Recommending Customizable Products: A Multiple Choice Knapsack Solution

Aravind Sivaramakrishnan, Madhusudhan Krishnamachari, Vidhya Balasubramanian
Department of Computer Science & Engineering*, Amrita School of Engineering,
Amrita Vishwa Vidyapeetham (University), Coimbatore, India

ABSTRACT

Recommender systems have become very prominent over the past decade. Methods such as collaborative filtering and knowledge based recommender systems have been developed extensively for non-customizable products. However, as manufacturers today are moving towards customizable products to satisfy customers, the need of the hour is customizable product recommender systems. Such systems must be able to capture customer preferences and provide recommendations that are both diverse and novel. This paper proposes an approach to building a recommender system that can be adapted to customizable products such as desktop computers and home theater systems. The Customizable Product Recommendation problem is modeled as a special case of the Multiple Choice Knapsack Problem, and an algorithm is proposed to generate desirable product recommendations in real-time. The performance of the proposed system is then evaluated.

CCS Concepts

•**Information systems** → **Content ranking; Recommender systems;** *Personalization;* E-commerce infrastructure;

Keywords

Recommender System, Customization, Customizable Products

1. INTRODUCTION

In today's ever-growing competitive global marketplace, manufacturers are moving away from non-customizable products to customizable products in order to fulfill consumer requirements. Product customization is becoming more prevalent in a large range of products, from laptops and mobile phones to clothing and footwear. Such customization options have left the customers with a varied amount of choices. However, an abundance of choice can be confusing to the customer and may hinder the decision making process. To mitigate this, recommendation systems that help customers select the right customized product is the need of the hour.

The problem with existing recommendation systems is that they have been developed extensively only for non-customizable products, where the product specifications are already fixed by the manufacturers. In the case of customizable products, the user has the option to choose the individual features of the product. However, the user might not have enough domain knowledge or experience and thus, may not be able to provide a complete set of feature specifications. In such cases, the system must be able to provide novel product suggestions based on user requirements. It is desirable to adapt existing recommender systems to operate on customizable products.

There are several challenges in designing such a recommender system. Consider for example, the case where the customer provides a set of feature specifications for a BMW 7 Series, and a cost threshold. These specifications are in the form of constraints on the final product. In some cases, the user might not be able to provide a complete set of constraints for all the features of a product. In such cases, when some of the features are left to the discretion of the system, the system must still be able to provide suggestions based on the specified constraints. It must also be able to recommend products without any prior knowledge about the user. Since the range of options is huge, approaches based on gathering purchase data of users might not be suitable for this problem. Therefore the recommender system must primarily be based on the features of a product. In addition, such a recommendation system for customizable products must be able to provide suggestions to a user, in situ.

One of the challenges of this problem lies in representing the feature specifications and cost threshold accurately. In the example discussed previously, the number of possible variations in the BMW 7 Series could reach 10^{17}[19]. Thus, providing good recommendations subject to the given constraints may prove to be difficult mainly due to the sheer size of the search space. When such a system is deployed for real-time commercial use, it must be able to provide quick recommendations that are both effective and efficient. Care must be taken to provide recommendations that do not overspend on a particular feature. Thus, the recommendation system problem boils down to an optimization problem

*This work was done as part of the research activities at the Amrita Multi-Dimensional Data Analytics Lab.

WIMS '15 July 13 - 15, 2015, Larnaca, Cyprus

ACM 978-1-4503-3293-4/15/07 ...$15.00.
http://dx.doi.org/10.1145/2797115.2797116

as follows: given a set of feature specifications, each with corresponding cost and selection reward, the system must determine which instance of each feature to include in the product so that the total cost is within the threshold, and the total reward is as high as possible.

In this paper, we propose a suitable representation for the feature specifications and cost threshold. The problem of finding the best possible customization is modeled as a special case of the Multiple Choice Knapsack Problem (MCKP), which is an *NP-Hard* problem. Since we are primarily interested in a set of recommendations, i.e., the candidate solutions, they can be identified in polynomial time using a greedy algorithm [4].

The main contributions of this paper are as follows.

- A formal definition of the Customizable Product Recommendation problem as an optimization problem.
- Modelling the Customizable Product Recommendation problem as a special case of the Multiple Choice Knapsack Problem (MCKP).
- Results of applying a greedy algorithm for identifying solutions to the Customizable Product Recommendation Problem.

The rest of this paper is structured as follows. In Section 2, we survey the available literature on various state-of-the-art recommender system approaches. In Section 3, we formally define the Customizable product recommendation problem and model it as a special case of the MCKP. In Section 4, we describe a greedy algorithm to identify candidate solutions to the MCKP. We apply the algorithm to the case of customizable desktop computers and customizable home theater systems. The results are discussed in detail in Section 5, and avenues for future work are proposed in Section 6.

2. RELATED WORK

The problem of finding and suggesting similar products to users has attracted a lot of attention in the field of Information Retrieval [3] in recent years. Recommender systems have been developed for a variety of applications. Collaborative filtering [15] is a commonplace technique used by E-commerce websites like Amazon to suggest similar products to the user. Under collaborative filtering, users are recommended items that people with similar tastes and preferences have liked in the past. Other E-commerce services that suggest movies to the user employ content-based recommender systems[9], which keep track of the movies a user browses, builds a user profile and suggests similar movies to the user. Algorithms such as k-nearest neighbors[2] and the Pearson correlation have been used for the measurement of similarity scores.

In recent years, hybrid recommender systems[1] have been widely used by websites such as Netflix wherein a collaborative filtering recommender system and a content-based system work in tandem. Knowledge based recommender systems [16] have been developed for scenarios where collaborative filtering and content-based filtering cannot be applied, or when sufficient information has not been gathered for such systems to operate effectively. They rely on knowledge models of the product domain in order to provide effective recommendations. Various deep learning techniques[17] have also been proposed for providing effective product recommendations to users. However, most of these approaches focus primarily on user behavior, but not on product properties, which must also play a major role in the recommendations.

The above mentioned approaches focus primarily on off-the-shelf products. However, approaches for recommending customizable products (along with their customizable components) have not gained much prominence. One of the few solutions that address this problem is by Moon et al who proposed an agent-based recommender system [11] that recommends products after learning customer preferences through a market-based learning mechanism. A mass customization recommendation system for the automotive industry was proposed by Mavridou et al [10], which mines user preferences from feedback and survey data filled out by people who have already purchased cars. Stormer proposed an approach [14] that applies collaborative filtering to calculate the most common product options and propose them as recommendations for customers. More recently, Wang et al [18] proposed a Customizable Product Recommendation system that builds a probabilistic model based on whether users bought a particular item given a set of specifications.

Such systems are a step forward in providing customizable product recommendations to the user, but they generally require a large amount of user preference data before they can provide suggestions[12]. In the case of a customizable product, each user might be having varying requirements in the form of feature specifications. Hence, a recommendation system for customization must be able to provide suggestions using only feature specifications from the user. Designing a knowledge-based recommender system for customizable products would require the recommendation knowledge to be well-defined in an explicit fashion, which is an arduous task[8]. A robust system must also be able to use the same knowledge representation across different product domains.

Our goal is to propose a greedy approach for providing recommendations for customizable products, using primarily the feature properties of the product, and the initial user specifications. Subsequent user feedback, such as in the case of systems discussed above, are beyond the scope of this paper. Although a knapsack optimization based approach to recommender systems has been proposed for non-customizable products[5], the same strategy has not yet been investigated for the Customizable Product Recommendation problem. Our approach models the Customizable Product Recommendation problem as a special case of the Multiple Choice Knapsack Problem (MCKP), first proposed by Sinha and Zoltners [13].

3. PROBLEM FORMULATION

In order to solve the Customizable Product Recommendation problem, it is necessary to formally define the problem. In this section, we first formally define the Customiz-

able Product Recommendation problem and then proceed to model it as a special case of the Multiple Choice Knapsack Problem (MCKP).

3.1 Formal Definition of the Customizable Product Recommendation Problem

Let a customizable product X be defined by a set of n features, $\{N_1, N_2, ..., N_n\}$. Each feature N_i consists of m models $\{x_{i1}, x_{i2}, ...x_{im}\}$. The value of m for each N_i may vary. Each feature example x_{ij} has a corresponding cost w_{ij}. The user may provide values from $\{x_{i1}, x_{i2}, ...x_{im}\}$ for a particular feature N_i, or he may set $N_i = 0$, indicating that no preference is specified. Thus, each feature example N_{ij} will have a desirability value p_{ij} which is dependent on the user selection for N_i and the value of x_{ij}. The user should also specify a cost threshold W for the recommended product. The Customizable Product Recommendation problem can be defined as follows.

Given a minimum user requirement specification for n features $\{N_1, N_2, ..., N_n\}$, the system must return a candidate set of solutions S, where each candidate solution $Y \in S$ is specified by $\{y_1, y_2, ..., y_n\}$. The top c values of $\sum_{i=1}^{n} p_{ij}, j \in y_i$ are chosen for every $Y \in S$ subject to the following constraints:

$$(i) \sum_{i=1}^{n} w_{ij} \leq W, j \in y_i \tag{1}$$

$$(ii) w_{ij} \leq \epsilon (1 \leq i \leq m, j \in y_i) \tag{2}$$

The recommended product may have a higher configuration than the user specification, but care must be taken to avoid the recommendation of lower configurations. For example, in the case of a recommender system for customizable laptops, if the user has specified requirement of 4GB of RAM, the system must be able to prune all values of RAM that are less than 4GB. However, the system may recommend 6GB of RAM to the user, as long as it is within the user's cost specification, and all other requirements have been met. The constraint specified by (1) ensures that the recommended product does not exceed the cost threshold specified by the user. Constraint (2) ensures that the system does not overspend on a particular feature.

Thus, the customizable product recommender system has to select one feature example for each feature, such that the total cost of all feature examples does not exceed the cost threshold. In addition, the feature examples selected must meet the minimum user requirements. The recommendation problem thus defined is a constrained optimization problem and can be mapped directly to a special case of the Knapsack Problem, i.e., the Multiple Choice Knapsack Problem (MCKP), which is described below.

3.2 MCKP Definition

Consider a collection of items subdivided into n classes. Each set of items belonging to class i is denoted by N_i. Each item $x_{ij} \in N_i$ has an associated profit p_{ij} and a selection cost w_{ij}. The objective is to pick exactly one item from each class, with maximal total profit, while obeying that the total cost of the chosen items must not exceed W. Let s_{ij} be a binary value that denotes the selection of an item x_{ij}. Then, the MCKP can be defined as

$$maximize \sum_{i=1}^{n} \sum_{j \in N_i} p_{ij} s_{ij} \tag{3}$$

subject to

$$\sum_{i=1}^{n} \sum_{j \in N_i} w_{ij} s_{ij} \leq W,$$
$$\sum_{j \in N_i} s_{ij} = 1, \text{ for all } 1 \leq i \leq n$$
$$s_{ij} \in \{0, 1\}, \text{ for all } 1 \leq i \leq n \text{ and } x_{ij} \in N_i$$

Thus, if a particular product is being defined by n features, with each feature example $x_{ij} \in N_i$ having a selection cost of w_{ij} and a selection reward p_{ij}, it is easy to model the Customizable Product Recommendation problem as a special case of the MCKP. The cost of each feature example w_{ij} is the price of the component. W is the cost threshold, as specified by the user. The profit for each feature example p_{ij} can be mapped to the component's desirability value as explained below.

3.3 Assigning Desirability Values

All feature examples within a class N_i are initially sorted in increasing order of component quality. For example, in the case of desktop computer configuration, a sorted feature array for the RAM feature would be {2GB, 4GB, 6GB,...}.

In the case of quantifiable features such as RAM or Monitor size, the desirability value p_{ij} can be defined as follows. When the user specification for N_i is x_{ij}, the base desirability value is defined as $p_{ij} = 1$.

Let v_{ij} be a function that returns the value of the feature example x_{ij}. For example, $v(2\text{GB}) = 2$, $v(6\text{GB}) = 6$, and so on. Every selection x_{ik}, where $v_{ik} \geq v_{ij}$ must hence be proportionally rewarded. Let x_{im} be the final feature example in N_i. First, we define unit value of a feature example $\tilde{v}_{ik}$ as

$$\tilde{v}_{ik} = \frac{v_{ik}}{v_{ij}} \tag{4}$$

Then the desirability value p_{ik} can be defined as

$$p_{ik} = \left(1 - \frac{\tilde{v}_{ik}}{w_{ik}}\right) \times \tilde{v}_{ik} \tag{5}$$

In the case of non-quantifiable features, such as Operating System in the case of desktop computers, the desirability value p_{ij} for every item can be defined as follows. When the user specification for N_i is x_{ij}, the base desirability value $p_{ij} = 1$. The desirability value of x_{ik} is denoted by p_{ik} and can be defined as

$$p_{ik} = k - j + 1 \tag{6}$$

The cost constraint specified by (1) is modeled by the cost constraint of the MCKP. Also, since one feature example has to be chosen from each feature, overspending will not occur, and thus, the constraint specified by (2) is upheld as well.

As the MCKP is an NP-Hard problem, recommending a single product based on the user specifications cannot be achieved in polynomial time. However, as a set of candidate solutions needs to be provided as recommendations, we make use of a greedy algorithm that generates these candidate solutions in polynomial time. A greedy approach is preferable as the recommended products will have maximum desirability value for the given cost. In the next section, we describe in detail the greedy algorithm used to solve the customized product recommendation problem.

4. PROPOSED ALGORITHM

In this section, we use a classic greedy approach proposed for the MCKP[7] to solve the Customizable Product Recommendation problem. The described greedy algorithm solves the MCKP by reducing it into an instance of the 0-1 knapsack problem (KP), and then selecting a single item from each class. We also propose different approaches for generating multiple recommendations.

4.1 Algorithm Description

Initially, feature examples with $p_{ij} < 1$ are removed from the search space to ensure that configurations lower than the user specifications are not returned by the recommender system. Next, LP-Dominated items are removed from each class N_i. An item x_{ia} with cost w_{ia} and profit p_{ia} is said to LP-Dominate x_{ib} with cost w_{ib} and p_{ib} in a class N_i iff $w_{ib} > w_{ia}$, and $p_{ib} < p_{ia}$. Item x_{ib} is said to be LP-Dominated[7]. Once the LP-Dominated items are removed from N_i, the subset of N_i obtained is denoted by R_i. The LP-Bound solution to the MCKP can be found by using the algorithm outlined in Algorithm 1.

1. For each class N_i, remove all items with $p_{ij} < 1$.
2. For each class N_i, sort the items according to increasing costs and derive R_i. The size of R_i is r_i.
3. Construct an instance of a KP by setting $\tilde{p_{ij}} := p_{ij}$ - $p_{i,j-1}$, $\tilde{w_{ij}} := w_{ij}$ - $w_{i,j-1}$ for each class R_i and j = 2,...,r_i.
4. Sort the items according to decreasing incremental efficiencies $\tilde{e_{ij}} := \tilde{p_{ij}} \ / \ \tilde{w_{ij}}$ and store them in e.
5. Use Algorithm Greedy-Replace (Algorithm 2) to return the candidate solutions.

Algorithm 1: MCKP-Greedy

Initialize the knapsack with $x_{i1} \forall N_i$;
Set *iter* = 1;
Return knapsack and Continue;
while *iter* $\leq$ *length(e)* **do**
  Replace x_{ij} with $e[iter]$, where $e[iter] \in N_i$;
  if *cost of new knapsack* $\leq W$ **then**
    Set *iter* = *iter* + 1;
    Return new knapsack and Continue;
  else
    Stop;
  end
end

Algorithm 2: Greedy-Replace

We shall illustrate the working of the described algorithm with an example. Consider a product defined by three features $\{N_1, N_2, N_3\}$. The desirability values have been allocated for each component as per the user specifications as shown in Table 1. The cost threshold provided by the user is $W = 60$.

i	j	w_{ij}	p_{ij}
1	1	5	1
	2	10	2
	3	25	3
	4	70	3
	5	65	4
2	1	10	1
	2	40	1
	3	35	2
3	1	10	1
	2	20	2
	3	95	2
	4	40	3
	5	85	4

Table 1: Instance of MCKP

After removal of the LP-dominated items and renumbering the remaining items, we have $R_1 = \{1,2,3,4\}$, $R_2 = \{1,2\}$, $R_3 = \{1,2,3,4\}$ as given in Table 2. The incremental efficiencies

i	j	w_{ij}	p_{ij}
1	1	5	1
	2	10	2
	3	25	3
	4	65	4
2	1	10	1
	2	35	2
3	1	10	1
	2	20	2
	3	40	3
	4	85	4

Table 2: Instance of MCKP after removal of LP-Dominated items

in sorted order can be given as $e = \{e_{12}=\frac{1}{5}, e_{32} = \frac{1}{10}, e_{13} = \frac{1}{15}, e_{33} = \frac{1}{20}, e_{22} = \frac{1}{25}, e_{14} = \frac{1}{40}, e_{34} = \frac{1}{45}\}$. Choosing the smallest costs in each class, the capacity of the knapsack $w = w_{11} + w_{21} + w_{31} = 25$. The Greedy-Replace algorithm first chooses component (1,2), increasing the capacity to 30, then it chooses component (3,2) getting $w = 40$, component (1,3) with $w = 55$. When component (3,3) is chosen, the capacity of the knapsack exceeds W, and hence the optimal solution becomes $\{x_{13}, x_{21}, x_{32}\}$. The total desirability value of the solution is $z = 3 + 1 + 2 = 6$. Other candidate solutions for the knapsack can be obtained by returning the intermediate solutions. In this particular example, $\{x_{12}, x_{21}, x_{32}\}$ is also a candidate solution with a total desirability value $z = 5$.

The cost of removal of LP-Dominated items from a single class N_i is $O(n \log n)$. After the Knapsack instance is constructed, the cost of sorting the items according to their incremental efficiencies is also $O(n \log n)$. Sorting the candidate solutions also takes $O(n \log n)$ time. Thus, the worst

case running time of the above algorithm can be described as $O(\sum_{N_i}(n_i \log n_i) + n \log n)$.

The Greedy-Replace approach described in Algorithm 2 is guaranteed to find a solution with a high desirability value per unit cost in $O(n)$ time, but it might not generate enough solution choices. In the above example, only a set of 4 recommendations are returned by Greedy-Replace. A good recommender system must be able to produce a diverse set of results for the user to choose from. Also, these results must be returned in a short amount of time. Hence we design solutions for generating a candidate set of recommendation. There are several approaches that can be used for the selection of candidate solutions, and they are discussed below.

Initialize the knapsack with $x_{i1} \forall N_i$;
set $iter = 1$;
Return knapsack and continue;
while *iter* $\leq$ *length(e)* **do**
  Replace x_{ij},$x_{i+1,j}$,...$x_{i+n-1,j}$;
  if *cost of new knapsack* $\leq W$ **then**
    Set $iter = iter + 1$;
    Return new knapsack;
    Initialize the knapsack with $x_{i1} \forall N_i$;
  else
    Stop;
  end
end

Algorithm 3: n-Swap

Initialize the knapsack with $x_{i1} \forall N_i$;
set $iter = 1$;
Return knapsack and continue;
while *iter* $\leq$ *length(e)* **do**
  Replace x_{ij} with $e[iter]$, where $e[iter] \in N_i$;
  if *cost of new knapsack* $\leq W$ **then**
    Set $iter = iter + 1$;
    Return new knapsack and continue;
  else
    Remove e[0];
    Initialize the knapsack with $x_{i1} \forall N_i$;
    Set $iter = 1$;
  end
end

Algorithm 4: 1-Pop

4.2 Alternative Selection Approaches

The most obvious way to generate multiple recommendations is by selecting components from e, and swapping them with the components present in the initial knapsack if the new cost does not exceed the threshold. Therefore, the first approach that we propose is the n-Swap algorithm, outlined in Algorithm 3. In this approach, a single instance of the knapsack is generated after replacing the components in the knapsack with n components from the incremental efficiencies list e. Then, the initial knapsack is restored and the process is repeated until all components in e are exhausted. The top c candidate solutions are then returned by removing duplicate solutions and sorting the candidate solutions in decreasing order of profits per unit value.

In the above example, the n-Swap algorithm selects n components from e such that each component belongs to a different class and swaps them with the components present in the initial state of the knapsack $\{x_{11}, x_{21}, x_{31}\}$. For example, 2-Swap may select components (3,2) and (2,2) from e, and swap them with (3,1) and (2,1) in the initial knapsack to obtain a candidate solution $\{x_{11}, x_{22}, x_{32}$ with a total cost $c = 5 + 35 + 20 = 60$ and a total desirability value $z = 5$. For a given value of k, k-Swap has a worst-case complexity of $O(n^k)$.

For a given value of n, 1-Swap is performed, followed by 2-Swap, and so on, until n-Swap. The value of n cannot exceed the total number of features. For high values of n, n-Swap may be computationally expensive. The desirability value of the recommendations may also not be comparable to Greedy-Replace. Therefore, we propose an approach, 1-Pop, described in Algorithm 4, that iterates Greedy-Replace over different versions of e. In this approach, after the knapsack obtained through Greedy-Replace has exceeded the cost threshold, the first item from the incremental efficiencies list e is popped, and Greedy-Replace is repeated. This process is repeated until all items in e is exhausted. The top c candidate solutions are then returned by removing duplicate solutions and sorting the candidate solutions in decreasing order of profits per unit value.

In the above example, the 1-Pop algorithm runs a single iteration of Greedy-Approach, and then pops e_{12} from e. Then it chooses component (3,2), with a capacity of 35. When component (1,3) is chosen, the capacity of the knapsack increases to 55. In the next iteration, the capacity of the knapsack exceeds W, and hence, the intermediate solutions are returned, and e_{32} is popped from e. This process is repeated until e is empty. As 1-Pop involves n iterations of Greedy-Replace, the worst case complexity of 1-Pop is $O(n^2)$.

5. EXPERIMENTAL RESULTS

In this section, we evaluate our approach for the Customizable Product Recommendation problem for two types of customizable products: desktop computers and home theater systems, so as to qualify the adaptivity of the system.

5.1 Experimental Setup

We first describe the data that was used to obtain the experimental results. The indexing of the various feature components within the dataset is imperative to obtain the correct results, and is discussed in Section 5.1.1. Then, we outline the architecture of the system used as well as a detailed description of the same.

5.1.1 Dataset

The dataset for both desktop computers and home theater systems were sourced from multiple e-commerce websites. For desktop computers, the features chosen were: Hard Disk capacity, Graphics Card memory, Monitor size, Operating System, Processor and RAM. Out of these features, Hard Disk capacity, Graphics Card memory, Monitor Size and

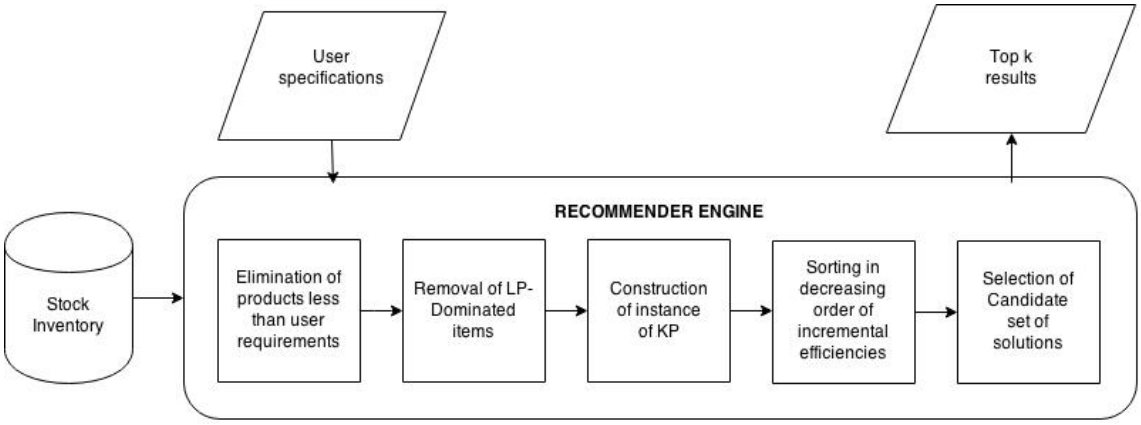


Figure 1: System architecture

Feature	Capacity	Instance	Price (in $)	Desirability value
RAM	4GB	Corsair Vengeance DDR3	46.99	0.9997
	4GB	Transcend DDR3-1333 DDR3	36.99	0.9996
	8GB	Corsair Vengeance DDR3	90.49	1.9993
	8GB	Kingston HyperX DDR3	88.49	1.9993
	16GB	G.Skill Ripjaws X DDR3	218.49	3.9988
Operating System	Linux/DOS	DOS	0	0.0000
	Windows 7	Microsoft Windows 7 Home Premium	89.99	1.0000
	Windows 7	Microsoft Windows 7 Professional	128.49	1.0000
	Windows 8.1	Microsoft Windows 8.1 Professional	160.99	2.0000
Hard Disk Capacity	250GB	Seagate	51.49	0.4999
	500GB	WD Caviar Blue	57.99	0.9997
	500GB	HP SATA	52.99	0.9997
	1TB	Toshiba	59.49	1.9989
	2TB	WD Caviar Green	93.99	3.9972

Table 3: Sample dataset for desktop computers

RAM are quantifiable features, and hence, Equation 5 can be used to calculate their respective desirability values. Operating System and Processor cannot be quantified, and hence, their desirability values are naively calculated using Equation 6. For home theater systems, the feature set consisted of: Television screen size, Blu-Ray player, Receiver, Universal Remote and Speakers. Television screen size and Receivers are the only quantifiable features in the case of home theatre systems.

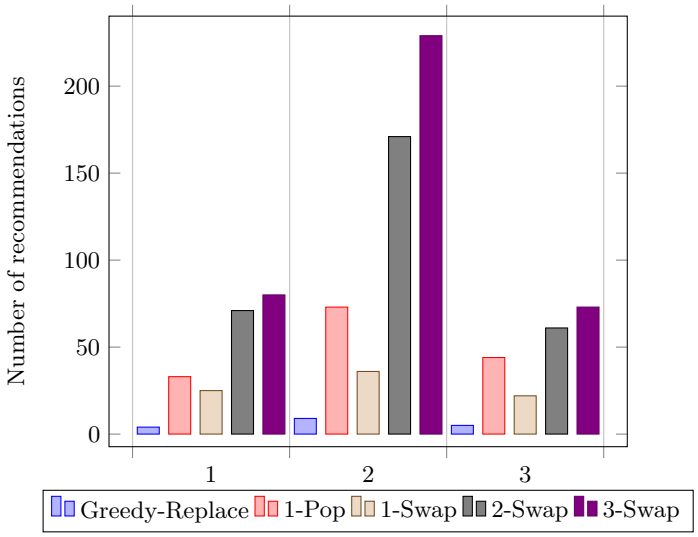
In both cases, the feature examples are ordered in increasing order of component quality. For example, in the case of RAM for desktop computers, 4GB examples are followed by 8GB examples, and so on. In cases where the "quality" of a component is ambiguous, the feature examples are ordered in increasing order of cost, as general user perception would associate a high cost in a particular product with a high product quality as well. For example, in the case of Operating systems for desktop computers, the "quality" of a Linux OS, which is free, and a Windows 8.1 operating system, which lies in the $120-180 range, is ambiguous. Hence, Linux OS is placed at a lower index than Windows 8.1. A sample of the dataset for desktop computers is provided in Table 3. The indicated desirability values are for a user selection of 4GB RAM, Windows 7 OS, and 500 GB Hard Disk Capacity.

5.1.2 System Architecture

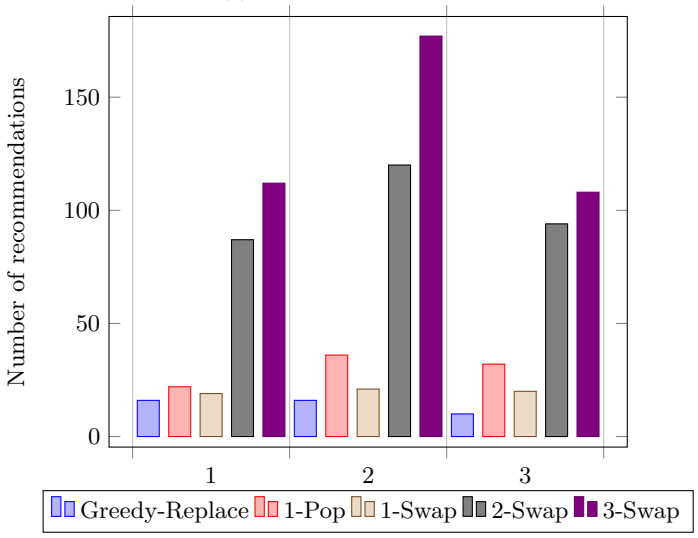
Given the dataset we now discuss the architecture of the proposed recommendation system. The overall system architecture is shown in Figure 1. The dataset is stored offline in the form of a stock inventory. The user gives a set of feature specifications to the recommender engine, which then applies the MCKP-Greedy algorithm to generate a set of candidate solutions. The top c candidate solutions from this set are returned to the user as recommendations for the provided feature specifications. The selection of the top c candidate solutions depends on the method used for replacing the items in the current state of the knapsack with the items present in the incremental efficiencies list e. The candidate results are displayed to the user, as a set of recommendations. Using this system, we evaluate the proposed methodology. We first compare the selection approaches so as to identify the best approach for identifying the final candidate list. The best approach is chosen, and the results returned by that approach are evaluated using standard metrics as described in in Section 5.3.

5.2 Comparative study of selection approaches

In order to measure the performance of the system, it is necessary to identify the best approach for identifying candidate solutions. We evaluate the following selection approaches: Greedy-Replace, 1-Pop, 1-Swap, 2-Swap and 3-Swap using

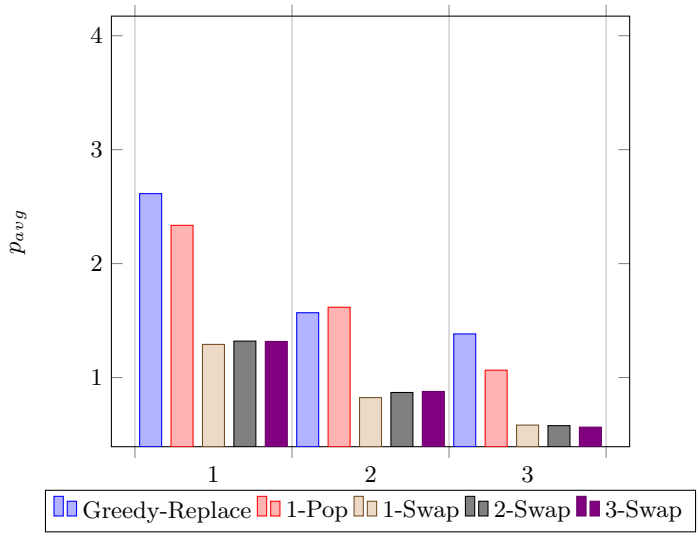


(a) Desktop computers

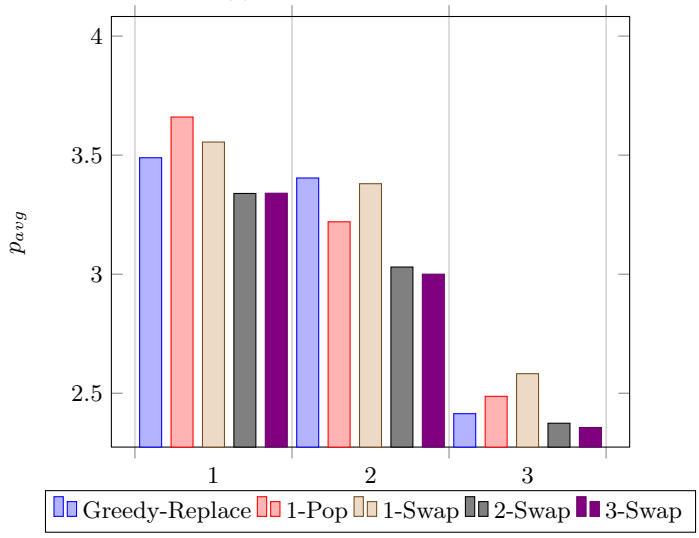


(b) Home theater systems

Figure 2: Number of recommendations



(a) Desktop computers



(b) Home theater systems

Figure 3: Average desirability value per unit cost

the following metrics: number of recommendations, desirability value per unit cost and time complexity.

5.2.1 *Input case description*

The experimental results discussed in the remainder of this section were obtained on a computer with the following configurations: Intel(R) Core(TM) i5-2450M having clock speed of 2.50GHz CPU with 5.90GB of usable RAM. For both desktop computers and home theater systems, three cases of user input were taken to evaluate the various selection approaches. The different user requirements specifications for desktop computers are:

- Case 1: 500 GB Hard Disk capacity, 1GB Graphics memory, 24 inch Monitor, Windows 7 OS, Intel i5 Processor, 4GB RAM (cost threshold: $ 650)
- Case 2: 1TB Hard Disk capacity, 2GB Graphics memory, 15.6 inch Monitor, Windows 8 OS Intel i7 Processor, 8GB RAM (cost threshold: $ 885)
- Case 3: 2TB Hard Disk capacity, 3GB Graphics memory, 24 inch Monitor, Linux/DOS, Intel i7 Processor, 16GB RAM (cost threshold: $ 1210)

In the case of home theater systems, no user specification was provided for the universal remote feature. The different user requirements specifications are:

- Case 1: 40 inch TV, Blu-Ray player with WiFi, 5 channel Receiver, Bookshelf speakers (2 pair) (Cost threshold: $ 2500)
- Case 2: 48 inch TV, Blu-Ray player (Universal), 7 channel Receiver, Sound Bar (Cost threshold: $ 4500)

Method	User Input 1	User Input 2	User Input 3
Greedy-Replace	3.01	1.803	1.512
1-Pop	3.01	1.803	1.512
1-Swap	2.206	1.658	1.112
2-Swap	2.206	1.658	1.112
3-Swap	2.206	1.658	1.112

(a) Desktop computers

Method	User Input 1	User Input 2	User Input 3
Greedy-Replace	5.143	4.722	2.883
1-Pop	5.143	4.722	2.883
1-Swap	5.143	4.722	2.883
2-Swap	5.152	4.722	2.883
3-Swap	5.152	4.722	2.883

(b) Home theater systems

Figure 4: Maximum desirability value per unit cost

- Case 3: 32 inch TV, Blu-Ray player with WiFi, 7 channel Receiver, Floor Standing speakers (2 pair) (Cost threshold: $ 3000)

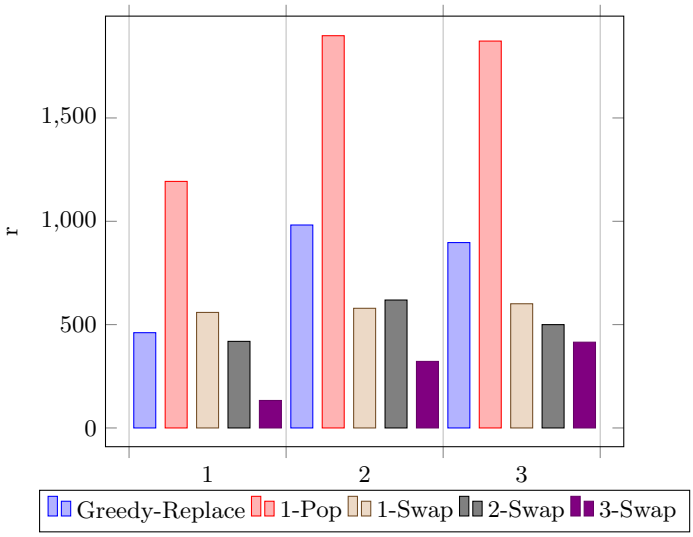
5.2.2 *Number of recommendations*

The number of recommendations returned by each of the selection approaches discussed above for all the given user specifications is provided in Figure 2. 1-Pop generates more recommendations than Greedy-Replace as it performs several iterations of Greedy-Replace. Both 2-Swap and 3-Swap are able to generate a large number of recommendations for the given user specifications as they perform more swaps than Greedy-Replace and 1-Pop. However, a diverse set of solutions is not the only factor that determines the effectiveness of the system.

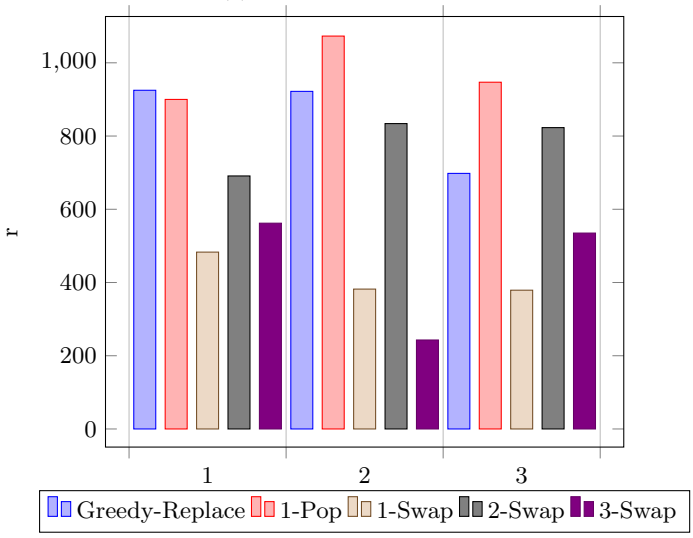
5.2.3 *Desirability value per unit cost*

The desirability of the returned results also plays a role in deciding the effectiveness of the approach. By calculating the desirability value per unit cost for all the recommendations returned, the average desirability value per unit cost p_{avg} and the maximum desirability value per unit cost p_{max} are estimated for all the selection approaches. The values for p_{avg} for all the selection approaches are shown in Figure 3. 2-Swap and 3-Swap are found to have lower values of p_{avg} when compared to other methods such as Greedy-Replace and 1-Pop. This is because each step in a single iteration of Greedy-Replace results in an increase in desirability value. As 2-Swap and 3-Swap consider all possible combinations by swapping feature examples, they are not guaranteed to return recommendations with high desirability values.

It can be inferred from Figure 4 that both Greedy-Replace and 1-Pop methods return high values of p_{avg}, However, from Figure 2, we see that 1-Pop returns more recommendations than Greedy-Replace. From Figure 4, where we have tabulated the values of p_{max} for the different selection approaches, it is observable that in some cases, such as for desktop computers (Figure 4(a)), only Greedy-Replace and 1-Pop are able to return the items with maximum desirability value per unit cost for a given set of user specifications. In the case of home theatres (Figure 4(b)), all approaches are found to suggest the recommendation that has the maximum desirability value per unit cost for the given set of user specifications.



(a) Desktop computers



(b) Home theater systems

Figure 5: Ratio of number of recommendations to time taken

5.2.4 *Time complexity*

It is also essential to measure the time complexity of the various selection approaches so that they can be scaled to real-time scenarios. However, as all approaches return different number of recommendations, it is counter-intuitive to compare their running times. Hence, to measure the time complexity of a particular approach, we shall define the ratio between the number of recommendations returned to the time taken to generate those recommendations as r. The values of r for the different selection approaches are plotted in Figure 5.

In most cases, 1-Pop is able to achieve high values of r. Thus, we are able to conclude that 1-Pop is a computationally efficient and an effective approach for finding the candidate

set of solutions for a given set of user specifications. It is able to generate a sufficiently large number of recommendations with high desirability values in a short amount of time. We therefore select 1-pop as our chosen strategy for generating candidate solutions and the overall recommendation algorithm using 1-pop is evaluated in Section 5.3.

5.3 Evaluation Metrics

In order to measure the performance of the proposed recommender system, it is necessary to evaluate the results suggested by the system against a ground truth. To establish the ground truth for the customized product recommendation problem, a sample population of users were presented with the results of the recommender system for both desktop computers and home theater systems. Among these results, they were asked to select a total number of 10 products that they would consider buying given the requirements specifications. Each product was then assigned a preference score equal to the number of users who preferred that particular product. The products were then ranked according to their preference value, forming the set R_G. These results were then compared against the top 10 recommendations returned by the 1-Pop selection approach for the same user specifications, denoted by R_S. The elements in R_S are ranked according to their desirability value per unit cost.

An important performance of the relevance of results returned by the recommender system is given by its precision and recall values. The precision of a recommender system is the proportion of recommendations returned by the system that the user has judged to be good recommendations from the ground truth. The recall of a recommender system is the proportion of the good recommendations as judged by the user that are returned by the system. Thus, we can define precision and recall as follows.

$$precision = \frac{|R_S| \cap |R_G|}{|R_S|} \tag{7}$$

$$recall = \frac{|R_S| \cap |R_G|}{|R_G|} \tag{8}$$

After measuring the relevance of the results returned by the system, it is necessary to evaluate the system's rankings of the recommended products. We can measure the ranking quality of the system using the Normalized Discounted Cumulative Gain (NDCG) measure [6]. Given the preference scores for each recommendation rel_i in R_S, we can define the Discounted Cumulative Gain (DCG) as

$$DCG_c = \sum_{i=1}^{c} \frac{2^{rel_i} - 1}{\log_2 (i+1)} \tag{9}$$

The DCG of the rankings in R_G can be defined as the ideal DCG, or the IDCG. Thus we can define the NDCG of the recommendations as

$$nDCG_c = \frac{DCG_c}{IDCG_c} \tag{10}$$

The precision, recall and NDCG values for the different user specifications are provided in Figure 6. In the case of desktop computers, an average of 6-7 recommendations that are returned by the system are preferred by the user. In the case of home theater systems, an average of 5-6 recommendations that are returned by the system are preferred by the user. This performance we believe is due to the modelling of the desirability value of the product and its components, which is able to provide recommendations that capture the user preference. Also, as we use the 1-Pop approach for the MCKP, most of the recommendations obtained have high desirability values per unit cost. The user is likely to agree with an average of 77% of the rankings returned by the system in the case of desktop computers, and with an average of 86% of the rankings returned by the system in the case of home theater systems. Thus, ranking the recommendations in decreasing order of desirability value per unit cost proves to be an effective ranking measure.

User Input	Precision	Recall	NDCG
Case 1	0.8	0.8	0.8344
Case 2	0.6	0.6	0.7928
Case 3	0.6	0.6	0.6900

(a) Desktop computers

User Input	Precision	Recall	NDCG
Case 1	0.7	0.7	0.8705
Case 2	0.6	0.6	0.8784
Case 3	0.3	0.3	0.8300

(b) Home theater systems

Figure 6: Precision, Recall and NDCG values

6. CONCLUSION AND FURTHER WORK

In this paper, we have presented an approach to solve the Customizable Product Recommendation problem by modelling it as a special case of the Multiple Choice Knapsack Problem. We proposed a greedy algorithm for solving the Customizable Product Recommendation problem, as well as additional selection approaches for generating a more diverse set of recommendations. We were also able to identify one of those approaches, 1-Pop, as the best method overall. The performance of our recommendation system was then evaluated on two types of products: desktop computers and home theater systems, using empirical measures such as precision, recall and NDCG.

The results generated by the recommender system are highly dependent on the desirability value of a particular component. Hence, better understanding of user behavior and modelling of component desirability is necessary to build a system which achieves high degree of performance. In this paper, we have assumed that the selection of feature components belonging to a particular class are independent of the selection of feature components from another class. However, this might not always be the case. A more robust system must be developed to account for feature classes that are dependent on each other. Subsequent user feedback after the initial recommendations has not been covered in this paper, and is an interesting avenue to explore in the future.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge*

and Data Engineering, IEEE Transactions on, 17(6):734–749, 2005.

[2] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[3] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[4] K. Dudziński and S. Walukiewicz. A fast algorithm for the linear multiple-choice knapsack problem. *Operations Research Letters*, 3(4):205–209, 1984.

[5] L. Gao. A product recommendation algorithm based on knapsack optimization. 2012.

[6] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[7] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer Science & Business Media, 2004.

[8] S. L. Kendal and M. Creen. *An introduction to knowledge engineering*. Springer, 2007.

[9] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[10] E. Mavridou, D. D. Kehagias, D. Tzovaras, and G. Hassapis. Mining affective needs of automotive industry customers for building a mass-customization recommender system. *Journal of intelligent manufacturing*, 24(2):251–265, 2013.

[11] S. K. Moon, T. W. Simpson, and S. R. Kumara. An agent-based recommender system for developing customized families of products. *Journal of Intelligent Manufacturing*, 20(6):649–659, 2009.

[12] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[13] P. Sinha and A. A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

[14] H. Stormer. Improving product configurators by means of a collaborative recommender system. *International Journal of Mass Customisation*, 3(2):165–178, 2009.

[15] L. Terveen and W. Hill. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, 1:487–509, 2001.

[16] S. Trewin. Knowledge-based recommender systems. *Encyclopedia of Library and Information Science: Volume 69-Supplement 32*, page 180, 2000.

[17] A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pages 2643–2651, 2013.

[18] Y. Wang and M. M. Tseng. Customized products recommendation based on probabilistic relevance model. *Journal of Intelligent Manufacturing*, 24(5):951–960, 2013.

[19] X. Zhu, S. J. Hu, Y. Koren, and S. P. Marin. Modeling of manufacturing complexity in mixed-model assembly lines. *Journal of Manufacturing Science and Engineering*, 130(5):051013, 2008.