-336, 2011.
[12] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting Stars: The k Most Representative Skyline Operator," *Proc. Int'l Conf. Data Eng. (ICDE),* 2007.
[13] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Trans. Database Systems,* vol. 30, no. 1, pp. 41-82, 2005.
[14] D. Sacharidis, S. Papadopoulos, and D. Papadias, "Topologically-Sorted Skylines for Partially-Ordered Domains," *Proc. Int'l Conf. Data Eng. (ICDE),* 2009.
[15] I. Stanoi, M. Riedewald, D. Agrawal, and A.E. Abbadi, "Discovery of Influence Sets in Frequently Updated Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2001.
[16] K.-L. Tan, P. Eng, and B. Ooi, "Efficient Progressive Skyline Computation," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2001.
[17] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-Based Representative Skyline," *ICDE '09: Proc. IEEE Int'l Conf. Data Eng.,* pp. 892-903, 2009.
[18] Q. Wan, R.C.-W. Wong, I.F. Ilyas, M.T. Özsu, and Y. Peng, "Creating Competitive Products," *Proc. VLDB Endowment,* vol. 2, pp. 898-909, 2009.
[19] Q. Wan, R.C.-W. Wong, and Y. Peng, "Creating Top-K Profitable Products," technical report, http://www.cse.ust.hk/~raywong/paper/createTopKProfitableProduct-technical.pdf, 2010.
[20] Q. Wan, R.C.-W. Wong, and Y. Peng, "Finding Top-K Profitable Products," *Proc. Int'l Conf. Data Eng. (ICDE),* 2011.

[21] R.C.-W. Wong, J. Pei, A.W.-C. Fu, and K. Wang, "Mining Favorable Facets," *Proc. 13th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* 2007.
[22] T. Xia, D. Zhang, E. Kanoulas, and Y. Du, "On Computing Top-k Most Influential Spatial Sites," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB),* 2005.
[23] C. Yang and K.-I. Lin, "An Index Structure for Improving Nearest Closest pairs and Related Join Queries in Spatial Databases," *Proc. Int'l Symp. Database Eng. and Applications (IDEAS),* 2002.
[24] Z. Zhang, L. Lakshmanan, and A.K. Tung, "On Domination Game Analysis for Microeconomic Data Mining," *ACM Trans. Knowledge Discovery from Data,* vol. 2, article 18, 2009.

Yu Peng received the BSc degree from the South China University of Technology (SCUT) in 2006 and the MPhil degree from the Sun Yat-sen University (SYSU) in 2008, respectively. She is currently working toward the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology (HKUST). Her research interests are in the areas of data mining and database.

Raymond Chi-Wing Wong received the BSc, MPhil, and PhD degrees in computer science and engineering from the Chinese University of Hong Kong (CUHK) in 2002, 2004, and 2008, respectively. He joined the Department of Computer Science and Engineering, Hong Kong University of Science and Technology as an assistant professor in 2008. His research interests include database, data mining and security.

Qian Wan received the BSc degree in computer science from Nankai University in 2008, and the MPhil degree from the Hong Kong University of Science and Technology (HKUST) in 2010, respectively. He is currently working toward the PhD degree from the Department of Computer Science, University of Wisconsin-Madison. His research interests are in the area of spatial queries, approximation algorithm, data mining, computational finance modeling and map reduce.