Finding Top-k Profitable Products

Qian Wan, Raymond Chi-Wing Wong, Yu Peng

Computer Science and Engineering Department, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{qwan,raywong,gracepy}@cse.ust.hk

Abstract— **The importance of dominance and skyline analysis has been well recognized in multi-criteria 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 profitable 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. In this problem, 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. A straightforward solution is to enumerate all possible subsets of size k and find the subset which gives the greatest profit. However, there are an exponential number of possible subsets. In this paper, we propose solutions to find the top-k profitable products efficiently. An extensive performance study using both synthetic and real datasets is reported to verify its effectiveness and efficiency.**

I. Introduction

Dominance analysis is important in many multi-criteria 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 I shows four packages: p_1, p_2, p_3 and p_4. We know that lower price and shorter distance-to-beach are more preferable. Thus, p_3 dominates p_4 because p_3 has lower price and shorter distance-to-beach than p_4. However, p_3 does not dominate p_1 because p_1 has lower price than p_3. Similarly, p_1 does not dominate p_3 because p_3 has 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 [10], [6], [14], [5], [9], [15]. The packages in the skyline are the best possible tradeoffs among the two factors in question. In Example 1, p_3 is in the skyline because it is not dominated by p_1, p_2 and p_4. However, p_4 is not in the skyline because it is dominated by p_3.

Consider that a new travel agency wants to start some new packages from a pool of potential packages as shown in Table II. Table II 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.

TABLE I
PACKAGES IN THE EXISTING MARKET

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

TABLE II
POTENTIAL PACKAGES IN THE NEW TRAVEL AGENCY

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

Example 2 (Profitable Price with One Package): Suppose that we select only one *new package, says 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 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 Example 2, 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 price of q_1 is set to \$250 and the price of q_2 is set to \$300. 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 profit of q_1 and the profit of q_2 are \$150 and \$100, respectively. The sum of these profits is equal to \$250.* □

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 k packages from Q such that the sum of the profits of the selected packages is maximized and each selected package is not dominated by any packages in the existing market and any selected new packages where k is a positive integer and is a user parameter. We call this problem *finding top-k profitable products (TPP)*. For example, k is equal to 1 in Example 2 while k is equal to 2 in Example 3.

A naive way for this 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.

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. However, 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.

Our contribution are summarized as follows.

1) To the best of our knowledge, we are the first to study how to find top-k profitable products. Finding top-k profitable 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.
2) 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.
3) We present a systematic performance study using both real and synthetic datasets to verify the effectiveness and the efficiency of our proposed approaches. The experimental results show that finding top-k profitable products is interesting.

The rest of the paper is organized as follows. In Section II, we formally define our problem. In Section III, we describe an algorithm for finding the sum of the profits given a set of k selected packages. The dynamic programming approach is developed in Section IV while the greedy approaches are presented in Section V. In Section VI, we give some discussions of the proposed approaches. A systematic performance study is reported in Section VII. In Section VIII, we describe some related work. The paper is concluded in Section IX.

II. Problem Definition

We first describe the background about skyline in Section II-A. Then, we give some notations used in our problem in Section II-B and our problem in Section II-C.

A. Background: Skyline

A skyline analysis involves multiple attributes. The values in each attribute can be modeled by a partial order on the attribute. A *partial order* $\preceq$ is a reflexive, asymmetric and transitive relation. A partial order is also a total order if for any two values u and v in the domain, either $u \preceq v$ or $v \preceq u$. We write $u \prec v$ if $u \preceq v$ and $u \neq v$.

By default, we consider tuples in an w-dimensional[1] space $\mathbb{S} = x_1 \times \cdots \times x_w$. For each dimension x_i, we assume that there is a partial or total order. For a tuple p, $p.x_i$ is the projection on dimension x_i. For dimension x_i, if $p.x_i \preceq q.x_i$, we also simply write $p \preceq_{x_i} q$. We can omit x_i if it is clear from the context.

For tuples p and q, p *dominates* q with respect to $\mathbb{S}$, denoted by $p \prec q$, if for any dimension $x_i \in \mathbb{S}$, $p \preceq_{x_i} q$, and there exists a dimension $x_{i_0} \in \mathbb{S}$ such that $p \prec_{x_{i_0}} q$. If p dominates q, then p is more preferable than q.

Definition 1 (Skyline): Given a dataset $\mathcal{D}$ containing tuples in space $\mathbb{S}$, a tuple $p \in \mathcal{D}$ is in the *skyline* of $\mathcal{D}$ (i.e., a *skyline tuple* in $\mathcal{D}$) if p is not dominated by any tuples in $\mathcal{D}$. The skyline of $\mathcal{D}$, denoted by $SKY(\mathcal{D})$, is the set of skyline tuples in $\mathcal{D}$. □

For example, in Table I where $\mathcal{D} = \{p_1, p_2, p_3, p_4\}$, since p_1, p_2 and p_3 are not dominated by any tuples in $\mathcal{D}$, $SKY(\mathcal{D})$ is equal to $\{p_1, p_2, p_3\}$.

B. Notations

We have a set P of m tuples in the existing market, namely $p_1, p_2, ..., p_m$. Each tuple p has l attributes, namely $A_1, A_2, ..., A_l$. The domain of each attribute is $\mathbb{R}$. 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 $q_1, q_2, ..., q_n$. Similarly, each tuple q has the

[1] In this paper, we use the terms "*attribute*" and "*dimension*" interchangeably.

same l attributes, namely $A_1, A_2, ..., A_l$. The value of attribute A_j for tuple q is denoted by $p.A_j$ 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 $A_1, A_2, ..., A_{l-1}$). 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 running example, P contains 4 tuples, namely p_1, p_2, p_3 and p_4 (Table I), and Q contains 3 tuples, namely q_1, q_2 and q_3 (Table II). Attribute A_1 and attribute A_2 are "Distance-to-beach" and "Price", respectively. Attribute C is "Cost".

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 $D = \{0, \sigma, 2\sigma, 3\sigma, ...\}$ 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 life applications where attribute Price involves discrete values instead of continuous real values. For example, attribute Price has value "\$500.00" and value "\$1000.00" but no value "\$1000.0007". In the following, we set $\sigma = 50$ for our running example.

C. 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

$$v - q.C$$

In our running example, if we set $q_1.A_l$ to be \$300, then $\triangle(q_1, 300)$ is equal to \$300 - \$100 = \$200. If we set it to be \$250, then $\triangle(q_1, 250)$ is equal to \$250 - \$100 = \$150.

We define a *price assignment vector* of Q', denoted by $\mathbf{v}$, in form of $(v_1, v_2, ..., v_n)$. v_i is said to be the i-th 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.

Suppose that q is selected in the final selection set Q'. Our objective is to make sure that after we set the price of q, q is not dominated by not only the tuples in the existing market (P) but also the newly selected tuples in Q'. That is, q should be in the skyline with respect to $P \cup Q'$.

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 $P \cup Q'$.

Example 4 (Feasible Vector): In Example 3, $k = 2$. Suppose that $Q' = \{q_1, q_2\}$.

Consider that we set the price of q_1 to \$300 and the price of q_2 to \$300. The price assignment vector $\mathbf{v}$ is $(300, 300, 0)$. $\mathbf{v}$ is not feasible. As described in Example 3, after we set the price for q_1 and the price for q_2, q_1 is not in the skyline with respect to $P \cup Q'$ because q_1 is dominated by q_2.

Consider that we set the price of q_1 to \$250 (instead of \$300) and the price of q_2 to \$300. The price assignment vector $\mathbf{v}'$ is $(250, 300, 0)$. $\mathbf{v}'$ is feasible because after we set the price for q_1 and the price for q_2, both q_1 and q_2 are in the skyline with respect to $P \cup Q'$. □

Definition 2 (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 $(v_1, v_2, ..., v_n)$. The *profit* of Q' with $\mathbf{v}$, denoted by $Profit(Q', \mathbf{v})$, is defined to be

$$\sum_{q_i \in Q'} \triangle(q_i, v_i)$$

□

Example 5 (Profit of Selection): In Example 3, $k = 2$. Suppose that $Q' = \{q_1, q_2\}$.

Consider that the selection set is Q'. We set the price of q_1 to \$250 and the price of q_2 to \$300. The price assignment vector $\mathbf{v}'$ is $(250, 300, 0)$. From Example 4, we know that this vector is feasible.

$Profit(Q', \mathbf{v}')$ is equal to

$$\begin{aligned} & \triangle(q_1, v_1) + \triangle(q_2, v_2) \\ = \; & (\$250 - \$100) + (\$300 - \$200) \\ = \; & \$150 + \$100 \\ = \; & \$250 \end{aligned}$$

Consider that the selection set is still Q'. We set the price of q_1 to \$150 (instead of \$250) and the price of q_2 to \$300. The price assignment vector $\mathbf{v}''$ is $(150, 300, 0)$. It is easy to verify that this vector is also feasible. Similarly, we obtain that $Profit(Q', \mathbf{v}'')$ is equal to \$200. □

In the above example, we learn that even though we select the same two packages (i.e., q_1 and q_2), different price assignment vectors give different profits. It is obvious that price assignment vector $\mathbf{v}'$ is better than price assignment vector $\mathbf{v}''$ because it gives more profit.

Since the company wants to maximize its profit, it wants to find a price assignment vector which maximizes its profit.

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

We just learnt 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 and k tuples will be selected by some customers.

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 top-k profitable products (TPP)*.

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 IV and Section V.

III. Finding Optimal Price Assignment

In this section, we present an algorithm for finding the optimal price assignment called *AOPA* in $O(k(\log(m+n)+N))$ 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 *quasi-dominate* p' if (1) p dominates p' with respect to the first $l-1$ attributes, namely $A_1, A_2, ..., A_{l-1}$, or (2) p has the same $l-1$ attribute values as p'. In our running example, $l = 2$. Suppose that $Q' = \{q_1, q_2\}$, p_2 quasi-dominates both p_1 and q_1 since p_2 dominates both p_1 and q_1 with respect to attribute "Distance-to-Beach". p_2 also quasi-dominates q_2. Let $\gamma(X, q_i)$ be a set containing all tuples in X which quasi-dominate q_i. For example, suppose that $Q' = \{q_1, q_2\}$. $\gamma(X, q_1) = \{q_2, p_2, p_3, p_4\}$ and $\gamma(X, q_2) = \{p_2, p_3, p_4\}$.

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

Lemma 1: Suppose that $p \in X$ and $q_i \in Q'$. Consider that we are given a price assignment vector of Q' equal to $\mathbf{v} = (v_1, v_2, ..., v_n)$ 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'.
  In our running example, let $Q' = \{q_1, q_2\}$. Consider q_1. Since p_1 is not in $\gamma(X, q_1)$, we know that p_1 does not dominate q_1.

TABLE III
THE f VALUE OF EACH TUPLE $p \in Q$

Tuple	f
q_1	5.0
q_2	4.5
q_3	0.5

- **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.
  For example, consider q_1 again. Consider a price assignment vector $\mathbf{v} = (300, 300, 0)$. Note that q_2 is in $\gamma(X, q_1)$. After we set the prices of q_1 and q_2 with vector $\mathbf{v}$, q_2 dominates q_1.
  Consider another price assignment vector $\mathbf{v}' = (250, 300, 0)$. After we set the prices of q_1 and q_2 with vector $\mathbf{v}'$, q_2 does 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 quasi-dominating 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 quasi-dominance 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' quasi-dominating 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 follows.

$$f(q) = \sum_{i=1}^{l-1} q.A_i$$

In our running example, $l = 2$. The f value of each tuple $q \in Q$ can be found in Table III. The ordering of tuples in X is q_3, q_2 and q_1.

With this function f, we know the following lemma.

Lemma 2: Suppose p and p' are in X. If p quasi-dominates p', then $f(p)$ is smaller than or equal to $f(p')$. □

For example, since p_2 quasi-dominates p_1, $f(p_2)(= 4)$ is smaller than $f(p_1)(= 7)$.

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 $\mathbf{v}$
```

Algorithm 2 Algorithm **findOptimalIncrementalPrice**($q_i, Q_{i-1}, \mathbf{v}_{i-1}$)

```
Input: A set $Q_{i-1}(= \{q_1, q_2, ..., q_{i-1}\})$, tuple $q_i$ in $Q'$ 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 $\mathbf{v}_i$
```

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 $q_1, q_2, ..., q_k$ 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 $i = 1, 2, 3, ..., k$.

Lemma 3: 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 $\mathbf{v}_{i-1} = (v_1, v_2, ..., v_n)$ 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 the 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.

With Lemma 3, it is easy to verify the following theorem.

Theorem 1: Given a set Q' of k tuples selected from Q, Algorithm AOPA returns the optimal price assignment of Q'.

□

Implementation and Time Complexity: In Algorithm 2, the most time-consuming operation is the step of finding all tuples in $P \cup Q'$ which quasi-dominate q_i. One possible implementation is to build an R*-tree index on dataset $P \cup Q'$ according to the first $l-1$ attributes. If we perform a range query with the range equal to "$A_i \leq q_i.A_i$" for each $i \in [1, l-1]$, then we can find all tuples in $P \cup Q'$ which quasi-dominate q_i. However, with this implementation, we have to build different indexes on $P \cup Q'$ for different selection sets Q', which is not efficient.

Another possible implementation is to build an R*-tree index on dataset $P \cup Q$ (instead of $P \cup Q'$) according to the first $l-1$ attributes. Similarly, we perform a range query with the same range as above and find all tuples in $P \cup Q$ which quasi-dominate q_i. In this implementation, since $Q' \subseteq Q$, we can do a post-processing step to select all tuples in $P \cup Q'$ which quasi-dominate q_i. This implementation has its advantage that we only need to build an index once for any selection set. We adopt this implementation in our experiment.

Suppose that we are given an R*-tree index on dataset $P \cup Q$ in the second implementation. We want to analyze the time complexity of the step of finding all tuples in $P \cup Q'$ which quasi-dominate q_i. In most cases, the cost of a range query is $O(\log(|P| + |Q|) + N)$ where N is the total number of tuples returned in a range query. Typically, N is extremely small compared with $(|P| + |Q|)$. That is, $N << |P| + |Q|$. The post-processing step takes $O(N)$ time. Thus, the step of finding all tuples in $P \cup Q'$ which quasi-dominate q_i takes $O(\log(|P| + |Q|) + N)$ time.

After we analyze the time complexity of this time-consuming operation, it is easy to verify that Algorithm 2 takes $O(\log(|P| + |Q|) + N)$ time.

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

IV. Dynamic Programming

In this section, we present a dynamic programming approach which finds an optimal solution for problem TPP when $l = 2$.

Consider $l = 2$. Without loss of generality, we assume that $q_1, q_2, .., q_n$ are sorted in ascending order of the f values. $Q(i)$ is defined to be a set of tuples in Q such that these tuples are quasi-dominated by q_i. Let $S(i, t)$ denote 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 $Q(i)$. Let $\mathbf{v}(i, t)$ denote the optimal price assignment vector of set $S(i, t)$. Let $T(i, t)$ denote the (optimal) profit of set $S(i, t)$.

Let us use a symbol $\alpha(q_i, S, \mathbf{v})$ to represent **findOptimalIncrementalPrice**($q_i, S, \mathbf{v}$).

In the following, we describe how to find three variables, namely $\mathbf{v}(i, t)$, $T(i, t)$ and $S(i, t)$. Consider two cases.

- **Case 1:** q_i is included in the final selection of size t.
  By Lemma 3, the optimal price assignment vector of $S(i, t)$, denoted by $\mathbf{v}(i, t)$, can be obtained from the

optimal price assignment vector of $S(i-1,t-1)$, denoted by $\mathbf{v}(i-1,t-1)$.
Thus, we have the following equation.

$$\mathbf{v}(i,t) = \alpha(q_i, S(i-1,t-1), \mathbf{v}(i-1,t-1)) \quad (1)$$

Thus,

$$T(i,t) = T(i-1,t-1) + v \quad (2)$$

where v is the i-th entry in $\mathbf{v}(i,t)$. Let $T_{select} = T(i-1,t-1)+v$.
Similarly, $S(i,t)$ can be obtained as follows.

$$S(i,t) = S(i-1,t-1) \cup \{q_i\} \quad (3)$$

- **Case 2:** q_i is not included in the final selection.
  We have

$$\mathbf{v}(i,t) = \mathbf{v}(i-1,t) \quad (4)$$

  We have

$$T(i,t) = T(i-1,t) \quad (5)$$

  and

$$S(i,t) = S(i-1,t) \quad (6)$$

  Let $T_{notSelect} = T(i-1,t)$.

Note that we want to maximize the profit of the selection set of size t. Obviously, if $T_{select} \geq T_{notSelect}$, we should select q_i in the selection set (which corresponds to Case 1). Otherwise, we should not select q_i (which corresponds to Case 2).

The pseudo-code of the dynamic programming approach is shown in Algorithm 3.

Theorem 2: Algorithm Dynamic Programming returns an optimal solution Q' of size k for problem TPP (i.e., the set Q' of size k with the greatest profit) when $l = 2$.

Time Complexity: Consider Algorithm 3. Statements from line 1 to line 3 takes $O(n)$ time while statements from line 4 to line 6 takes $O(k \cdot k(\log(m+n)+N)) = O(k^2(\log(m+n)+N))$ time. Consider statements from line 7 to line 22. There are $O(kn)$ iterations where the statements from line 9 to line 22 correspond to an iteration. It is easy to verify that each iteration takes $O(k(\log(m+n)+N)+n)$ time. Thus, statements from line 7 to line 22 takes $O(kn(k(\log(m+n)+N)+n)) = O(k^2 n(\log(m+n)+N)+kn^2)$ time.

Algorithm 3 Dynamic programming approach

```
Input: $P, Q$ and $k$
Output: the final selection $Q'$ of size $k$ and the optimal price
    assignment vector $\mathbf{v}$ of $Q'$
1: for $i = 1$ to $n$ do
2:   $T(i,0) \leftarrow 0$
3:   $S(i,0) \leftarrow \emptyset$
4: for $t = 1$ to $k$ do
5:   $T(1,t) \leftarrow \alpha(q_1, \emptyset, (0,0,...,0))$
6:   $S(1,t) \leftarrow \{q_1\}$
7: for $t = 1$ to $k$ do
8:   for $i = 1$ to $n$ do
9:     $\mathbf{v}_{select} \leftarrow \alpha(q_i, S(i-1,t-1), \mathbf{v}(i-1,t-1)$
10:    $v \leftarrow$ the $i$-th entry in $\mathbf{v}_{select}$
11:    $T_{select} \leftarrow T(i-1,t-1) + v$
12:    $T_{notSelect} \leftarrow T(i-1,t)$
13:    if $T_{select} \geq T_{notSelect}$ then
14:      // Case 1: $q_i$ is selected
15:      $\mathbf{v}(i,t) \leftarrow \alpha(q_i, S(i-1,t-1), \mathbf{v}(i-1,t-1))$
16:      $T(i,t) \leftarrow T_{select}$
17:      $S(i,t) \leftarrow S(i-1,t-1) \cup \{q_i\}$
18:    else
19:      // Case 2: $q_i$ is not selected
20:      $\mathbf{v}(i,t) \leftarrow \mathbf{v}(i-1,t)$
21:      $T(i,t) \leftarrow T_{notSelect}$
22:      $S(i,t) \leftarrow S(i-1,t)$
23: $Q' \leftarrow Q(n,t)$
24: $\mathbf{v} \leftarrow \mathbf{v}(n,t)$
25: return $Q'$ and $\mathbf{v}$
```

V. Greedy Algorithm

In the previous section, 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 as follows.

Theorem 3: When $l > 2$, problem TPP is NP-hard. □

The proof can be found in the appendix.

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. The second greedy algorithm performs well empirically.

A. 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 *standalone profit* of q_i.

Definition 4 (Standalone Profit): Given a tuple q in Q, the *standalone profit* of q, denoted by $SP(q)$, is defined to $Profit_o(\{q\})$.

The first version of the greedy algorithm is described as follows. Specifically, for each tuple q in Q, we find the standalone profit of q. Then, we choose k tuples which have the greatest standalone profits. This version is shown in Algorithm 4.

Algorithm 4 Greedy algorithm (Version 1)

```
1: $\mathbf{v} \leftarrow (0, 0, ..., 0)$
2: for each $q \in Q$ do
3:     find the standalone profit of $q$
4: $Q' \leftarrow$ a set of the $k$ tuples which have the greatest standalone profits
5: return $Q'$
```

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 $Profit_o(Q')$). It is easy to verify that

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

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 4: Let O be the optimal selection set and Q' be the selection set returned by Algorithm 4.

$$Profit_o(O) - \epsilon_{add} \leq Profit_o(Q')$$

where

$$\epsilon_{add} = \frac{k(k-1)}{2}\sigma$$

□

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

Theorem 5: Let O be the optimal selection set and Q' be the selection set returned by Algorithm 4. Suppose that $Profit_o(Q') > 0$. Let $\Delta = \sum_{q_i \in Q'} SP(q_i)$. Algorithm 4 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 Algorithm 4. We need to calculate the standalone profit of q for each tuple $q \in Q$. This step takes $O(k(\log(m+n) + N))$ time. Then, we need to choose the k tuples which have the greatest standalone profits, which can be done in $O(k \log k)$ time. Thus, the time complexity of Algorithm 4 is $O(k(\log(m+n) + N) + k \log k)$. Since $k = O(m+n)$, the complexity becomes $O(k(\log(m+n) + N)) = O(k \log(m+n) + kN)$.

Algorithm 5 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 $Q'$
```

B. Greedy Based Selection II

In the previous subsection, we describe the first version of the greedy algorithm which does not consider the price correlation. In this subsection, we describe the second version of the greedy algorithm which select tuples in Q iteratively considering the price correlation.

The second version of our greedy algorithms is shown in Algorithm 5.

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

VI. 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 top-k profitable products (TPP) 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.

All the formulations described in Section II can also be used. The only change we need is the definition of a *feasible* price assignment vector. With the h-dominance constraint, 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 $P \cup Q'$ and each $q_i \in Q'$ dominates at least h tuples in P.

Algorithm 6 Algorithm **findOptimalIncrementalPrice**($q_i, Q_{i-1}, \mathbf{v}_{i-1}$) with the h-dominance constraint

```
Input: A set $Q_{i-1}(= \{q_1, q_2, ..., q_{i-1}\})$, tuple $q_i$ in $Q'$ and
    the optimal price assignment vector $\mathbf{v}_{i-1}$ of $Q_{i-1}$
Output: the optimal price assignment vector $\mathbf{v}_i$ of $Q_i$ with
    the $h$-dominance constraint
1:  $\mathbf{v}_i \leftarrow \mathbf{v}_{i-1}$
2:  // Skyline constraint
3:  find a set $Y$ containing all tuples in $P \cup Q'$ which quasi-dominate $q_i$
4:  if $Y \neq \emptyset$ then
5:      $v_{skyline} \leftarrow (\min_{p \in Y} p.A_l) - \sigma$
6:  else
7:      $v_{skyline} \leftarrow \infty$
8:  // The $h$-dominance constraint
9:  find a set $Z$ containing all tuples in $P$ which are quasi-dominated by $q_i$
10: if $|Z| \geq h$ then
11:     $v_{dom} \leftarrow$ the $h$-th greatest price (i.e., $A_l$) among all tuples in $Z$
12: else
13:     $v_{dom} \leftarrow -\infty$
14: // Combining the above two constraints
15: $v \leftarrow \min\{v_{skyline}, v_{dom}\}$
16: set the $i$-th entry in $\mathbf{v}_i$ to $v$
17: return $\mathbf{v}_i$
```

Problem TPP with the h-dominance constraint is more general than problem TPP without the h-dominance constraint. This is because if we set $h = 0$, then the new problem becomes problem TPP without the h-dominance constraint.

After we consider the additional h-dominance constraint, the price assignment vector of a selection set may also be affected.

In this new problem, we only need to modify Algorithm 2 to Algorithm 6. Algorithm 6 is nearly the same as Algorithm 2. Algorithm 6 involves three major parts. The first part contains the statements related to the *skyline constraint* which can be found from Line 2 to Line 7. The second part contains the statements related to the *h-dominance constraint* which can be found from Line 8 to Line 13. The third part contains the statements related to combining the above two constraints which can be found from Line 14 to Line 16.

With the following theorem (which is similar to Lemma 3), it is easy to prove the correctness of Algorithm 6 (i.e., under the h-dominance constraint, Algorithm 6 returns the optimal price assignment vector of Q_i given a set $Q_{i-1} = \{q_1, q_2, ..., q_{i-1}\}$, tuple q_i in Q' and the optimal price assignment vector of Q_{i-1}).

Theorem 6: 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 $\mathbf{v}_{i-1} = (v_1, v_2, ..., v_n)$ such that we set the price of each q_j (i.e., $q_j.A_l$) in Q_{i-1} to v_j.

- **Skyline Constraint:** Suppose that $\gamma(X, q_i) \neq \emptyset$. Let $v_{skyline} = (\min_{p \in \gamma(X, q_i)} p.A_l) - \sigma$.
- **The h-dominance Constraint:** Let $\theta(q_i)$ be the set containing all tuples in P which are quasi-dominated by q_i. Suppose that $\theta(q_i) \neq \emptyset$. Let v_{dom} be the h-th greatest price (i.e., A_l) among all tuples in Z.

Let $\mathbf{v}_i$ be a price assignment vector equal to $\mathbf{v}_{i-1}$ except that the i-th entry of $\mathbf{v}_i$ is set to $\min\{v_{skyline}, v_{dom}\}$. $\mathbf{v}_i$ is the optimal price assignment vector of Q_i. □

Note that this problem with the h-dominance constraint is a general problem of problem TPP. Since problem TPP is NP-hard, this problem is also NP-hard.

Theorem 7: The problem with the h-dominance constraint is NP-hard. □

All the proposed algorithms, namely the dynamic approach and the two version of the greedy approach, can also be adopted except that we need to call a new version of algorithm **findOptimalIncrementalPrice** (Algorithm 6). Details can be found in [13].

VII. Empirical Studies

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

We measured the algorithms with four measurements, namely (1) *Execution Time*, (2) *Preprocessing Time*, (3) *Memory Cost* and (4) *Profit*. (1) The execution time of an algorithm corresponds to the time it takes to find the final selection. (2) The preprocessing of an algorithm corresponds to the time it builds a R*-tree index for quasi-dominance checking. (3) The memory cost of an algorithm is the memory occupied by the algorithm. (4) The profit of an algorithm corresponds to the profit returned by the algorithm.

A. Dataset Description

The experiments are conducted over real datasets and synthetic datasets.

1) Real Dataset: For the real datasets, same as [12], we obtain real datasets from Priceline.com and Expedia.com.

For the website of Priceline.com, we obtained all packages on Jan 15, 2009 for a round trip traveling from San Francisco to New York for a period from March 1, 2009 to March 7, 2009. We have 149 packages. These packages form the set P of existing tuples. Each package has 6 attributes, namely *quality-of-room*, *customer-hotel-grading*, *hotel-class*, *hotel-price*, *class-of-flight*, *no-of-stops* and *price*.

For the website of Expedia.com, we obtained all flights and all hotels on the same day (i.e., Jan 15, 2009) for the same round trip with the same travel period. We have 1014 hotels and 4394 flights. According to these hotels and these flights, we adopt the method proposed by [12] to generate all *competitive packages*. Details can be found in [12]. These competitive products form set Q. In this dataset, we have 4787 competitive packages. Similarly, each package in Q has 6 attributes (including attribute *price*). Note that each package in Q is associated with an additional cost attribute. In order to generate the cost attribute, for each package q in this package set, 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 dataset because our problem is to find these values.

2) Synthetic Dataset: For synthetic datasets, we adapt the dataset generator of [1]. We observe from the real dataset that some attributes have large cardinalities but some have small cardinalities. For example, in the real dataset, *price* may have thousands of possible values, but *no-of-stops* can have only 2 or 3 possible values. We divide the attributes into two groups of nearly equal size. Note that P has l attributes only while Q has an additional attribute C in addition to the l attributes. The first group contains the first half of attributes (or more specifically, $A_1, \dots A_{\lfloor l/2 \rfloor}$) each of which has the cardinality of 10. The second group contains the second half of attributes (or more specifically, $A_{\lfloor l/2 \rfloor+1}, \dots, A_l$) and attribute C where each attribute in this group has the cardinality of 10k.

We generate P and Q in the same way except that generating P involves l attributes but generating Q involves the first $l-1$ attributes and attribute C. Note that attribute A_l of Q is not considered because in our problem definition, attribute A_l is to be found. The dataset generation process is described as follows. Firstly, we used the dataset generator provided by [1] to generate an anti-correlated dataset where each attribute value is a real number in a range between 0.0 and 1.0. Secondly, we perform a postprocessing step so that each attribute in the first group has the cardinality of 10 and each attribute in the second group has the cardinality of 10k. For an attribute in the first group, it can be easily done by multiplying a value in this attribute by 10 and rounding it to be an integer. We can also do a similar step for an attribute in the second group.

B. Result over Small Synethetic Dataset

It is known that a dynamic programming approach is not scalable to large datasets. Besides, this dynamic programming approach solves problem TPP when $l = 2$. In this section, we conducted some experiments to compare all proposed algorithms over a small two-dimensional synthetic dataset where $|P| = 10,000$ and $|Q| = 10,000$. We set the default parameters as $h = 5, d = 0.5$ and $\sigma = 200$.

We vary k to study the performance of the proposed algorithms. Figures 1, 2 and 3 are the results for execution time, profit and the memory cost of each algorithm, respectively.

In Figure 1, the execution time of *BF* is very large and is very unscalable. Note the *PREP* is the preprocessing time of *GR1* and *GR2*. It is nearly equal to the time for *GR1* and *GR2* to find the selection set for problem TPP. In Figure 2, the profit of *DP* and *BF* is the greatest but the profit of *GR1* and *GR2* is also high. In most cases, *GR2* returns higher profit than *GR1*. In Figure 3, as expected, *DP* occupies much more memory than other approaches.

Since *BF* is not scalable, in the following, we do not compare all algorithms with *BF*.

TABLE IV
EXPERIMENTAL SETTING

Parameter	Values
$\|P\|$	0.5M, **1M**, 1.5M, 2.0M
$\|Q\|$	0.5M , 1M **2M**, 3M
d	0.25, **0.5**, 0.75, 1
l	2, **5**, 10,15, 20
k	10, **20**, 50, 100
σ	50, **100**,150, 200
h	0, **10**,20,30

C. Result over Large Synthetic Dataset

In the previous section, we conducted experiments over small synthetic datasets. In these experiments, although *GR1* and *GR2* are heuristical, they also give a high profit. In this section, we conducted experiments over large synthetic datasets 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 in the experiments are given in Table IV where the default values are in bold.

Figures 4, 5 and 6 shows some selected results. For the sake of space, we do not show some figures.

1) Execution time: Figures (a) show the measurement of execution time. In all figures, *GR2* runs slower than *GR1*. As we discussed in Section V, the time complexity of *GR2* is higher than that of *GR1*. For factor k (Figure 5(a), when k increases, the execution time of *GR2* increases exponentially but the execution time of *GR1* does not change much. This is because the time complexity of *GR2* is quadratic with respect to k but the time complexity of *GR1* is not.

2) Preprocessing time: Figures (b) show the preprocessing time of the algorithms. This involves the step of building the index. When $|P|, |Q|$ and l increase, the preprocessing times of *GR1* and *GR2* increase.

3) Memory cost: Figures (c) show the memory cost of the algorithms. Since the memory cost of both *GR1* and *GR2* is the memory occupied by the spatial index R*-tree on dataset $P \cup Q$, when $|Q|$ increases and $|P|$ increases, the memory cost increases, as shown in Figures 4.

4) Profit: Figures (d) show the profit returned by the algorithms. In most cases, *GR1* and *GR2* gives similar profits. For factor k (Figure 5(d)), when k increases, the profits of *GR1* and *GR2* increase because more tuples are selected to contribute the profit of the final selection. For factor h (Figure 6(d)), when h increases, the profits of both algorithms decreases. This is because if h is larger, then the price of each selected tuple in the final selection should be set lower in order that each of tuples dominates at least h tuples in the existing market.

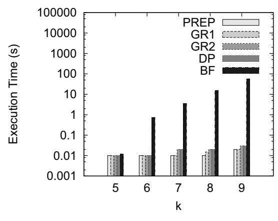

Fig. 1. Execution time of all algorithms (small dataset)

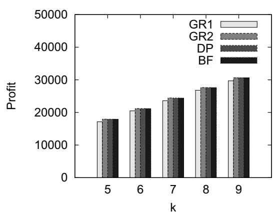

Fig. 2. Profit of all algorithms (small dataset)

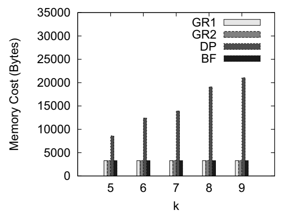

Fig. 3. Memory cost of all algorithms (small dataset)

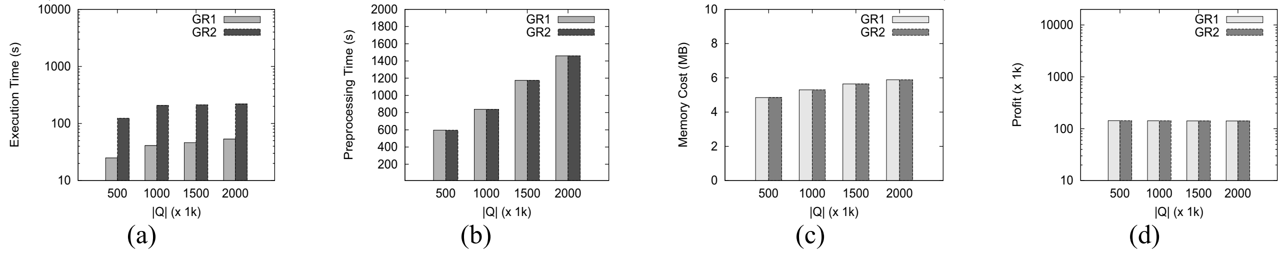

Fig. 4. Effect of $|Q|$ (the number of potential new tuples)

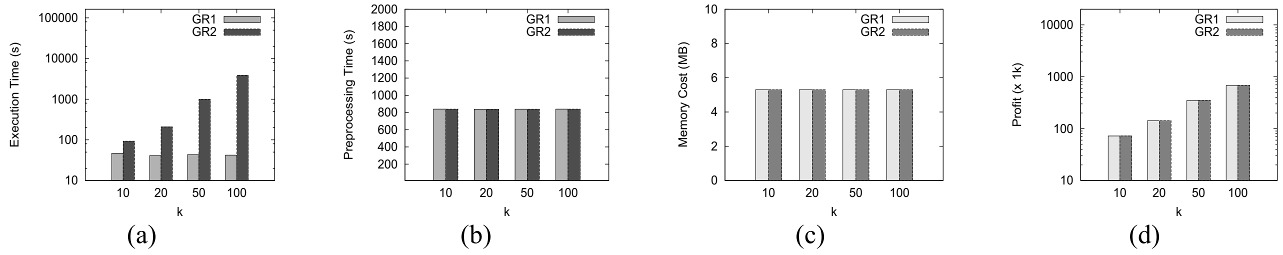

Fig. 5. Effect of k (the size of the final section set)

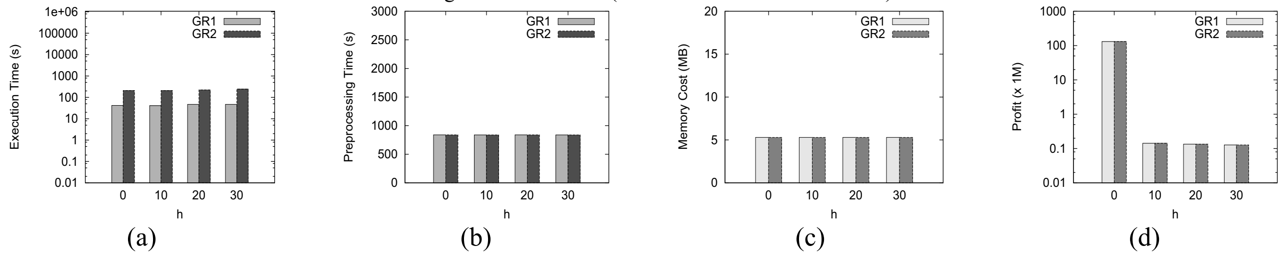

Fig. 6. Effect of h (the minimum number of tuples dominated by each tuple in the selection set)

h	15, **20**, 25, 30
σ	1000 ,2000, **5000** , 10000
d	0.4, **0.6**, 0.8, 1.0
k	100, **150**, 200, 250

TABLE V

EXPERIMENT PARAMETERS ON REAL DATASET

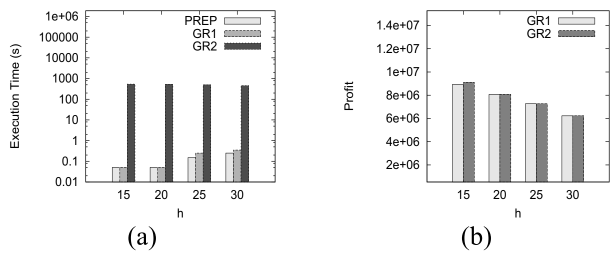

Fig. 7. Effect of h (the minimum number of tuples dominated by each tuple in the selection set)

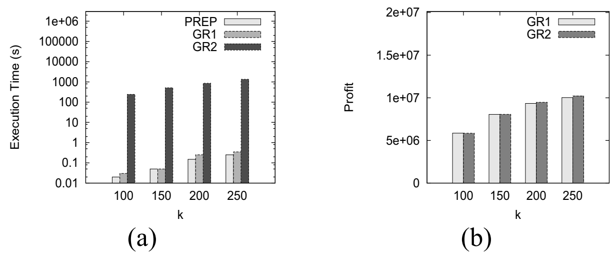

Fig. 8. Effect of k (the size of the final selection set)

D. Result over Real Dataset

We also conducted experiments on real datasets. We varied four factors, namely h, k, d and σ. For the interest of space, we only show the results with two factors h and k as shown in Figures 7 and 8, respectively. The default setting configuration is: $k = 150$, $h = 20$, $d = 0.6$ and $\sigma = 50$. The results for real datasets are similar to those for synthetic datasets.

Summary: Although *DP* finds the optimal solution for problem TPP, it is not scalable and is limited to problem TPP when $l = 2$. *GR1* and *GR2* is scalable to large datasets. It is shown that they can find a selection set Q' with high profits. In most cases, *GR1* and *GR2* returns similar profits. However, *GR2* sometimes gives a higher profit than *GR1*.

VIII. Related Work

Skyline queries have been studied since 1960s in the theory field where skyline points are known as Pareto sets and admissible points [4] or maximal vectors [2]. However, earlier algorithms such as [2], [3] are inefficient when there are many data points in a high dimensional space. Skyline queries in database was first studied by Borzsonyi [1] in 2001.

After that, a lot of 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 [10], a nearest neighbor (NN) algorithm [6], and branch and bound skylines (BBS) method [8].

Top-K queries about skyline were studied in [8], [7], [11]. [8] 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 VI.

[7], [11] studied *representative skyline queries*. The problem is to select k points among all skyline points according to a pre-defined objective function. The k points in the output are said to be representative.

[7] was the first to introduce *representative skyline queries*. [7] finds 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 [7] cannot be applied in our problem because we consider both the profitability of products and the dominance relation of products, but [7] considers 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 [11]. In [11], representative skyline queries is 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. Firstly, 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. Secondly, the concept of *profits* is considered in this paper but not in the above studies.

The most closely related work is [12]. Given a set P of existing tuples and a number of source tables, [12] finds 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 [12] can be regarded as set Q described in this paper. However, in [12], too many competitive products are generated. In their experimental studies, there are 10,000 competitive products in a real dataset. In most cases, it is good to choose some of the competitive products instead of all competitive products for promotion. One criterion is to maximize the total profit of the selection set which is studied in this paper.

IX. Conclusion

In this paper, we identify and tackle the problem of finding top-k profitable products, which has not been studied before. We propose methods to find top-k profitable products efficiently. An extensive performance study using both synthetic and real datasets is reported to verify its effectiveness and efficiency. As future work, finding top-k profitable products with dynamic data and finding top-k profitable products with additional constraints (e.g., representative properties [7], [11]) are interesting topics.

Acknowledgements: The research is supported by HKRGC GRF 621309 and Direct Allocation Grant DAG11EG05G.

References

[1] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.
[2] J. L. B. et al. On the average number of maxima in a set of vectors and applications. In *Journal of ACM, 25(4)*, 1978.
[3] J. L. B. et al. Fast linear expected-time algorithms for computing maxima and convex hulls. In *SODA*, 1990.
[4] O. B.-N. et al. On the distribution of the number of admissable points in a vector random sample. In *Theory of Probability and its Application, 11(2)*, 1966.
[5] B. Jiang, J. Pei, X. Lin, D. W.-L. Cheung, and J. Han. Mining preferences from superior and inferior examples. In *SIGKDD*, 2008.
[6] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, 2002.
[7] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: the k most representative skyline operator. In *in ICDE*, 2007.
[8] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. In *ACM Transactions on Database Systems, Vol. 30, No. 1*, 2005.
[9] D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically-sorted skylines for partially-ordered domains. In *ICDE*, 2009.
[10] K.-L. Tan, P. Eng, and B. Ooi. Efficient progressive skyline computation. In *VLDB*, 2001.
[11] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 892–903, Washington, DC, USA, 2009. IEEE Computer Society.
[12] Q. Wan, R. C.-W. Wong, I. F. Ilyas, M. T. Ozsu, and Y. Peng. Creating competitive products. In *VLDB*, 2009.
[13] Q. Wan, R. C.-W. Wong, and Y. Peng. Creating top-k profitable products (technical report). In *http://www.cse.ust.hk/~raywong/paper/createTopKProfitableProduct-technical.pdf*, 2010.
[14] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang. Mining favorable facets. In *SIGKDD*, 2007.
[15] Z. Zhang, L. Lakshmanan, and A. K. Tung. On domination game analysis for microeconomic data mining. In *TKDD*, to appear.

Appendix

A. Proof of Lemmas/Theorems

Proof of Lemma 1: If p dominates q_i, and $p \notin \gamma(X, q_i)$, then p does not quasi-dominate q_i. So, there exists at least one attribute on which q_i is better than p according to the quasi-dominance definition. This conflicts with our premise that p dominates q_i. $\square$

Proof of Lemma 2: If p quasi-dominates p', according to the definition of f, this indicates that (1) p dominates p' with respect to the first $l-1$ attributes, namely $A_1, A_2, ..., A_{l-1}$, or (2) p has the same $l-1$ attribute values as p'. In case 1, obviously, $f(p) < f(p')$. In case 2, $f(p) = f(p')$. Thus, $f(p) \leq f(p')$ □

Proof of Lemma 3: Suppose that we have a better assignment vector $\mathbf{v}'$ of Q_i in form of $(v'_1, v'_2, \dots v'_n)$ such that $Profit(Q_i, \mathbf{v}') > Profit(Q_i, \mathbf{v}_i)$.

Consider two cases. *Case 1:* $v'_i \leq v_i$. In this case, $v'_i - q_i.C \leq v_i - q_i.C$. In other words, $\triangle(q_i, v'_i) \leq \triangle(q_i, v_i)$. In addition, we know that $\mathbf{v}_{i-1}$ is the optimal price assignment vector of Q_{i-1}, which means $Profit(Q_{i-1}, \mathbf{v}') \leq Profit(Q_{i-1}, \mathbf{v}_{i-1})$. Thus, we have

$$\begin{aligned} Profit(Q_i, \mathbf{v}') &= Profit(Q_{i-1}, \mathbf{v}') + \triangle(q_i, v'_i) \\ &\leq Profit(Q_{i-1}, \mathbf{v}_{i-1}) + \triangle(q_i, v_i) \\ &= Profit(Q_i, \mathbf{v}_i) \end{aligned}$$

Thus, $Profit(Q_i, \mathbf{v}') \leq Profit(Q_i, \mathbf{v}_i)$. This leads to a contradiction.

Case 2: $v'_i > v_i$. It is easy to verify that $\mathbf{v}'$ is not a feasible price assignment vector. For the sake of space, we do not include the details. □

Proof of Theorem 4: Let P be the set of existing tuples, and Q be the set of the newly created tuples. Let O be the optimal selection and Q' be the selection returned by Algorithm 4. Given $Q' \subseteq Q$ and $q_i \in Q'$, we define $\Delta_o(q_i, Q') = \Delta(q_i, v_i)$ where v_i is the i-th entry of the optimal price assignment vector $\mathbf{v}$ of Q'. Suppose $O = \{o_1, o_2, \dots o_k\}$ and $Q' = \{q_1, q_2, \dots q_k\}$, where o_i and q_i are sorted in ascending order of the f values described in Section III.

Let n_i be the greatest possible number of tuples in Q' quasi-dominating q_i. It is easy to verify that $\Delta_o(q_i, Q') \geq SP(q_i) - n_i\sigma$. Note that $\sum_{i=1}^{k} n_i \leq \frac{k(k-1)}{2}$. We derive that

$$\begin{aligned} \sum_{q_i \in Q'} \Delta_o(q_i, Q') &\geq \sum_{q_i \in Q'} (SP(q_i) - n_i\sigma) \\ &= \sum_{q_i \in Q'} SP(q_i) - \sum_{q_i \in Q'} n_i\sigma \\ &\geq \sum_{q_i \in Q'} SP(q_i) - \frac{k(k-1)}{2}\sigma \end{aligned}$$

Thus, we conclude that

$$\sum_{q_i \in Q'} \Delta_o(q_i, Q') \geq \sum_{q_i \in Q'} SP(q_i) - \frac{k(k-1)}{2}\sigma$$

Note that

$$\begin{aligned} \sum_{o_i \in O} \Delta_o(o_i, O) &\leq \sum_{o_i \in O} SP(o_i) \\ &\leq \sum_{q_i \in Q'} SP(q_i) \end{aligned}$$

Therefore, we have

$$\begin{aligned} Profit_o(O) - \epsilon_{add} &= \sum_{o_i \in O} \Delta_o(o_i, O) - \frac{k(k-1)}{2}\sigma \\ &\leq \sum_{q_i \in Q'} SP(q_i) - \frac{k(k-1)}{2}\sigma \\ &\leq \sum_{q_i \in Q'} \Delta_o(q_i, Q') \\ &= Profit_o(Q') \end{aligned}$$

□

Proof of Theorem 5: According to Theorem 4, we have that $\Delta - \frac{k(k-1)}{2}\sigma \leq Profit_o(Q')$. By the nature of optimality, we also have $Profit_o(Q') \leq Profit_o(O)$. For any tuple $q \in Q'$, the real profit cannot be larger than the standalone profit. Thus, $\Delta - \frac{k(k-1)}{2}\sigma \leq Profit_o(Q') \leq Profit_o(O) \leq \Delta$.

Therefore, if $Profit_o(Q') > 0$, we have

$$\begin{aligned} \frac{Profit_o(Q')}{Profit_o(O)} &\geq \frac{\Delta - \frac{k(k-1)\sigma}{2}}{\Delta} \\ &= 1 - \frac{k(k-1)\sigma}{2\Delta} \end{aligned}$$

Therefore,

$$Profit_o(Q') \geq (1 - \epsilon_{mult})Profit_o(O)$$

□

Proof of Theorem 6: It is very easy to verify that $\mathbf{v}_i$ is a feasible assignment vector, so we only prove the optimality. Suppose that $\mathbf{v}_i = (v_1, v_2 \dots v_n)$ is not optimal. Let $\mathbf{u} : (u_1 \dots u_n)$ be another feasible price assignment vector such that $Profit(Q_i, \mathbf{u}) > Profit(Q_i, \mathbf{v}_i)$. Since $q_1 \dots q_n$ are sorted in ascending order of $f(q)$, we have $Profit(Q_i, \mathbf{v}_i) = Profit(Q_{i-1}, \mathbf{v}_{i-1}) + \Delta(q_i, v_i)$.

Consider two cases. *Case 1:* $u_i \leq v_i$. In this case, $\Delta(q_i, u_i) \leq \Delta(q_i, v_i)$. Since $\mathbf{v}_{i-1}$ is the optimal price assignment vector of Q_{i-1}, thus $Profit(Q_{i-1}, \mathbf{v}_{i-1}) \geq Profit(Q_{i-1}, \mathbf{u})$. Therefore, we have

$$\begin{aligned} Profit(Q_i, \mathbf{v}_i) &= Profit(Q_{i-1}, \mathbf{v}_{i-1}) + \Delta(q_i, v_i) \\ &\geq Profit(Q_{i-1}, \mathbf{u}) + \Delta(q_i, u_i) \\ &= Profit(Q_i, \mathbf{u}) \end{aligned}$$

which conflicts with our previous assumption.

Case 2: $u_i > v_i$. In this case, since $v_i = \min\{v_{skyline}, v_{dom}\}$, $v_i = v_{skyline}$ or $v_i = v_{dom}$. For any one of two cases, if $u_i > v_i = v_{skyline}$ or $u_i > v_i = v_{dom}$, we can immediately verify that $\mathbf{u}$ breaks the *skyline constraint* or the *h-dominance constraint.* □