A Bi-Objective Approach for Product Recommendations

Idir Benouaret, Sihem Amer-Yahia, Christiane Kamdem-Kengne, Jalil Chagraoui

A Bi-Objective Approach for Product Recommendations

1st Idir Benouaret, 2nd Sihem Amer-Yahia
Univ. Grenoble Alpes
Grenoble, France
firstname.lastname@univ-grenoble-alpes.fr

3rd Christiane Kamdem-Kengne, 4th Jalil Chagraoui
TOTAL
Paris, France
firstname.lastname@total.com

***Abstract*—We propose a bi-objective formulation for product recommendations. Our formulation goes beyond traditional recommendations by capturing two conflicting objectives: *utility* that serves customers' interests, and profit *margin*, a business-oriented goal. To satisfy the needs of our business partners, we formulate a new problem, namely generating a result containing all sets of k products such that there does not exist any other set of k products that dominates the returned sets, i.e., whose *cumulative* values for each objective is higher than a set of k products in the result. We study properties of k-Pareto sets that enable us to reduce the number of candidates, as well as the number of dominance tests between candidate sets. We develop a dynamic programming algorithm that leverages those properties to prune the space of solutions. We generalize traditional measures of recommendation accuracy to be applicable to sets of k products. Our experiments on a large set of real customer transactions validate the need for a bi-objective optimization to reconcile customer and business interests, and the scalability of our solution.**

***Index Terms*—Recommendation , Customer Margin, Bi-Objective Optimization**

I. Introduction

Product recommendation is traditionally seen as finding products a user is most likely to purchase. This goal relies on the assumption that prediction accuracy approximates well recommendation effectiveness. In practice however, several other factors are considered when designing recommendations. In particular, the business impact of recommendations in terms of revenue loss and gain, plays a major role in determining which products to recommend to which customers. In this paper, we investigate a new way of formulating recommendations as an optimization problem that tackles two objectives at once: one customer-centric and one business-oriented. Product recommendation is formulated as the optimization of *utility* and *margin*, two conflicting goals.

Product managers play an important role in the Retail industry. They are the ones who understand customer interests for a product, a collection of products, or a product categories. They spend time and effort experimenting with various *customer utility* formulas and studying *the profit margin* of individual products, defined as a percentage of the selling price. Those values are used to enforce business rules in product recommendation. A common rule is to recommend to a customer products that are of high utility to that customer, and generate a high profit margin. Existing recommendation approaches in the literature are mostly concerned with maximizing utility, a measure that is considered a good proxy for predicting the products a customer is most likely going to purchase, and hence increase recommendation accuracy. Augmenting existing recommendation approaches with the ability to also account for profit margins is a non-obvious question due to *conflicts* between the two objectives. Products with a high profit margin are not necessarily the ones customers want the most, and vice versa. It is therefore important for product managers to *automate the process of finding candidate recommendations to customers among all possible combinations of products that collectively have a high utility and a high profit margin*.

Our first contribution is a formalization. With two conflicting objectives at hand, customer utility and product profit margin, we face a bi-objective optimization (maximization in our case) problem. This problem is well-studied in research in the case where the goal is to return *one set of items* such that there does not exist any other item whose value for each objective is higher. In other terms, *all dominating items are returned*. In this paper, we are facing a different question: product managers would like to generate all sets of k items such that there does not exist any other set of k items whose *cumulative* values for each objective is higher than a set in the result. In other terms, we are looking to compute *all sets of k products that are not dominated by any other set of size k*, which we refer to as *k-Pareto sets*.

Our second contribution is computational. There are two ways to address a bi-objective optimization problem: *scalarization* and *pareto-based* approaches. The first solves a single objective function formed as a linear combination of utility and profit margin (our two objectives). While this is a good baseline, in practice, it may miss several solutions. The second formulation suffers from a high computational cost induced by the fact that generating all candidate sets is practically infeasible. Hence our solution generates dynamically the candidate sets in a progressive way and exploits pruning opportunities. More precisely, our algorithm relies on dynamic programming which takes one product at a time as input, and

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

generates candidate sets containing that product, and repeats the process for other products. Therefore, the only question that remains to be solved is how to avoid generating all candidate sets. We study properties of k-Pareto sets that enable us to reduce the number of candidates, as well as the number of dominance tests between candidate sets. Our algorithm uses those properties to prune all non-Pareto products that are dominated by at least k products and all candidate sets that contain any product i but does not contain all products dominated by i.

Our third contribution is empirical. Our dataset is provided by our project partners at *TOTAL*[1] and represents the purchase data of customers owning a loyalty. It contains about 30 million unique receipts, generated by $425,406$ customers at $3,463$ gas stations in France, over 28 months (from January 2017 to April 2019). We first justify the need for a bi-objective optimization approach by comparing the accuracy of the recommendations it generates to the ones generated with scalarization. We study the accuracy of our recommendations in terms of precision and profit margin, and show that our algorithm generates recommendations offering different trade-offs between those two objectives. We discuss how that can be beneficial to our business partners in choosing the best recommendation alternative. In particular, we also study the effect of product prices on the overall accuracy and interpret the impact of price on precision and profit margin. Finally, we validate the necessity of our pruning strategies to ensure good response times.

We briefly summarize our contributions.

- To match real needs in retail, we propose a new bi-objective formulation of product recommendations that maximizes both customer utility and product margin. Our formulation returns all sets of k products, none of which is dominated by other sets of k products.
- We explore two solutions for our problem: scalarization and pareto-based. We study properties of the space of solutions and devise new pruning strategies applicable to our new formulation.
- We conduct experiments with a real dataset that justify the need for a bi-objective optimization approach for recommendations, and validate our pruning strategies.

Section II introduces our data model and problem formulation. Section III discusses strategies for solving our problem. Section IV presents our algorithm. Experiments are reported in Section V. The related work is provided in Section VI and we conclude in Section VII.

II. Model and problem

Let $\mathcal{U} = \{u_1, u_2, ..., u_m\}$ be the set of customers and $\mathcal{I} = \{i_1, i_2, ..., i_n\}$ be the set of products. For a given customer u, $H_u \subseteq \mathcal{I}$ denotes the purchase history of u, i.e. the set of products purchased by u.

[1]TOTAL S.A. is a French multinational integrated oil and gas company founded in 1924 and one of the seven "supermajor" oil companies in the world

Our dataset represents customers owning a loyalty card and who purchased products at different gas stations that are geographically distributed in France. The complete dataset contains over 30 million unique receipts generated at $3,463$ gas stations. The set of customers $\mathcal{C}$ contains over $425,406$ million unique records. Each customer has demographic attributes such as *age, location, gender* and *membership status*. The set of products $\mathcal{I}$ contains over $16,891$ entries that belong to one of 54 categories including *gas, lubricants, car wash, hot drinks,* and *sweets*.

We note $\boldsymbol{X}$ the purchase matrix of the m customers in $\mathcal{U}$ over the n products in $\mathcal{I}$, where each column corresponds to a product, and each row corresponds to a customer. An entry $x_{u,i}$ in the purchase matrix contains a boolean value (0 or 1), where $x_{u,i} = 1$ means that product i was purchased by customer u (0 means the opposite).

A. Optimization objectives

Following our discussion with product managers and analysts at our business partner, we identified the need to account for the following two criteria when designing product recommendations:

- Suggesting products with a high value (i.e., products with high interest for the customer)
- Improvement of margin gain (Products with a high generated revenue)

Our goal is to recommend products with the highest utility to a customer and that generate the highest margin. Intuitively a good product recommendation has to satisfy two criteria: that the product has a high chance to be of interest to the target customer, thus increasing the probability that the customer effectively purchases the product; that the product has to generate a high margin. For example, our analysts stated that *coffee* products are among the products that generate a high margin, but it would not be useful to recommend those products to a customer who is not a *coffee drinker*, which justifies the need to account for both objectives.

1) Utility: Following [1] and [2], we use association rules to estimate the utility of a product for a customer. To target a customer u with relevant recommendations we need to measure the utility of a product i with respect to a customer u that has not already bought i, we note this utility $Utility(u, i)$. To do that we leverage the purchase history of our customers. We rely on association rules mining using *bigram* rules to compute an association matrix $\boldsymbol{A}$. The matrix $\boldsymbol{A}$ is computed from the purchase matrix $\boldsymbol{X}$, where each entry $a_{j,l}$ corresponds to the confidence of the association rule $j \rightarrow l$.

$$a_{j,l} = confidence(j \rightarrow l) = \frac{X_{\bullet j}^T X_{\bullet l}}{||X_{\bullet j}||_1}, ||X_{\bullet j}||_1 = \sum_{i=1}^{n} |x_i| \quad (1)$$

Given a product i and a customer u, intuitively the estimated utility $Utility(u, i)$ is high if the association rules between products $j \in H_u$ in the purchase history of customer u and the target product i, i.e., $j \rightarrow i$, have high confidence values. More formally: $Utility(u, i) = \sum_j^{H_u} confidence(j \rightarrow i)$

2) Margin: In addition to recommend products with a high utility to a given customer u, we need also to account for the profit margin that is generated by these products. For a given product i, its margin $Margin(i)$ is calculated as follows: $Margin(i) = \beta \times sp(i)$

where $sp(i)$ is the selling price of product i. β is a value between 0 and 1 that depends on the product category and the station where it is purchased.

3) Station and customer-specific margin: In practice, β depends on the attractiveness of a product category at a station. The gas stations visited by our customers are classified into one of three types: COCO (Company owned, Company operated), CODO (Company owned, Dealer operated) and DODO (Dealer owned, Dealer operated). Station types differ mainly in the customers they attract and the overall revenue they generate. Additionally, the margin of a product also depends on the behavior of the customer. Each customer has different purchasing habits and visits different types of stations. To reflect that, we revisit the profit margin of a product i and define it as a weighted sum, where each term is determined by a station specific value of β and the frequency of visits of a customer u in each station type:

$$Margin(u,i) = (F_u^{\texttt{COCO}} \times \beta_{\texttt{COCO}} + F_u^{\texttt{CODO}} \times \beta_{\texttt{CODO}} + F_u^{\texttt{DODO}} \times \beta_{\texttt{DODO}}) \times sp(i) \quad (2)$$

where $F_u^{\texttt{COCO}}, F_u^{\texttt{CODO}}, F_u^{\texttt{DODO}}$ correspond to the frequency the customer visits to station types COCO, CODO and DODO respectively.

As an example, for a coffee product for which we assume that its selling price is 2 euros, we have $\beta_{\texttt{COCO}} = 0.8$, $\beta_{\texttt{CODO}} = 0.3$ and $\beta_{\texttt{DODO}} = 0.3$. Given two customers u and v with $F_u^{\texttt{COCO}} = 0.6, F_u^{\texttt{CODO}} = 0.4, F_u^{\texttt{DODO}} = 0$ and $F_v^{\texttt{COCO}} = 0, F_v^{\texttt{CODO}} = 0.5, F_v^{\texttt{DODO}} = 0.5$, respectively, we will have: $Margin(u,i) = 1.2$ and $Margin(v,i) = 0.6$.

B. Product recommendation problem

Product managers at our business partner expressed the need for generating alternative recommendations for their customers. As a result, we set out to solve the following problem: find all sets of k products, each of which maximizes both customer utility and generated margin. Therefore, we formalize our recommendation problem as a bi-objective optimization problem.

Given a target customer u with her purchase history H_u, the set of all available products $\mathcal{I}$, an integer constant k, our problem is to select *all sets* $S \subseteq \mathcal{I}$, such that each set S satisfies:

- $Utility(u,S)$ is maximized;
- $Margin(u,S)$ is maximized;
- $|S| = k$
- $S \cap H_u = \emptyset$

where

$$\begin{cases} Utility(u,S) = \sum_{i \in S} Utility(u,i) \\ Margin(u,S) = \sum_{i \in S} Margin(u,i) \end{cases}$$

TABLE I: A dataset with 6 products and corresponding values for utility and margin

Product	Utility	Margin
i_1	0.4	0.4
i_2	0.1	0.5
i_3	0.5	0.2
i_4	0.3	0.4
i_5	0.3	0.3
i_6	0.3	0.2

The output is k-sets of product recommendations, where each set S satisfies the above conditions.

III. Approaches for solving a bi-objective problem

A. Transforming our problem into a single objective optimization

The main challenge in designing an algorithm for our problem is its bi-objective nature. If optimizing one objective leads to an optimized value for the second objective, the problem becomes single-objective. Another approach is scalarization which combines the two objectives into a single one with a weighted linear combination. Another popular method is ϵ-constraints [3] (thresholding), where one objective is optimized and the other is constrained. For example, in our setting, we could maximize utility subject to the constraint that margin is above a threshold ϵ. Another popular method we can use is Multi-Level Optimization [4]. The principle of this approach is to consider a single objective at a time. For example, in our setting, we could maximize utility and then among the returned set of recommendations, choose the ones with the maximum margin.

All the methods we describe work well in practice in the case where the objectives are not conflicting. When all objectives are independent and conflicting, none of the above methods are possible. As we will see in our experiments (Section V-E1), our two objectives are conflicting which justifies the need for a bi-objective optimization.

B. Addressing the two objectives at once

When dealing with more than one objective to optimize, there may be many incomparable k sets of products. As an example, consider the dataset in Table I with 6 products. Each product is associated with utility and margin scores for a given customer u. If we are about to choose between two items i_1 and i_2 with $Utility(u,i_1) = 0.4$, $Margin(u,i_1) = 0.4$ and $Utility(u,i_2) = 0.1$, $Margin(u,i_2) = 0.5$, we notice that each product has its own characteristics: the first one has a higher margin and the second one has a higher utility, thus i_1 and i_2 are incomparable. Another product i_5 with $Utility(u,i_5) = 0.3$, $Margin(u,i_5) = 0.3$ has no advantage compared to i_1, in this case we say that i_5 is *dominated* by i_1 and that i_1 *dominates* i_5. In our example stated in Table I, as we prefer higher values for both utility and margin, products i_1, i_2 and i_3 are the best choices because they are not dominated by any other product, while products i_4, i_5 and i_6 are all inferior choices because they are dominated by i_1. We refer to i_1, i_2 and i_3 as *Pareto* products.

TABLE II: 3-*Pareto* sets

Set of Products	Sum of Utilities	Sum of Margins
$\{i_1, i_2, i_3\}$	1	1.1
$\{i_1, i_2, i_4\}$	0.8	1.3
$\{i_1, i_3, i_4\}$	1.2	1
$\{i_1, i_4, i_5\}$	1	1.1

Algorithms that retrieve the set of *Pareto* solutions have received a lot of attention in the research community [5], [6]. However such methods are only suitable for applications where the user has to choose only a single option among all available ones. In our case, a product manager looking to recommend products may receive a set of k non-dominated products, and needs to choose one such set.

In our case, we intend to target a customer with multiple sets of k products. Thus, conventional methods for *Pareto* set computation are inadequate to address our problem. The reason is that our setting requires to analyze not just individual products of a dataset but also combinations of k products. Let us go back to our example in Table I, it is not clear how to answer the following question: *Our product manager wants to target a customer with 3 products. Which set of 3 products is the best choice?* to answer this common question, there is a need to analyze sets of 3 products. Thus, we need to *revisit the notion of dominance relation between sets of products by comparing their aggregated values on individual optimization objectives.* In our scenario we use the sum Σ as the aggregation function over sets.

Consider again the dataset in Table I. Our concern was to retrieve sets of 3 products whose sums of utility and sums of margins are not worse than any other set of 3 products. After examining all possible combinations (20 different sets), we obtain in Table II 4 sets of 3 products each that are not dominated by any other set of 3 products.

We refer to these sets as 3-*Pareto* sets of the dataset, where 3 indicates the number of products in each set. More generally, given a constant factor k specifying the number of products in each set, our problem is to find all k-*Pareto* sets. i.e., those solutions that are not dominated by any other solutions.

The main challenge in developing an algorithm for retrieving all k-*Pareto* sets is to overcome its computational complexity. For instance, for n products, there can be $\binom{n}{k}$ different candidate sets. A naive solution is to retrieve all $\binom{n}{k}$ candidates sets, compute the aggregated values for each set, and then using any traditional algorithm to identify all the non dominated sets. It is obvious that such an approach cannot work in practice. Therefore, to address this challenge, we need to develop an algorithm that enables pruning of the search space so that we do not explore all possible combinations of products.

We study the properties of k-Pareto sets that enable us to reduce the number of generated candidates, as well as the number of dominance tests between candidate sets. As a preliminary step, we begin by presenting formal definitions and the notations that we use. Table III summarizes our notations.

Definition 3.1: (**Dominance**)
Product i dominates product j, if i has a greater or equivalent value for at least one objective and has a strictly greater value for the other objective.

From now on, we will write $i \succ j$ to mean that product i dominates product j. We will also write $i \not\succ j$ to mean that product i does not dominate product j.

Definition 3.2: (**Pareto product**)
A product i is a *Pareto* product, if there exists no other product j that dominates it, i.e., $j \succ i$.

Given the set of products $\mathcal{I}$, we define the set of *Pareto* products of $\mathcal{I}$ as follows:

$$\mathcal{P}(\mathcal{I}) = \{i \in \mathcal{I} | \forall j \in \mathcal{I}, j \not\succ i\}$$

To compare between two sets of products, we extend the notion of the dominance relation between two sets of the same size, i.e., containing the same number of products. We define the dominance relation between two sets using the aggregated sum of their corresponding values for our both objectives (utility and margin).

Now let us formally define set dominance. We refer to a subset of $\mathcal{I}$ containing k products with S_k, and write $\mathcal{C}_k(\mathcal{I})$ the set of all sets of k products in $\mathcal{I}$:

$$\mathcal{C}_k(\mathcal{I}) = \{S_k | S_k \subseteq \mathcal{I}, |S_k| = k\}$$

We define the dominance relation $S_k \succ S'_k$ between two k-sets S_k and S'_k as follows:

Definition 3.3: (**Set dominance**)
Let us suppose $S_k = \{i_1, ..., i_k\}$ and $S'_k = \{j_1, ..., j_k\}$. We say that S_k dominates S'_k and write $S_k \succ S'_k$ if S_k has a greater aggregated value for one of our objectives and has a strictly greater aggregated value for the other objective. More formally,
$\sum_{i \in S_k} Utility(u, i) \geq \sum_{i' \in S'_k} Utility(u, i')$ and
$\sum_{i \in S_k} Margin(u, i) > \sum_{i' \in S'_k} Margin(u, i')$.
Or $\sum_{i \in S_k} Utility(u, i) > \sum_{i' \in S'_k} Utility(u, i')$ and
$\sum_{i \in S_k} Margin(u, i) \geq \sum_{i' \in S'_k} Margin(u, i')$
We also write $S_k \not\succ S'_k$ to mean that S_k does not dominate S'_k

Definition 3.4: (k-**Pareto set**)
A set of products S_k is a k-Pareto set, if there exists no other set of products S'_k such that $S'_k \succ S_k$

Then given the set of products $\mathcal{I}$, we define the set of all k-Pareto sets as follows:

$$\mathcal{P}_k(\mathcal{I}) = \{S_k \in \mathcal{C}_k(\mathcal{I}) | \forall S'_k \in \mathcal{C}_k(\mathcal{I}), S'_k \not\succ S_k\}$$

C. Properties of k-Pareto sets

Since we cannot compute all k-Pareto sets in polynomial space and time, we propose to examine specific properties of k-Pareto sets that can help us to consider fewer input products and also consider fewer intermediate candidate sets.

TABLE III: Summary of our notations

Notation	Meaning
u	customer
i, j	products
$\mathcal{I}$	set of all products
S_k	set of k products
C_k	set of all sets of k products
P_k	k-Pareto set
$\mathcal{P}(\mathcal{I})$	Pareto set of $\mathcal{I}$
$\mathcal{P}_k(\mathcal{I})$	set of all k-Pareto sets of $\mathcal{I}$

Since a k-Pareto set represents a set of k "good" products as a whole, it is likely to be a combination of k Pareto products. A naive algorithm that computes the set of all Pareto products $\mathcal{P}(\mathcal{I})$ and check for combinations of k products of $\mathcal{P}(\mathcal{I})$ to retrieve the set of k-Pareto sets does not work. Indeed, Observation III-C.1 shows that a set of k Pareto products does not necessarily form a k-Pareto set.

Observation III-C.1: Suppose a set of k Pareto products $S_k \subset \mathcal{P}(\mathcal{I})$. Then it is possible to have $S_k \notin \mathcal{P}_k(\mathcal{I})$

Proof: Consider a customer u and a set of 4 products $\mathcal{I} = \{i_1, i_2, i_3, i_4\}$. Let us note for a product i: $i = (x, y)$ to mean that $Utility(u, i) = x$ and $Margin(u, i) = y$. Then, let's consider $i_1 = (0.1, 0.5)$, $i_2 = (0.2, 0.3)$, $i_3 = (0.3, 0.2)$ and $i_4 = (0.5, 0.1)$. We notice from this example that all products are not dominated, but we see that $\{i_1, i_4\}$ dominates $\{i_2, i_3\}$.

We now prove the contrapositive of Observation III-C.1.

Observation III-C.2: Suppose a k-Pareto set $S_k \in \mathcal{P}_k(\mathcal{I})$. Then it is possible to have $S_k \notin \mathcal{P}(\mathcal{I})$

Proof: Consider again the example in Tables I and II. i_4 and i_5 are both non Pareto products, but $\{i_1, i_4, i_5\}$ is a 3-Pareto set.

Hence, according to these two observations, to compute k-Pareto sets we need not only to consider Pareto products but also non-Pareto products. But do these observations mean that we have to treat all non-Pareto products in the same way? Can we safely prune a portion of candidate sets?

D. Pruning candidate sets

We now present our strategy for pruning the search space that enables us to consider fewer candidates sets. Lemma III-D.1 demonstrates that a k-Pareto set may contain a non-Pareto product i, but if so, then it has to contain all the products that are dominating i.

Lemma III-D.1:

Suppose a k-Pareto set $S_k \in \mathcal{P}_k(\mathcal{I})$, and suppose a product $i \in S_k$. If there exists a product $j \in \mathcal{I}$ such that $j \succ i$, then we must have $j \in S_k$.

Proof: We prove lemma III-D.1 by contradiction. Let us suppose $S_k = \{i_1, i_2, ..., i_k\}$ and that $j \notin S_k$. Without loss of generality, let's assume that $i = i_k$. So, if there exists a product $j \in \mathcal{I}$ such that $j \succ i$, it's obvious that with replacing i by j in S_k we will end up with a set dominating the original one. i.e., $(S_k - \{i\}) \cup \{j\} \succ S_k$, this contradicts the assumption that S_k is a k-Pareto set.

Thus, Lemma III-D.1 is at the basis of the pruning strategy that we use to generate less candidate sets. Given a product i, we safely prune all candidate sets containing i and not containing products dominating i.

E. Pruning input products

We now present a strategy that enables us pruning the input products even before starting to search for the set of k-Pareto sets. Indeed, Lemma III-E.1 demonstrates that we can safely prune all non-Pareto products in the original input $\mathcal{I}$, that are dominated by at least k products.

Lemma III-E.1:

Suppose a product $i \in \mathcal{I}$ that is dominated by k other products. This implies that there exists no k-Pareto set $S_k \in \mathcal{P}_k(\mathcal{I})$ such that $i \in S_k$

Proof: We also prove Lemma III-E.1 by contradiction. Let us assume that $S_k \in \mathcal{P}_k(\mathcal{I})$ is a k-Pareto set, and that S_k contains a product i that is dominated by k other products. Then, using Lemma III-D.1, S_k has also to contain all those k products that are dominating i. This obviously violates the cardinality size k for S_k, since it has to contain at the same time the product i and also the k products dominating i, which contradicts the assumption that $|S_k| = k$.

An interesting result from Lemma III-E.1 is that we can safely discard all products in $\mathcal{I}$ that are dominated by at least k other products, which can significantly reduce the number of products to consider for retrieving the set of all k-Pareto sets in $\mathcal{I}$.

IV. Algorithm

The biggest challenge in designing an algorithm to solve our problem is computational. We need to deal with the exponential space complexity. Generating all candidate sets is practically infeasible. For a set of products with size $n = 1000$, if we want to retrieve a recommendation set of size $k = 4$, there might be $\binom{1000}{4} = 41417124750$ candidate sets of the specified size. Lemmas III-D.1 and III-E.1 in the previous section, form the basis of the algorithm we develop to retrieve all k-Pareto sets. Given a customer u and the set of products $\mathcal{I}$, we exploit Lemma III-E.1 to discard all the products that are dominated by k or more products. Then we consider only those candidate sets that satisfy the condition stated in Lemma III-D.1.

Our solution does not generate all candidate sets at once, rather it generates dynamically the candidate sets in a progressive way and exploits pruning opportunities. More precisely, our algorithm relies on dynamic programming which takes a product i and generates candidate sets containing i and repeats the process for other products. Our algorithm is based on the following intuition. Let us assume that we have a subset of products $\mathcal{L} \subset \mathcal{I}$ such that $|\mathcal{L}| \geq k$ and containing a product i. Thus, to compute $\mathcal{P}_k(\mathcal{L})$ we can compute it from $\mathcal{P}_k(\mathcal{L} - \{i\})$ and $\mathcal{P}_{k-1}(\mathcal{L} - \{i\})$. Let us consider a k-Pareto set S_k in $\mathcal{P}_k(\mathcal{L})$, Then we may encounter two possibilities, we may either have $i \in S_k$ or $i \notin S_k$.

Proposition 4.1: Suppose that $i \notin S_k$. This implies that S_k must be in $\mathcal{P}_k(\mathcal{L} - \{i\})$

Proof: We prove this proposition by contradiction. Let us assume that $S_k \notin \mathcal{P}_k(\mathcal{L} - \{i\})$. Then there must exist a k-Pareto set $S'_k \in \mathcal{P}_k(\mathcal{L} - \{i\})$ that dominates S_k. Thus we face two cases: (1) $S'_k \in \mathcal{P}_k(\mathcal{L})$, since $S'_k \succ S_k$ it contradicts that S_k is a k-Pareto set in $\mathcal{P}_k(\mathcal{L})$. (2) $S'_k \notin \mathcal{P}_k(\mathcal{L})$, then there exists another k-Pareto set $S''_k \in \mathcal{P}_k(\mathcal{L})$ that dominates S'_k, thus, we have $S''_k \succ S'_k \succ S_k$ which contradicts that S_k is a k-Pareto set in $\mathcal{P}_k(\mathcal{L})$. Hence the proof.

Proposition 4.2: Suppose now that $i \in S_k$. This implies that the set $(S_k - \{i\})$ must be in $\mathcal{P}_{k-1}(\mathcal{L} - \{i\})$

Proof: We also prove this proposition by contradiction. Let us assume that $(S_k - \{i\}) \notin \mathcal{P}_{k-1}(\mathcal{L} - \{i\})$. Then there exists a $(k-1)$-Pareto set S_{k-1} that dominates $(S_k - \{i\})$, i.e., $S_{k-1} \succ S_k - \{i\}$. Thus, the union of product i on both sides gives $S_{k-1} \cup \{i\} \succ S_k$, which contradicts the fact that S_k is a k-Pareto set.

Algorithm 1: INPUT PRUNING

Input: a user u, a set of products $\mathcal{I}$, a maximum size of recommended sets k

Output: a set of products $\mathcal{I}' \subset \mathcal{I}$ containing only products that are not dominated by at least k products, a Dominance Hashmap $\mathcal{D}$ materializing for each product the set of products dominating it

$\mathcal{I}' \leftarrow []$
foreach *product* $i \in \mathcal{I}$ **do**
  Compute $Utility(u, i)$
  Compute $Margin(u, i)$
$\mathcal{L} \leftarrow Desc\_Sort(\mathcal{I}, Utility(u, i) + Margin(u, i))$
for *x = 1 to size(*$\mathcal{L}$*)* **do**
  $i \leftarrow \mathcal{L}[x]$
  $\mathcal{D}[i] \leftarrow \emptyset$
  $\mathcal{I}'.append(i)$
  for *y = 1 to x-1* **do**
    if $size(\mathcal{D}[i]) \geq k$ **then**
      remove i from $\mathcal{I}'$ // prune i
      $break$
    $j \leftarrow \mathcal{L}[y]$
    if *j dominates i* **then**
      $\mathcal{D}[i].add(j)$
return $\mathcal{I}'$, $\mathcal{D}$

Therefore, an important result of these two propositions is that given a set of products $\mathcal{L}$ and a single product $i \notin \mathcal{L}$ that we are considering, in order to compute $\mathcal{P}_k(\mathcal{L} + \{i\})$, we only have to consider those sets which are k-Pareto sets in $\mathcal{P}_k(\mathcal{L})$ (according to Proposition 4.1) and new candidate sets obtained from $\mathcal{P}_{k-1}(\mathcal{L})$ by growing the set with product i (according to Proposition 4.2). In the latter case, we exploit Lemma III-D.1 to generate only candidate sets from $\mathcal{P}_{k-1}(\mathcal{L})$ that are containing all the products dominating i, other sets can be safely pruned. In this way, we generate a minimal number of intermediate candidate sets.

Algorithm 2: *Pareto* $(k, \mathcal{I}_n)$: Dynamic programming algorithm for retrieving the set of k-Pareto sets

Input: a user u, a set of products $\mathcal{I}_n = \{i_1, ..., i_n\}$, a maximum size of recommended sets k, a dominance hashmap $\mathcal{D}$

Output: $\mathcal{P}_k^n$: the set of all k-pareto sets

if $\mathcal{P}_k^n$ *is already computed* **then**
  return $\mathcal{P}_k^n$
if $k < n$ **then**
  $\mathcal{P}_k^{n-1} \leftarrow$ *PARETO* $(k, \mathcal{I}_{n-1})$
  $Cand \leftarrow \mathcal{P}_k^{n-1}$
else
  $Cand \leftarrow \emptyset$
if $k == 1$ **then**
  $Cand' \leftarrow \{\{i_n\}\}$
else
  $Cand' \leftarrow \emptyset$
  $\mathcal{P}_{k-1}^{n-1} \leftarrow$ *PARETO* $(k-1, \mathcal{I}_{n-1})$
  foreach *Set* $S \in \mathcal{P}_{k-1}^{n-1}$ **do**
    if $\mathcal{D}[i_n] \subseteq S$ **then**
      $S' \leftarrow S \cup \{i_n\}$
      $Cand' \leftarrow Cand' \cup \{S'\}$
$Cand_k^n = Cand \cup Cand'$
foreach *set* $S \in Cand_k^n$ **do**
  if S *is not dominated by any other set in* $Cand_k^n$ **then**
    $\mathcal{P}_k^n$.add(S)
return $\mathcal{P}_k^n$

We design a strategy in two stages. Algorithm 1 is called to prune the set of products before searching for k-Pareto sets. It is also used to materialize a hashmap associating a product with the set of products dominating it. The algorithm first computes for each product the corresponding utility and margin, then we sort the product according to the sum of utility and margin for each product. This is an optimization that avoids unnecessary dominance checks. When products are sorted in this way, we make sure that a product can only be dominated by products that appear before in the sorted list. The algorithm returns the list of products that are not dominated by k or more products, in addition it also returns a dominance Hashmap which associates to each product the set of products dominating it. The list of input products and the associated Hashmap are then fed to Algorithm 2.

Algorithm 2 computes the set of all k-Pareto products. It is a dynamic programing approach that takes one product at a time and generates the set of candidates following Propositions 4.1 and 4.2. $\mathcal{P}_k^n$ denotes the set of k-Pareto sets for the first n products. Thus, to compute $\mathcal{P}_k^n$ we recursively compute

TABLE IV: Summary of the characteristics of our dataset

	Our dataset
Domain	retail, gas and oil industry
Time span	January 2017 –>April 2019
Number of customers	$425,406$
Number of products	$16,891$
Number of purchases	$2,822,310$
Sparsity	99.96 %

$Cand = \mathcal{P}_k^{n-1}$ which is the set of k-Pareto sets for the first $n-1$ products, and $Cand' = \mathcal{P}_{k-1}^{n-1}$ the $k-1$-Pareto sets for the first $n-1$ products, we grow these sets with i_n to generate new candidates of size k by exploiting Lemma III-D.1 (lines 13-16). The algorithm returns the set of all k-Pareto sets for the target customer u.

V. Experiments

We conduct several experiments whose purpose is to examine the behavior of our algorithm and the solutions it finds. Our implementation is in Python 3.7.0 and is running on a 2.7 GHz Intel Core i7 machine with a 16 GB main memory, running OS X 10.13.6.

A. Summary of results

Our results can be summarized as follows. We first justify the need for a bi-objective optimization approach by comparing the accuracy of the recommendations it generates to the ones generated with scalarization. We then study the accuracy of our recommendations, namely their precision and profit margin, and show that our algorithm can generate recommendation alternatives offering different tradeoffs that can further be beneficial to our business partner. We then study the effect of product prices on the overall accuracy. Finally, we validate the necessity of our pruning strategies to ensure good response times.

B. Dataset

Our dataset, provided by our business partners, represents customers who own a loyal card and who purchase products at different gas stations that are geographically distributed in France. It contains the purchase history of those customers for a period of approximatively 28 months, from January 2017 to April 2019. Each transaction record is associated to a customer, a product and a timestamp. the dataset contains $425,406$ different customers and $16,891$ different products. Each product is associated with a category such as *car wash, lubricants, hots drinks, coffee, etc*. Table IV shows a summary of the characteristics of the dataset.

C. Experimental protocol

The widely used strategy for evaluating accuracy of recommendations in offline experiments splits the dataset into training and test sets. The test set is used to simulate future transactions (ratings, clicks, purchases, etc) and it usually contains a fraction of transactions. The remaining interactions are kept in the training set and are fed to the recommendation algorithm which usually outputs a list of top-k product recommendations. The accuracy of recommendations is then evaluated on the test set. The split ratio between the training and test sets is usually chosen to be $80/20$.

In our experiments we refined the splitting by leveraging the timestamps of purchases with a chronological split, inspired from existing work [7], [8]. The availability of timestamps in the purchase records enables us to attempt a more realistic experiment. We hence train our algorithm on past purchases and test the results on future purchases. This setting is illustrated in Figure 1 that contains an example of a timestamped dataset. We split the dataset according to a given point in time (the vertical line in the figure), which acts as our "present" (the time we apply our algorithm). Past purchases which are located to the left of the split point are used for training, whereas future purchases located on the right side are used for testing. Customers on the right side of the split point, with purchase histories timestamped only in the "future" are discarded, while "past customers" appearing on the left side of the split point are kept in the training phase. In particular for our dataset, we used purchase records from January 2017 to December 2018 for training and records from January 2019 to April 2019 for testing.

As it is often practiced in the literature of recommender systems for retail datasets [9], we also discard "cold start" customers having a purchase history below a certain threshold. In our experiments, we only keep those customers with at least 10 past purchases.

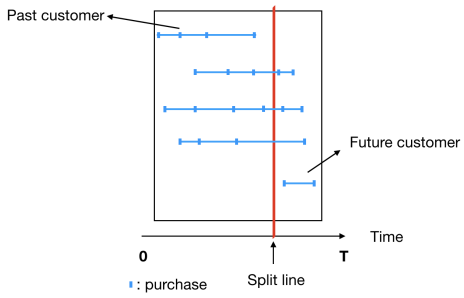


Fig. 1: Training and test sets in a timestamped dataset

D. Evaluation measures

Our goal is to evaluate the effectiveness of our recommendations with respect to recommendation accuracy and also the generated profit margin. We use two evaluation measures, the first one is *Precision* and the second is the average margin.

Precision is the most widely used recommendation measure to assess the accuracy of recommendations [10]. A recommendation algorithm outputs a sorted list of top-k products given the purchase history of a customer. *Precision* indicates the proportion of relevant recommended products from the total number of products in the final recommended list (i.e. k), which is defined as:

$$prec_u@k = \frac{card(S_u@k \cap test_u)}{k}$$

where, given a customer u, $S_u@k$ is the set of top-k recommendations and $test_u$ is the target test set of customer u that contains the products purchased in the test set.

The second evaluation measure is used to quantify the average profit margins generated by the top-k recommended products. We define it as follows:

$$margin_u@k = \frac{\sum_{i \in S_u@k} Margin(u,i)}{k}$$

where, given a customer u, $S_u@k$ is the set of top-k recommendations.

The value k is chosen by our business partners.

To compute the final performance values for both measures, we average the results over all the customers present in the test set. The final precision score is given by:

$$prec@k = \frac{\sum_{u \in test_U} prec_u@k}{|test_U|}$$

and similarly, the final margin score is given by:

$$margin@k = \frac{\sum_{u \in test_U} margin_u@k}{|test_U|}$$

where $test_U$ is the set of customers present in the test set of customers $\mathcal{U}$.

E. Detailed results

In this section we run our experiments and report detailed results. We first start by reporting the results of scalarization. We then examine more thoroughly the behavior of our bi-objective algorithm.

1) Scalarization: To show the effectiveness of our work, and the need to account for a bi-objective optimization, we investigate the use of scalarization to transform our setting into a single objective optimization problem by optimizing a new objective function. In this function, a parameter λ is used to combine the two optimization dimensions: utility and margin. The objective function is written as: $(1-\lambda) \times Utility(u,i) + \lambda \times Margin(u,i)$ and λ is in the range $[0,1]$. In our experiment, The values of λ are drawn from $\{0, 0.3, 0.5, 0.7, 1\}$, and for each value, we compute the average $precision@5$ and $margin@5$ over all test customers. We chose $k = 5$ because customers usually purchase a small number of products in a single visit to a gas station, thus 5 recommendations are a reasonable choice according to our business partners. We have a total of $4,589$ test customers. Test customers are those that have at least 5 purchases in the test set (data from January 2019 to April 2019) and who also have at least 10 purchases in the training set (data from January 2017 to December 2018). Unless otherwise stated, this is the number of customers we have in the remainder of the experiments.

Results are shown in Figure 2 for *Precision* and Figure 3 for margin. As we increase λ, the weight given to precision decreases and its value decreases accordingly. On the other hand, higher values of λ shift the weight to margin thereby increasing its value. Note that the highest precision value is 40% which is reasonable given the high sparsity of our purchase matrix. These results confirm the conflicting nature of our two objectives and the need for an algorithm optimizing both objectives at once.

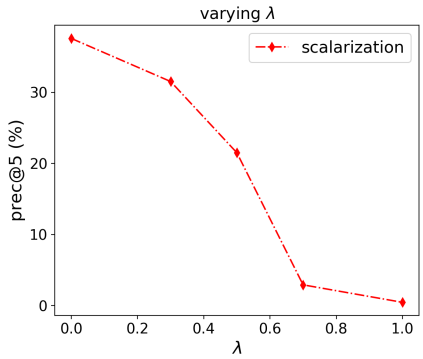


Fig. 2: Average precision for different values of λ

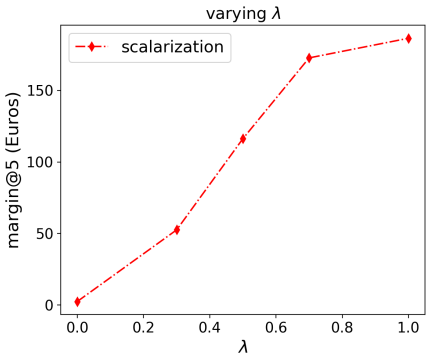


Fig. 3: Average profit margin for different values of λ

2) Our Approach: Our algorithm can return multiple optimal tradeoff solutions for one customer in only one run, while traditional recommender algorithms output only one solution. In this way, our algorithm provides multiple alternatives to product managers at our business partner, offering different tradeoffs between recommendation utility and generated profit margins. Based on the concept of Pareto dominance, no solution is better or worse than another as each solution presents a different tradeoff. Table V contains an example of different 5-Pareto sets that are generated by our algorithm for a random customer. The table reports the aggregated utilities and margins of each 5-Pareto set. The example clearly indicates that the algorithm cannot select one best recommendation set from the returned k-Pareto sets. Therefore, to compute the accuracy of all sets, we need to *aggregate precision and margin* of different k-Pareto sets generated for a customer.

In the following experiments, we report three different ways of aggregating precision and margin in different k-Pareto sets. Each test customer receives a different number of k-Pareto sets. For instance, for $k = 5$, the number of solutions ranges from 16 to 49. We first aggregate precision and margin

TABLE V: An example of 5-Pareto sets generated for a random customer

Recommended set of products	utility	margin
$\{i_1, i_2, i_3, i_4, i_5\}$	1.79	23.11
$\{i_1, i_4, i_5, i_6, i_7\}$	2.41	18.64
$\{i_1, i_2, i_3, i_5, i_8\}$	1.36	24.53
$\{i_2, i_3, i_5, i_9, i_{10}\}$	0.46	26.76
$\{i_2, i_3, i_7, i_{10}, i_{11}\}$	0.04	37.44

for each test customer and then average over all $4,589$ test customers. The first aggregation, referred to as AVG computes the average precision and average margin over all the k-Pareto sets. The second, referred to as $Max\_utility$ selects the recommendation set that offers the highest precision value, and computes the associated average margin. The third and last, Max_margin, selects the recommendation set that offers the highest margin value, and computes the associated average precision.

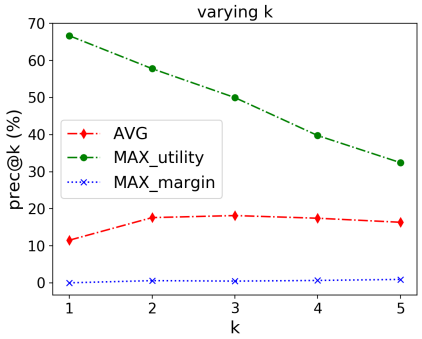


Fig. 4: Average precision over test customers: $Prec@k$

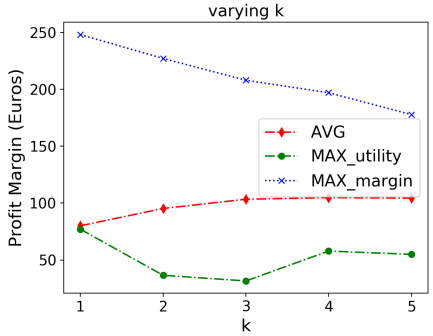
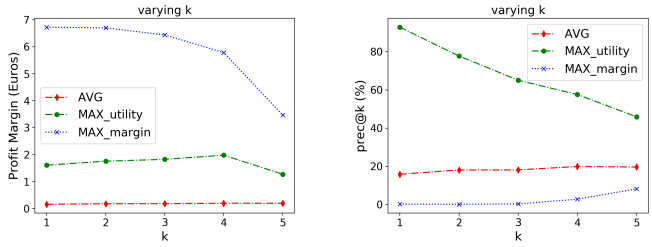


Fig. 5: Average profit margin over test customers

The results in Figures 4 and 5 and show that $Max\_utility$ achieves the highest precision values but at the same time the worst margins. Max_margin which essentially recommends expensive products achieves high values for generated margins at the cost of very low precision, independently of the size of the recommendation sets k. AVG shows that when considering all recommendation sets, we achieve a good balance between precision and margin. These results show that our algorithm can return different recommendation alternatives for each customer, which may be beneficial in helping products managers choose the best final recommendation that will *increase the profit margin without a significant loss in recommendation precision.*

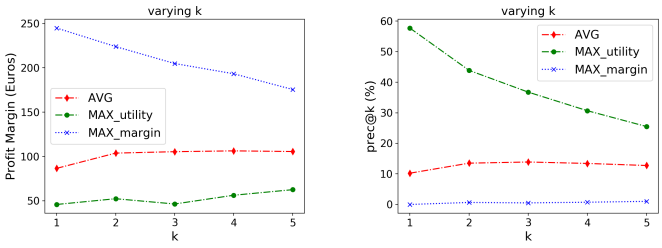
To study the effect of product prices on our optimization, we run additional experiments where we control the distribution of product prices in the training set. We created two separate datasets in which we only keep the cheapest products in the first one and the most expensive products in the second one. Figures 6b, 6a, 7b and 7a, report the values of precision and margin for different aggregation functions (AVG, $Max\_utility$, Max_margin) for our test customers using both sampled datasets. The results are consistent with the original dataset. The higher precision values observed on the dataset with the cheapest products can be explained by the low impact of price on utility (Figure 6b). Interestingly, precision in this case attains higher values (close to 90% for $Max\_utility$) than for the full dataset (around 70% for $Max\_utility$).

The lower precision values for the dataset with the most expensive products (close to 60% in Figure 7b) is justified by the preference of the algorithm on maximizing profit margin. This experiment suggests the need to involve product managers in choosing the best k-Pareto set for different customers according to their spending habits. This step can also be automated in the future to better strike a balance between utility and profit margin while also accounting for customers' expectation in terms of product price.



(a) (b)

Fig. 6: Average margin (a) and precision (b) over test customers where price of products is low: $prec@k$



(a) (b)

Fig. 7: Average margin (a) and precision (b) over test customers where price of products is high: $prec@k$

TABLE VI: Execution time in seconds for different values of k, averaged over 10 randomly selected customers

	Our algorithm	Exhaustive algorithm
k=2	0.02	0.12
k=3	0.16	9.99
k=4	1.18	128.17
k=5	4.93	481.45

F. Response time

The purpose of this last experiment is to validate the effectiveness of our pruning strategies. We compare our bi-objective algorithm with an exhaustive (naive) approach that generates all possible candidate sets. This is only possible after applying Algorithm 1 for pruning products dominated by more than k products.

Table VI reports the average running time in seconds for our bi-objective algorithm and the exhaustive approach. We can notice that our algorithm runs much faster, since it does not generate all possible candidate sets and thus computes only necessary dominance tests, thanks to our pruning strategies.

VI. Related work

We summarize two research areas that are related to our work.

Multi-objective optimization. A number of approaches have been developed to solve multi-objective problems [11], [12]. We examined the applicability of scalarization in our case (Section III-A). Another approach is multi-level optimization [4] which relies on a meaningful hierarchy between objectives. Since our objectives are independent and conflicting, multi-level optimization does not apply. Another popular method is ϵ-constraints [13] where one objective is optimized and others are constrained. This formulation could be used in our context in the case where a specific profit amount is desired and products must be chosen accordingly. Another category of works, focus on computing the pareto-frontier of the solution space [14], [15]. However, the problem we tackle is more complex since we are interested in returning multiple sets of k products. Indeed, we showed that computing the pareto-frontier and then check for combinations of k products is suboptimal.

Recommendation systems. Our work is obviously related to the rich area of recommendations [16], where the focus has mostly been on optimizing utility only. Few works have been proposed to also optimize the generated margin. [17] proposes a simple profit-aware recommendation system, where products are ranked in decreasing order of expected profit. [18] proposes a re-ranking based algorithm that takes as input a sorted recommendation list and re-ranks it according to the generated revenues. This is akin to multi-level optimization [4], which is suboptimal when the objectives are conflicting.

VII. Conclusion

In this paper, we formalized recommendation as a bi-objective optimization problem that maximizes customer utility and profit margin. We designed a dynamic programming algorithm that exploits properties of the solution space to prune unqualifying results. Our experiments on real retail datasets examine the balance between customer-centric and business-oriented goals and assess the ability of our algorithm to return accurate and profit-making recommendations. Our immediate work is to extend our formulation to the design of customer-centric promotional offers that take into account both customers' spending habits and business-oriented targets.

References

[1] C. Kim and J. Kim, "A recommendation algorithm using multi-level association rules," in *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*. IEEE, 2003, pp. 524–527.

[2] B. Sarwar, G. Karypis, J. Konstan, J. Riedl *et al.*, "Analysis of recommendation algorithms for e-commerce," in *EC*, 2000, pp. 158–167.

[3] C. H. Papadimitriou and M. Yannakakis, "On the approximability of trade-offs and optimal access of web sources," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000, pp. 86–92.

[4] A. Migdalas, P. M. Pardalos, and P. Värbrand, *Multilevel optimization: algorithms and applications*. Springer Science & Business Media, 2013, vol. 20.

[5] I. Bartolini, P. Ciaccia, and M. Patella, "Salsa: computing the skyline without scanning the whole sky," in *Proceedings of the 15th ACM international conference on Information and knowledge management*. ACM, 2006, pp. 405–414.

[6] ——, "Efficient sort-based skyline evaluation," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 4, p. 31, 2008.

[7] D. Paraschakis, B. J. Nilsson, and J. Holländer, "Comparative evaluation of top-n recommenders in e-commerce: An industrial perspective," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 1024–1031.

[8] B. Pradel, S. Sean, J. Delporte, S. Guérif, C. Rouveirol, N. Usunier, F. Fogelman-Soulié, and F. Dufau-Joel, "A case study in a recommender system based on purchase data," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 377–385.

[9] A. Said, A. Bellogın, and A. De Vries, "A top-n recommender system evaluation protocol inspired by deployed systems," in *LSRS Workshop at ACM RecSys*. Citeseer, 2013.

[10] H. Liu, Z. Hu, A. Mian, H. Tian, and X. Zhu, "A new user similarity model to improve the accuracy of collaborative filtering," *Knowledge-Based Systems*, vol. 56, pp. 156–166, 2014.

[11] I. Trummer and C. Koch, "Approximation schemes for many-objective query optimization," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014.

[12] G. Tsaggouris and C. Zaroliagis, "Multiobjective optimization: Improved fptas for shortest paths and non-linear objectives with applications," *Theory of Computing Systems*, vol. 45, no. 1, pp. 162–186, 2009.

[13] C. H. Papadimitriou and M. Yannakakis, "On the approximability of trade-offs and optimal access of web sources," in *FOCS*, 2000.

[14] M. T. Ribeiro, N. Ziviani, E. S. D. Moura, I. Hata, A. Lacerda, and A. Veloso, "Multiobjective pareto-efficient approaches for recommender systems," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 4, p. 53, 2015.

[15] M. Rodriguez, C. Posse, and E. Zhang, "Multiple objective optimization in recommender systems," in *Proceedings of the sixth ACM conference on Recommender systems*. ACM, 2012, pp. 11–18.

[16] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011.

[17] L.-S. Chen, F.-H. Hsu, M.-C. Chen, and Y.-C. Hsu, "Developing recommender systems with the consideration of product profitability for sellers," *Information Sciences*, vol. 178, no. 4, pp. 1032–1048, 2008.

[18] A. Azaria, A. Hassidim, S. Kraus, A. Eshkol, O. Weintraub, and I. Netanely, "Movie recommender system for profit maximization," in *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 2013, pp. 121–128.