Profit Maximization with Sufficient Customer Satisfactions

CHENG LONG, Queen's University Belfast
RAYMOND CHI-WING WONG, The Hong Kong University of Science and Technology
VICTOR JUNQIU WEI, The Hong Kong University of Science and Technology

In many commercial campaigns, we observe that there exists a trade-off between the number of customers satisfied by the company and the profit gained. Merely satisfying as many customers as possible or maximizing the profit is not desirable. To this end, in this paper, we propose a new problem called *k-Satisfiability Assignment for Maximizing the Profit* (k-SAMP) where k is a user parameter and a non-negative integer. Given a set P of products and a set O of customers, k-SAMP is to find an assignment between P and O such that at least k customers are satisfied in the assignment and the profit incurred by this assignment is maximized. Although we find that this problem is closely related to two classic computer science problems, namely maximum weight matching and maximum matching, the techniques developed for these classic problems cannot be adapted to our k-SAMP problem. In this work, we design a novel algorithm called *Adjust* for the k-SAMP problem. Given an assignment A, *Adjust* iteratively increases the profit of A by *adjusting* some appropriate matches in A while keeping at least k customers satisfied in A. We prove that *Adjust* returns a global optimum. Extensive experiments were conducted which verified the efficiency of *Adjust*.

CCS Concepts: • **Information systems** → *Social recommendation*; *Content match advertising*; *Online auctions*;

Additional Key Words and Phrases: Assignment, profit maximization, users' satisfactions

ACM Reference format:
Cheng Long, Raymond Chi-Wing Wong, and Victor Junqiu Wei. 2017. Profit Maximization with Sufficient Customer Satisfactions. 1, 1, Article 1 (June 2017), 35 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Assigning products to customers is a very common scenario and finds many applications in real life. Some examples are school admission [10], profile matching [32], facility allocation [12], Internet auctions [25] and product bidding [27]. Here, the terms "products" and "customers" are general and can refer to different things in different applications. For example, consider the "Name Your Own Price" (NYOP) service [27] from Priceline.com, which assigns packages/hotels (as products) to bidders (as customers), as follows.

Example 1.1 (Name Your Own Price (Priceline.com)). "Name Your Own Price" (NYOP) [27] is a hotel booking service launched by Priceline.com where customers specify their requirements on the hotels (e.g., check in/out dates, quantity, location and star level) and provide their biding prices (i.e., name your own price). Then, Priceline.com returns an assignment between the customers and the hotels each with a cost. As part of the agreement, a customer cannot reject the products that are assigned to him/her.

Consider that customers are looking for vacation packages to Bordeaux, France using Priceline.com. They use two criteria for choosing packages, namely *price* and *distance-to-beach* (in short, *distance*). Table 1 shows six customer preferences on packages. In this table, each customer gives

XXXX-XXXX/2017/6-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1. Customer preferences on packages

Name	Price ($)	Distance (km)
Alice	200	7
Bob	40	10
Clement	300	5
David	400	4
Emily	100	9
Fred	550	1

Table 2. Packages provided by a travel agency

Package	Cost ($)	Distance (km)
p_1	120	6
p_2	80	6.5
p_3	510	1.5
p_4	310	2
p_5	280	5.5
p_6	290	2.5

a value on each criterion denoting the greatest possible acceptable value for this criterion. For example, Alice is looking for a package with price at most $200 where the distance between a hotel in the package and a beach is at most 7km.

Besides, suppose Priceline.com has six packages in stock as shown in Table 2. In this table, attribute *Cost* corresponds to the cost of a package.

If Priceline.com sets the price of package p_1 to be $200, then package p_1 satisfies Alice's preference and the agency can make a profit of $200 - $120 = $80 if p_1 is assigned to Alice. □

In the application of NYOP, two aspects should be taken into consideration (from the perspective of a travel agency such as Priceline.com). The first aspect is, of course, the *profit*. The second aspect is the *satisfiability* (which corresponds to the number of customers who have been assigned with products, i.e., they are satisfied) since if it is not considered (i.e., the goal is to maximize the profit only), the number of customers who are satisfied with the assignment (i.e., the satisfiability) could be undesirably low (which is verified theoretically in Section 3 and empirically in Section 7), and this would then discourage the customers from using this service. An intuitive solution for this problem is to maximize the profit such that a reasonable number of customers are satisfied. For example, the travel agency can set it a goal to satisfy *at least three* customers in our running example and at the same time, the profit should be maximized, which we illustrate in detail in the following example.

Example 1.2 (Profit Maximization With Sufficient Customer Satisfactions). For the sake of illustration, we first consider that the agency wants to satisfy exactly one customer. When we assign p_2 to Alice and set the price of p_2 to $200, the agency can make the greatest profit of $200 - $80 = $120.

Next, consider that it wants to satisfy exactly two customers. Suppose that we want to satisfy Alice and David. We can assign p_2 and p_6 to them, respectively, which corresponds to an assignment $\{(p_2, \text{Alice}), (p_6, \text{David})\}$. In this case, we set the price of p_2 to \$200 and the price of p_6 to \$400. So, p_2 satisfies Alice (with a profit gain of \$200 - \$80 = \$120) and p_6 satisfies David (with a profit gain of \$400-\$290 = \$110). So, the profit of these two packages is equal to \$230 (which is *greater* than \$120, the greatest profit the agency can make when it satisfies exactly one customer.) In fact, this assignment $\{(p_2, \text{Alice}), (p_6, \text{David})\}$ maximizes the total profit if the agency satisfies exactly two customers.

One may raise a question: "Is it a must that we can make more profit if we satisfy more customers?" The answer is interestingly no. Consider back our scenario that the agency wants to satisfy *at least three* customers and maximize its profit. With similar derivations, we obtain the assignment $\{(p_2, \text{Alice}), (p_4, \text{David}), (p_6, \text{Clement})\}$ satisfying three customers, which maximizes the total profit (=\$220) (which is *smaller* than \$230, the profit the agency can make when it satisfies exactly two customers.) □

Note that in Example 1.2, compared with satisfying two customers (with profit \$230), when the agency satisfies three customers (with profit \$220), it loses \$10. In other words, it *pays* \$10 to satisfy the third customer. One may question why it is willing to sacrifice \$10 to satisfy the third customer. The first reason is the reputation consideration, i.e., if the total number of customers satisfied by products is very small, the *reputation* of this company will become low and these customers will likely not buy any products from this company again in the future. The second reason is that the agency can promote its company and its products easily to the public when it satisfies more customers. Customers who are satisfied with a company are eager to promote the company to their friends, relatives or families via their personal relationship, which can be regarded as an effective marketing strategy [22, 24]. Thus, this \$10 sacrifice is not just regarded as the cost of satisfying more customers. It can also be used to raise the awareness of other customers.

If an assignment satisfies at least k customers, we say that this assignment is *k-satisfiable*. Thus, the last assignment in Example 1.2 is 2-satisfiable and also 3-satisfiable.

Problem k-SAMP: In this paper, we propose a new problem called *k-Satisfiability Assignment for Maximizing Profit* (k-SAMP) where k is a user parameter and a non-negative integer. Specifically, given a set P of product types and a set O of customers, k-SAMP is to find a k-satisfiable assignment between P and O such that the profit incurred by this assignment is maximized. In our running example, k is equal to 3.

The k-SAMP problem is closely related to two classic computer science problems, namely the *maximum weight matching* problem [26] and the *maximum matching* problem [6]. Firstly, our k-SAMP problem is a general problem of the maximum weight matching problem in the sense that if we set k to 0 in our k-SAMP problem, then our problem becomes the profit maximization problem (or equivalently the maximum weight matching problem). Clearly, the solution of the maximum weight matching problem cannot be used for our k-SAMP problem. In our running example, the solution of the maximum weight matching problem for profit maximization is $\{(p_2, \text{Alice}), (p_6, \text{David})\}$ satisfying two customers only and giving the profit of \$230, which does not satisfy the constraint that at least three customers must be satisfied. Secondly, the maximum matching problem is exactly a *customer satisfiability maximization* problem in our problem setting without considering the profit objective. In our running example, the solution of the maximum matching problem is the assignment $\{(p_1, \text{Emily}), (p_2, \text{Bob}), (p_4, \text{David}), (p_5, \text{Alice}), (p_6, \text{Clement})\}$ satisfying five customers and *loosing* the profit of \$40, which is not a desirable solution. Although the two classic problems have been well-studied, their solutions cannot be applied to solve our k-SAMP problem.

The k-SAMP problem can easily be formulated as an *integer linear programming* (ILP) problem (the details will be introduced in Section 7), however, as will be demonstrated in our experiments, solving the k-SAMP problem as an ILP problem by using existing ILP solvers is very expensive and not scalable (e.g., it took more than 6 hours and occupied more than 10GB memory on a real dataset with 17,274 and 1,711 products and customers, respectively).

In this paper, we design an efficient algorithm called *Adjust* for k-SAMP. *Adjust* first initializes a k-satisfiable assignment and then conducts appropriate adjustments iteratively on this assignment such that the resulting assignment is still k-satisfiable and has more profit, until no such adjustments are available. We prove that the final assignment is k-satisfiable with the greatest profit.

Contribution and Organization: We summarize our main contributions as follows. Firstly, to the best of our knowledge, we are the first to propose the k-SAMP problem, which is more general and practical than the existing assignment problems and thus has more extensive real-life applications. Secondly, we propose an efficient method called *Adjust* for k-SAMP. Thirdly, we conducted extensive experiments which verified our *Adjust* algorithm.

The rest of the paper is organized as follows. We formulate our k-SAMP problem in Section 2 and clarify the relations of k-SAMP to existing problems in Section 3. We introduce the *Adjust* algorithm in Section 4 and further discuss it in Section 6. We give the empirical studies in Section 7 and conclude the paper in Section 8.

2 PROBLEM DEFINITION

Let P be a set of product types and O be a set of customers. Each product type p in P has its *capacity*, denoted by $p.w$, representing the total number of products of type p the company provides. Each customer o in O has his/her *demand*, denoted by $o.w$, representing the number of products s/he needs. Each product type p is associated with a *cost* attribute and a set $\mathcal{A}$ of attributes other than the cost attribute describing this product type. For each product type p, we use $p.cost$ to denote the cost of p and $p.a$ to denote the value of p on attribute $a \in \mathcal{A}$. We assume that for any two values, v and v', in each attribute, if $v < v'$, then v is more preferable. In the case that a larger value is more preferable on an attribute, we can just negate all values on this attribute. Consider Example 1.1, $\mathcal{A} = \{\text{distance}\}$. If $a =$"distance", $p_1.a = 6$ and $p_1.cost = 120$.

Similar to [9, 13, 15, 17, 18, 21, 28, 29], each customer o in O has his/her own preference on the attributes of product types (or simply products). Customer preferences 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 [14]. Besides, they can also be obtained directly by some online systems such as "Name Your Own Price" [27] where customers can provide their acceptable prices called *bidding prices* directly. Similar to a product type, each customer o in O has the same set $\mathcal{A}$ of attributes and an attribute called $price$. We use $o.price$ to denote the bidding price provided by customer o and use $o.a$ to denote the greatest acceptable value on each attribute $a \in \mathcal{A}$ given by o.

In order to encourage customers to give "reasonable" bidding prices (i.e., they do not fool the system by providing very low bidding prices), we have the following strategies. First, as what the "Name Your Own Price" service does, we can suggest a customer an "average" bidding price for the hotels that satisfy the requirements of this customer, which could be computed based on the historical data easily. Second, we propose a strategy called "local profit loss tolerance" requirement which main idea is that we only *accept* the bidding prices which should not be too low compared with the cost of a *potential* hotel. More specifications of this strategy will be introduced later in this section.

In Example 1.2, we need to set the price of a package p for a customer o assigned. In order to gain more profit (if any) or lose less money (otherwise), we should set the price of p to be $o.price$. Thus, given a customer o and a product type p, we define the *unit profit* of p assigned to o, denoted by $profit(p, o)$, to be $o.price - p.cost$.

We know that when a product type p is assigned to a customer o, $profit(p, o)$ can be positive or negative. A positive value means that the company gains profit for this pair (p, o) but a negative value means that it loses money.

We want to assign customers with product types such that each product type p assigned to a customer o can "satisfy" both the *company's need* and the *customer o's need*. If a given pair (p, o) satisfies both needs, we say that (p, o) is *matchable*.

Company's Need: There are two scenarios. The first scenario is that some companies want to enforce the *local profit gain* requirement: for each assigned pair (p, o), $profit(p, o) > 0$ (i.e., $o.price > p.cost$). Note that the *overall* profit among all products assigned to customers in this scenario must be positive. The second scenario is that some companies do not enforce this requirement. Specifically, for an assigned pair (p, o), it is possible that $o.price > p.cost$ but for another assigned pair (p', o'), it is possible that $o'.price \leq p'.cost$. In this case, the overall profit can be positive or negative. Under this scenario, different companies may still have different requirements. One requirement can be the *local profit loss tolerance* requirement: given a user parameter $\alpha \in [0, 1]$, for each assigned pair (p, o), $o.price > p.cost \times \alpha$. If α is set to 1, this scenario becomes the first scenario. This requirement is enforced to avoid each customer o setting $o.price$ too low. The other requirement can be the *global profit gain* requirement which means that the overall profit should be greater than 0. Under this requirement, if the overall profit obtained is positive, we will return customers with their assigned product types as an output. Otherwise, we will return nothing (since there exists on solution with positive overall profit).

In the following, we illustrate our concepts under the first scenario (i.e., $profit(p, o) > 0$). The second scenario will be discussed in Section 6.

Customers' Need: Each customer o requires that a product type p assigned to him/her can meet his/her preference.

We define formally the concept of "matchable" as follows.

Given a pair (p, o), we say that (p, o) is **matchable** if and only if $o.price > p.cost$ and for each $a \in \mathcal{A}$, $o.a \geq p.a$.

We define a **match** to be a triplet in the form of (p, o, w) where (p, o) is matchable and w is a positive integer denoting the number of products of type p assigned to o. w is said to be the *weight* of this match. We define an **assignment** between P and O to be a set A of matches such that (1) the total number of products of each type assigned to customers should be at most its capacity (i.e., $\forall\, p \in P, \sum_{(p,o,w)\in A} w \leq p.w$), and (2) the total number of products assigned to each customer should be at most his/her demand (i.e., $\forall\, o \in O, \sum_{(p,o,w)\in A} w \leq o.w$).

Given an assignment A, we define the **satisfiability** of A, denoted by $sat(A)$, to be $\sum_{(p,o,w)\in A} w$. A is said to be *k-satisfiable* if $sat(A) \geq k$.

Let A be an assignment and (p, o, w) be a match in A. The profit of (p, o, w), denoted by $profit(p, o, w)$, is $w \cdot (o.price - p.cost)$. We define the profit of A, denoted by $profit(A)$, to be the sum of the profits of all matches in A, i.e., $profit(A) = \sum_{(p,o,w)\in A} profit(p, o, w)$.

Problem 1 (k-SAMP). *The goal of k-**SAMP** is to find the k-satisfiable assignment A with the greatest $profit(A)$.* □

In the following, for the ease of illustration, we focus on the *un-weighted* version of k-SAMP where for each $o \in O$ and each $p \in P$, $o.w = 1$ and $p.w = 1$. In this case, since each match (p, o, w)

Table 3. Summary of notations

P (O)	the set of product types (customers)
p (o)	a product type in P (a customer in O)
$p.w$ ($o.w$)	product type p's capacity (customer o's demand)
$\mathcal{A}$ (a)	the set of attributes of a product type (an attribute in $\mathcal{A}$)
$p.a$ ($o.a$)	product type p's value on attribute a (customer o's requirement on attribute a)
$p.cost$ ($o.price$)	product type p's cost (customer o's bidding price)
$profit(p, o)$	the unit profit gained if a product of product type p is assigned to o
(p, o, w)	a match meaning that w products of product type p are assigned to customer o
(p, o)	an un-weighted match (equivalent to $(p, o, 1)$) or a feasible pair
A	a set of matches, i.e., an assignment
$sat(A)$	the satisfiability of assignment A (i.e., the amount of products assigned in A)
$profit(A)$	the profit of assignment A (i.e., the sum of the profits of all matches in A)
$\mathbb{A}$	the set of all possible assignments between P and O
S	a feasible sequence, usually in the form of $(p_1, o_1, p_2, o_2, ..., p_m, o_m)$
$M(S)$	the set of all internal matches in feasible sequence S
$o(S, A)$	the adjusting operation on feasible sequence S in assignment A
$N(S)$	the set of all new matches formed after $o(S, A)$ is done
$PG(S, A)$	the profit gain due to $o(S, A)$
$SG(S, A)$	the satisfiability gain due to $o(S, A)$
$\mathcal{T}$	the set of all possible sequences
$\mathcal{T}^*$	the set of all critical sequences
$G_A(V, E)$ (G_A)	the directed graph based on assignment A
π	a path in $G_A(V, E)$
$path(S)$	the mapped path of sequence S in $G_A(V, E)$
$seq(\pi)$	the mapped feasible sequence of path π in A
C	a cycle in $G_A(V, E)$

has its weight w equal to 1, we simply write a match as (p, o). Similarly, we can simply write $profit(p, o, w)$ as $profit(p, o)$.

The original version of k-SAMP where $o.w$ and $p.w$ can be equal to any positive integer can be transformed easily to the un-weighted version by duplicating each o $o.w$ times and p $p.w$ times. Thus, the following proposed technique for the un-weighted version can also solve the original version. Note that we have a more concise technique for the original version which is developed based on the concepts in the un-weighted version, and the details could be found in Section 5.

The notations used throughout this paper are summarized in Table 3.

3 RELATED WORK

Let $\mathbb{A}$ be the set of all possible assignments between P and O. We define sat_{max} to be $\max_{A \in \mathbb{A}} sat(A)$ and define $profit_{max}$ to be $\max_{A \in \mathbb{A}} profit(A)$.

Maximum Matching: Given a bipartite graph G containing two sets of vertices, P and O, and a set of edges each of which connects a vertex $p \in P$ and a vertex $o \in O$, the *maximum matching* problem [6] is to find an assignment A between P and O such that the total number of the matches in A is the greatest. Let X be the set of multiple assignments which have the greatest number of matches. A variation of the maximum matching problem considers the same bipartite graph where each edge is associated with a weight w, and finds the assignment in X which has the greatest

sum of the weights of all matches involved in the assignment. This variation can be solved by a well-known min-cost flow algorithm [1]. In the following, we focus on this variation.

Given an instance of the k-SAMP problem, we can construct the variation of the maximum matching problem as follows. Firstly, all product types in P form one vertex set in the bipartite graph while all customers in O form another vertex set. Secondly, for each $p \in P$ and $o \in O$, if (p, o) is matchable, we create an edge connecting the vertex for p and the vertex for o and set the weight of this edge to be $profit(p, o)$. It is easy to verify that the satisfiability of the assignment for the maximum matching problem is equal to sat_{max}.

Lemma 3.1 (Solution Existence). *There exists an assignment for k-SAMP if and only if $k \leq sat_{max}$.* □

Lemma 3.2 (Inapplicable Maximum Matching). *Let A be the assignment for the constructed maximum matching problem and A_o be the optimal assignment for k-SAMP. There exists a problem instance such that $\frac{profit(A)}{profit(A_o)} \approx 0$.* □

Maximum Weight Matching: Given the same bipartite graph G with weights described in the variation of the maximum matching problem, the maximum weight matching problem [26] is to find an assignment A between P and O such that the sum of the weights of the matches in A is the greatest.

Given an instance of the k-SAMP problem, we can construct the maximum weight matching problem in the same way as the construction for the variation of the maximum matching problem. Clearly, the sum of the weights of the matches in the assignment for the constructed maximum weight matching problem is equal to $profit_{max}$.

Lemma 3.3 (Max. Weight Matching). *The constructed maximum weight matching problem is a special case of k-SAMP. If we set $k = 0$, k-SAMP becomes the constructed maximum weight matching problem.* □

Lemma 3.4 (Inapplicable Max. Weight Matching). *Let A be the assignment for the constructed maximum weight matching problem and A_o be the optimal assignment for k-SAMP. There exists a problem instance such that $\frac{sat(A)}{sat(A_o)} \approx 0$.* □

Assignment Problems: Different assignment problems have been studied extensively in the literature [1, 10, 16, 20, 23]. Among them, *maximum weight perfect matching* [1, 16, 23] is to find a matching with n matches between a set of n objects and another set of n objects, which has the *greatest* sum of the weights of the matches, *cardinality assignment* [20] is to find a matching with exactly a given number of matches, which has the *smallest* sum of the weights of the matches, and *stable marriage matching* [10] is to find a matching between two sets such that there exist no *unstable* matches where whether a match is stable depends on the preferences of objects (i.e., a match of (p, o) is said to be unstable if p (o) prefers some other objects than o (p)). To the best of our knowledge, none of the existing assignment problems is the same as the k-SAMP problem proposed in this paper.

Recently, [28, 32, 35] studied the assignment problem under the spatial setting. Specifically, they are the *spatial matching* problem [32], the *capacity constrained assignment* problem [35] and the *fair assignment* problem [28]. However, all these studies aim at either satisfying the *greatest* number of customers in O with some objectives or maximizing the total weight (or profit in our case), which is different from our problem satisfying *at least* k customers. [4] provides an extensive survey on

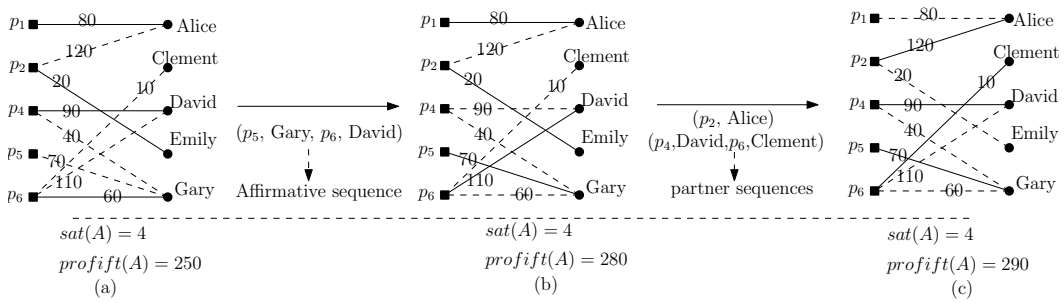


Fig. 1. Our running example

assignment problems. To the best of our knowledge, no existing studies have been performed on k-SAMP.

Besides, in the literature of Recommendation Systems, there are several studies which study the recommendation problem for profit maximization [2, 5, 8, 19]. Those studies are different from the one in this paper since their focus is mainly on how to model the adoption probabilities of users for a specific product based on which a product is recommended to a limited number of users for a potentially optimal profit but user satisfaction as defined in this paper is not considered. Some other related work includes [11] which studies a tour recommendation problem by considering both the travel cost and tourists' interests, [33] which exploits the temporal behaviour patterns in the buying processes of the business customers and develops a B2B (business to business) marketing campaign recommender system, and [36] which studies an assignment problem which maximizes the total social surplus without any guarantee over the satisfiability.

4 ALGORITHM

In this section, we propose an algorithm called *Adjust* for k-SAMP with the following *framework* in two phases. Phase 1 is to initialize a k-satisfiable assignment. Phase 2 is to conduct appropriate adjustments iteratively on this assignment such that the resulting assignment is still k-satisfiable and has more profit, until no such adjustments are available. Here, we focus on the first scenario and the techniques developed will be extended to the second scenario in Section 6.

4.1 Concepts

In this subsection, we introduce a new concept called "feasible sequence" which is a component used for the assignment adjusting in Phase 2 of *Adjust*. After that, we describe an operation on it, which is one of the fundamental steps for the assignment adjusting.

4.1.1 Feasible Sequence. We first define the concept of *feasible un-matched pair* (or *feasible pair* in short) which is the basic unit in a "feasible sequence".

Definition 4.1 (feasible pair). Let A be an assignment. Given $p \in P$ and $o \in O$, we say that (p, o) is a **feasible pair** wrt A if (p, o) is matchable and p is *not* matched with o in A. □

Given two feasible pairs wrt A, namely (p, o) and (p', o'), if (p', o) is a match in A, we say that (1) these two feasible pairs are *adjacent* wrt A, (2) match (p', o) is the *connector* between (p, o) and (p', o') wrt A, (3) (p, o) appears *before* (p', o') wrt A, and (4) (p', o') appears *after* (p, o) wrt A. Note that any connector is a match in A. In the following, we simply say a feasible pair wrt A a feasible pair if there is no ambiguity of A.

Example 4.2 (Feasible Pair). In order to illustrate better, we include one more customer, Gary, in Table 1 where his preference on (Price, Distance) is ($350, 5.5). We present the customers in Table 1 and the packages in Table 2 with a bipartite graph in Figure 1, where each black dot represents a customer and each black square box represents a package. For each pair of package p and customer o that is matchable, we draw *solid* line between p and o if p is matched with o and draw a *dashed* line between p and o in the other case. Besides, we associate with each line (either solid or dashed) between p and o a number equal to $profit(p, o)$. For simplicity, we ignore in the bipartite graph those customers and packages that are not involved in any matchable pairs (e.g., p_3). Consider an assignment $A = \{(p_1, \text{Alice}), (p_2, \text{Emily}), (p_4, \text{David}), (p_6, \text{Gary})\}$ as presented in the bipartite graph in Figure 1(a). Then, (p_5, Gary) is a feasible pair since (p_5, Gary) is matchable and p_5 is not matched with Gary in A. Similarly, (p_6, David) is also a feasible pair . Since (p_6, Gary) is a match in A, (p_5, Gary) and (p_6, David) are adjacent. Besides, (p_6, Gary) is the connector between (p_5, Gary) and (p_6, David), and we say (p_5, Gary) appears before (p_6, David) and (p_6, David) appears after (p_5, Gary) . □

Definition 4.3 (Feasible Sequence). Let A be an assignment. Consider m feasible pairs wrt A, namely $(p_1, o_1), (p_2, o_2), ..., (p_m, o_m)$. A sequence S in the form of $(p_1, o_1, p_2, o_2, ..., p_m, o_m)$ is said to be a **feasible sequence** wrt A iff (p_i, o_i) appears before (p_{i+1}, o_{i+1}) for $1 \leq i \leq m-1$. Each of these m pairs is also said to be a feasible pair of S, and S is said to involve these m feasible pairs. □

In the above definition, the connector between (p_i, o_i) and (p_{i+1}, o_{i+1}) is said to be an **internal match** in S for $1 \leq i \leq m-1$. Note that S has $m-1$ internal matches. p_1 (o_m) is defined to be the **head** (**tail**) in S, which must come from P (O). If there exists a match involving p_1 (o_m) in A, then this match is said to be the **head match** (**tail match**) in S.

Example 4.4 (Feasible Sequence). Continue Example 4.2. Let S be a sequence in the form of (p_5, Gary, p_6, David). Since we know that (p_5, Gary) and (p_6, David) are feasible pairs, and (p_5, Gary) appears before (p_6, David), we conclude that S is a feasible sequence. Besides, (p_6, Gary) (which is the connector between (p_5, Gary) and (p_6, David)) is an internal match in S. Since there are no other internal matches in S, (p_6, Gary) corresponds to the only internal match in S. p_5 is the head in S and David is the tail in S. Since there is no customer in O currently matched with p_5, the head in S, in A, there is no head match in S. Since p_4 is matched with David, the tail in S, in A, (p_4, David) is the tail match in S. □

In the following, for clarity, when we write "sequence", we mean "feasible sequence".

4.1.2 Operation on Sequence. Next, we define an operation called an *adjusting operation* on a feasible sequence. Given a sequence S, $M(S)$ is defined to be the set containing all internal matches in S, the head match in S (if its head is matched) and the tail match in S (if its tail is matched). Consider the sequence S=(p_5, Gary, p_6, David) in Example 4.4. $M(S)$ corresponds to {(p_6, Gary), (p_4, David)}.

Definition 4.5 (Adjusting Operation). Let A be an assignment. Consider a feasible sequence S wrt A. The *adjusting operation* on S in A, denoted by $o(S, A)$, is defined to be an operation which takes S and A as inputs and outputs the assignment A' which is formed by breaking each match in $M(S)$ and forming new matches for all feasible pairs of S. We say that this operation *adjusts* S in A to *form* A'. A' is said to be the *adjusted assignment* of A by S. □

Each feasible pair of sequence S (originally un-matched) becomes matched after the adjusting operation on S. We define $N(S)$ to be the set of all new matches formed during this operation.

Example 4.6 (Adjusting Operation). Consider the sequence S =(p_5, Gary, p_6, David) in Example 4.4. The adjusting operation on S in A breaks each match in $M(S)$ (which is equal to {(p_6, Gary), (p_4, David)}) and forms new matches for all feasible pairs of S (i.e., (p_5, Gary) and (p_6, David) which form the set $N(S)$). Consequently, we form a new assignment A' = {(p_1, Alice), (p_2, Emily), (p_5, Gary) and (p_6, David)} (which is presented in the bipartite graph in Figure 1(b)). □

Given an assignment A and a sequence S wrt A, we capture the effect of $o(S, A)$ on the profit (satisfiability) of A by the concept "profit gain" ("satisfiability gain").

Definition 4.7. Let A be an assignment and S be a feasible sequence wrt A. Let A' be the adjusted assignment of A by S. The *profit gain* of adjusting S in A, denoted by $PG(S, A)$, is defined to be $profit(A') - profit(A)$. The *satisfiability gain* of adjusting S in A, denoted by $SG(S, A)$, is defined to be $sat(A') - sat(A)$. □

It is easy to verify that $PG(S, A) = profit(N(S)) - profit(M(S))$ and $SG(S, A) = sat(N(S)) - sat(M(S))$ because the adjusting operation on S only affects the matches broken in $M(S)$ and the matches newly formed in $N(S)$.

Example 4.8 (Satisfiability/Profit Gain). Consider the sequence S =(p_5, Gary, p_6, David) in Example 4.4. We have $N(S)$ = {(p_5, Gary), (p_6, David))} and $M(S)$ = {(p_6, Gary), (p_4, David)}. We have $profit(p_5, \text{Gary}) = 70$, $profit(p_6, \text{David}) = 110$, $profit(p_6, \text{Gary}) = 60$ and $profit(p_4, \text{David}) = 90$. Thus, $profit(N(S)) = 70 + 110 = 180$ and $profit(M(S)) = 60 + 90 = 150$. Besides, $sat(N(S)) = 2$ and $sat(M(S)) = 2$. Therefore, $PG(S, A) = profit(N(S)) - profit(M(S)) = 180 - 150 = 30$ and $SG(S, A) = sat(N(S)) - sat(M(S)) = 2 - 2 = 0$. □

Consider an assignment A. Let S be a feasible sequence wrt A involving m feasible pairs. Let A' be the adjusted assignment of A by S. If $PG(S, A)$ is positive, then the profit will increase after we adjust sequence S in A. However, if it is negative, the profit will decrease after we adjust S. Interestingly, $SG(S, A)$ is equal to one of the *three* possible values, 1, -1 or 0.

Lemma 4.9 (Possible Values of $SG(S, A)$). *Let A be an assignment. Consider a feasible sequence S wrt A. $SG(S, A)$ is equal to one of the three values, 1, -1 or 0.* □

Let $S = (p_1, o_1, p_2, o_2, ..., p_m, o_m)$ be a sequence. $N(S) = \{(p_i, o_i) | 1 \leq i \leq m\}$ and thus $sat(N(S)) = m$. $M(S)$ corresponds to the set $\{(p_{i+1}, o_i) | 1 \leq i \leq m - 1\}$ augmented with the head match (if any) and the tail match (if any). Thus, $sat(M(S))$ could be $m - 1$ (S has no head match nor tail match), $m + 1$ (S has both its head match and its tail match which are distinct) or m (the other cases). Recall that $SG(S, A) = sat(N(S)) - sat(M(S))$ and thus $SG(S, A)$ belongs to $\{1, -1, 0\}$.

4.1.3 Three Kinds of Sequences. Based on the satisfiability gain and the profit gain, we define the following three kinds of sequences, namely *affirmative sequence*, *profitable sequence* and *gratifying sequence* which will be used in our *Adjust* algorithm.

Definition 4.10. Let A be an assignment and S be a sequence in A. S is said to be

- *affirmative* in A iff $SG(S, A) \geq 0$ and $PG(S, A) > 0$;
- *profitable* in A iff $SG(S, A) < 0$ and $PG(S, A) > 0$;
- *gratifying* in A iff $SG(S, A) > 0$ and $PG(S, A) \leq 0$. □

We do not define other kinds of sequences like the sequence S which has $SG(S, A) < 0$ and $PG(S, A) < 0$ since they are not useful for our algorithm.

Example 4.11. Consider the sequence S =(p_5, Gary, p_6, David) used in Example 4.4. According to Example 4.8, $PG(S, A) = 30 > 0$ and $SG(S, A) = 0$. Therefore, S is an affirmative sequence. Similarly, in the assignment $A' = \{(p_1, Alice), (p_2, Emily), (p_5, Gary)$ and $(p_6, David)\}$ presented in the bipartite graph in Figure 1(b), we can verify that sequence (p_2, Alice) is a profitable sequence and sequence (p_4, David, p_6, Clement) is a gratifying sequence. □

Affirmative, profitable and gratifying sequences are very useful to develop our *Adjust* algorithm. For example, consider an assignment A. When we find an affirmative sequence in A and execute an operation on this sequence, the profit of A will increase and meantime, the satisfiability of A is kept (i.e., not decreased). More discussions will be found in the next subsection.

According to Lemma 4.9, we conclude that if S is affirmative, then $SG(S, A)$ must be 1 or 0. If S is profitable, then $SG(S, A)$ must be -1 . If S is gratifying, then $SG(S, A)$ must be 1.

4.2 Challenges

Consider back the framework of our proposed algorithm which considers a number of iterations each of which performs an assignment adjustment on a k-satisfiable assignment maintained in the algorithm. Suppose that A is the k-satisfiable assignment at an iteration. No matter what $sat(A)$ is, if we find an affirmative sequence S and adjust S in A, we obtain the adjusted assignment of A by S which has a larger profit and at least the satisfiability of A. Besides, if $sat(A)$ is strictly larger than k, in addition to an affirmative sequence, we can find a profitable sequence S in A, adjust S in A and obtain the adjusted assignment A' of A by S which has a larger profit but has $sat(A')$ equal to $sat(A) - 1$ (since $SG(S, A)$ = -1). In this case, though $sat(A')$ is smaller than $sat(A)$, we know that $sat(A')$ is still at least k (because $sat(A') = sat(A) - 1$ and $sat(A) > k$). Thus, finding an affirmative sequence no matter what $sat(A)$ is and finding a profitable sequence when $sat(A) > k$ is a good way for our strategy.

One may ask: "*Is this the only way to increase the profit of an assignment while the satisfiability is at least k?*" The answer is "no". In fact, one may come up with the idea that we can find a *series* of feasible sequences and after we execute the adjusting operations on these sequences *one by one*, the profit of the resulting assignment increases and the satisfiability of the resulting assignment is at least k. For example, suppose that this series involves three sequences, S_1, S_2 and S_3. Let A be the original k-satisfiable assignment. We find S_1 in A and adjust S_1 in A. Then, we obtain a resulting assignment A'. Next, we do it similarly. We find S_2 in A' and adjust S_2 in A', finally obtaining a resulting assignment A''. The last step is to find S_3 in A'' and perform the adjusting operation. Suppose that sequence S_1 has $SG(S_1, A) = 1$ and $PG(S_1, A) = -40$, S_2 has $SG(S_2, A') = -1$ and $PG(S_2, A') = 30$ and S_3 has $SG(S_3, A'') = 0$ and $PG(S_3, A'') = 20$. After executing all three adjusting operations, we obtain that the profit gain is 10 and there is no satisfiability change.

This idea is promising but it has two challenges.

Challenge 1: The first challenge is that each series may involve *a lot of* feasible sequences which means that the computational cost of finding a series is very high. In the above example, it involves three sequences in order to increase the profit. Fortunately, we show that it is *sufficient* to find a series involving *at most two* feasible sequences, which can *significantly* lower down the cost of finding a series.

Challenge 2: The second challenge is related to the computational cost of finding such a series even if it involves two feasible sequences only. According to the above idea, the second feasible sequence in the series is to be found on the assignment obtained *after* the adjusting operation on the first feasible sequence in the series is executed. However, in this paper, we show that we can find the first feasible sequence and the second feasible sequence *simultaneously* based on one interesting

property. The property is that the first feasible sequence and the second feasible sequence are *compatible* (or *independent*). This means that we can find the second feasible sequence based on the original assignment instead of the assignment obtained due to the adjusting operation on the first feasible sequence. With this compatible property, we can design a more efficient algorithm.

Definition 4.12 (Compatible). Consider an assignment A. Two distinct feasible sequences S and S' are said to be *compatible* wrt A iff $M(S) \cap M(S') = \emptyset$. □

Example 4.13 (Compatible). Consider the assignment A as shown in the bipartite graph in Figure 1(b). Consider two sequences S and S' where S is (p_2, Alice) and S' is (p_4, David, p_6, Clement). Since $M(S) = \{(p_1, \text{Alice}), (p_2, \text{Emily})\}$ and $M(S') = \{(p_6, \text{David})\}$, i.e., $M(S) \cap M(S') = \emptyset$, S and S' are compatible. □

Lemma 4.14. *Consider an assignment A. If two sequences S and S' are distinct and compatible, then $N(S) \cap N(S') = \emptyset$.* □

According to the above lemma, whenever two distinct feasible sequences S and S' wrt an assignment A are compatible (i.e., $M(S) \cap M(S') = \emptyset$), we have $N(S) \cap N(S') = \emptyset$, which implies that the matches to be broken and the new matches to be formed during the adjust operation on S are totally different from those matches during the adjust operation on S'. In other words, S and S' are independent. Continue Example 4.13. $N(S) = \{(p_2, \text{Alice})\}$ and $N(S') = \{(p_4, \text{David}), (p_6, \text{Clement})\}$. That is, $N(S) \cap N(S') = \emptyset$.

In contrast, if two sequences S and S' wrt the current assignment A are not compatible, i.e., $M(S) \cap M(S') \neq \emptyset$, we show that the adjusting operation on one would make the other one not a feasible sequence wrt the resulting assignment any more and thus they cannot be performed simultaneously. Let match (p, o) be one of the common matches of $M(S)$ and $M(S')$. Let A' be the adjusted assignment of A by S. According to Definition 4.5, the adjusting operation on S breaks down (p, o) resulting in (p, o) no longer a match in A'. As a result, S' is no longer a feasible sequence wrt A' and thus we cannot perform the adjusting operation on S' after the adjusting operation on S (this essentially tells that the adjusting operations on S and S' cannot be performed simultaneously).

4.3 Theoretical Properties and Algorithm

Now, we can explain how we address the two challenges we described with two lemmas.

Lemma 4.15 (Compatible). *Let A be an assignment. Suppose that there exist no affirmative sequences in A. If there exist a profitable sequence S and a gratifying sequence S' in A such that $PG(S, A) + PG(S', A) > 0$, then S and S' are* compatible. □

If there exist no affirmative sequences, and a profitable sequence S and a gratifying sequence S' wrt an assignment A satisfy $PG(S, A) + PG(S', A) > 0$, we say that these two sequences are two *partner sequences* in A. Obviously, the profit of the resulting assignment A' due to the adjusting operations on these two sequences is greater than the original profit. Besides, the satisfiability of this resulting assignment A' is exactly equal to the original satisfiability because $SG(S, A) + SG(S', A) = 0$ (since $SG(S, A) = -1$ and $SG(S', A) = 1$).

This lemma suggests a powerful tool to design our assignment adjustment as follows.

- We first determine whether there exists any affirmative sequence in A. If yes, we can perform the adjusting operation on an affirmative sequence. Otherwise, we do the next step.
- We have the following two cases.

ALGORITHM 1: Algorithm *Adjust*

```
1: //Phase 1
2: initialize a k-satisfiable assignment A
3: //Phase 2
4: while true do
5:    if sat(A) = k then
6:       if there exists an affirmative sequence S in A then
7:          A ← o(S, A)
8:       else if there exist 2 partner sequences in A, S and S′ then
9:          A ← o(S, A);A ← o(S′, A)
10:      end if
11:      else return A
12:   else
13:      // sat(A) > k
14:      if there exists a sequence S which is affirmative or profitable in A
15:      then A ← o(S, A)
16:      else return A
17:   end if
18: end while
```

  - If $sat(A) > k$, we determine whether there exists any profitable sequence in A. If yes, we can perform the adjusting operation on a profitable sequence.
  - If $sat(A) = k$, we proceed to the next step.
- We determine whether there exist two partner sequences. If yes, we can also perform the adjusting operations on these sequences.

This strategy can address the second challenge *provided that* this strategy is sufficient to find the optimal assignment. The following lemma states that this strategy is sufficient for the optimal solution.

Lemma 4.16 (Optimal). *Given an assignment A, A is optimal if one of the following two conditions is satisfied:* C1: $sat(A) > k$ *and there does not exist any affirmative sequence and any profitable sequence in A, and* C2: $sat(A) = k$ *and there does not exist any affirmative sequence and any two partner sequences in A.* □

The above lemma suggests that it is sufficient to use an affirmative sequence or a profitable sequence when $sat(A) > k$ and use an affirmative sequence or two partner sequences when $sat(A) = k$ for the assignment adjustment where the number of sequences involved in each assignment adjustment is *at most two*. Thus, the first challenge can be addressed well.

We present *Adjust* in Algorithm 1.

Example 4.17 (Adjust). Consider our running example shown in Figure 1. Let k be 4. Assume that initialized assignment is the one presented in the bipartite graph shown in Figure 1(a). Note that $sat(A) = 4$ and $profit(A) = 250$. At iteration 1, since $sat(A)$ is equal to k, it aims to search affirmative sequences or two partner sequences. Assume that *Adjust* finds sequence S =(p_5, Gary, p_6, David) which is an affirmative sequence. Then, it adjusts S in A and the bipartite graph in Figure 1(b) presents the resulting assignment. It then updates A as the resulting assignment. Note that $profit(A)$ is increased to 280. At iteration 2, $sat(A) = k$. There exist no affirmative sequences but two partner sequences wrt A namely S =(p_2, Alice) and S' =(p_4, David, p_6, Clement). According to Lemma 4.15, S and S' are compatible. Therefore, *Adjust* performs the adjusting operations on S

and S' simultaneously and the bipartite graph in Figure 1(c) presents the resulting assignment. It then updates A as the resulting assignment. Note that $profit(A)$ is increased to 290. At iteration 3, $sat(A) = 4$ and there exist no affirmative sequences nor two partner sequences. Thus, it returns A as the optimal k-satisfiable assignment. □

We verify the correctness of *Adjust* with Theorem 4.18.

Theorem 4.18 (Correctness). *The* Adjust *algorithm returns the optimal assignment for the k-SAMP problem.* □

Note that in a case when k is set too large that a k-satisfiable assignment is not available, our *Adjust* algorithm (by its current definition in Algorithm 1) would stop in Phase 1 (i.e., the initialization phase) by returning an assignment with the greatest possible satisfiability, says k' ($k' < k$). Fortunately, *Adjust* has a very good feature that its Phase 2 is essentially a procedure which improves the profit of an assignment while retaining its satisfiability for any given assignment, and as a benefit of this feature, *Adjust* can deal with the above case well by performing Phase 2 on the k'-satisfiable assignment returned by Phase 1 and finally returns a k'-satisfiable assignment with the greatest profit, which is the best assignment we can expect (it has the greatest profit among all assignments with the greatest possible satisfiability).

4.4 Detailed Steps and Time Complexity

In Phase 1, we have to initialize a k-satisfiable assignment A. We propose four methods for this phase, namely *Pure*, *P-Match*, *Random* and *P-Seq*. In *Pure*, we iteratively find a gratifying sequence S and perform an adjusting operation on S for k times. The effect of *Pure* is exactly the effect obtained when we execute an algorithm for the maximum matching problem finding k matches only. In *P-Match*, for each customer o, we form a match (p, o) where p is the product which is not matched yet and p has the largest value of $profit(p, o)$ among all possible p's which are matchable with o. If all matches form a k-satisfiable assignment A, we return A. Otherwise, we execute the iterative process in *Pure* repeatedly until A becomes k-satisfiable. *Random* is identical to *P-Match* except that it selects products in P randomly instead of using the profit heuristic. *P-Seq* is the same as *Pure* except that it finds the gratifying sequence S with the greatest value of $PG(S, A)$ among all gratifying sequences for each iteration.

In Phase 2, we have an iterative process. For each *iteration*, we find affirmative sequences and partner sequences in an assignment A when $sat(A) = k$, and find affirmative sequences and profitable sequences in A when $sat(A) > k$. Thus, when $sat(A) = k$, any affirmative sequence and any two partner sequences (together) are *desirable* wrt A. When $sat(A) > k$, any affirmative/profitable sequence is *desirable* wrt A.

Consider an iteration. A straightforward implementation is to enumerate *all* possible feasible sequences one by one and check whether each sequence is desirable. Enumerating all possible feasible sequences is too costly. Instead, we will show that it is *sufficient* to only enumerate some *critical* sequences.

Given a sequence S, we say that S is *critical* if S has the greatest profit gain among all sequences that have the same heads and tails as S. Let $\mathcal{T}$ denote the set of all possible sequences and $\mathcal{T}^*$ denote the set of all critical sequences. We observe the following property.

Lemma 4.19 (Critical Sequences). *Given an assignment A, there exist desirable sequences wrt A in $\mathcal{T}$ iff there exist desirable sequences wrt A in $\mathcal{T}^*$.* □

According to Lemma 4.19, it is sufficient to focus on $\mathcal{T}^*$ for searching desirable sequences. It could be verified that $|\mathcal{T}^*|$ is bounded by $|P| \cdot |O|$ since for each pair of p and o, there exists at most one critical sequence. In the following, we discuss our method for finding all critical sequences in $\mathcal{T}^*$.

We construct a directed graph $G_A(V, E)$ based on assignment A from which we can find all critical sequences. For each product $p \in P$ (customer $o \in O$), we create a vertex p (o) in V. For each *feasible pair* (p, o) wrt A, we create a directed edge (p, o) and assign its weight to be $-profit(p, o)$. For each *match* (p', o') in A, we create a directed edge (o', p') and assign its weight to be $profit(p', o')$.

Note that for a sequence S in the form of $(p_1, o_1, ..., p_m, o_m)$ wrt assignment A, there exists a path π in $G_A(V, E)$ which is in the form of $(p_1, o_1, ..., p_m, o_m)$. We say that π is the *mapped* path of sequence S and vice versa. Besides, we denote the mapped path of sequence S by $path(S)$ and denote the mapped sequence of a path π (if exist) by $seq(\pi)$.

Given an assignment A and its corresponding graph G_A, we have the following interesting property.

Lemma 4.20 (Shortest Path). *Given an assignment A, a sequence S is critical wrt A iff path(S) is the* shortest *simple path from p to o in G_A, where p (o) is the head (tail) of S.* □

Here, the shortest simple path corresponds to the simple path with the minimum total weight along the path.

According to Lemma 4.20, we can compute all critical sequences by computing all shortest pathes from the product types $p \in P$ to the customers $o \in O$. This is very promising that we can find all critical sequences. Unfortunately, since the weights of some edges in G_A are negative, and thus there might exist negative-weight cycles in G_A, finding the shortest simple path from a product type to a customer in G_A with negative-weight cycles is NP-hard [6].

Fortunately, we observe the following interesting property, under our problem setting, that there is *no need* to calculate the shortest simple path when there exists a negative-weight cycle in G_A. This is because the negative-weight cycle corresponds an affirmative sequence, which achieves our major goal to find desirable sequences.

Lemma 4.21 (Negative-weight Cycle). *Given an assignment A and its corresponding graph G_A, for any negative-weight cycle C in G_A, seq(C) is affirmative.* □

Here, cycle C can be represented in the form of a path $\pi : (v_1, v_2, .., v_m)$ where vertex v_i and vertex v_{i+1} are connected for each $i \in [1, m-1]$, and vertex v_1 and vertex v_m are connected. $seq(C)$ is defined to be $seq(\pi)$.

The above discussion suggests the following ways to find desirable sequences. Step 1: We first construct G_A (which takes $O(|V| + |E|)$ time). Step 2: We can then use an existing algorithm (e.g., the Bellman-Ford's algorithm [6]) to check whether there exists a negative-weight cycle in G_A. We denote the time complexity of this existing algorithm by α. Note that the *worst-case* time complexity of the Bellman-Ford's algorithm is $O(|V| \cdot |E|)$ and thus $\alpha = O(|V| \cdot |E|)$. This algorithm can significantly be speeded up in practice with the techniques introduced in [34] (e.g., *early termination*) which are adopted in our experiments. If there exists a negative cycle, then we can find that cycle C out and return $seq(C)$ as an affirmative sequence. Otherwise, we proceed the next step. Step 3: For each product type $p \in P$, we adopt the Johnson's algorithm [6] to find the shortest simple paths on the graph containing no negative-weight cycle from p to all customers (which takes $O(|V| \log |V| + |E|)$ time). Whenever we find paths such that the corresponding sequences are desirable, we returns these sequences. Here, we process product types according to one of the three heuristics, namely *"Capacity"*, *"Profit"* and *"Capacity+Profit"*. Details will be discussed in Section 7.

If no desirable sequences are found after we process all product types, then we know that A is optimal (according to Lemma 4.16). Let γ be the total number of product types we need to process in Step 3. Note that γ is bounded by $|P|$ and is usually $0.15 \cdot |P|$ on average in our experiments. The overall time complexity of finding critical sequences is $O(\alpha + \gamma \cdot (|V| \log |V| + |E|))$.

Time Complexity. As could be verified, the cost of Phase 1 is dominated by the cost of Phase 2. Thus, we focus on the time complexity of Phase 2 (i.e., *Adjust*) only. Let I be the total number of times of executing the while-loop in *Adjust* and the β be the cost of finding critical sequences. Then, the time complexity of *Adjust* is $O(I \cdot \beta)$ which is equal to $O(I \cdot (\alpha + \gamma \cdot (|V| \log |V| + |E|)))$.

5 WEIGHTED K-SAMP

In the weighted k-SAMP problem, for each $p \in P$ ($o \in O$), $p.w$ ($o.w$) can be any arbitrary positive integer. Although we can transform from weighted k-SAMP to un-weighted k-SAMP by duplicating each $o \in O$ $o.w$ times and each $p \in P$ $p.w$ times, and thus can use *Adjust* (Algorithm 1) on this un-weighted problem to find the solution for the weighted problem, this transformation is cumbersome and undesirable, because duplicating each object multiple times is costly. For example, suppose that there is a customer $o \in O$ and a product $p \in P$ such that $o.w = 10,000$ and $p.w = 20,000$. We need to duplicate objects 30,000 times. If all duplicates of o are matched with duplicates of p in the optimal assignment for the weighted k-SAMP, the adapted algorithm have to find an (un-weighted) match 10,000 times. A concise method is to find a *weighted* match *once* with its weight equal to 10,000.

In this section, we first describe the difference between the un-weighted version and the weighted version. Based on this difference, we design some adapted concepts for our weighted version.

In the weighted version, different from the un-weighted version, each object (o or p) in one dataset can be matched with multiple objects in the other dataset. Even an object has a weighted match with another object, it is possible that this object has its *remaining* weight which is not involved in all of its matches. According to this difference, we define the following concepts.

Let A be a weighted assignment. For a product $p \in P$, we define its *free capacity*, denoted by $p.free$, to be the amount of p not assigned in A. That is, $p.free = p.w - \sum_{(p,o,w)\in A} w$. We say that product p has *no free capacity* if $p.free = 0$. For a customer $o \in O$, we define its *deficient demand*, denoted by $o.deficient$, to be the amount of o's demand not satisfied in A. That is, $o.deficient = o.w - \sum_{(p,o,w)\in A} w$. We say customer o has *no deficient demand* if $o.deficient = 0$. Besides, p (o) is said to be *fully matched* with o (p) in A if there exists a weighted match (p, o, w) in A such that $w = \min\{p.w, o.w\}$ in A.

In the weighted version, we also have the concept of feasible sequences which is made up of a number of feasible pairs. In the un-weighed version, for any two adjacent feasible pairs, there is a connector between these two pairs which is a match in the given assignment. Note that in the un-weighted setting, each object involved in this match has no free capacity or no deficient demand. In the weighted version, we also have a similar concept of "adjacency" for any two feasible pairs as follows.

Let A be a (weighted) assignment between P and O. Given $p \in P$ and $o \in O$, we say (p, o) is a *feasible un-matched weighted pair* (or *feasible pair* in short) in A if (p, o) is matchable and p is not fully matched with o in A. Given two feasible pairs in A, namely (p, o) and (p', o'), if (p', o, w) is a (weighted) match in A where w is a positive integer, $o.deficient = 0$ and $p.free = 0$, we say that (1) these two feasible pairs are *adjacent* in A, (2) the weighted match (p', o, w) is the *connector* between (p, o) and (p', o'), (3) (p, o) appears *before* (p', o') in A, and (4) (p', o') appears *after* (p, o) in A.

With the above definition of "adjacency", we have the same definition of feasible sequences in the weighted version (Definition 4.3).

Next, we describe how we define the adjusting operation of a feasible sequence in the weighted version. In the un-weighted version, a sequence has at most one head match and at most one tail match. In the weighted version, a sequence can have multiple head matches and multiple tail matches. Even if it has multiple head/tail matches, its head/tail can have its free capacity/deficient demand.

In this weighted version, we adopt the principle that we should *first* use up all the free capacity of each product (or satisfy the deficient demand of each customer) (if we can increase the profit) and *then* consider breaking some matches involved in each product or customer. Under this principle, we can increase the profit gradually if we execute the adjusting operations on some sequences. Given a sequence S, if S has its head match, the *best head match* in S is defined to be the head match in S in form of (p, o, w) which has the minimum value of $profit(p, o)$. Similarly, we have the definition for the *best tail match* in S. Given a sequence S, we define $M(S)$ to be the set containing all internal matches in S, the best head match in S (if the head in S has no free capacity) and the best tail match in S (if the tail in S has no deficient demand).

With the definition of $M(S)$, we can define the adjusting operation on a sequence S similar to the one in the un-weighted version (Definition 4.5). But, we need to *generalize* this operation which can incorporate the *weight* information. As we described at the beginning of this section, we may execute the (un-weighted) adjusting operation 10,000 times. In order to solve this problem, we associate a *weight* to each adjusting operation where the weight corresponds to the total number of matches involved in the assignment to be broken or the total number of matches to be formed.

With this weight concept associated to each sequence, we have to find the greatest possible amount of matches to be broken (or to be formed) for this sequence as follows. Let $\mathcal{W}$ to be the set of the weights of all internal matches in S. Given a sequence S, we define w_{head} to be the weight of its best head match if its head has no free capacity and to be the free capacity of its head otherwise. Similarly, we have a similar definition of w_{tail} for its tail. Thus, we define this weight associated with a sequence S to be $\min\{w_{head}, w_{tail}, w_{internal}\}$ where $w_{internal} = \min_{w \in \mathcal{W}} w$.

Similar to the un-weighted version, we have the same definition of "compatible" (Definition 4.12) in the weighted version.

Finally, the two important Lemmas 4.15 and 4.16 in the un-weighted problem can still be satisfied under the setting of the weighted problem. With these two lemmas, we can also design a weighted version of algorithm *Adjust*, called *Weighted Adjust*.

Theorem 5.1 (Correctness). *The* Weighted Adjust *algorithm returns the optimal assignment for the weighted k-SAMP problem.* □

6 DISCUSSION

In the previous sections, we focus on the first scenario. In this section, we discuss how our proposed method can be extended to the second scenario.

Firstly, we re-define the concept of "matchable" based on the second scenario. In the second scenario, when we consider the local profit loss tolerance requirement, given a pair (p, o), we say that (p, o) is *matchable* if $o.price > p.cost \times \alpha$ and for each $a \in \mathcal{A}$, $o.a \geq p.a$. When we do not consider this requirement, given a pair (p, o), we say that (p, o) is *matchable* if for each $a \in \mathcal{A}$, $o.a \geq p.a$.

In general, the second scenario is different from the first scenario in the following way. In the second scenario, it is possible that we can assign o with p where $o.price \leq p.cost$ but in the first scenario, it is not possible.

All concepts described in Section 4 can be used for the second scenario. However, we need to introduce one concept related to a match (p, o) where $o.price \leq p.cost$.

Let A be an assignment. In Definition 4.3, we know that a feasible sequence contains at least one feasible pair. Here, we *generalize* the definition of a feasible sequence which covers not only the original definition containing at least one feasible pair, called a *regular feasible sequence*, but also the new definition to be introduced containing no feasible pair, called a *special feasible sequence*.

Given an assignment A and a match $(p, o) \in A$ where $o.price \leq p.cost$, a *special feasible sequence* S wrt A is a sequence of zero length where (1) there is no feasible pair involved, (2) its head match is defined to be (p, o), and (3) both its tail match and its internal matches are defined to none. p is said to be the *head* in S. Note that $SG(S, A) = -1$.

Note that the adjusting operation on a regular feasible sequence creates at least one match based on the feasible pairs wrt this sequence. Besides, it breaks its internal matches (if any), its head match (if any) and its tail match (if any). However, we define the adjusting operation on a special feasible sequence which creates no match (since it involve no feasible pair). Besides, it breaks its head match since the sequence contains its head match only.

For example, suppose that A contains a match (p_5, Alice) where Alice.$price \leq p_5.cost$. Thus, there is a special feasible sequence S wrt A with its head match (p_5, Alice) and its head as p_5.

All lemmas/theorems can be kept the same except the parts related to the new concept. Specifically, only the proofs of Lemma 4.15 and Lemma 4.16 should be updated. The updated proofs can be found in the appendix.

As a result, since we still keep using the same set of lemmas/theorems, the same algorithm *Adjust* can still be used.

7 EMPIRICAL STUDIES

We introduce the experimental setup in Section 7.1 and present the experimental results in Section 7.2.

7.1 Experiment Setup

Datasets. We used both real datasets and synthetic datasets in our experiments. The real datasets are Package [30] and NBA [28]. Package corresponds to a set of trip packages (e.g., hotels and flights) crawled from Priceline.com and Expedia.com and contains 4,936 trip packages each containing 5 attributes and a cost. Details of how the trip packages were crawled can be found in [30]. NBA corresponds to the Great NBA Players' technical statistics from 1960 to 2001, which contains 17,247 players and each has 17 attributes such as points, rebounds, and assists etc. [1]. Since a larger attribute value of a player means that he is a better player, we regard each player's cost as the sum of all of its attribute values. All attribute values and cost are normalized in range $[0, 1]$. For each real product type set P, we generated its corresponding customer set O of size equal to N with two steps where N is the targeted number of customer preferences. First, for each $p \in P$, we generated a customer preference o such that for each attribute a, $o.a$ is set to be $p.a$ plus a random offset falling a Gaussian distribution $\mathcal{N}(\delta, \delta/2)$, where δ a user parameter. As a result, we obtained $|P|$ customer preferences. Second, we randomly sampled N customer preferences from the $|P|$ customer preferences generated at the first step and included them into O. We assumed $N \leq |P|$ in our experiments. The summaries of real datasets are shown in Table 4.

The synthetic datasets were generated with the method in [3] which follows existing studies on preference queries [3, 7, 31]. We used three types of distributions for the product type set, namely *anti-correlated*, *correlated* and *independent*. The anti-correlated distribution indicates that a product that has a good attribute value in one attribute would probably has poor attribute values in other attributes. In contrast, the correlated distribution corresponds to the phenomenon that

[1] NBA Statistics v2.1. http://basketballreference.com/stats download.htm.

a product that has a good attribute value in one attribute would likely have desirable attribute values in other attributes as well. The independent distribution means that the attribute values of a product are generated in a random and independent way. These three types of distributions are common benchmarks for preference-based queries [3, 7, 31], and following [3, 7, 31], we used anti-correlated products by default. All attribute values fall in range $[0, 1]$. For each product type set P, we generated its corresponding set O with N customer preference by first generating N customer preferences in the same way as generating product types and then for each generated customer preference o, adding $o.a$ for each attribute a by a random offset falling a Gaussian distribution $\mathcal{N}(\delta, \delta/2)$, where δ a user parameter.

For the un-weighted version of k-SAMP, the capacity (demand) of each product type (customer) is 1. For the original version (we call it the *weighted* version), we set the capacity (demand) of each product type (customer) to be an integer in range $[1, 10)$ randomly.

Algorithms. We studied our algorithm *Adjust* and also a baseline algorithm *CBC* (Coin-OR Branch and Cut) [2] which is the one of the fastest open-sourced integer linear programming (ILP) solvers according to some existing empirical study [3]. In the following, we introduce the details of formalizing the k-SAMP problem as an ILP problem.

We let n_1 denote the number of customers (i.e., $n_1 = |O|$), n_2 denote the number of product types (i.e., $n_2 = |P|$), r denote the number of pairs of p and o which are matchable ($r = |\{(p, o) \in P \times O | (p, o)\ \textit{is matchable}\}|$). We then order the set of customers, the set of product types, and the set of matchable pairs and let $c[h]$ denote the capacity of the h-th customer in O where $h = 1, 2, .., n_1$, $d[i]$ denote the demand of the i-th product type in P where $i = 1, 2, ..., n_2$, and $f[j]$ denote the profit of the j-th matchable pair where $j = 1, 2, ..., r$.

We define B to be the bipartite graph $O \times P$ where there exists an edge between a customer o and a product type p if (p, o) is matchable. Besides, we order the vertices in B as the 1st customer, the 2nd customer, ..., the n_1-th customer, the 1st product type, the 2nd product type, ..., the n_2-th product type and the edges as the 1st matchable pair, the 2nd matchable pair, ..., the r-th matchable pair. We then define $\mathbf{A}$ to be the *incident matrix* of B. In other words, $\mathbf{A}[i][j] = 1$ if the i-th vertex of B is incident to the j-th edge of B and $\mathbf{A}[i][j] = 0$ otherwise, where $i = 1, 2, ..., n$ where $n = n_1 + n_2$ and $j = 1, 2, ..., r$.

We define $\mathbf{b}$ to be the vector $\langle c[1], c[2], ..., c[n_1], d[1], d[2], ..., d[n_2] \rangle^T$, $\mathbf{f}$ to be the vector $\langle f[1], f[2], ..., f[r] \rangle^T$, and $\mathbf{x}$ to be an r-dimensional non-negative vector. Then, we formulate the k-SAMP problem as an integer linear programming (ILP) problem as follows.

$$\begin{aligned} \max\ & \mathbf{f}^T \cdot \mathbf{x} \\ \text{s.t. } & \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \\ & \mathbf{1} \cdot \mathbf{x} \geq k \\ & \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbb{Z}^r. \end{aligned}$$

In this ILP, the entries in $\mathbf{x}$ are variables, the objective captures the goal of k-SAMP which is to maximize the profit, the first constraint guarantees that the demand constraint and also the capacity constraint are satisfied, the second constraint guarantees that the satisfiability is at least k, and the third constraint guarantees that a product can only be assigned to a customer as a whole.

Factors and Measurements. We studied the following 6 factors, namely the parameter δ, the parameter k (for convenience, we use relative values for k in this section, e.g., $k = 0.6$ means that 60% customers need to be satisfied), the number of product types called P size, the number of

[2] https://projects.coin-or.org/Cbc

[3] http://www.gurobi.com/resources/switching-to-gurobi/open-source-solvers

Table 4. Real datasets

	Package	NBA
$\|P\|$	4,787	17,274
$\|O\|$	488	1,711
Dim.	6	17

Table 5. Synthetic datasets

Factors	Settings
$\|P\|$	5k, **10k**, 15k, 20k, 25k
$\|O\|$	100, 250, 500, 750, **1000**
Dim.	1, **2**, 3
Offset(δ)	**.10**, .11, .12, .13, .14, .15
k	0.6, 0.7, **0.8**, 0.9, 1
r	5,000,000 - 8,000,000

customers called O size, the dimensionality, and the number of matchable pairs r. The settings of the above factors are given in Table 5. We evaluated our algorithm with the following 2 measurements, namely the running time and memory usage.

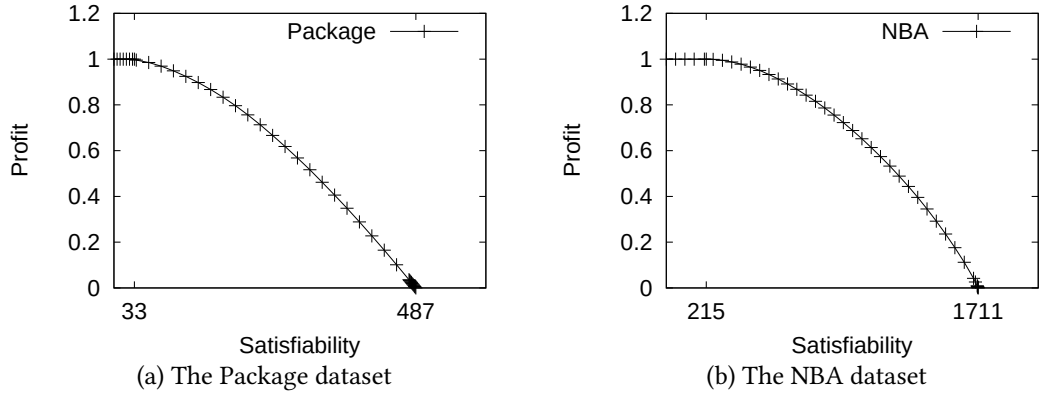
All algorithms were implemented in C/C++, and all experiments were conducted on an IBM X3650 M3 server with 2x6-Core 2.66GHz and 48GB RAM operated by a CentOS linux distribution.

7.2 Experimental results

7.2.1 Profit vs. Satisfiability. In this section, we compared our algorithm with two related algorithms. The first algorithm Alg_{sat} is the algorithm which maximizes its satisfiability while the second algorithm Alg_{profit} is the algorithm which maximizes its profit. Alg_{sat} can be accomplished by the min-cost flow algorithm and Alg_{profit} can be completed by the maximum weight matching algorithm.

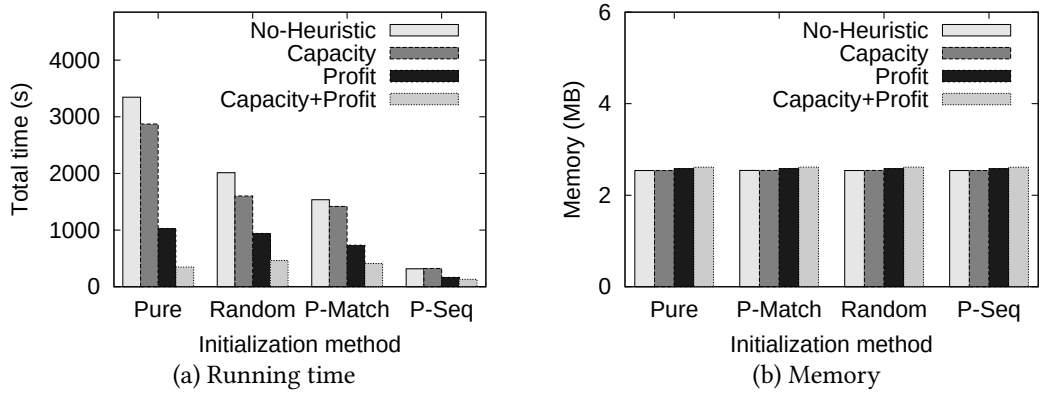
We conducted experiments on our real datasets, Package and NBA. For the sake of comparison, we normalized the profit in range [0, 1]. Consider the results on the Package dataset. We found that Alg_{sat} returned an assignment A_{sat} where $sat(A_{sat}) = 487$ and $profit(A_{sat}) \approx 0$, while Alg_{profit} returned an assignment A_{profit} where $sat(A_{profit}) = 33$ and $profit(A_{profit}) = 1.0$. We also conducted experiments on the effect of the parameter k over the profit of a k-SAMP assignment, and the results are shown in Figure 2(a) on the Package dataset and in Figure 2(b) on the NBA dataset. According to the results, we can see that there is always a non-increasing pattern (or negative correlation) between the parameter k (i.e., the least satisfiability an assignment needs to have) and the greatest profit of a k-satisfiable assignment. This is simply because by definition, each $(k+1)$-satisfiable assignment is also a k-satisfiable assignment but not vice versa. The trend in general is that when k increases from a small number to a large number, the greatest possible profit keeps being the maximum one for a while (when k is small) and then decreases strictly (when k becomes large). The intuition behind this phenomenon is that when k is small, in which case the satisfiability requirement is loose, there is quite much freedom to assign between the products and the customers and a maximum profit could be achieved and when k gets larger, due to the satisfiability requirement, some matches that generate larger profits need to be broken in order for some more matches that generate smaller profits to be formed, and as a result, the satisfiability increases (for satisfying the satsifiability requirement) while the profit decreases.

7.2.2 Initialization Methods and Heuristics. In this part, we evaluated the 4 initialization methods in Phase 1 described in Section 4.4, namely *Pure*, *Random*, *P-Match* and *P-Seq*. Besides, we tested the effectiveness of the 3 heuristics in Phase 2, i.e., *"Capacity"*, *"Profit"* and *"Capacity+Profit"*, by comparing them with the algorithm without any heuristic, which is denoted by *"No-Heuristic"*. In *"Capacity"*, the product types in P are sorted in descending order of their free capacities (the amount of capacity not assigned yet). In *"Profit"*, each product type p is given a *label* equal to



(a) The Package dataset (b) The NBA dataset

Fig. 2. Profits of k-SAMP assignments with varying k's



(a) Running time (b) Memory

Fig. 3. Effects of initialization methods & heuristics (Un-weighted version)

$\min_{(p,o,w)\in A} profit(p,o)$ if it has no free capacity and equal to 0 otherwise. The product types are sorted in ascending order of their labels. In *"Capacity+Profit"*, the product types are sorted in descending order of their free capacities and then in ascending order of their labels.

In our experiment, we studied each possible combination of 4 initialization methods and 4 heuristics (including *"No-Heuristic"*), and the results are shown in Figure 3. According to Figure 3(a), it is clear that the combination of *P-Seq* (Initialization method) and "*Capacity+Profit*" (Heuristic) runs the fastest. According to Figure 3(b), we know that choices of an initialization method and a heuristic have no significant effect on the memory usage of the *Adjust* algorithm, and this is because the component that dominates the memory usage of *Adjust* is the component for storing matchable pairs and the matches of the assignment to be computed which is independent of the choices of an initialization method and a heuristic. In conclusion, the combination of *P-Seq* (Initialization method) and "*Capacity+Profit*" (Heuristic) runs faster and occupies slightly more memory than all other combinations. Therefore, in the rest of experiments, we adopted the combination of *P-Seq* and "*Capacity+Profit*" only for our *Adjust* algorithm and when we say the "*Adjust*" algorithm, we mean the version that employs the combination of *P-Seq* and "*Capacity+Profit*".

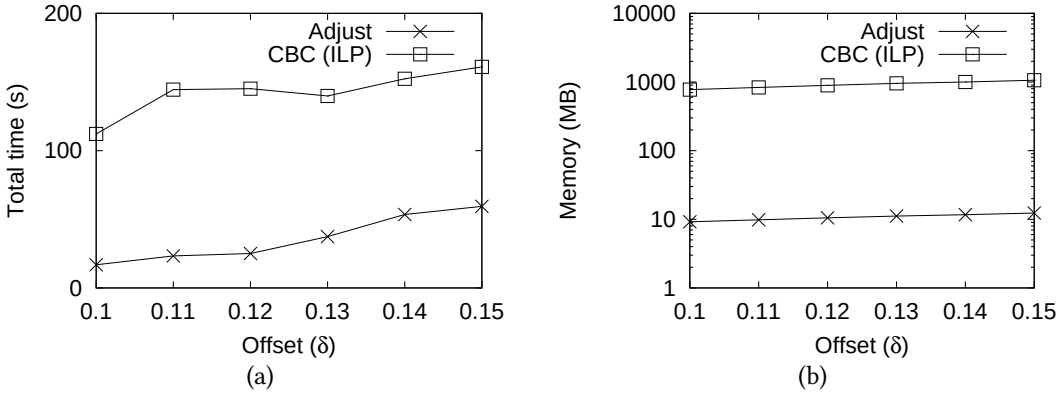


Fig. 4. Effects of the offset parameter (δ) (Real dataset Package, Un-weighted version)

7.2.3 Performance studies. In this part, we conducted performance studies on the two algorithms, *Adjust* and CBC (which we denote by CBC (ILP) in the figures), by varying different settings for the factors and seeing their effects on the measurements.

Offset parameter δ. We varied δ with values from $\{0.10, 0.11, 0.12, 0.13, 0.14, 0.15\}$, and the results on the Package dataset, the NBA dataset, synthetic datasets are shown in Figure 4, Figure 5, and Figure 6, respectively. Consider the results in Figure 4 for example. We can see that (1) the running times of both *Adjust* and CBC increase with δ in general; (2) the running time of CBC is up to 7 times larger than that of CBC (in the default setting of $\delta = 0.10$, the running time of CBC is 112.1s and that of *Adjust* is 16.8s); (3) the memories of both algorithms increase slightly with δ also; and (4) the memory of CBC is consistently about 2 orders of magnitudes larger than that of *Adjust*. The high-level idea of explaining (1) and (3) is that when δ increases, the attributes of the customers generated become larger and it further implies that more pairs of customers and product types would be matchable. As a result, more candidates need to be considered for the matching process, and thus more computation cost (e.g., running time and memory) is incurred (in the case of the *Adjust* algorithm, it means that the input graph is larger and in the case of the CBC algorithm, it means that the number of variables is bigger). The reason for (2) is probably that *Adjust* is an algorithm specifically designed for the k-SAMP problem while CBC is an algorithm designed for all general ILP problems and does not necessarily work efficiently for the k-SAMP problem. Last, the reason for (4) is that CBC needs to maintain a $n \times r$ matrix which is very expensive since $n \times r$ has a cubic complexity in terms of $|P|$ and/or $|O|$ while *Adjust* has its at most quadratic.

Parameter k. We varied the parameter k with values from $\{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, and the results on the dataset Package, the dataset NBA, synthetic datasets are shown in Figure 7, Figure 8, and Figure 9, respectively. Consider the results on the dataset Package for example. We can see that (1) the effects of parameter k on both the running time and memory for two algorithms are insignificant, (2) the running time of CBC is about 10 times larger than that of *Adjust*; (3) the memory of CBC is about 2 orders of magnitude larger than that of *Adjust* except for the setting of $k = 1$; and (4) the running time and memory of two algorithms drop when k changes from 0.9 to 1. The reason for (1) is that in the case of *Adjust*, when k increases, though it takes more time (or iterations) to initialize a k-satisfiable assignment during Phase 1, there would be less room left for adjusting the assignment and thus it needs less time for adjusting during Phase 2, and in the case of CBC, the number of variables in the underlying ILP simply does not relies on parameter k. The reason

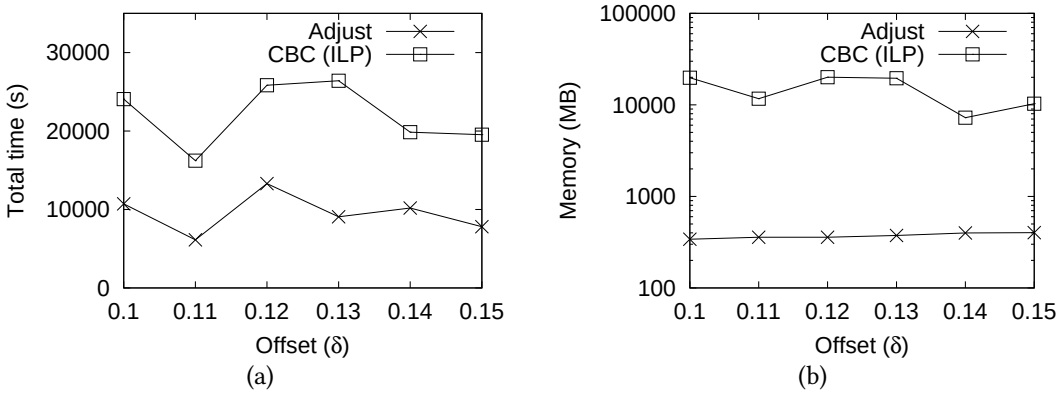


Fig. 5. Effects of the offset parameter (δ) (Real dataset NBA, Un-weighted version)

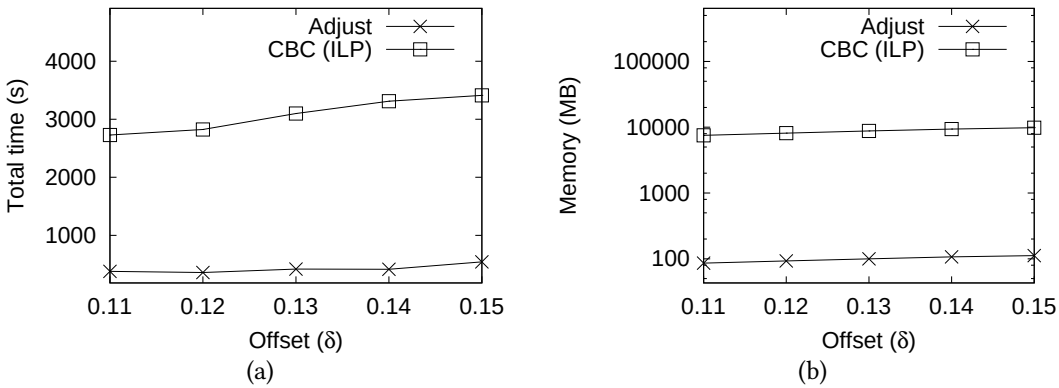


Fig. 6. Effects of the offset parameter (δ) (Synthetic dataset, Un-weighted version)

for (2) and (3) is similar to that for explaining (2) and (4) of varying the offset. Last, the reason for (4) is that in the setting of $k = 1$ which means all customers need to be satisfied, both algorithms terminate earlier after detecting this is the case (e.g., in the case of *Adjust*, Phase 1 for initializing a k-satisfiable assignment cannot be finished and then Phase 2 is skipped).

P size $|P|$. We varied $|P|$ with values from $\{5k, 10k, 15k, 20k, 25k\}$, and the results are shown in Figure 10. We can see that (1) the running times and memories of both algorithms increase with $|P|$; (2) the running time of CBC is up to 11 times larger than that of *Adjust* (e.g., in the setting of $|P| = 25k$, CBC ran for 38,963.9s and *Adjust* ran for 3,449.8s) and the gap increases with $|P|$; and (3) the memory of CBC is consistently larger than that of *Adjust* by 2 orders of magnitude. The reason for (1) is simply that when $|P|$ increases, in the case of *Adjust*, the underlying graph becomes larger and in the case of CBC, the number of variables becomes larger, and as a result, the computation cost (running time and memory) increases. Again, (2) and (3) can be explained similarly as when varying the offset and parameter k.

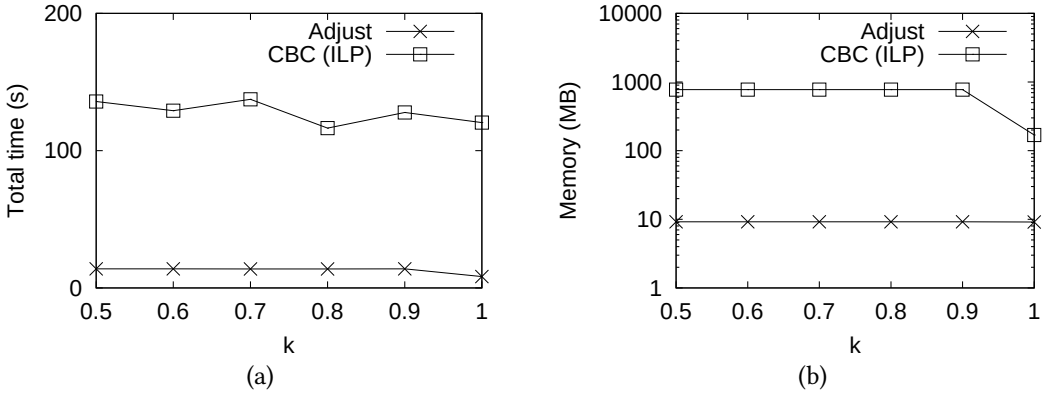


Fig. 7. Effects of k (Real dataset Package, Un-weighted version)

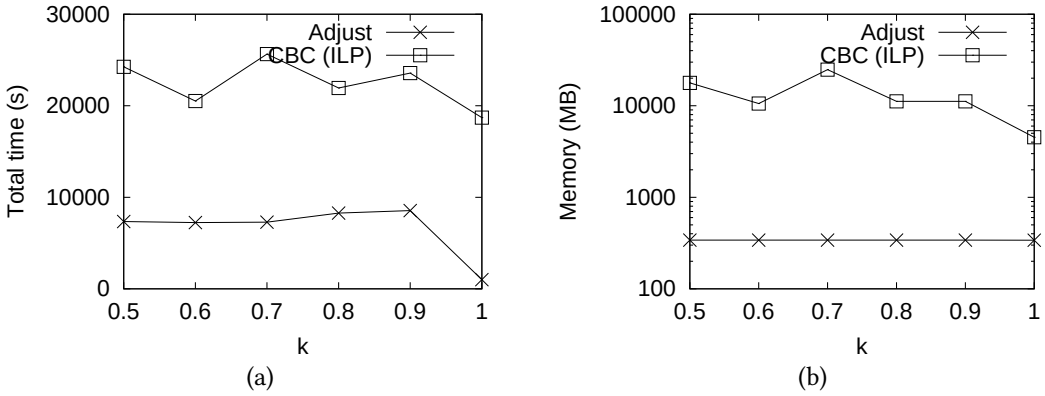


Fig. 8. Effects of k (Real dataset NBA, Un-weighted version)

O size $|O|$. We varied $|O|$ with values from $\{100, 250, 500, 750, 1000\}$, and the results are shown in Figure 11. The results are quite similar to those when varying $|P|$ and could also be explained similarly.

No. of matchable pairs (r). We used real datasets for this experiment and generated datasets with different number of matchable pairs by using different offset values (recall that a larger offset would result in a dataset with more matchable pairs). We varied the number of matchable pairs r with some values from $5,000,000$ to $8,000,000$, and the results on the dataset NBA are shown in Figure 12 (those on the dataset Package are similar than they are omitted). We can see that the running times and memories of both algorithms increase with the number of matchable pairs and this is clearly the reasonable results since the number of matchable pairs determines the complexity of the problem to an significant extent (more matchable pairs means more variables in the ILP problem for CBC and bigger graph for *Adjust*).

Dimensionality. We varied the dimensionality with values from $\{1, 2, 3\}$, and the results are shown in Figure 13. We can see that (1) the running times and memories of both both algorithms decrease when the dimensionality increases; and (2) the gap of running time between CBC and

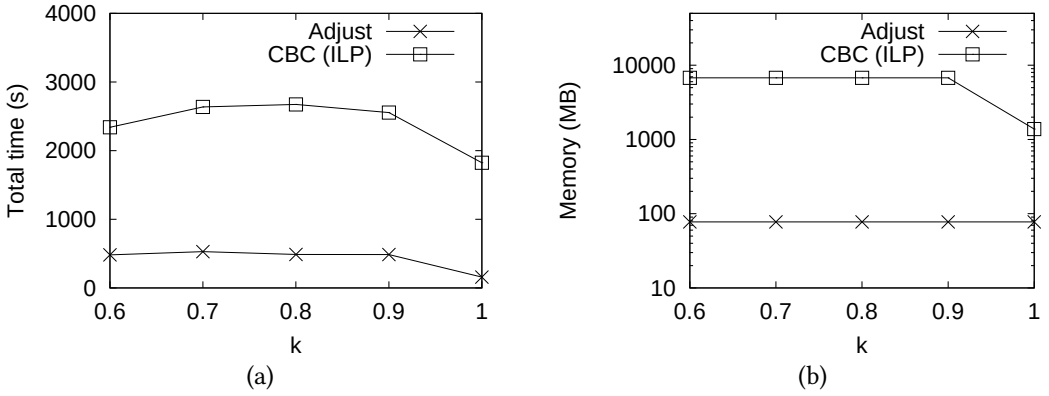


Fig. 9. Effects of k (Synthetic dataset, Un-weighted version)

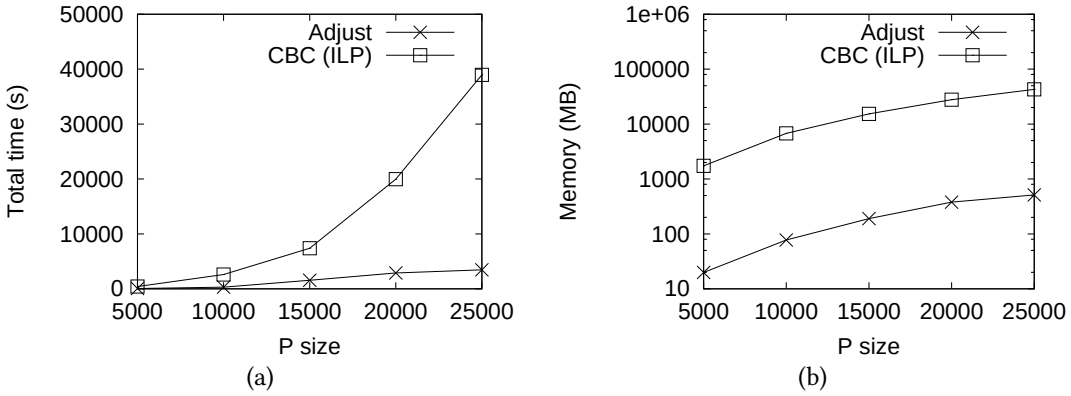


Fig. 10. Effects of P size ($|P|$) (Synthetic dataset, Un-weighted version)

Adjust becomes smaller when the deimensionality increases. The reason for (1) is that a larger dimensionality means that a customer gives more attribute values on the product types, and thus it is more likely that this customer cannot be satisfied, which further implies that each customer can be satisfied by fewer candidates (product types), i.e., there are fewer matchable pairs. As a result, less computation cost (running time and memory) is needed. The reason for (2) is probably that the number of matchable pairs affects CBC more directly than *Adjust* since it determines the number of variables on which the performance of CBC directly rely.

Scalability test. We vary $|P|$ with values from $\{25k, 50k, 75k, 100k\}$ for scalability test. In each setting of $|P|$, $|O| = |P|/10$. The results of our *Adjust* algorithm are shown in Figure 14 and those of the CBC algorithm are not shown since it ran out of memory (i.e., it occupied more than 48GB memory) in the settings of $75k$ and $100k$ and could not be finished within a reasonable amount of time in the other settings. According to results, we can see that *Adjust* ran for less than 1.5 days and occupied about 12GB when $|P| = 100k$ (and $|O| = 10k$).

Weighted version. We performed similar experiments for the weighted version of k-SAMP and the results are similar to those for the un-weighted version. For simplicity, we only show the results

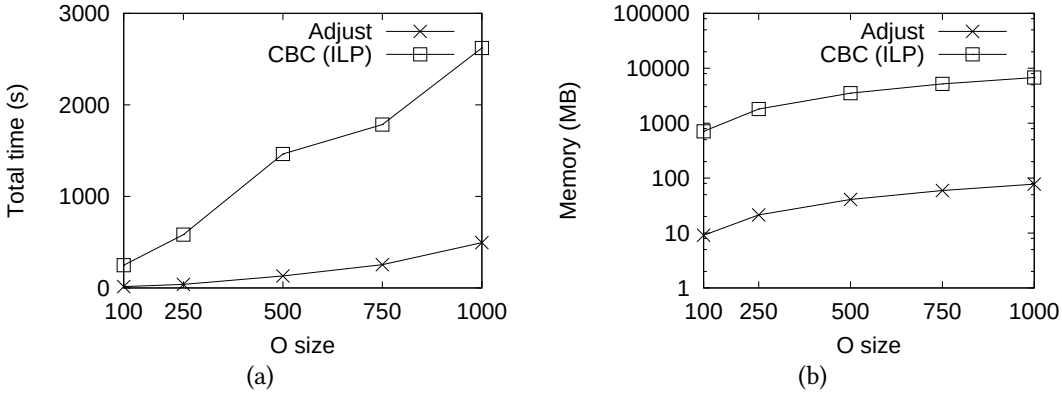


Fig. 11. Effects of O size ($|O|$) (Synthetic dataset, Un-weighted version)

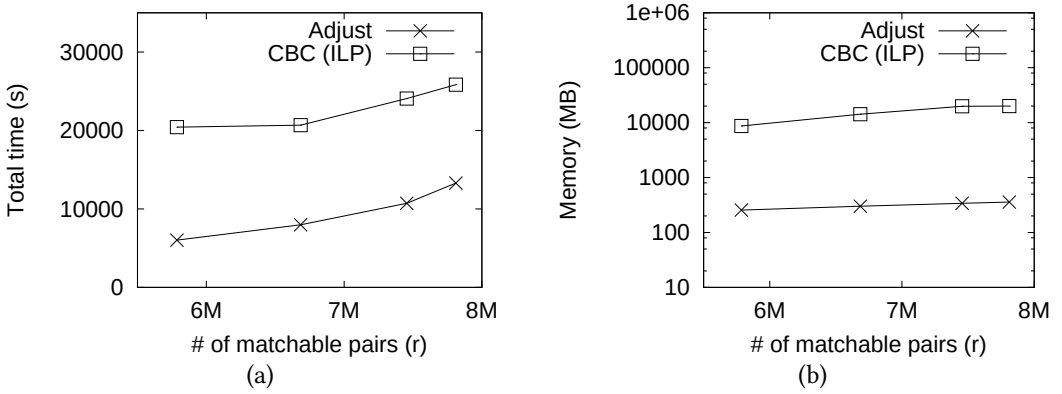


Fig. 12. Effects of no. of matchable pairs (Real dataset NBA, Un-weighted version)

on one of the real datasets Package in Figure 15 (varying the offset parameter δ) and Figure 16 (varying the parameter k).

7.2.4 Conclusion of the Experimental Results. Our *Adjust* algorithm is about 5-10 times faster than the CBC algorithm for the majority of the experimental settings and occupies less memory than the CBC algorithm by about 2 orders of magnitude.

8 CONCLUSION

In this paper, we propose a new problem called k-SAMP, which has a lot of practical applications for decision-making. We propose an efficient algorithm *Adjust* for this problem. Finally, we conducted experiments which verified the efficiency of our algorithm. There are several interesting future directions. One direction is to study the problem of maximizing the (average) profit that we can earn from a *single* customer satisfied. Another direction is to study the k-SAMP problem with an additional constraint that each customer can be matched with at most one product type. Besides, one can consider the k-SAMP problem with other definitions of $sat(A)$, e.g., the total number of customers that are satisfied *completely* in assignment A. Lastly, one possible direction is to study

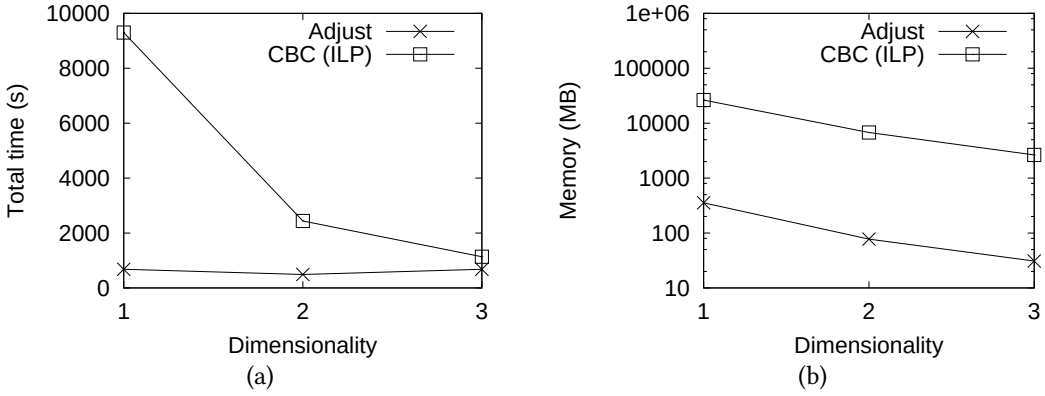


Fig. 13. Effects of dimensionality (Synthetic datasets, Un-weighted version)

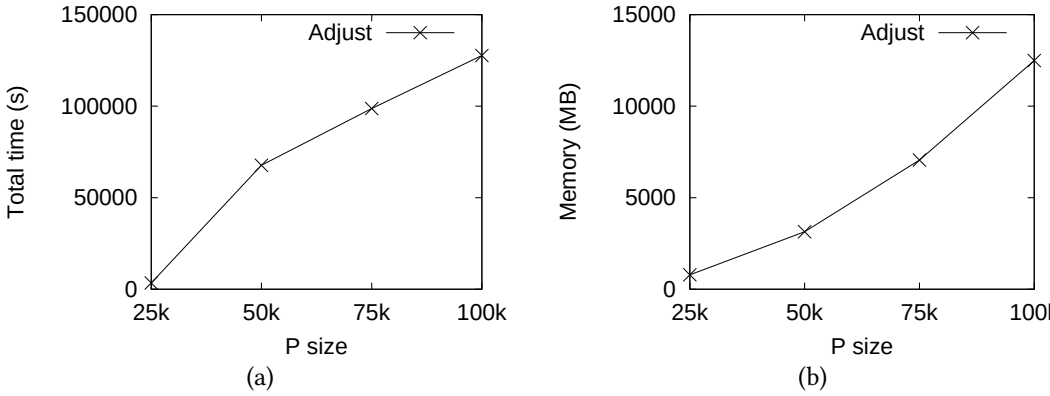


Fig. 14. Scalability test (Synthetic datasets, Un-weighted version)

how to set k when the minimum profit to be earned is specified. Another direction is to study the k-SAMP problem in the context where customers' preferences and satisfactions are modeled with latent factors, which are computed with collective behavioral data.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. 1993. *Network flows: theory, algorithms, and applications.* Prentice Hall.
[2] Amos Azaria, Avinatan Hassidim, Sarit Kraus, Adi Eshkol, Ofer Weintraub, and Irit Netanely. 2013. Movie recommender system for profit maximization. In *Proceedings of the 7th ACM conference on Recommender systems.* ACM, 121–128.
[3] S. Börzsönyi, D. Kossmann, and K. Stocker. 2001. The skyline operator. In *ICDE.* 421–430.
[4] R. Burkard, M. Dell'Amico, and S. Martello. 2009. *Assignment problems.* Society for Industrial Mathematics.
[5] Long-Sheng Chen, Fei-Hao Hsu, Mu-Chen Chen, and Yuan-Chia Hsu. 2008. Developing recommender systems with the consideration of product profitability for sellers. *Information Sciences* 178, 4 (2008), 1032–1048.
[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms.* The MIT Press.
[7] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. 2000. Closest pair queries in spatial databases. *ACM SIGMOD Record* 29, 2 (2000), 189–200.
[8] Aparna Das, Claire Mathieu, and Daniel Ricketts. 2009. Maximizing profit using recommender systems. *arXiv preprint arXiv:0908.3633* (2009).
[9] M. Endres and W. KieSSling. 2008. Optimization of Preference Queries with Multiple Constraints. In *VLDB.*

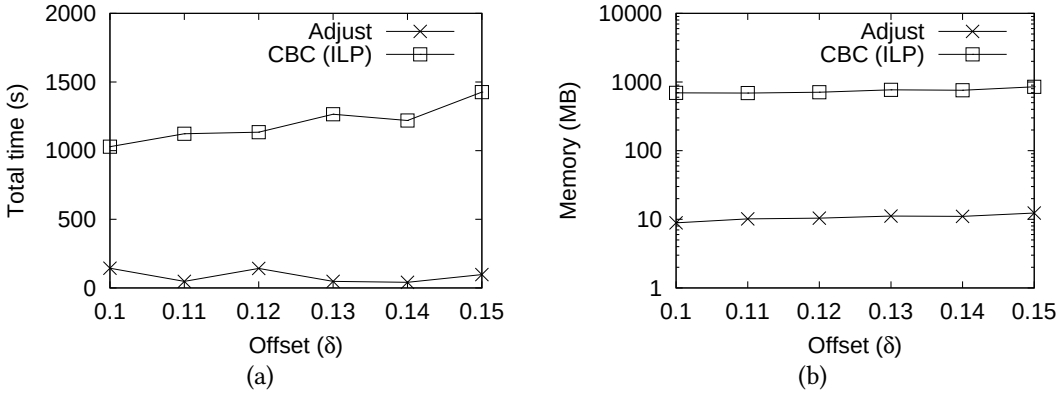


Fig. 15. Effects of the offset parameter (δ) (Real dataset Package, Weighted version)

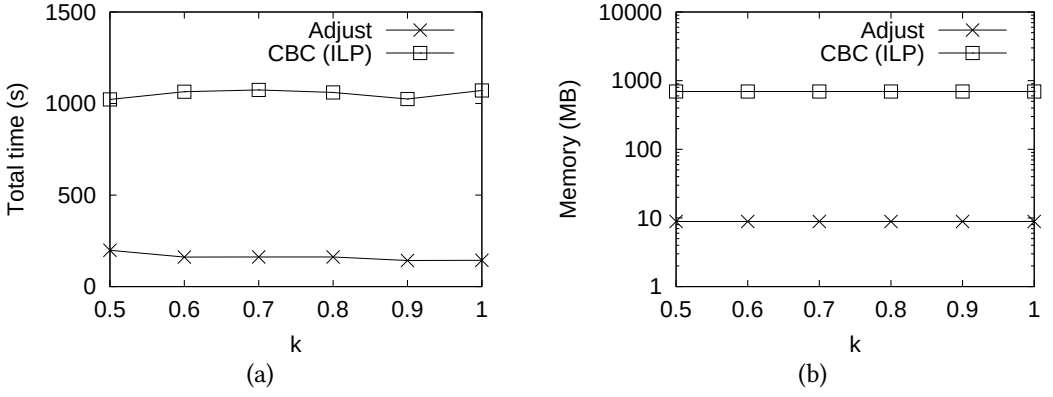


Fig. 16. Effects of parameter k (Real dataset Package, Weighted version)

[10] D. Gale and L. S. Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* (1962).

[11] Yong Ge, Qi Liu, Hui Xiong, Alexander Tuzhilin, and Jian Chen. 2011. Cost-aware travel tour recommendation. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 983–991.

[12] W. Gu and X. Wang. 2010. Studies on the Performance of a Heuristic Algorithm for Static and Transportation Facility Location Allocation Problem. In *High Performance Computing and Applications*.

[13] V. Hristidis, N. Kourdas, and Y. Papakonstantinou. 2001. Prefer: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *SIGMOD*.

[14] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han. 2008. Mining Preferences from Superior and Inferior Examples. In *KDD*.

[15] W. Kiebling. 2002. Foundation of Preferences in Database Systems. In *VLDB*.

[16] T. C. Koopmans and M. Beckmann. 1957. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society* (1957), 53–76.

[17] C. Li, B. C. Ooi, A. K.H. Tung, and S. Wang. 2006. DADA: A Data Cube for Dominant Relationship Analysis. In *SIGMOD*.

[18] X. Lian and L. Chen. 2008. Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases. In *SIGMOD*.

[19] Wei Lu, Shanshan Chen, Keqian Li, and Laks VS Lakshmanan. 2014. Show me the money: dynamic recommendations for revenue maximization. *Proceedings of the VLDB Endowment* 7, 14 (2014), 1785–1796.
[20] Silvano Martello Mauro Dell Amico. 1997. The k-cardinality assignment problem. *Discrete Applied Mathematics* 76 (1997), 103–121.
[21] D. Mindolin and J. Chomicki. 2009. Discovering Relative Importance of Skyline Attributes. In *VLDB*.
[22] I. R. Misner. 1999. *The World's best known marketing secret: Building your business with word-of-mouth marketing*. Bard Press, 2nd edition.
[23] J. Munkres. 1957. Algorithms for the assignment and transportation problems. *J. Soc. Indust. Appl. Math.* 5, 1 (1957), 32–38.
[24] J. Nail. 2004. The Consumer Advertising Backlash. *Forrester Research* (2004).
[25] A. Ockenfels and A. E. Roth. 2006. Late and multiple bidding in second price Internet auctions: Theory and evidence concerning different rules for ending an auction. *Games and Economic Behavior* 55, 2 (2006), 297–320.
[26] C. H. Papadimitriou and K. Steiglitz. 1998. *Combinatorial optimization: algorithms and complexity*. Dover Pubns.
[27] Priceline.com. 2013. Name Your Own Price. In *http://www.priceline.com/*.
[28] L. H. U, N. Mamoulis, and K. Mouratidis. 2009. A fair assignment algorithm for multiple preference queries. In *VLDB*.
[29] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag. 2010. Reverse Top-k Queries. In *ICDE*.
[30] Q. Wan, R. C. W. Wong, and Y. Peng. 2011. Finding top-k profitable products. In *ICDE*. 1055–1066.
[31] R. C. W. Wong, J. Pei, A. W. C. Fu, and K. Wang. 2007. Mining favorable facets. In *KDD*. 804–813.
[32] R. C. W. Wong, Y. Tao, A. W. C. Fu, and X. Xiao. 2007. On efficient spatial matching. In *VLDB*. 579–590.
[33] Jingyuan Yang, Chuanren Liu, Mingfei Teng, Hui Xiong, March Liao, and Vivian Zhu. 2015. Exploiting temporal and social factors for b2b marketing campaign recommendations. In *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 499–508.
[34] Jin Y. Yen. 1970. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quart.Appl.Math* 27, 4 (1970), 526–530.
[35] M. L. Yiu, K. Mouratidis, and N. Mamoulis. 2008. Capacity constrained assignment in spatial databases. In *SIGMOD*.
[36] Yongfeng Zhang, Qi Zhao, Yi Zhang, Daniel Friedman, Min Zhang, Yiqun Liu, and Shaoping Ma. 2016. Economic recommendation with surplus maximization. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 73–83.

Proof of Lemmas/Theorems

Proof of Lemma 3.1: This could be verified by the fact that sat_{max} is the greatest possible satisfiability. □

Proof of Lemma 3.2: We prove Lemma 3.2 by constructing a problem instance such that $profit(A)/profit(A_o) \rightarrow 0$. Consider Figure 17(a). We have two products, p_1 and p_2, and two customers, o_1 and o_2. For each matchable pair in the form of (p, o), we link p and o with a dashed line, and the number along the dashed line represents $profit(p, o)$. For example, (p_1, o_1) is matchable with $profit(p_1, o_1)$ equal to $profit$.

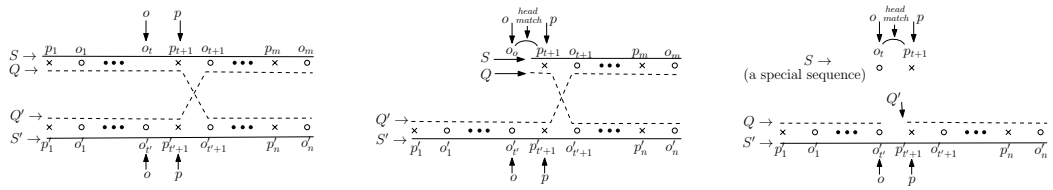
It is easy to verify that $A = \{(p_1, o_2), (p_2, o_1)\}$ with its satisfiability equal to 2 (2 is the greatest possible satisfiability for this problem instance). As a result, we have $profit(A) = 2$. Assume $profit > 2$ and the parameter k is equal to 1 in our k-SAMP problem. With this configuration, A_o would be $\{(p_1, o_1)\}$, since $sat(A_o)$ is at least k (i.e., 1) and A_o gives the greatest possible profit. Note that $profit(A_o) = profit$. Thus, we have $profit(A)/profit(A_o) = 2/profit$. Clearly, when $profit \rightarrow \infty$, $profit(A)/profit(A_o) \rightarrow 0$. □

Proof of Lemma 3.3: The k-SAMP problem with $k = 0$ maximizes the profit of the assignment imposing no requirement on the satisfiability of the assignment, which is identical to the maximum weight matching problem. □

Proof of Lemma 3.4: We prove Lemma 3.4 by constructing a problem instance such that $sat(A)/sat(A_o) \approx 0$, where A is the assignment for the maximum weight matching and A_o is the optimal assignment for our k-SAMP problem. Consider Figure 17(b). We have n products, p_i for $1 \leq i \leq n$, and n customers, o_i for $1 \leq i \leq n$. For each matchable pair in the form of (p, o), we link p and o with a dashed line and the number along the dashed line represents $profit(p, o)$.

(a) (b)

Fig. 17. Problem instances



(a) (b) (c)

Fig. 18. Proof of Lemma 4.15

Clearly, A corresponds to $\{(p_1, o_1)\}$ with the profit equal to 1. Thus, $sat(A) = 1$. Assume that $k = n$ in our k-SAMP problem. Then, A_o is $\{(p_i, o_i)|1 \le i \le n\}$. Thus, $sat(A_o) = n$. As a result, we have $sat(A)/sat(A_o) = 1/n$, which approaches 0 when $n \to \infty$. □

Proof of Lemma 4.9: If a sequence S contains no head match and no tail match, $SG(S, A) = 1$. If it contains exactly one of the head match and the tail match, $SG(S, A) = 0$. If it contains both its head match and its tail match, we consider two cases. The first case is that these two matches are distinct, then $SG(S, A) = -1$. The second case is that they are identical, then $SG(S, A) = 0$. □

Proof of Lemma 4.14: We prove by contradiction. Assume that $N(S) \cap N(S') \neq \emptyset$. Let (p, o) be one such feasible pair in $N(S) \cap N(S')$. We consider 2 cases with p and o. Case 1: none of p and o is matched in A. In this case, both S and S' correspond to the same sequence (p, o). This leads to a contradiction since S and S' are distinct. Case 2: at least one of p and o is matched in A. Without loss of generality, assume p is matched, say with o', in A. In this case, $(p, o') \in M(S)$ and $(p, o') \in M(S')$. Thus, $M(S) \cap M(S') \neq \emptyset$, which leads to a contradiction. This completes the proof. □

Proof of Lemma 4.15: We prove by contradiction. Let the profitable sequence S be $(p_1, o_1, p_2, o_2, ..., p_m, o_m)$ and the gratifying sequence S' be $(p'_1, o'_1, p'_2, o'_2, ..., p'_n, o'_n)$. Suppose that S and S' are not compatible. Thus, $M(S) \cap M(S') \neq \emptyset$. Let (p, o) be one of the matches in $M(S) \cap M(S')$.

Consider S. Since S is profitable, it has its head match (p_1, o_0) and its tail match (p_{m+1}, o_m). Besides, (p_1, o_0) and (p_{m+1}, o_m) cannot be identical (i.e., $p_1 \neq p_{m+1}$ and $o_0 \neq o_m$) (This is because otherwise, $SG(S, A)$ becomes 0 and thus S is not profitable, which leads to a contradiction). Clearly, $M(S) = \{(p_{i+1}, o_i)|i = 0, 1, ..., m\}$. Since $(p, o) \in M(S)$, we know $\exists\, t \in [0, m]$ such that $(p, o) = (p_{t+1}, o_t)$ (i.e., p is p_{t+1} and o is o_t).

Consider S'. Since S is gratifying, it has no head match and no tail match. Clearly, $M(S') = \{(p'_{i+1}, o'_i)|i = 1, 2, ..., n-1\}$. Since $(p, o) \in M(S')$, we know $\exists\, t' \in [1, n-1]$ such that $(p, o) = (p'_{t'+1}, o'_{t'})$ (i.e., p is $p'_{t'+1}$ and o is $o'_{t'}$).

Consider two cases. *Case 1 (General Case):* $0 < t < m$ and $0 < t' < n$. We divide the matches in $M(S)$ into two parts. The first part involves (p_{i+1}, o_i) for $i = 0, 1, ..., t$. The second part involves (p_{i+1}, o_i) for $i = t+1, t+2, ..., m$. Let $L_1 = \sum_{i=0}^{t} profit(p_{i+1}, o_i)$ and $L_2 = \sum_{i=t+1}^{m} profit(p_{i+1}, o_i)$. We also divide the feasible pairs of S into two parts. The first part involves (p_i, o_i) of S for $i = 1, 2, ..., t$. The second part involves (p_i, o_i) of S for $i = t+1, t+2, ..., m$. Let $G_1 = \sum_{i=1}^{t} profit(p_i, o_i)$ and $G_2 = \sum_{i=t+1}^{m} profit(p_i, o_i)$. It could be verified that $PG(S, A) = G_1 + G_2 - (L_1 + L_2)$.

Similar to S, we define L'_1, L'_2, G'_1 and G'_2 for S'. Specifically, $L'_1 = \sum_{i=1}^{t} profit(p'_{i+1}, o'_i)$, $L'_2 = \sum_{i=t+1}^{n-1} profit(p'_{i+1}, o'_i)$, $G'_1 = \sum_{i=1}^{t} profit(p'_i, o'_i)$ and $G'_2 = \sum_{i=t+1}^{n} profit(p'_i, o'_i)$. As a result, $PG(S', A) = G'_1 + G'_2 - (L'_1 + L'_2)$.

Next, we construct two sequences, Q and Q', based on S and S'. Specifically, Q is constructed as $(p_1, o_1, p_2, o_2, ..., p_t, o_t, p'_{t'+1}, o'_{t+1}, ..., p'_n, o'_n)$ (the first part of S concatenated with the second part of S') and Q' is constructed as $(p'_1, o'_1, p'_2, o'_2, ..., p'_{t'}, o'_{t'}, p_{t+1}, o_{t+1}, ..., p_m, o_m)$ (the first part of S' concatenated with the second part of S). Note that (p, o), (p_{t+1}, o_t) and $(p'_{t'+1}, o'_{t'})$ corresponds to the same match with different symbols. See Figure 18(a) for illustration, where nodes that pass along *solid* lines form sequences S and S' while those pass along *dashed* lines form sequences Q and Q'. As a result, we can verify that $PG(Q, A) = G_1 + G'_2 - (L_1 + L'_2)$ and $PG(Q', A) = G'_1 + G_2 - (L'_1 + L_2)$. It follows that

$$PG(Q, A) + PG(Q', A) = PG(S, A) + PG(S', A) \tag{1}$$

Note that Q has its head match but no tail match. Thus, $SG(Q, A) = 0$. Similarly, Q' has its tail match but no head match. Thus, $SG(Q', A) = 0$. We further conclude that $PG(Q, A) \leq 0$ and $PG(Q', A) \leq 0$ (since no affirmative sequences exist in A). Thus, $PG(Q, A) + PG(Q', A) \leq 0$. With Equation (1), we deduce that $PG(S, A) + PG(S', A) \leq 0$, which contradicts our assumption that $PG(S, A) + PG(S', A) > 0$.

Case 2 (Boundary Case): Case 1 is a general case where t is not equal to 0 and m, and t' is not equal to 0 and n. There are different boundary cases. The proof for each boundary case is similar. Here, we consider one boundary case that $t = 0$ and $0 < t' < n$. We construct Q as $(p_{t+1}, o'_{t'+1}, p'_{t'+2}, o'_{t'+2}, ..., p'_n, o'_n)$ and Q' as $(p'_1, o'_1, p'_2, o'_2, ..., o'_{t'}, p'_{t'+1}, o_{t+1}, p_{t+2}, o_{t+2}, ..., p_m, o_m)$ (See Figure 18(b) for illustration). After that, we can also come up with the same contradiction as Case 1. □

Proof of Lemma 4.16: In this proof, we first give the following lemma, which is used to prove Lemma 4.16.

Lemma .1 (Transformable). *Let A be a k-satisfiable assignment and A_o be the optimal assignment for k-SAMP. There exists a set $\mathcal{T}$ of sequences in A such that any two sequences in $\mathcal{T}$ are compatible and the resulting assignment A' obtained due to the adjusting operations on all sequences in $\mathcal{T}$ is A_o.*

□

We prove Lemma 4.16 by contradiction. Suppose that A (which satisfies C1 or C2) is not optimal. Let A_o be the optimal assignment. We have $profit(A) < profit(A_o)$. According to Lemma .1, there exists a set $\mathcal{T}$ of sequences $\{S_1, S_2, ..., S_m\}$ such that any two sequences in $\mathcal{T}$ are compatible and the adjusting operations on all sequences from $\mathcal{T}$ transforms A to A_o.

Let $\mathcal{T}_p$ be the set containing all sequences in $\mathcal{T}$ where each sequence S in this set has $PG(S, A) > 0$. First, $\mathcal{T}_p \neq \emptyset$. Since otherwise, $profit(A_o) \leq profit(A)$, which leads a contradiction. Second, $SG(S, A) < 0$ for each sequence $S \in \mathcal{T}_p$ (since otherwise, a sequence $S \in \mathcal{T}_p$ with $SG(S, A) \geq 0$ corresponds to an affirmative sequence, which contradicts our assumption). We denote by S_m the sequence with the greatest profit gain in $\mathcal{T}_p$. Note that S_m is profitable (since $SG(S_m, A) < 0$ and $PG(S_m, A) > 0$).

Let $\mathcal{T}_g$ be the set containing all sequences S in $\mathcal{T}$ with $SG(S, A) > 0$. We know $\mathcal{T}_g \cap \mathcal{T}_p = \emptyset$ (since otherwise, a sequence that is contained by both $\mathcal{T}_p$ and $\mathcal{T}_g$ corresponds to an affirmative sequence, which contradicts our assumption). We deduce that each sequence S in $\mathcal{T}_g$ has $PG(S, A) \leq 0$. Let S'_m be the sequence with the greatest profit gain in $\mathcal{T}_g$. Note that S'_m is gratifying (since $SG(S'_m, A) > 0$ and $PG(S'_m, A) \leq 0$).

Consider two cases. *Case 1*: $sat(A) = k$. Then,

$$|\mathcal{T}_g| \geq |\mathcal{T}_p| \tag{2}$$

This is because otherwise, A_o cannot be k-satisfiable (each sequence $S \in \mathcal{T}_p$ has $SG(S, A) = -1$ and each sequence $S' \in \mathcal{T}_g$ has $SG(S', A) = 1$). Besides,

$$PG(S_m, A) + PG(S'_m, A) \leq 0 \tag{3}$$

This is because otherwise, S_m and S'_m correspond to two partner sequences, which leads a contradiction.

With Equation (2) and Equation (3), we further deduce that

$$\begin{aligned} \textstyle\sum_{S \in \mathcal{T}_p} PG(S, A) + \sum_{S' \in \mathcal{T}_g} PG(S', A) &\leq \\ |\mathcal{T}_p| \cdot PG(S_m, A) + |\mathcal{T}_g| \cdot PG(S'_m, A) &\leq \\ |\mathcal{T}_g| \cdot (PG(S_m, A) + PG(S'_m, A)) &\leq 0 \end{aligned} \tag{4}$$

Considering Equation (4) and the fact that for any $S \in \mathcal{T} - (\mathcal{T}_p \cup \mathcal{T}_g)$, $PG(S, A) < 0$, we know $profit(A_o) \leq profit(A)$, which leads to a contradiction.

Case 2: $sat(A) > k$. In this case, we have $\mathcal{T}_p = \emptyset$. Since otherwise, there exist either profitable sequences or affirmative sequences in A, which leads to a contradiction. It immediately follows that $profit(A_o) \leq profit(A)$.

In both cases, we have $profit(A_o) \leq profit(A)$, which contradicts our assumption and thus we finish our proof. □

Proof of Theorem 4.18: The correctness follows Lemma 4.16. □

Proof of Lemma 4.19: Consider the '$\Longleftarrow$' direction. The proof is trivial since $\mathcal{T}^* \subseteq \mathcal{T}$.

Consider the '$\Longrightarrow$' direction. Let S be a desirable sequence wrt A in $\mathcal{T}$. We consider the case where S is an affirmative sequence, one type of desirable sequences, for illustration. Thus, we have $PG(S, A) > 0$ and $SG(S, A) \geq 0$. Let p and o be S's head and S's tail, respectively. Consider two cases. Case 1: $S \in \mathcal{T}^*$. We are done. Case 2: $S \notin \mathcal{T}^*$. As a result, there exists a critical sequence S^* in $\mathcal{T}^*$ with its head p and its tail o such that $PG(S^*, A) > PG(S, A) > 0$. It could be verified that $SG(S^*, A) = SG(S, A) \geq 0$ since S^* and S have the same head and tail. As a result, S^* is also affirmative wrt A and thus S^* is desirable wrt A. Since $S^* \in \mathcal{T}^*$, there exists a desirable sequence (i.e., S^*) wrt A in $\mathcal{T}^*$. Therefore, we are done.

The proofs for other cases of S are similar and are omitted. □

Proof of Lemma 4.20: Before we give the proof of Lemma 4.20, we introduce a lemma, which shows the relationship between the profit gain of a sequence and the length of its mapped path.

Lemma .2. *Let A be an assignment and G_A be the directed graph based on A. Consider a sequence S wrt A. Let w_p (w_o) be the profit of S's head (tail) if S has its head (tail) match, and be 0 otherwise. Let tag be 0 if (1) S has its head match p and its tail match o and (2) $(p, o) \in A$, and be -1 otherwise. Then, $PG(S, A) = -\pi.length - w_p + tag \cdot w_o$, where π is $path(S)$ and $\pi.length$ is the length of π.* □

Let A be an assignment and S be a sequence wrt A. Let p (o) be S's head (tail). Let π be S's mapped path in G_A.

Consider the '$\Longrightarrow$' direction. We have the condition that S is a critical sequence wrt A. We prove by contradiction. Assume that the shortest simple path between p and o, denoted by π^*, is not π. Let S^* be π^*'s mapped sequence wrt A. By using Lemma .2, we deduce that $PG(S^*, A) > PG(S, A)$, which, however, contradicts the assumption that S is a critical sequence.

Consider the '$\Longleftarrow$' direction. Thus, we have the condition that π is the shortest simple path in G_A. Again, by using Lemma .2, we can deduce that S has the greatest profit gain among all sequences with the head of p and the tail of o. That is, S is critical wrt A. □

Proof of Lemma 4.21: Let A be the assignment and C, a negative-weight cycle in G_A, be represented in the form of a path $\pi : (p_1, o_1, ..., p_m, o_m)$. Let $S = seq(C)$.

First, we show that $SG(S, A) = 0$. $M(S) = \{(p_{i+1}, o_i) | 1 \leq i \leq m-1\} \cup \{(p_1, o_m)\}$. Thus, $|M(S)| = m$. $N(S) = \{(p_i, o_i) | 1 \leq i \leq m\}$. Thus, $|N(S)| = m$. As a result, $SG(S, A) = |N(S)| - |M(S)| = m - m = 0$.

Second, we show that $PG(S, A) > 0$. We use $C.length$ to denote the length of C, which is equal to $\pi.length + profit(p_1, o_m)$ (Note: The addition of term "$profit(p_1, o_m)$" is due to the directed edge (o_m, p_1)). Since C is a negative-weight cycle, we have $C.length < 0$. According to Lemma .2, $PG(S, A) = -\pi.length - w_p + tag \cdot w_o$, where $w_p = w_o = profit(p_1, o_m)$ and $tag = 0$ in this case. As a result, $PG(S, A) = -C.length > 0$.

In summary, we have $SG(S, A) = 0$ and $PG(S, A) > 0$. Thus, S is an affirmative sequence. □

Proof Sketch of Lemma .1: Let $A_d = A_o - A$ and $A_c = A_o \cap A$. Thus, $A_o = A_c \cup A_d$. For each match (p, o) in A_d, (p, o) is a feasible pair since (p, o) is matchable and p is not matched with o.

Firstly, we *construct* a set $\mathcal{T}$ of sequences such that for any two sequences in $\mathcal{T}$, these two sequences are compatible. We construct a graph $G = (V, E)$ where V is a set of vertices each of which corresponds to $p \in P$ or $o \in O$, and E is a set of edges equal to $A_d \cup A$. Suppose that there are m connected components in G each of which contains at least one edge (p, o) in A_d. Let these components be $C_1, C_2, ..., C_m$. It is easy to verify that each component contains a single path without any branches and along this path, an edge in A_d and an edge in A appear alternatively. Consider a component C_i containing w edges in A_d where one of them is (p, o). Let D be the direction of a path from p to o. Within C_i, we traverse the path from one end to another end in direction D to find the first edge in A_d, says (p_1, o_1), and the last edge in A_d, says (p_w, o_w), along the path. Let the path be $(p_1, o_1, p_2, o_2, ..., p_w, o_w)$. Next, we construct a sequence for this component C_i to be the sequence denoting the path. For each component, we can construct a sequence. All sequences form a set $\mathcal{T}$.

Secondly, since the sequences are generated from non-overlapping paths, for any two sequences, S and S' in A, we have $M(S) \cap M(S') = \emptyset$, and thus S and S' are compatible.

Thirdly, we show that each match in A_c (where $A_c \subseteq A$) is not broken when we perform the adjusting operation on each sequence from $\mathcal{T}$ in A. This can be proved by contradiction.

Fourthly, we show that the resulting assignment A' due to the adjusting operations on each sequence from $\mathcal{T}$ in A is equal to A_o. Each pair in A_d becomes a match in A' (since each pair in A_d appears as a feasible pair in one sequence in $\mathcal{T}$). We have $A_d \subseteq A'$. Besides, each match in A_c is not broken in the resulting assignment A'. Thus, we have $A_c \subseteq A'$. We conclude that $(A_c \cup A_d) \subseteq A'$ and thus $A_o \subseteq A'$. Let $Y(A', A_o)$ be the set of all matches (p, o) in $A' - A_o$. We consider two cases. *Case 1:* $Y(A', A_o) = \emptyset$. We derive that $A' = A_o$ because there are no matches (p, o) in $A' - A_o$. *Case 2:* $Y(A', A_o) \neq \emptyset$. Since $profit(p, o) > 0$, we know the profit of A' is greater than the profit of A_o, which leads a contradition. Thus, Case 2 is not possible. □

Proof of Lemma .2: Let S be $(p_1, o_1, p_2, o_2, ..., p_m, o_m)$. Then, $N(S) = \{(p_i, o_i) | i = 1, 2, ..., m\}$. Let $M'(S)$ be $M(S)$ excluding S's head match (if any) and S's tail match (if any)), i.e., $M'(S) =$

$\{(p_{i+1}, o_i)|i = 1, 2, ..., m-1\}$. We have

$$\begin{aligned} & profit(N(S)) - profit(M'(S)) \\ = \ & \sum_{i=1}^{m} profit(p_i, o_i) - \sum_{i=1}^{m-1} profit(p_{i+1}, o_i). \end{aligned} \tag{5}$$

Consider $\pi.length$. Since the weight of edge (p_i, o_i) is $-profit(p_i, o_i)$ for $1 \leq i \leq m$, and the weight of edge (o_i, p_{i+1}) is $profit(p_{i+1}, o_i)$ for $1 \leq i \leq m-1$, We have

$$\pi.length = \sum_{i=1}^{m} - profit(p_i, o_i) + \sum_{i=1}^{m-1} profit(p_{i+1}, o_i). \tag{6}$$

According to Equation (5) and Equation (6),

$$profit(N(S)) - profit(M'(S)) = -\pi.length \tag{7}$$

In the following, we consider 4 cases (4 combinations of the existence of S's head match and the existence of S's tail match).

Case 1: S has its head match (p_1, o_0) and its tail match (p_{m+1}, o_m). We further consider two subcases. Case 1(a): (p_1, o_0) and (p_{m+1}, o_m) are identical (i.e., $p_1 = p_{m+1}$ and $o_0 = o_m$). That is, (p_1, o_m) is a match in A. Thus, $tag = 0$ and $M(S) = M'(S) \cup \{(p_1, o_m)\}$. Besides, $w_p = w_o = profit(p_1, o_m)$. Then, $PG(S, A) = profit(N(S)) - profit(M'(S)) - profit(p_1, o_m) = \pi.length - w_p + tag \cdot w_o$.

Case 1(b): (p_1, o_0) and (p_{m+1}, o_m) are not identical. Thus, $tag = -1$ and $M(S) = M'(S) \cup \{(p_1, o_0)\} \cup \{(p_{m+1}, o_m)\}$. Besides, $w_p = profit(p_1, o_0)$ and $w_o = profit(p_{m+1}, o_m)$. Then, $PG(S, A) = profit(N(S)) - profit(M'(S)) - profit(p_1, o_0) - profit(p_{m+1}, o_m) = \pi.length - w_p + tag \cdot w_o$.

The correctness of the remaining three cases could be verified similar to Case 1 and thus they are omitted here. □

Proof of Theorem 5.1: The proof is similar to that of Theorem 4.18. □

Updated Proof of Lemma 4.15 described in Section 6: The proof is the same as the original proof of Lemma 4.15. However, we need to consider the additional case that S (a profitable sequence) can be a special feasible sequence. Note that S' (a gratifying sequence) cannot be a special sequence. Let (p, o) be the head match in S and $(p'_{t'+1}, o'_{t'})$ be the match in S' such that $(p'_{t'+1}, o'_{t'}) = (p, o)$. Consider two sequences, Q and Q', where $Q = (p'_1, o'_1, ..., p'_{t'}, o'_{t'})$ and $Q' = (p'_{t'+1}, o'_{t'+1}, ..., p'_n, o'_n)$. See Figure 18(c) for illustration. Recall that S' is gratifying. Thus, both p'_1 and o'_n are not matched in A. That is, Q has no head match and Q' has no tail match. Considering that $(p'_{t'+1}, o'_{t'})$ is a match in A, we know Q has a tail match and Q' has a head match. As a result, $SG(Q, A) = 0$ and $SG(Q, A) = 0$. Besides, it is not hard to verify that $PG(Q, A) + PG(Q', A)$ is exactly equal to $PG(S, A) + PG(S', A)$. According to the assumption, $PG(S, A) + PG(S', A) > 0$. Thus, we have $PG(Q, A) + PG(Q', A) > 0$. It follows that at one least one of $PG(Q, A)$ and $PG(Q', A')$ is greater than 0. Without loss of generality, suppose $PG(Q, A) > 0$. As a result, Q corresponds to an affirmative sequence, which contradicts the assumption.

Updated Proof of Lemma 4.16 described in Section 6: The proof of Lemma 4.16 can be kept intact. However, the proof of Lemma .1 used in the proof of Lemma 4.16 should be updated as follows. The proof is the same as the original proof of Lemma .1 except Case 2 ($Y(A', A_o) \neq \emptyset$). We change the proof as follows. In this case, we perform the following post-processing. For each match $(p, o) \in Y(A', A_o)$ with $profit(p, o) \leq 0$, we insert a special sequence containing its head match equal to (p, o) into $\mathcal{T}$. Note that (p, o) is to be broken in the adjusting operation on this special sequence. Besides, (p, o) must not be in A_o (since otherwise, A_o is not optimal). Suppose that we execute the adjusting operations on these special sequences on A' and obtain a resulting assignment A''. Since each broken match $(p, o) \in A'$ is not in A_o and $A_o \subseteq A'$, we deduce that $A_o \subseteq A''$. In addition, there does not exist any matches (p, o) in $A'' - A_o$ where $profit(p, o) \leq 0$. Consider $Y(A'', A_o)$. We show $Y(A'', A_o) = \emptyset$ by contradiction. Assume that $Y(A'', A_o) \neq \emptyset$. Let (p, o) be a

match in $Y(A'', A_o)$. We have $profit(p, o) > 0$ (since all matches (p', o') with $profit(p', o') \leq 0$ have been broken during the above adjusting operations). As a result, we have $profit(A'') > profit(A_o)$, which leads to a contradiction. Thus, A'' is A_o and we finish our proof. □