Inventory Reduction via Maximal Coverage in E-Commerce

Shay Gershtein
Tel Aviv University
shayg1@mail.tau.ac.il

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Slava Novgorodov
eBay Research
snovgorodov@ebay.com

ABSTRACT

Many e-commerce platforms serve as an intermediary between companies and consumers, receiving a commission per purchase. To increase sales, these platforms tend to offer as many items as possible. However, in many situations a reduced subset of the items should be offered for sale, e.g., when opening an express delivery branch, starting operations in a new region, or disposing of redundant items to improve data quality and decrease maintenance costs. In all these cases it is imperative to select a reduced inventory which maximally covers consumer needs. A naïve, yet popular, solution is to focus on the top selling items. This however ignores the hidden relations between items, and in particular the tendency of shoppers to buy, in the absence of an item they are looking for, a satisfying alternative.

In this paper we introduce the *Preference Cover problem*, and investigate its application to practical inventory reduction. Given a large set of items, a bound on the number of items that can be retained and consumer preferences in terms of items popularity and suitability as alternatives, the goal is to select a reduced inventory which maximizes the likelihood of a purchase. We first model the problem via a dedicated weighted directed graph which captures the relevant information, then study two problem variants, which differ in their interpretation of the probabilistic dependencies between consumer preferences. We prove both variants are NP-hard, and characterize their approximation hardness. Since in the practical application the overall number of items and the bound on the reduced item set are very large - in the order of magnitude of millions - we propose a highly parallelizable and scalable algorithm along with approximation guarantees. Finally, we present an end-to-end solution that fits the real-world e-commerce application, and provide an extensive set of experiments demonstrating the efficiency and effectiveness of our solution.

1 INTRODUCTION

Due to the rapid growth of the e-commerce industry, online selling has become one of the most trending businesses of today. Many e-commerce platforms serve as an intermediary between companies and consumers, receiving a commission per purchase. To increase the number of sales, such sites tend to offer a large number of items[1]. Nonetheless, they often pursue complementary objectives where selecting and offering a reduced inventory is required. Common such scenarios, reported by our industry collaborators, are the following:

Launching an express delivery store. When large companies provide express delivery services (alongside existing services) offering items for same-day-delivery, these items should be available in different warehouses for immediate shipping. It is not feasible to ensure immediate availability, in terms of storage space and maintenance overhead, except for a significantly reduced inventory, as seen for example in the Amazon Prime same-day-delivery catalog, which offers a small percentage of the entire inventory.

Opening a branch overseas. When e-commerce platforms start operations in new regions, it is often done gradually, initially offering a small backlog of items. This is in part because one needs to require the vendors to ship abroad, with regulations often restricting the number of products that are allowed to be distributed. A notable example is AliExpress (consumer facing branch of Alibaba), which due to regulations restricts the number of items offered for shipping abroad.

Reducing maintenance costs. Maintaining large inventories incurs substantial data maintenance overhead (e.g., in data cleaning, validation, entity resolution and semantic enhancement [5]). Hence, companies periodically dispose of some small percentage of items deemed to be least valuable.

These examples demonstrate the need to select a reduced inventory that minimizes the loss in the number of sales. A naïve, yet popular, solution is to focus on the top selling items. This approach however entirely ignores the *hidden* relations between items. In particular, studies show that consumers are flexible, and when searching for a specific item are often willing to buy in its absence what they consider to be a reasonable alternative [34]. For example, in the absence of a specific 19” LG TV a customer may be willing to settle for a slightly bigger LG TV or for the same size TV from Samsung. Retaining a set of items which are not only popular in-and-of-themselves, but are also likely to “cover" the inventory and serve as suitable alternatives for omitted items, can significantly improve the overall satisfaction of the customers.

To model consumer preferences and item alternatives we use a *preference graph* - a directed graph with weights on both the nodes and the edges. The nodes correspond to the items, and the node weights reflect the items' purchase popularity (% out of total sold items). A (directed) edge from item A to item B indicates that, in A's absence, consumers consider B as a possible alternative[2]. The edge weight reflects the probability that a consumer is willing to buy B as an alternative to A, if A is missing. (We discuss how edge weights are derived via customary techniques from click-stream data, commonly available to e-commerce companies, in the Experiments Section.)

We use the *preference graph* to devise effective algorithms for the selection of items. However, before presenting our results, let us first illustrate through a simple example how the information provided in the graph is employed.

Example 1.1. Consider the preference graph depicted in Figure 1. Assume that of the five available items we wish to choose two. We can see that A is the best selling item (purchased by 33% of customers) while D is the least sold (6%). We can also see that consumers interested in E are likely to settle in its absence for D, but will not transitively buy C. Such behavior is common, for instance, when D (resp. C) is a one-step upgrade of E (D): people are often flexible and willing to add a small amount of

[1] Note that “item" here refers to a specific item type from a specific seller, e.g., Silver iPhone Xs 256GB by Best Buy, rather than to individual instances of this item.

[2] We assume that all transitive relationships, when/if exists, are directly represented in the graph by single edges.

© 2020 Copyright held by the owner/author(s). Published in Proceedings of the 23rd International Conference on Extending Database Technology (EDBT), March 30-April 2, 2020, ISBN 978-3-89318-083-7 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

Series ISSN: 2367-2005

10.5441/002/edbt.2020.63

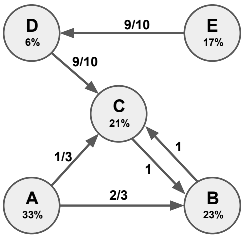


Figure 1: Sample graph of items

money in return for an upgrade, but a two-steps delta may be overly expensive. We can also see that consumers interested in C (B) will settle in its absence for B (C), and that B is a more likely replacement for A than C. Selecting the two best-sold items, A and B, is likely to satisfy about 77% of the customers (those interested in C, who in its absence are likely to purchase B, and those interested in A and B). Interestingly, a more careful analysis (which we only describe here intuitively and will formalize later in the paper) shows that in fact retaining B and D (the least sold item!) is the optimal solution, covering 87.3% of requests. Intuitively this is because B covers most requests for A, B, and C, whereas D covers itself and most of E. In this simple example the sets of requests served by the two retained items are disjoint, but a similar analysis can be applied to general overlapping cases.

The probability of a purchase is contingent on the dependencies between choosing different alternatives. Such dependencies can be extremely complicated, hence a practical model should simplify them in a manner which approximates well real life settings. We define in this paper two variants of the Preference Cover problem, which we have observed to be most prevalent in real-world e-commerce applications (see technical discussion in Section 5.2), the *Independent* and the *Normalized* variants. These variants differ by the semantics of edge dependencies. The Independent variant assumes independence between all alternatives. Whereas, the Normalized variant assumes that each consumer considers at most one item as an alternative she will actually buy, and thus the sum of weights of outgoing edges from any item is bounded by 1 (hence the name Normalized). In the experiments section, we show that both variants capture real-world consumer behavior. We will discuss the similarity and differences between these two problem variants (and when each is suitable) in depth in the following sections and propose effective algorithms to solve both.

To get an idea on what one may hope to achieve in terms of efficiency, we first study the computational complexity of the two problem variants. We prove that both are NP-hard but have different approximation bounds. Nevertheless, we devise a single greedy scheme that, with only minor adaptations, is able to support both variants, providing in both cases approximation guarantees. For the Independent variant we also prove this to be the optimal approximation ratio (matching the inapproximability bound we proved). For the Normalized variant, we show (by proving equivalence to the Max Vertex Cover problem) that while tighter theoretical upper bounds do exist, their corresponding algorithms are not scalable and thus impractical for our setting. Indeed, in the e-commerce application that we study here, both the overall number of available items and the number of items to be selected are very large - in the order of magnitude of millions. Thus, scalability is critical. Our solution consequently trades off the tightness of approximation guarantee in return for improved performance.

The simple greedy scheme which we use to solve both problem variants is highly parallelizable and scalable. It further has the added value of allowing to directly solve the complementary minimization problem, where, instead of an upper bound on the number of items to retain, one is given a lower bound on the percentage of item requests that should be covered, and the goal is to identify the smallest set of items that achieves the required coverage. Note that a naive solution for this complementary problem can be obtained via binary search on the target set size, by running any algorithm for the original problem. But this incurs a $O(\log n)$ factor overhead, n being the number of available items. Our direct greedy approach allows to avoid this overhead.

Our contributions can be summarized as follows.

(1) We formulate the Preference Cover problem and propose a graph-based model to capture it, with two natural possible variants.
(2) We study the theoretical computational complexity of the two problem variants and prove their NP-hardness and approximation hardness.
(3) We propose efficient algorithms to solve both variants, and prove their approximation guarantees.
(4) To demonstrate the practicality of our approach we describe the process of creating the preference graph from data available to actual e-commerce companies, based on well agreed upon inference methods in e-commerce research.
(5) We present an extensive experimental study, based on real-world data from a large e-commerce company, for both variants of the problem, demonstrating the effectiveness and efficiency of our algorithms.

Finally, we note that in the problem setting that we study here (which is common to many intermediary platforms [1]) the commission-per-purchase (or the perceived gain per purchase) is considered fixed and the intermediary platform is indifferent to the items' cost/revenue or required physical storage. Extending our work to support varying revenues and storage considerations, in a scalable manner, is an intriguing future work. We overview, in the Related Work, results in the field of operational research that incorporate such factors but in more complex models that do not lend to practical big data solutions. Other lines of work that bear resemblance to ours are recommendation systems and query results diversification. However, as we explain in the Related Work, the optimization problems they study differ from ours, yielding different complexity results and algorithms.

A first prototype of our system was implemented with help of our industry collaborators, and demonstrated at CIKM'19 [15]. The short paper accompanying the demonstration gives only a brief overview of the system architecture, whereas the present work provides a comprehensive description of the underlying model, algorithms and applications.

Paper outline. Section 2 provides the necessary definitions and formalism behind our problem. Sections 3 and 4 each study one of the two variants of our problem. Implementation and experimental studies are presented in Section 5. Finally, the related work appears in Section 6, and we conclude in Section 7.

2 PRELIMINARIES

We introduce here the *Preference Cover* problem. We start by formally describing the general model along with two concrete variants - the Independent and the Normalized variants.

Recall that the main motivation for our problem is an e-commerce setting, where consumers are interested in specific items but, if not available, may be willing to settle for some alternatives. Given a bound on the number of different item types a store may offer, our goal is to retain those which maximize the likelihood of purchase.

Formally, we represent consumer preferences via a *Preference Graph* which, along with an integer k, serve as input for the Preference Cover problem. A preference graph $G = (V, E, W_V, W_E)$ is a directed graph with weighted vertices and edges. The vertex set V corresponds to n items. For each vertex $v \in V$, its weight, $W_V(v) \in [0, 1]$, is defined as the probability of v being requested by a consumer. The sum of all node weights is therefore 1. We say that u is a *neighbor* of v if there exists an outgoing edge from v into u. The neighbors of a node v are all the items that are considered by consumers as possible alternatives. For each edge from v into a neighbor node u, its weight, $W_E(v, u) \in (0, 1]$, implies the probability of u matching a request for v as an alternative. We explain later how these edge weights are computed and interpreted. For simplicity we omit in the sequel subscripts of weighting functions when clear from context.

Given a number k, our goal is to choose a subset $S \subseteq V$ of k items, marking them as *retained*. Given a request for item v, if it is retained, the request is considered *matched*. Otherwise, if v is not retained, a request has some probability of being matched by another retained neighboring item of v, as indicated by the weights of edges outgoing from v into its retained neighbors. We define a target function $C : 2^V \to [0, 1]$, s.t. assuming S is the retained set of items, $C(S)$ is the probability a request drawn from the distribution indicated by the node weights is matched. The *Preference Cover problem* aims to compute $\arg\max_{S,|S|=k} C(S)$. We are ultimately only interested in whether or not a request is matched, and it makes no difference theoretically which item matches it. This corresponds to a real-life setting where intermediary platforms value the selling of each item as equally beneficial, and accordingly seek to maximize the number of sales.

We term $C(\cdot)$ as the *Cover function*, and say that the value $C(S)$ is the *cover of S*. Similarly, we call the probability a request for v is matched by a retained set S as the cover of v by S.

An explicit formula for computing $C(\cdot)$ is contingent on the dependencies, if such exist, between the probabilities indicated by the edges. In this paper we study two variants of the problem which, as our analysis of real data indicates, approximate well common real life scenarios: the *Independent* variant assumes that the probabilities modeled by edges are independent, while the *Normalized* variant assumes that each consumer considers at most one item as a most suitable alternative. In both cases, the goal is to retain the set of items which covers most of the predicted requests as implied by the preferences model.

In both variants when considering alternatives for a request for item v, we only take into account v's immediate neighbors. This is because, as mentioned earlier, the possible transitive processes of considering an alternative followed by an alternative to that alternative and so on is already taken into account when constructing the graph and assigning edge weights, which means, intuitively, that the preference graph is the transitive closure of a graph modeling the probabilities to correspond to such replacement paths.

We now formally describe the two variants we study in this paper. In the presentation below, given a retained set S we denote the retained neighbors of node v by $R_v(S) = \{u|(v, u) \in E, u \in S\}$

2.1 Independent variant

In the *Independent* variant we assume complete independence between the edges. Namely, the probability a given alternative matches a request is not affected by whether or not a different alternative matches it. Thus, the probability of the event of **not** matching a request for a non-retained item v, which occurs when no retained alternative is suitable, is, due to independence, the product of all such probabilities, $\prod_{u \in R_v(S)}(1 - W(v, u))$. The probability of the complement of this event, which is matching the request, is $1 - \prod_{u \in R_v(S)}(1 - W(v, u))$. We next formally define the Independent variant of the Preference Cover problem.

Definition 2.1 (IPC_k). Given a preference graph G and an integer k, the Independent Preference Cover problem (IPC_k) is computing $\arg\max_{S,|S|=k} C(S)$, where

$$C(S) = \sum_{v \in S} W(v) + \sum_{v \in V \setminus S} \left[W(v) \cdot (1 - \prod_{u \in R_v(S)} (1 - W(v, u))) \right]$$

The first addend is due to requests for retained items being matched with probability 1. The second addend corresponds to summing over all items not in S, for each such item v adding the probability it is both requested ($W(v)$) and covered by S ($(1 - \prod_{u \in R_v(S)}(1 - W(v, u)))$. Recall that we are computing the purchasing probability of a single consumer session, which in the hypothetical case where all items are available would have resulted in a purchase. The overall expected number of sales, given only S is retained, is thus the number of such sessions times $C(S)$. Cases where a consumer is looking to purchase several items, or several copies of the same items, are modeled as separate sessions.

2.2 Normalized Variant

In the *Normalized* variant we assume that each consumer considers at most one item as a suitable alternative (i.e. the item she will actually buy). Neighbors are therefore dependent, in the sense that for any requested item v, a retained neighbor matching the request implies that all other neighbors do not. It follows that the sum of the weights of all edges outgoing from any given node is at most 1, and given a request for a non-retained item v, the probability it is matched is $\sum_{u \in R_v(S)} W(v, u)$. We next formally define the Normalized variant of the Preference Cover problem.

Definition 2.2 (NPC_k). Given a preference graph G and an integer k, the Normalized Preference Cover problem (NPC_k) is computing $\arg\max_{S,|S|=k} C(S)$, where

$$C(S) = \sum_{v \in S} W(v) + \sum_{v \in V \setminus S} \left[W(v) \cdot \sum_{u \in R_v(S)} W(v, u) \right]$$

Here again the first addend, $\sum_{v \in S} W(v)$, is due to requests for retained items being matched with probability 1. The second addend corresponds to summing over all items not in S, and for each such item v adding the probability it is both requested ($W(v)$) and covered by S ($\sum_{u \in R_v(S)} W(v, u)$).

We discuss the choice of these particular variants and which real-life settings they correspond to in Section 5. Intuitively, the Independent variant asserts that the opinion on the suitability of a given alternative is not demonstrated to be strongly dependent for most consumers on their opinion of other alternatives. The dependencies are either insignificant overall or tend to cancel out when summed over the entire user base. In contrast, the Normalized variant is suited for domains where it is demonstrated that consumer requests are often very specific in nature, and the number of suitable alternatives is very small, which the Normalized variant models as a single alternative at most per request (though this can be a different alternative per each request for the same item). Finally, note that the weight of a node does not necessarily represent the probability of a premeditated and explicit request for the item, rather, more generally, the probability, given all items are available and a purchase is made, of this specific item being the one purchased.

2.3 Set functions

We next present some general definitions and results pertaining to set functions ($f : 2^V \rightarrow \mathbb{R}$, given universe V), which will be useful in the following sections for formally characterizing $C(\cdot)$.

Definition 2.3. f is **nonnegative** if $\forall S \subseteq V$: $f(S) \geq 0$.

Definition 2.4. f is **monotone** if $\forall S \subseteq V, \forall v \in V$: $f(S \cup \{v\}) \geq f(S)$.

Definition 2.5. f is **submodular** if $\forall S \subseteq T \subseteq V, \forall v \in V$: $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$.

The following is a key result in submodular optimization.

LEMMA 2.6 ([22]). *Given universe V, $k \leq |V|$, and a nonnegative, monotone submodular function $f : 2^V \rightarrow \mathbb{R}$, there exists a polynomial algorithm achieving an approximation ratio of at least $(1 - \frac{1}{e})$ in maximizing $f(\cdot)$ over subsets of size k. This algorithm, at each of its k iterations, selects an element maximizing the marginal gain to $f(\cdot)$.*

2.4 Related problems

Finally, we present definitions and results pertaining to related problems, which will also serve us in the formal analysis.

Definition 2.7. In the *Directed Max Dominating Set Problem (DS_k)* the input is a directed graph and a number k, and the goal is to find a subset of the vertices of size k such that the number of vertices adjacent to this subset is maximized.

Definition 2.8. In the *Max Vertex Cover Problem (VC_k)* the input is a number k and an undirected graph (self edges allowed) with positive weights assigned to its edges, and the goal is to select a subset of the vertices of size k such that the weight of edges incident to this subset is maximized.

We now provide hardness results pertaining to the above problems. The following theorem was proven in [21] for undirected graphs, but trivially extends to directed graphs, as undirected graphs are a special case where for each edge there exists a parallel edge in the opposite direction.

THEOREM 2.9. *[21, 25] The DS_k problem is NP-hard. It has no polynomial approximation algorithm of a factor higher than (1-1/e), unless $P = NP$. The problem is NP-hard even when the maximal degree in the graph is bounded by a constant.*

THEOREM 2.10. *[14, 27] The VC_k problem is NP-hard, and is hard to approximate to within $(1 - \delta)$ factor for some (small) $\delta > 0$, unless $P = NP$. Moreover, VC_k is NP-hard even for graphs of degree at most 3.*

Tighter approximation hardness bounds for VC_k are not known, and existing algorithms are discussed in Section 3.2. We also note that the NP-hardness of the bounded degree cases in both of the above theorems was proven by [14] and [25] for the minimization versions of the Vertex Cover (VC) and Dominating Set (DS) problems, resp. In the minimization versions the goal is to find the smallest set covering the entire graph. However, the hardness extends to our top-k versions, as VC (resp. DS) can be solved by solving VC_k (resp. DS_k) at most n times for varying values of k.

In each of the following two sections we study in detail one of the two variants of the Preference Cover problem. We analyze the Normalized variant first, as it is more complicated technically. Moreover, part of the discussion of the Normalized variant (in particular the proposed algorithm) applies, with minor adaptations, to the Independent variant as well.

Table 1: Approximation ratios of the greedy algorithm and best known polynomial algorithms for VC_k

Range of k/n	Greedy Algorithm	Best Known
$o(1)$	$(1 - 1/e)$	0.75+ϵ (SDP) [11]
$\Theta(1)$, [0, ≈0.39)	$(1 - 1/e)$	0.92 (SDP) [19]
(≈0.39, ≈0.72)	$(1 - (1 - \frac{k}{n})^2)$	0.92 (SDP) [19]
(≈0.72, 0.74)	$(1 - (1 - \frac{k}{n})^2)$	≈0.93 (SDP) [17]
[0.74, 1]	$(1 - (1 - \frac{k}{n})^2)$	$(1 - (1 - \frac{k}{n})^2)$ [11]

3 NORMALIZED VARIANT

3.1 Theoretical Analysis

We begin this section by studying the complexity and the approximation hardness of NPC_k, the Normalized variant of the Preference Cover problem. In both these respects we prove an equivalence to the VC_k problem, which has been studied extensively in the literature, implying in particular that both upper and lower bounds on the approximation of VC_k apply for NPC_k as well. We then discuss algorithms for solving NPC_k. Here again we utilize the equivalence to VC_k to adapt its known algorithms to our setting. Concretely, we present the currently best known approaches in terms of the approximation ratio guarantee, which vary for different ranges of k, and discuss their performance w.r.t. scalability. We note the trade-off between performance and approximation guarantees, and as our goal is to provide a scalable solution for big data settings, we focus on a fast algorithm, for which we provide an efficient parallelizable implementation, and demonstrate it to be highly scalable in our experiments. Moreover, its approximation factor is the best known for high values of k, and is not far off for lower ranges. We further argue that all algorithms known to provide a better approximation guarantee for lower ranges are not scalable at all. We discuss additional advantages of our approach in Section 3.2.

THEOREM 3.1. *The NPC_k problem is hard to approximate to within a $(1 - \delta)$ factor for some (small) $\delta > 0$, unless $P = NP$. The problem is NP-hard, even when the maximal degree (disregarding edge orientation) is 3. Furthermore, any α-approximation algorithm for VC_k implies an α-approximation algorithm for NPC_k, and vice versa.*

PROOF. We first reduce NPC_k to VC_k (Definition 2.8), and show that any α-approximation algorithm for VC_k implies an algorithm for NPC_k with the same factor. Given an instance $I = \langle G, k \rangle$ of NPC_k, we add a self edge to every node whose sum of outgoing edge weights is less than 1, and assign to it the weight which completes the total outgoing weight to 1. This added weight intuitively represents the relative part of requests for this item which cannot be covered by any alternative. Observe that this change has no bearing on the cover function $C(\cdot)$, as when a node is retained, its weight is covered entirely all the same. Next we reduce this instance I to an instance $I' = \langle G', k \rangle$ of VC_k, such that G' has the same nodes as G only without weights, the same edges only without orientation, and the weight of every edge (v, u) changes from $W(v, u)$ to $W(v) \cdot W(v, u)$ (multiplied by the weight of its origin node). Note that G' may have pairs of nodes connected by 2 parallel edges, if edges in both directions connected these nodes in G. This is equivalent, w.r.t. VC_k, to replacing both edges with a single edge whose weight is the combined weight of the original two. However, we do not combine parallel edges, as we analyze their weight contributions separately.

We now argue that for any choice of a node set $S \subseteq V$, its cover weight in G' is equal to the cover $C(S)$ in G. Indeed, let E_S denote all edges that are **outgoing** from a node in S in G, then the sum of weights of the edges in G' originating from E_S is exactly $\sum_{v \in S} W(v)$, which corresponds to the first addend in the formula in Definition 2.2 (each node in S contributes its weight to $C(S)$); considering all remaining edges adjacent to S in G' which are not in E_S: the sum of weights of all such edges originating in any given node $v \notin S$ is $W(v) \cdot \sum_{u \in R_v(S)} W(v, u)$ (recall that $R_v(S) = \{u|(v, u) \in E, u \in S\}$), which, when summing over all such $v \notin S$, is exactly the remaining addend in the formula for $C(S)$, thus proving the equivalence.

As for the other direction, which is proving that NPC_k is just as hard to approximate as VC_k, from which the NP-hardness and hardness of approximation follow, assume we are given an instance $I' = \langle G', k \rangle$ of VC_k. We reduce it to an instance $I = \langle G, k \rangle$ of NPC_k such that all nodes in G are the same as in G', and the edges are also the same with the orientation chosen arbitrarily. Now for any node v let M_v denote the sum of weights of all its outgoing edges at this point, then we set $W(v) = M_v$, and for every outgoing edge e from v, we change its weight from its original weight in G', denoted by $W'(e)$ to $W(e) = \frac{W'(e)}{M_v}$. It follows that the sum of weights of all outgoing edges from any given node, which has at least one such edge, is 1. We set the weight of any node without any outgoing edges adjacent to it to 0.

Following this reduction the sum of weights of all nodes, denoted by N, is not necessarily 1. This requirement over the sum of node weights is due to the semantics of the problem, and is computationally insignificant. Indeed, we can normalize, and divide all node weights by N, and denote the resulting graph $\hat{G}$. It follows that the cover of any solution S, including the optimal solution, changes after this normalization from $C(S)$ in G to $\frac{C(S)}{N}$ in $\hat{G}$, and thus the approximation ratio is not changed. Given an α-approximation algorithm for NPC_k, we run it over the normalized graph $\hat{G}$, and it follows that this solution has the same ratio for the non-normalized instance I of NPC_k. Finally, observe that if we reduce I to an instance of VC_k, as we described in the first direction of the proof, we once again get I' (multiplying edge weights by the original node weight cancels out their normalization by the same factor), implying, by the same logic as before, that for any set S its cover weight in G' equals $C(S)$ in G. Therefore, the same node set S guarantees an α-approximation of the original VC_k instance. Note that as the reduction preserves the maximal degree, it follows that NPC_k is NP-hard for maximal degree 3. □

Due to the equivalence to VC_k, tight inapproximability bounds for NPC_k are also not known. We discuss below existing approximation algorithms, as VC_k algorithms can be adapted to NPC_k maintaining the same approximation factors. For a recent review of VC_k results see [19].

3.2 Algorithms

In this section we discuss algorithms for NPC_k, and focus on the implementation of a greedy algorithm, which we argue to be by far the most scalable option, on top of having high approximation guarantees.

The equivalence of NPC_k to VC_k combined with the linear approximation-preserving reductions described in the proof of Theorem 3.1, imply that when looking for an algorithm for NPC_k, one should examine the algorithms known for VC_k, an extensively studied problem. The algorithms providing the best known approximation guarantees vary for different ranges of k. Detailed results are presented in Table 1 (the second column pertains to a greedy algorithm on which we focus in the sequel). Nevertheless, for $k < 0.74n$, all top algorithms are based on the same core technique of semidefinite programming (SDP). SDP is a much more general extension of linear programming (LP). An algorithm based on LP also exists [2], with an approximation factor of 0.75. These SDP (and LP) based algorithms are important mainly for the theoretical analysis of the problem, in particular finding out the best approximation ratios. On the other hand, these solutions are not scalable, especially for big data settings, as they are known for having an impractical running time, even for medium sized programs [7, 37] (for example, the number of constraints in the program, as devised in [11], is of order $O(n^3)$).

Algorithm 1: Greedy Algorithm

```
  Input: G, k
1 $S = \emptyset$
2 foreach $1 \le i \le k$ do
3     foreach $v \in V \setminus S$ do
4         $C(S \cup v) = \text{Gain}(S, v)$
5     $\hat{v} = \arg\max_v C(S \cup v)$
6     $S, C(S) \leftarrow \text{AddNode}(S, \hat{v}, C(S))$
7 return $S, C(S)$
```

Another approach is a greedy algorithm, introduced in [16], and analyzed by [11] to have an approximation factor of $max\{(1 - 1/e), (1-(1-\frac{k}{n})^2)\}$. This factor positively correlates with k, and for $k \ge 0.74n$ it is the best known guarantee, exceeding a 0.93 factor. To the best of our knowledge, any algorithm which surpasses the approximation guarantee of the greedy algorithm, for any range of k, is based on SDP or LP. The greedy algorithm, unlike these alternatives, lends itself to an efficient implementation. As our goal is to provide a practical solution, we focus on the greedy algorithm, which we implement and evaluate. The implementation we provide is adapted directly into our setting without a reduction to VC_k. It is parallelizable, and as we prove in the experiments (Section 5), highly scalable even for graphs containing millions of nodes. Additional advantages of this approach are discussed towards the end of this section. As mentioned, Table 1 depicts, for various ranges of k, the best known approximation factor, alongside the factor of the greedy algorithm.

Greedy Algorithm The greedy algorithm (Algorithm 1) incidentally applies schematically for both the Normalized and the Independent variants (with latent distinctions, described in the next section, devoted to IPC_k, the Independent variant). We use an array I of size n, with an entry $I[v]$ for each $v \in V$, eventually set to the probability of v being both requested and matched by the produced solution S (the product of $W(v)$ and the cover of v by S). The summation of all entries in I, as indicated in Definition 2.2, equals $C(S)$. For simplicity, we assume the preference graph G and the array I are *global variables*, with I initialized to zeros.

Algorithm 1 maintains an initially empty solution set S (line 1). At each of its k iterations (line 2), it goes over all nodes currently not in S (line 3), and for each such node it computes the gain to $C(S)$ obtained by adding it to S (line 5). The node that maximized this gain is then chosen (line 6) and is added to S, with $C(S)$ updated accordingly (line 7). Finally, after completing k iterations, S and $C(S)$ are returned (line 8).

Algorithm 2: Gain - Normalized

```
   Input: S, v
   Global: G, I
1  $g = W(v) - I[v]$
2  foreach $u \in V \setminus S$ s.t. $W(u, v) \in E$ do
3      $g$ += $W(u) \cdot W(u, v)$
4  return $g$
```

The procedures *Gain* (Algorithm 2) and *AddNode* (Algorithm 3) are conceptually similar as both compute the marginal effects of adding a given node, with the main distinction being that *AddNode* also updates accordingly I and $C(S)$.

In Algorithm 3, line 1 adds to S the node v which was chosen in Algorithm 1 as maximizing the marginal gain. Line 2 adds to $C(S)$ the gain in the cover of itself and line 3 updates $I[v]$ to $W(v)$ as the newly added node covers itself completely. Next we iterate over all nodes outside of S with an edge into v, and for each such node u, we compute the marginal gain to its cover by v, and add it to $I[u]$ and $C(S)$ (lines 5 and 6, resp.). We can see that after each call to Algorithm 3 the array I is updated with each entry set to the contribution of covering the corresponding node by S to $C(S)$. As we mentioned, Algorithm 2 is the same as Algorithm 3, only focusing solely on the marginal gain, without updating I and $C(S)$.

Although we adapted the greedy algorithm directly to preference graphs, without reducing to VC_k, it is easy to show along the same lines as the proof of Theorem 3.1, that a reduction to VC_k would have resulted in choosing the same nodes. Hence, the approximation factor of $max\{(1 - 1/e), (1 - (1 - \frac{k}{n})^2)\}$, proven for VC_k in [11], holds here as well.

To illustrate the operation of Algorithm 1, consider the following example.

Example 3.2. Recall the preference graph depicted in Figure 1 and assume $k = 2$. The algorithm first computes the gain obtained by selecting each node and retains the most beneficial, which is B (66%, covering $W(B)$, $W(C)$ and 2/3 of $W(A)$). After B is retained, the algorithm proceeds to the second and final iteration, to choose the next node with the highest marginal gain, which is D. D itself is requested only by 6% of consumers, while A and C are 22% and 33%, resp. However, B being selected in the previous step reduces A's and C's marginal gain to 11% (the 1/3 of $W(A)$ corresponding to consumers not accepting B as an alternative to A) and 0% (all consumers who wanted C are happy to get B instead), resp. On other hand, D covers 6% (itself) and 15.3% (9/10 of $W(E)$ - consumers that wanted E, but also agree to have D as alternative), which gives a total of 21.3%. Finally, the retained items, B and D, cover 87.3% of consumer preferences (which in this case is also the optimal possible pair).

Performance Analysis We now analyze the complexity of the greedy algorithm. Let $d(v)$ denote the **incoming** degree of a node v, and let D denote the maximum incoming degree over all nodes. Observe that in both Algorithms 2 and 3 the number of operations performed is $\Theta(d(v)) = O(D)$ (v is the node whose marginal gain is evaluated). For each of the k iterations, Algorithm 2 is called $O(n)$ times, hence the overall time complexity is $O(nkD)$.

Furthermore, the algorithm is highly parallelizable. When Algorithm 1 iterates in line 3 over $O(n)$ nodes to compute their marginal gain, computations for each node are independent, and can be performed in parallel. Moreover, in each such call to Algorithm 2 (or 3) the iteration in line 2 over the $O(D)$ nodes adjacent to the added node can also be done in parallel. Concretely, for $N < nD$ threads, the complexity of each of the k iterations becomes $O(\frac{nD}{N})$ resulting in an overall $O(k + \frac{nkD}{N})$. The space complexity of the algorithm (excluding the input, which we treat as read-only), is $O(|V|)$ for storing I. We can also reduce the space complexity to $O(k)$ by doing away with I, at the expense of computing the value $I[v]$ from scratch in line 1 of Algorithm 2 (line 2 in Algorithm 3), which takes $O(D)$ operations, and does not affect the overall complexity.

Algorithm 3: AddNode - Normalized

```
   Input: S, v, C(S)
   Global: G, I
1  $S \leftarrow S \cup \{v\}$
2  $C(S)$ += $W(v) - I[v]$
3  $I[v] = W(v)$
4  foreach $u \in V \setminus S$ s.t. $W(u, v) \in E$ do
5      $C(S)$ += $W(u) \cdot W(u, v)$
6      $I[u]$ += $W(u) \cdot W(u, v)$
7  return $S, C(S)$
```

Additional Advantages Finally, we identify several additional benefits of our greedy algorithm. First, we can return I as part of the output. This enables to efficiently compute, for each non retained item u, its cover by S, which equals $\frac{I[u]}{W(u)}$. This provides important information about which item requests are affected by reducing the inventory and to what extent. Moreover, the incremental nature of the greedy approach, when the retained set is produced in the order in which the nodes were added to it, can provide solutions to related instances and problems. Namely, an ordered solution S of size k, also produces the solution for any $k' < k$, which is the first k' nodes in S (the same solution that running the algorithm with k' would have produced). Therefore, solving for $k = n$ provides at once the solutions for all k values, and, moreover, directly provides (an approximated) solution for the related problem where the goal is to retain the smallest set, such that the cover exceeds a given threshold.

4 INDEPENDENT VARIANT

4.1 Theoretical Analysis

We now study the theoretical properties of IPC_k, the Independent variant, and our proposed algorithm. We first prove our main result, stating that IPC_k is NP-hard (even given a constant bound on node degrees), and has a tight approximation factor of $(1 - 1/e)$ by a polynomial time algorithm. In fact, this factor is achieved by the same greedy approach as the one discussed in Section 3 for NPC_k, with small adjustments. The adaptations that need to be performed to previously presented algorithms and analysis of the adapted algorithms are then discussed in detail in Section 4.2.

THEOREM 4.1. *The IPC_k problem has a tight approximation bound of $(1 - 1/e)$ in polynomial time, unless P=NP. Moreover, it is NP-hard, even given a constant bound on the maximal node degree (disregarding edge orientation).*

PROOF. To prove hardness of approximation (and thereby the NP-hardness), we assume an α-approximation algorithm for IPC_k, and reduce an instance $I' = \langle G', k \rangle$ of DS_k to an instance $I = \langle G, k \rangle$ of IPC_k, such that an α-approximation solution is implied

Algorithm 4: Gain - Independent

```
  Input: S, v
  Global: G, I
1 $g = W(v) - I[v]$
2 foreach $u \in V \setminus S$ s.t. $W(u,v) \in E$ do
3     $g$ += $W(u,v) \cdot (W(u) - I[u])$
4 return $g$
```

for DS_k. The hardness results then follow from Theorem 2.9. The reduction preserves the maximal degree, implying NP-hardness even given a constant bound on it. Concretely, G has the same nodes and edges as G', except all edge orientations are **reversed**. All edges are assigned the weight 1, and all nodes are assigned the weight $\frac{1}{n}$.

We argue that for any solution $S \subseteq V$, the number of vertices it dominates[3] in G' is $n \cdot C(S)$ ($C(S)$ is the cover of S in G), proving the same approximation ratio. To see this, observe that all $|S|$ nodes in S dominate themselves in G', which corresponds to the first addend in the formula in Definition 2.1 (the sum of weights of nodes in S) which equals $\frac{|S|}{n}$. The remaining nodes dominated by S in G' form the set of all nodes outside of S that have an incoming edge from S, denoted by T. This corresponds to the remaining addend in the formula for $C(S)$, which (as the edges in G are reversed w.r.t. G') is the cover of the set of nodes outside of S with an outgoing edge into S in G, which equals $\sum_{v \in T} \frac{1}{n} = \frac{|T|}{n}$. Overall, we see that there is a fixed ratio of $\frac{1}{n}$ between the values of the target functions of the two problems for any S, proving the equivalence of the approximation. This proves a $(1-1/e)$ hardness bound on the approximation of IPC_k.

To show this bound is tight, we prove that all conditions specified in Lemma 2.6 (the function is nonnegative, monotone and submodular) hold for $C(\cdot)$, implying that a greedy incremental algorithm maximizing the marginal gain at each of its k iterations guarantees a $(1-1/e)$ approximation. Indeed, these properties are evident from the formula in Definition 2.1. $C(\cdot)$ is by definition nonnegative. The addition of any node to the solution maximally covers itself, and can only increase the cover of any other node (it decreases the product $\prod_{u \in R_v(S)}(1 - W(v,u))$ in the formula for $C(S)$, increasing the overall value), which proves monotonicity. Finally, $C(\cdot)$ is submodular: given two sets $S \subset T \subseteq V$ and a node u', we show that $f(S \cup \{u'\}) - f(S) \geq f(T \cup \{u'\}) - f(T)$. If u' belongs to any of these two sets then this follows trivially. Otherwise, as T covered u' at least as much as S (due to monotonicity), the complete covering of u' after its addition is at most the same for T compared to S. As for any other node v (which has an edge into u', as otherwise adding u' can have no effect on it): if $v \in T$ and $v \notin S$, then T already covered it completely, and the addition of u' to T adds nothing. Otherwise ($v \notin T$), for S and T, resp., the effect is that both $\prod_{u \in R_v(S)}(1 - W(v,u))$ and $\prod_{u \in R_v(T)}(1 - W(v,u))$ are multiplied by the $(1 - W(v,u'))$. As the second product (with T) is not bigger than the first (with S), then the additive difference after multiplying it by $(1 - W(v,u'))$ is also not bigger, leading to an overall smaller (or equal) increase in the cover (when added to T). As the increase in the cover of every node is not bigger when adding to T, submodularity follows.

□

Algorithm 5: AddNode - Independent

```
  Input: S, v, C(S)
  Global: G, I
1 $S \leftarrow S \cup \{v\}$
2 $C(S)$ += $W(v) - I[v]$
3 $I[v] = W(v)$
4 foreach $u \in V \setminus S$ s.t. $W(u,v) \in E$ do
5     $C(S)$ += $W(u,v) \cdot (W(u) - I[u])$
6     $I[u]$ += $W(u,v) \cdot (W(u) - I[u])$
7 return $S, C(S)$
```

4.2 Greedy Algorithm

We now discuss our proposed algorithm for IPC_k. Here again we opt for the same greedy approach as presented in Section 3.2 for NPC_k, which for IPC_k guarantees an optimal $(1 - 1/e)$ approximation factor, following Theorems 4.1 and 2.9. Moreover, the algorithm scheme is also depicted in Algorithm 1, with the distinction that the procedures *Gain* and *AddNode*, called, resp., in lines 4 and 6, have different implementations (resp., Algorithm 4 instead of Algorithm 2 and Algorithm 5 instead of Algorithm 3). These procedures, however, remain analogous to their counterparts for NPC_k (Algorithms 2 and 3), as the adjustments only reflect the technical difference between the formulas for $C(\cdot)$ in Definitions 2.1 and 2.2. Therefore, we do not discuss the algorithm in detail, as this is largely covered in Section 3.2, but rather focus on the distinctions from the NPC_k implementation.

Recall that I is an array, initialized with zeros, with an entry for each $v \in V$, denoted by $I[v]$, eventually set to the probability of v being both requested and matched by the produced solution S, and hence the summation of all entries in I equals $C(S)$. We once again set it to be a global variable whose value is saved between calls. In Algorithm 4 line 1 remains the same as in the analogous Algorithm 2, as in both variants it holds that a retained node is completely covered by itself. Line 3, however, is different. It pertains to the marginal gain obtained by adding the node v in the cover of node u, which has v as a neighbor. Let S and S' denote the solution set before and after resp., adding v, and let $I_S[u]$ and $I_{S'}[u]$ denote the correct value of $I[u]$ for solutions S and S', resp. The marginal gain by v over u is $I_{S'}[u] - I_S[u]$, which after doing the algebra is simplified into $W(u,v) \cdot (W(u) - I[u])$. This is the gain appearing in Line 3 of Algorithm 4. The correctness of this expression follows a straightforward computation, which we omit to avoid convoluted notation, and instead provide the intuition. Note that the computation of the probability u **not** being covered by S is the product of the probabilities of all its retained neighbours not being suitable alternatives. This probability can be easily computed from $I_S[u]$ (see the second sum in the formula in 2.1). The only change in S' is that this product is now multiplied by the probability v is also not a suitable alternative, which is $(1 - W(u,v))$. Therefore, when computing the product pertaining to S' (which implies $I_{S'}[u]$), we can reuse the computed product for S (implied by $I_S[u]$), and reduce the number of operations in computing the marginal gain to $O(1)$ per each such u. Moreover, the similarity of the computations for $I_S[u]$ and $I_{S'}[u]$ allows the simplification of the expression for the marginal gain. The adjustments made in Algorithm 5 are completely analogous.

Performance Analysis Following exactly the same considerations as in the NPC_k implementation (Section 3.2), both Algorithms 4 and 5 have the same time complexity, hence the overall

[3] A vertex is dominated by a solution S, if it is either in S or has an edge incoming from some node in S.

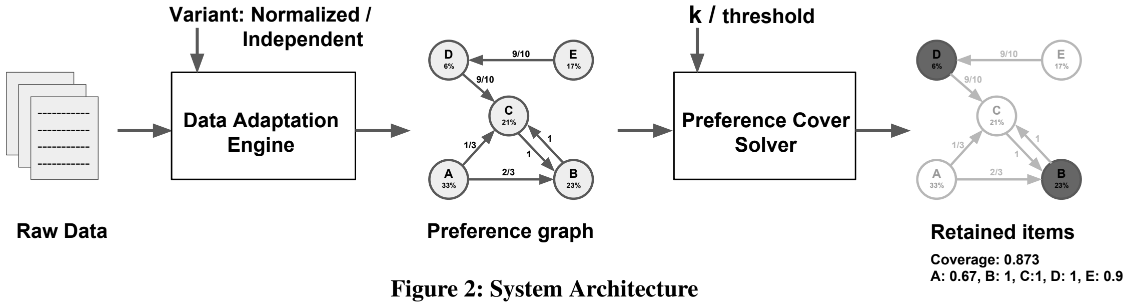


Figure 2: System Architecture

complexity of the greedy algorithm is $O(nkD)$ here as well (recall that D denotes the maximal incoming degree of a node in G). The potential parallelization is also the same, thus the complexity for $N < nD$ threads becomes $O(k + \frac{nkD}{N})$, same as in Section 3.2. As for the space complexity (excluding the input), while it is also $O(n)$, due to storing I, in the IPC_k case it is no longer true that we can do away with I to reduce the space to $O(k)$, without sacrificing efficiency (here the computation of the marginal gain is more reliant on previous computations). Finally, the additional benefits pertaining to the incremental nature of the greedy approach remain the same as for NPC_k, as described at the end of Section 3.2.

5 IMPLEMENTATION AND EXPERIMENTS

We start this section by describing the system architecture, focusing on the flow from raw data to a suggested list of k retained items. Afterwards, we discuss how real-life e-commerce data can be adapted to construct the preference graph. Then, we describe the experimental setup, where we introduce the real-life datasets we used for the experiments, and describe concretely which adaptations we made to these datasets to fit our models. Finally, we present the evaluation results.

5.1 System Architecture

The system architecture, depicted in Figure 2, demonstrates the end-to-end flow. The system consists of two main modules: the *Data Adaptation Engine* and the *Preference Cover Solver*. The *Data Adaptation Engine* takes as input the raw e-commerce data and the variant (Normalized or Independent), and builds the corresponding preference graph. Detailed discussion about what type of raw data is necessary, which of the two variants to choose in a given situation, and how the adaptation is actually performed is presented in the following section (Section 5.2). The constructed preference graph is then passed as input to the Preference Cover Solver, along with k, the desired number of retained items. The solver runs Algorithm 1, adapted to the specific variant (calling Algorithms 2 and 3 for the Normalized variant, and Algorithms 4 and 5, resp., for the Independent variant). The solver produces a list S of retained items (in the order in which the items were added by the algorithm), accompanied with metadata, such as $C(S)$ (the cover achieved by S), and the coverage percentage of every item (implied by the array I, used in our algorithms), i.e. how well the item is covered by the retained alternatives in S (the coverage of retained items is obviously 100%). For example, the rightmost part of Figure 2 highlights the produced set of retained items, B and D (for the input of $k = 2$ and the preference graph depicted in the center of Figure 2, originally introduced in Figure 1 as part of Example 1.1). The coverage of the non-retained item C is also 100%, since it is completely covered by B. The coverage of items A and E is 67% and 90% since they are covered by B and D, resp.

Note that, as explained at the end of in Section 3.2, the same architecture can be used to also solve the complementary problem where the goal is to retain the smallest set of items, such that the cover exceeds a given threshold.

5.2 Adaptation of Raw Data

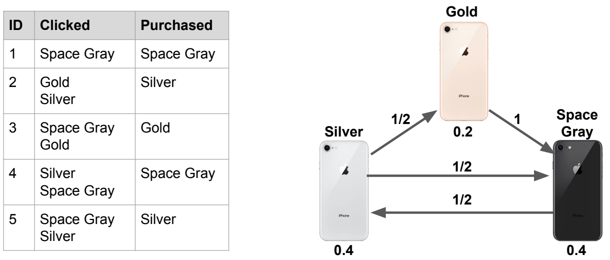
As mentioned in previous sections, adapting e-commerce data into our graph model is an essential phase, which is also incorporated in our architecture. E-commerce platforms collect tracking data from browsing sessions to learn consumer patterns and preferences to improve the shopping experience and increase revenue. The data is often stored in the format of a clickstream, consisting of the history of events performed by consumers during browsing, grouped by sessions. The information included in the clickstream usually contains the session id, date and time, search query, search page results, clicks, add-to-cart events, purchases and consumer related information, such as username, IP address, geolocation, etc. To ensure wide applicability, we assume that the available clickstreams include only minimal basic information: clicks and purchases grouped by sessions (which is true for most existing platforms and datasets [3])[4].

Graph construction process We now discuss the process of constructing a preference graph from the raw data. Recall that our model captures a session by identifying the "desired" item, which, if available, is the item the consumer would buy, and otherwise outgoing edges indicate her willingness to purchase concrete alternatives. Therefore, ideally, we would like to have for each session information identifying the desired item (for example, a search query specifying it explicitly), as well as a sufficient number of sessions where the specified item is not available, so as to accurately capture the suitability of alternatives, implied by the items purchased instead. However, in real life, when considering the main store which offers the product catalog in its entirety (which is the source for inferring the what-if probabilities necessary for curating the store with the limited inventory), it is overwhelmingly the case that all relevant items in a user session are in-stock. While this allows to identify the desired item as the one purchased[5], and derive an accurate estimation of each item's relative popularity, it is, nevertheless, harder to approximate user preferences pertaining to alternatives. In light of this limitation, we can use clicks on each item to estimate its suitability as an alternative to the purchased item. We note that assuming strong positive correlation between clicks and an intention to buy is a common practice employed by analysts in many e-commerce companies, when modeling consumer preferences[6], and it is also suggested in relevant studies [26, 32]. When viewing each click as an intention to buy (as an alternative), it is possible to overestimate this actual willingness to make a purchase, likely resulting in a diminished probability assigned to the event where no alternative is suitable. This can be

[4] One may also use semantic similarity between items to approximate edge weights, however we do not pursue this direction here.

[5] Sessions with no purchase, as all items are assumed to be available, are not driven by an intention to buy, and hence do not affect our modeling.

[6] Based on private conversations with analysts in multiple companies.



ID	Clicked	Purchased
1	Space Gray	Space Gray
2	Gold Silver	Silver
3	Space Gray Gold	Gold
4	Silver Space Gray	Space Gray
5	Space Gray Silver	Silver

a) Sample of sessions

b) Corresponding preference graph

Figure 3: Preference graph construction process

addressed by a more refined learning process where more variables are taken into account, subsequently normalizing the edge weights by a corrective factor. For example, by considering the amount of time spent viewing each item [32]. However, discussing such methods in detail is beyond the scope of this paper.

In view of the discussion above, we construct the preference graph, given clickstream data containing clicks and purchases per session, in the following way. The nodes in the graph correspond to the items. The node weights are assigned the percentage of purchases of an item out of total number of purchases. An edge from A to B exists only if the data contains a session where A was purchased, and B was clicked. The weight assigned to this edge is the fraction out of all sessions where A was purchased, in which B was also clicked.

Note that, it may seem at first logical to learn the edges in the opposite direction, i.e. to assign the weight of and edge from A to B based on sessions where A was clicked and later B was purchased, such that the edge direction “matches" the order of the operations. However, this does not fit the semantics of our model. Namely, we assume a “requested item" is bought with probability 1 when in stock, and an edge from A to B refers to sessions where A is a requested item which is out of stock, and B is an alternative. Since in most cases (and in the available data in particular) the items are in stock (examining only data where a desired item is out of stock will reduce the size of the relevant sessions to several thousands), clicking on A when available and then choosing another item implies that A is not the requested item. The other direction, which we opt for, is arguably more logical, given sessions where all relevant items are in stock, as the purchased item is almost certainly the most preferred item, and clicks on other items serve as considering these items as alternatives. Moreover, when estimating the weights of edges between A and B, we purposely avoid taking into account sessions where both A and B were clicked but neither was purchased. This is because the edges do not represent browsing probabilities (i.e. the probability B is clicked on next, when currently A is view, or vice versa), rather purchasing probabilities. Thus, our graph can intuitively be viewed as a transitive closure of a graph modeling “browsing" probabilities, with the purchased item viewed last (see discussion in Section 2). Observe that for items rarely clicked, the low number of corresponding sessions allows for more noise and the derived correlations to alternatives are less reliable. However, rarely clicked items have also (by definition) low weights, and hence these "noisier" items correspondingly have negligible influence over solution, as it focuses on more popular items.

How to choose the variant Finally, we explain when each of our two variants is a suitable choice given the data. We note that we focus in this work on these two models for edge dependencies, as in our inspection of clickstreams, we observed that in almost all cases one of the variants fits the data (in an approximated manner described next). Nevertheless, it is of course theoretically possible for other dependencies to exist, fitting neither of our models. We leave the adaptation of our techniques to such cases for future work, as here we focus on dependency schemes we have observed to be particularly prevalent.

The Normalized variant models the data well when at most one item, apart from the one purchased, was clicked. In practice it is unlikely any data perfectly adheres to this rule. However, when exceptions are rare we consider this to be a good approximation. In our experiments we applied the Normalized variant when in at least 90% of the sessions at most one alternative was clicked. In such datasets, when processing sessions where $t > 1$ other items were clicked, we “normalized” by counting each as a $\frac{1}{t}$-fraction of a click.

As for the Independent variant, it fits well when for any 2 alternatives (w.r.t. a given desired item) the fraction of sessions in which one was clicked remains the same when conditioned on clicking on the other (counting the fraction only out of sessions where the other was clicked). Once again, expecting any data to demonstrate such complete independence is not realistic. We, therefore, consider the Independent variant to be a reasonable approximation, when the following condition holds. We use a common measure for dependence of random variables, known as *Normalized mutual information* [31], which produces a value in $[0, 1]$, such that 1 indicates total dependence and 0 total independence. For any given item, we compute this measure for all pairs of alternatives, and take the average. Finally, we take the weighted average of these averages over all desired items, corresponding to the node weights (such that the average is not skewed by rarely purchased items)[7]. If this measure is below 0.1, analogously to the 90% cutoff in the Normalized variant, then we consider the Independent variant to be a a fitting choice of model.

To illustrate the process described above, consider Figure 3a, depicting a tiny sample of sessions taken from a real-life clickstream of users that purchased an iPhone 8 256GB. This smartphone comes in 3 different colors: Silver, Gold and Space Gray. The clickstream consists of these 3 items and 5 sessions, each ending in a purchase. The corresponding preference graph is depicted in Figure 3b. There are 2 purchases of the Space Gray iPhone, 2 of Silver and 1 of Gold. Hence, the node weights are 0.4, 0.4 and 0.2, resp. Out of the 2 times the Silver iPhone was purchased, each of the other 2 phones was clicked exactly once. Hence, the edge weights from Silver to Gold and Space Gray are $1/2$. Whereas, from the 2 sessions where Space Gray phone was purchased, one had no other clicks, and the other had 1 click on Silver. Hence, there is an edge from Space Gray to Silver with weight $1/2$. Finally, the Gold iPhone was purchased once, and in that session the Space Gray phone was clicked as well. Hence, there is a single edge of weight 1 from Gold to Space Gray. As for the problem variant, it is clear that the Normalized variant is a good fit, since no session implies more than one alternative.

Note that given a more detailed clickstream, an e-commerce platform can construct a more precise graph. For example, one can analyze the clickstream and combine with the information about out-of-stock items to find which items were purchased instead. Another idea for improvement is using the search query text and filter out items from the clickstream that are not matching the user’s intent. However, as mentioned before, such information is

[7] Of course, other thresholds and statistical distances can be applied just as well.

Table 2: The datasets used in the experiments

DS	Sessions	Purchases	Items	Edges
PE	10,782,918	10,782,918	1,921,701	9,250,131
PF	8,630,541	8,630,541	1,681,625	7,182,318
PM	8,154,160	8,154,160	1,396,674	5,826,429
YC	9,249,729	259,579	52,739	249,008

not always available in sufficient volume, hence it can be used as an enhancement on top of the proposed solution to adjust the weights. In general, more sophisticated data can be collected, with more resources invested in its analysis, resulting in a refined module for constructing the graph, which comes in place of our Data Adaptation Engine, with the rest of the architecture remaining the same. The methods we focus on here are chosen to fit actual information currently available to most e-commerce platforms.

5.3 Experimental Setup

We implemented our system using Python, and ran the experiments on a server with 128GB RAM and 32 cores. To evaluate our solution, we have performed a set of extensive experiments, both on a publicly available real-world dataset and on a bigger (private) dataset provided by a large e-commerce company[8]. We compare our approach to 4 baselines, using 4 evaluation methods.

Datasets. The first dataset, provided by a large e-commerce company, contains 5M items and 27M sessions, all ending with a single item purchase (we specifically requested such sessions). The dataset is private and comes as three independent parts, divided by domains - Electronics (PE), Fashion (PF) and Motors (PM). Due to its size, this dataset is particularly useful for scalability tests. The second dataset, marked as YC, is a public dataset, which was provided by the company YooChoose for the RecSys 2015 Challenge [3]. This dataset contains a clickstream with approximately 260K sessions ending in a single item purchase, covering a 6 month period in 2014 (from April 1st to September 30th). We have included this publicly available data, to allow the reader to reproduce the results.

The summary of these datasets is presented in Table 2. It includes the number of sessions, purchases, items and edges. Recall our discussion at the end of Section 5.2 regarding the conditions we set for each variant to fit a given dataset. It follows that the YC, PE and PF datasets fit the Independent variant, as in all three datasets our proposed independence measure is below 0.1. The PM dataset (whose items are parts and accessories for automobiles), however, is better captured by the Normalized variant, as in its sessions few alternatives were considered prior to purchasing. In particular, the percentage of sessions implying no more than a single alternative is above 90%.

Algorithms. We compare 5 different algorithms, over the same inputs (each input consists of a preference graph and a size bound k). The experiments are performed separately for both variants of the problem, and hence the following algorithms have in fact two versions, each with minor adaptations in accordance with the difference in the computation of the coverage function. We refer to the Normalized and Independent versions of the corresponding algorithm as NAME-**N** and NAME-**I**, resp. We next describe our selected algorithms.

- **Greedy** - Our proposed greedy algorithm (Algorithm 1).
- **BF** - A brute-force algorithm that evaluates all subsets of size k, and returns a set with the highest coverage. We use this baseline as it is the only one which guarantees the optimal solution, implying the exact approximation ratios achieved by our algorithm.
- **TopK-W** - An algorithm returning the top-k items by weight. This is the naïve baseline that considers each item individually without taking alternatives into account.
- **TopK-C** - An algorithm that returns the top-k items with the highest Coverage. This is a refined version of the previous baseline, which takes alternatives into account, however not from a global viewpoint as in our solution.
- **Random** - An algorithm that returns k items in a random manner. This is the simplest baseline.

Note that we did not include any SDP or LP based approximation algorithms, as they are not at all scalable (see discussion in section 3.2).

Evaluation Methods. We performed 4 complementary types of experiments to evaluate our end-to-end solution.

First, to examine the actual approximation factors Greedy attains in practice, we compare its coverage to that of BF, which is of course optimal. Greedy has a theoretical approximation guarantee in each variant, however it refers to the worst case, and in practice may achieve much better results. Moreover, we show in these experiments that approximation is necessary, as BF has impractical running times even on tiny inputs.

To show the quality of our algorithm in terms of the coverage it obtains on real-life high-scale data, we compare it to all other baselines, except BF, as it cannot scale to handle real-life data.

To demonstrate the scalability and parallelizability of our solution, in our third set of experiments we run it both on a single thread, varying the number of items (nodes), as well as on graphs of fixed size, varying the number of cores.

The fourth set of experiments was performed for the complementary minimization problem, where we set different thresholds and aim to find the smallest retained set whose cover exceeds the given threshold. We compare our adapted algorithm to analogous adaptations of the other algorithms, to demonstrate its effectiveness in terms of the size of the retained set.

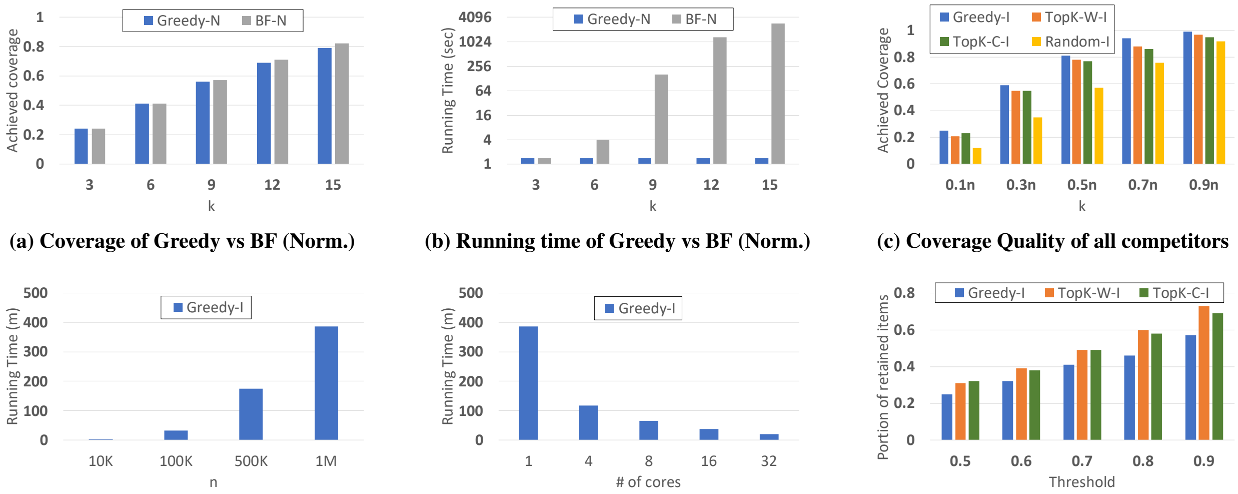
5.4 Evaluation Results

We present the results for the experiments detailed above.

Comparison to Brute Force. The brute-force algorithm does not scale to big graphs, since even for $n = 30$ and $k = 15$, there are $155M$ possible solutions. We show here a representative comparison of the algorithms on a subset of the YC dataset, reduced to 30 products (similar results were obtained for small subsets of all datasets). Figure 4a depicts the coverage achieved by Greedy compared to the optimal coverage achieved by BF. Figure 4b depicts (in log scale) the running times in seconds, over the Normalized variant (the Independent variant showed similar trends, hence omitted). We can see that the coverage of Greedy is very close to optimal, while having significantly better running times.

Coverage Quality. We compared all algorithms in terms of the achieved coverage quality, over all datasets. Figure 4c depicts the results on the YC dataset (Independent variant) for $k \in \{0.1n, 0.3n, ..., 0.9n\}$. The results over all other datasets (which include the Normalized variant) demonstrate a similar trend, hence omitted. BF cannot scale beyond small networks, and is excluded

[8] Company name omitted due to privacy considerations.



(a) Coverage of Greedy vs BF (Norm.)

(b) Running time of Greedy vs BF (Norm.)

(c) Coverage Quality of all competitors

(d) Scalability of Greedy

(e) Parallelizability of Greedy

(f) Complementary problem

Figure 4: Experimental results

from this experiment. As expected, Greedy is the top performing algorithm, while TopK-W and TopK-C lag behind, since they do not take into account, resp., alternatives, or overlaps in covers by different items. Random also achieves bad results (taking the best across 10 executions) as it makes no use of information pertaining to the popularity of items or their alternatives.

Scalability and Parallelizability. We performed the scalability tests on graphs of various sizes over all datasets. Note that we present only the running times of the algorithm, as the graph construction is considered to be an offline phase, hence not included in the measurements. Figure 4d depicts the running time of Greedy for $n \in \{10K, 100K, 500K, 1M\}$ and $k = 5K$, performed on subsets of the largest dataset (PE), while over other datasets similar trends were demonstrated, hence not shown here. The parallelizability analysis of Greedy, depicted in Figure 4e, was performed over the same dataset and an input graph of fixed size for varying the number of cores: 1, 4, 8, 16 and 32. The results show almost perfect parallelization, which scales well as the number of cores grows. For example, the execution of Greedy on 1 core compared to 32 cores runs 20x times faster.

Complementary problem. We conclude this section by evaluating our approach when adapted to the complementary minimization problem (as explained at the end of Section 3.2). The goal is to find the smallest set whose coverage exceeds a given threshold. Figure 4f depicts the results, in terms of the size of the produced set, obtained by our algorithm, when executed over the YC dataset (Independent variant), for thresholds $\in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, compared to the results obtained by TopK-W and TopK-C. These algorithms were also adapted to perform a binary search over a sorted list of nodes (by the relevant metric - weight or coverage, resp.), choosing the smallest prefix to exceed the threshold. The results demonstrate that the superiority of our approach carries over into this version of the problem, as it outperforms other baselines, producing a much smaller set. The results over the other datasets (and the Normalized variant) are similar.

6 RELATED WORK

E-commerce related problems attracted the interest of many researchers in recent years. Some extensively studied problems are product classification [9, 33], product ranking in search results [18] and automatic product content generation [10, 24]. Of such problems close to us in spirit are works on *diversity* [36], producing the top k most collectively dissimilar elements, typically out of a result set to a given search query. This relates to a similar concept in our problem, where we aim to avoid retaining items that match the same requests. The resemblance is even greater when elements are weighted by importance and there is a requirement, such as in [8], that each non-selected element is covered by a similar selected item, relating to our aim of choosing popular items and items covering as alternatives non-retained items. Nevertheless, there are many differences between their models and ours. Importantly, unlike in diversification problems, we do not aim to maximize diversity, rather it is a feature which is, to some extent, typical of good solutions to our problem, yet not at all necessary. Moreover, even when diversity is a constraint and the items are weighted [8], the goal is to maximize the total weight of the selection, in contrast to our model, where one must also (partially) count weights of adjacent items, which is a crucial property. Furthermore, as our edges represent choice probabilities (rather than item dissimilarity), we can support this original concept of covering neighboring items to a concrete and partial extent, which along with the aforementioned distinctions lead to vast differences in the algorithmic solutions and concrete computations.

Another line of work similar to ours is recommendations [28], as it also deals with selecting a subset of items to increase purchasing probability. However, there are important qualitative and quantitative differences. Primarily, recommendations are typically personalized w.r.t. a given user (and often a given product as well), and deal with a far smaller k. Some works on recommendations that derive product alternatives [20] may potentially serve as basis for another method of computing edge weights in preference graphs. We intend to investigate this approach in future work.

Other existing works in e-commerce that deal with finding top-k beneficial products to offer [35] focus on setting prices such that the predicted revenue (including costs) is maximized. However, in contrast to our models, they do not take into account consumers opting for alternatives.

Closest to our work is a subfield of Operations Research called Assortment Optimization. These works typically employ more complex models such as the Markov chain choice model [4, 23], multinomial logit model [29] and nested logit model [12]. The considered Markov chain model bears some resemblance to our Normalized variant, but is more complicated due to the consideration of varying item revenues and/or multiple-step graph paths (which are directly captured in our model by transitive edges). Consequently, their algorithms are also more complex and the work is geared towards theoretical analysis of the model rather than practical evaluation. The experiments, when exist, consider small scale item sets (order of 1000 items) [4], and the results are not scalable to big data. Furthermore, they mostly use synthetic datasets, and the process of model derivation from real-life data (which our end-to-end solution includes), is not considered there.

Our model is inspired by research in behavioral economics. In particular, [30] observed that consumers experience increased anxiety and are less likely to take action when faced with too wide of a selection. Additionally, [34] demonstrated that consumers, when searching for a specific item, are often willing to buy in its absence what they consider to be a reasonably satisfying alternative.

Our work draws on results in classical Top-k cover problems in graphs [13, 16]. Most notable is the Max Vertex Cover problem (VC_k) [11, 19], which was discussed in detail in Section 3. An existing direction in the study VC_k, whose practical adaptation to our setting would be an intriguing future work, is devising algorithms for graphs with bounded degree [13], as this special case arises in practice in our model. Similar problems include Max dominating set and Max edge domination [21]. All these problems can be viewed as special cases of the more abstract Maximum Coverage problem [6]. Moreover, each of these problems is strongly related to its more extensively researched variant, such as the Vertex Cover problem [25], where k is unspecified, and the goal is to find the smallest subset such that the entire graph is covered. Theoretical bounds and algorithms can often be adapted from one variant to the other, which is also the case for our problem as well, as discussed in Sections 3 and 4.

7 CONCLUSION

This paper introduces the *Preference Cover problem*, which aims to select a reduced inventory maximizing the likelihood of a purchase. We model consumer preferences via a *preference graph* and study two problem variants, *Normalized* and *Independent*, which differ in their interpretation of the probabilistic dependencies between the suitability of different alternatives. We study their approximation hardness, and since the overall number of items, and the bound on the retained set, tend to be very large in this context (in the order of magnitude of millions), we propose highly parallelizable and scalable algorithms that come with approximation guarantees. Finally, we present an end-to-end solution that maps real-world data into our model, and provide an extensive set of experiments on multiple datasets, demonstrating the efficiency and effectiveness of our approach.

In the problem setting studied here (which is common to intermediary e-commerce platforms [1]) the commission-per-purchase is considered fixed and the goal is to maximize the number of sales. Extending our work to support varying per-item revenues and storage considerations is an intriguing future work. Another interesting direction we are currently pursuing is incremental maintenance in response to changes over time.

Acknowledgements This work has been partially funded by the Israel Innovation Authority, the Israel Science Foundation, the Binational US-Israel Science foundation, Len Blavatnik and the Blavatnik Family foundation.

REFERENCES

[1] Marketplace Pricing Model. https://marketplace.webkul.com/marketplace-pricing-model-subscription-vs-commission/.

[2] A. A. Ageev and M. I. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. *IPCO*, 1999.

[3] D. Ben-Shimon, A. Tsikinovsky, M. Friedmann, B. Shapira, L. Rokach, and J. Hoerle. Recsys challenge 2015 and the yoochoose dataset.

[4] J. Blanchet, G. Gallego, and V. Goyal. A markov chain approximation to choice modeling. *Operations Research*, 64(4):886–905, 2016.

[5] M. Cao, Q. Zhang, and J. Seydel. B2c e-commerce web site quality: an empirical examination. *IMDS*, 105(5):645–661, 2005.

[6] R. Cohen and L. Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008.

[7] A. C. Doherty, P. A. Parrilo, and F. M. Spedalieri. Complete family of separability criteria. *Physical Review A*, 69(2):022308, 2004.

[8] M. Drosou and E. Pitoura. Multiple radii disc diversity: Result diversification based on dissimilarity and coverage. *TODS*, 2015.

[9] E. Dushkin, S. Gershtein, T. Milo, and S. Novgorodov. Query driven data labeling with experts: Why pay twice? In *EDBT*, 2019.

[10] G. Elad, I. Guy, S. Novgorodov, B. Kimelfeld, and K. Radinsky. Learning to generate personalized product descriptions. In *Proc. of CIKM*, 2019.

[11] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41(2):174–211, 2001.

[12] J. B. Feldman and H. Topaloglu. Capacity constraints across nests in assortment optimization under the nested logit model. *Operations Research*, 63(4):812–822, 2015.

[13] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[15] S. Gershtein, T. Milo, and S. Novgorodov. Reduce-comm: Effective inventory reduction system for e-commerce. In *Proc. of CIKM*, 2019.

[16] D. S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

[17] G. Jäger and A. Srivastav. Improved approximation algorithms for maximum graph partitioning problems. *Journal of combinatorial optimization*, 10(2):133–167, 2005.

[18] S. K. Karmaker Santu, P. Sondhi, and C. Zhai. On application of learning to rank for e-commerce search. In *SIGIR*, 2017.

[19] P. Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. In *SOSA 2019*.

[20] J. McAuley, R. Pandey, and J. Leskovec. Inferring networks of substitutable and complementary products. In *KDD*, 2015.

[21] E. Miyano and H. Ono. Maximum domination problem. In *The Australasian Theory Symposium*, pages 55–62, 2011.

[22] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, Dec 1978.

[23] K. Nip, Z. Wang, and Z. Wang. Assortment optimization under a single transition model. 2017.

[24] S. Novgorodov, I. Guy, G. Elad, and K. Radinsky. Generating product descriptions from user reviews. In *Proc. of WWW*, 2019.

[25] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *JCSS*, 43(3):425–440, 1991.

[26] C. Park, D. H. Kim, M. Yang, J. Lee, and H. Yu. Your click knows it: Predicting user purchase through improved user-item pairwise relationship. *CoRR*, abs/1706.06716, 2017.

[27] E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133–157, 1994.

[28] L. Qi, X. Xu, X. Zhang, W. Dou, C. Hu, Y. Zhou, and J. Yu. Structural balance theory-based e-commerce recommendation over big rating data. *IEEE Transactions on Big Data*, 4(3):301–312, 2018.

[29] P. Rusmevichientong, D. Shmoys, and H. Topaloglu. Assortment optimization with mixtures of logits. Technical report, 2010.

[30] B. Schwartz. The paradox of choice: Why more is less. 2004.

[31] A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *JMLR*, 3(Dec):583–617, 2002.

[32] Q. Su and L. Chen. A method for discovering clusters of e-commerce interest patterns using click-stream data. *ECRA*, 14(1):1–13, 2015.

[33] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 7(13):1529–1540, 2014.

[34] W. Verbeke, P. Farris, and R. Thurik. Consumer response to the preferred brand out-of-stock situation. *EJM*, 32(11/12):1008–1028, 1998.

[35] Q. Wan, R. C.-W. Wong, and Y. Peng. Finding top-k profitable products. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1055–1066. IEEE, 2011.

[36] Y. Wang, A. Meliou, and G. Miklau. Rc-index: Diversifying answers to range queries. *PVLDB*, 11(7):773–786, 2018.

[37] H. Wolkowicz. *Simple efficient solutions for semidefinite programming*. 2001.