Most Popular Package Design

Muhammed Miah
mmiah@suno.edu
Management Information Systems Department
Southern University at New Orleans
New Orleans, LA 70126, USA

Abstract

Given a set of elements, and a set of user preferences (where each preference is a conjunction of positive or negative preferences for individual elements), we investigate the problem of designing the most "popular package", i.e., a subset of the elements that maximizes the number of satisfied users. Numerous instances of this problem occur in practice. For example, a vacation package consisting of a subset of all possible activities may need to be assembled, that satisfies as many potential customers as possible, where each potential customer may have expressed his preferences (positive or negative) for certain activities. Likewise, the problem of designing new products, i.e., deciding which features to add to a new product that satisfies as many potential customers as possible, also falls under this framework. We present innovative optimal and approximate algorithms, and study their performance. Our experimental evaluation on real and synthetic datasets shows that our optimal and approximate algorithms are efficient for moderate and large datasets respectively.

Keywords: package design, popular package, maximize visibility, customer satisfaction, algorithms.

1. INTRODUCTION

Problem Motivation

Consider a travel agency that wishes to design one (or more) vacation packages, given the travel preferences of its clients. For example, a vacation package to Costa Rica can include some of the following elements: beaches such as *Puerto Vijeo*, *Jaco*, *Flamingo*, etc.; mountains and national parks such as *Arenal area*, *Monteverde*, *Tortuguero*, etc. The clients of the agency provide their preferences by specifying "yes", "no", or "don't care" for each element. The purpose of trip/vacation package design is to select a subset of these elements to satisfy as many customers as possible.

As another example, consider the problem of creating a social network and selecting the main topics of the network based on users' interests, with the goal of representing the collective group interests as optimally as possible. For example, assume one wants to create a new group focused on sports interests. One can leverage the users' profiles to select the main topic preferences of the network– *e.g., Basketball*, *Soccer*, *Baseball*–of the users.

The above examples can be generalized to an abstract problem, which we call the *Package Design (PD) Problem*. Assume that a package needs to be designed by selecting a subset of Boolean features (or elements, or attributes) from a large set of possible features. In particular, we focus on a specific and novel problem formulation, where we are given a set of user preferences in the form of a query log (or workload) of user queries, where each query is a conjunction of positive or negative preferences for some of the features, and we are asked to design the most *popular* package, i.e., the package that satisfies the maximum number of queries in the query log. We refer to this problem as the *Package Design* (*PD*) problem. Because of the vast use of the Internet nowadays, it is very

easy to collect online such query logs of user preferences for many such package design applications, and the new package can be designed based on real users' perception on desirable features.

Overview of Solutions

We propose an optimal algorithm, based on the *Binary Tree* (Wikipedia, 2011) data structure. We also provide an approximate algorithm for the problem. The algorithm does not have provable bounds, but is scalable and is shown to work very well in practice.

Summary of Contributions

1. We define the problem of designing an optimal package given user preferences, expressed as positive and negative preferences on the elements.

2. We present a feasible optimal (exact) algorithm based on the Binary Tree data structure.

3. We present fast approximate algorithm that work well in practice for large problem instances.

4. We perform detailed performance evaluations on real and synthetic data to demonstrate the effectiveness of our developed algorithms.

The rest of the paper is organized as follows. Related work discussed in Section 2. Section 3 provides formal problem definitions. Section 4 and Section 5 present the optimal and scalable approximate algorithms respectively. In Section 6 we present the result of extensive experiments. We conclude in Section 7. Section 8 provides the references.

2. RELATED WORK

Optimal product design or positioning is a well studied problem in Operations Research and Marketing. Shocker and Srinivasan (1974) first represented products and consumer preferences as points in a joint attribute space. After that, several approaches and algorithms ([Albers & Brockhoff, 1977], [Albers & Brockhoff, 1980], [Albritton & McMullen, 2007], [Gavish, Horsky, & Srikanth, 1983], [Gruca & Klemz, 2003], [Kohli, & Krishnamurti, 1989]) have been developed to design/position a new product. Works in this domain require direct involvement (one or two step) of consumers and users are usually shown a set of existing alternative products (predesigned) to choose or set preferences. Like our work, users in this domain in fact do not get to select the attributes or features they like and don't like. Instead of involving users directly in the process of designing new package, we use previous user search queries for the same package and it is easy to collect the preferences (search queries) for large number of Internet users nowadays. We also consider large query logs to design the new package and allow users to express their interests in attribute or feature level in terms of positive, negative and "don't care".

Recent works on dominant relationship (Li, Ooi, Tung, & Wang, 2006) and dominating neighborhood (Li, Tung, Jin, & Ester, 2007) uses skyline query semantics assuming that attributes are min/max, that is, all users have the same preference for an attribute (e.g., 2 doors is always better than 4 doors). Further, they assume there is a profitability plane which simplifies the algorithm given that the optimal solution is a point on the profitability plane. In contrast, in our work users may have opposite preferences for the same attribute, and our algorithms can be used with or without a profitability plane. Li et al. (2007) also considers spatial, non-preference attributes. Our algorithms can be modified to support skyline semantics; however, more efficient algorithms may be possible for this problem variant given its restrictive nature.

Works in (Miah, Das, Hristidis, & Mannila, 2008) tackled a related problem of maximizing the visibility of an existing object by selecting a subset of its attributes to be advertised. The main problem was: given a query log with conjunctive query semantics and a new tuple, select a subset of attributes to retain for the new tuple so that it will be retrieved by the maximum number of queries. The work did not consider negated conditions as in our work in this paper. In this paper, we consider designing an object (a new tuple), that is, assign values for all attributes instead of selecting subset of attributes.

3. PROBLEM FRAMEWORK

To define our problem more formally, we need to develop a few abstractions.

Attributes: Let $A = \{a_1 \ldots a_M\}$ be the set of Boolean attributes (or elements, or features).

Query (with negation): We view each user query as a subset of attributes and/or negation of attributes. The semantics is *conjunctive*, e.g., query $\{a_1, a_3\}$ is equivalent to "$a_1 = 1$ and $a_3 = 1$". We also consider queries *with negations*, e.g., $\{a_1, \sim a_2\}$ is equivalent to "$a_1 = 1$ and $a_2 = 0$". The remaining attributes for which values are not mentioned in the query are assumed to be "don't care", i.e., the value can be either 0 or 1.

Query Log or Workload: Let $Q = \{q_1 \ldots q_S\}$ be a collection of queries.

The problem definition is as follows:

Package Design (PD) Problem: *Given a query log Q with conjunctive semantics where a query can have negations, design a new tuple t (assign value [0, 1] for each attribute for the new tuple) such that the number of queries that retrieve t is maximized.*

Thus, we wish to ensure that the new package (or tuple) satisfies as many customers as possible.

Query ID	Beach	Boating	Casino	Fishing	Historical Site	Museum
q_1	1	0	1	?	?	?
q_2	1	?	0	?	1	?
q_3	0	?	1	?	1	?
q_4	?	0	0	1	1	?
q_5	?	1	?	0	?	1
q_6	1	0	0	?	?	0

Figure 1. Query Log Q for Running Example

Example 1. *Consider Figure 1 which shows a query log for a vacation package application, containing S=6 queries and M=6 attributes where each tuple (query) represents the preferences of a user. A query has values 1, 0, or ?, where 1 means the attribute must be present, 0 means the attribute must not be present, and "?" means "don't care". For this specific example, it is not hard to see that if we design a new package with Beach = 1, Boating = 0, Casino = 0, Fishing = 1, Historical Site = 1, Museum = 0 (i.e., new tuple t = [1, 0, 0, 1, 1, 0]), we can satisfy a maximum of 3 queries (q_2, q_4 and q_6). No other selection of attribute values for the new tuple will satisfy more queries.* □

4. A FEASIBLE OPTIMAL ALGORITHM

A naïve brute-force optimal approach seems to be a solution to design a new tuple (package) where we can generate all possible combination of attribute values and pick the combination (assignment of values) that is satisfied by the highest number of queries in the query log.

While the naïve algorithm is polynomial in the size of the query log, it is unfortunately exponential in number of attributes. Thus it is not feasible when the number of attributes is large since the algorithm has to generate an exponential number of possible combinations of attribute values.

We propose a novel optimal algorithm based on adaptations of the Binary Tree data structure which is much more efficient than the Naïve algorithm. Our algorithm works well for moderate problem instances. A binary tree is a tree data structure in which each node has at most two child nodes, usually distinguished as "left" and "right". Nodes with children are parent nodes, and child nodes may contain references to their parents. Outside the tree, there is often a reference to the "root" node (the ancestor of all nodes), if it exists. Any node in the data structure can be reached by starting at root node and repeatedly following references to either the left or right child. Figure 2 shows a simple binary tree of size 9 and height 3, with a root node whose value is 2. The above tree is unbalanced and not sorted.

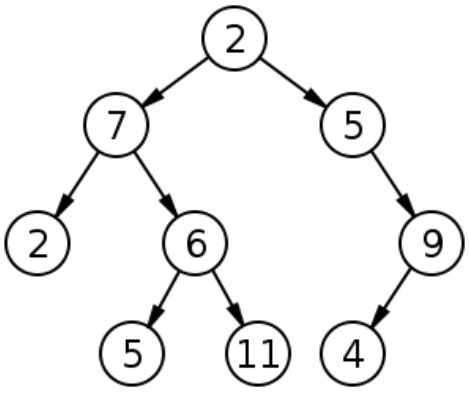

Figure 2. A Simple Binary Tree

Optimal Algorithm Based on Binary Tree Data Structure (*TreePD*)

We build a Binary tree structure like tree for our algorithm using the query log where each child (left and right) is created based on the attribute values. As described earlier in our running example (Figure 1), each attribute can have a value either 1 (attribute present in the query), 0 (attribute not present in the query), or "?" (don't care). We build the tree such that the query log is split into two groups based on an attribute value (which becomes a node in the tree) – the left child contains the queries with value 0 and "?"; and the right child contains the queries with values 1 and "?" for that specific attribute. The queries with attribute value "?" (don't care) go with both the left and right children of a node because in a newly designed package the value of that attribute could be either 1 or 0.

Once the tree is created, then we search the tree starting from the root of the tree and keep track how many queries can be satisfied the by assignment of the attribute values from root to the leaf. We pick the path (assignment of attribute values) with the highest count (number of satisfied queries) as the new package (tuple).

The *TreePD* optimal algorithm is much faster than the naïve optimal approach as we don't have to generate all possible combinations of attributes, but still the algorithm can be slow in case of very large number queries and attributes. So we propose approximate algorithm that work well for large dataset which is discussed next.

5. APPROXIMATE ALGORITHM BASED ON *MINSAT* HEURISTIC (*HeuristicPD*)

Package Design (PD) problem is the complement of the *MINSAT* problem (Kohli et al., 1994), which is an NP-complete problem. Given a set *U* of Boolean variables and a collection of disjunctive clauses over *U*, the goal of *MINSAT* problem is to find a truth assignment that minimizes the number of satisfied clauses. *PD*, which has conjunctive clauses (queries), can be converted to *MINSAT* as follows:

a) Complement the value of each attribute for each query in the query log, i.e., if an attribute has value 0 then convert it to 1 and vice versa.

b) Complement the conjunctive semantics to disjunctive semantics. Let $\sim Q$ denote the converted query log Q. Solving *MINSAT* on $\sim Q$, we get an assignment that satisfies the minimum number of queries in $\sim Q$; which corresponds to satisfying the maximum number of queries in the original query log Q.

Our algorithm adopting a greedy *MINSAT* heuristic (Kohli et al., 1994) operates as follows. Given any ordering of the variables, the greedy heuristic sequentially selects an assignment for each variable to satisfy the smallest number of additional clauses (clauses in $\sim Q$ in *PD*). Figure 3 displays the pseudocode of the algorithm.

Approx Algorithm: *HeuristicPD*
Let Q be the query log, A ($a_1 \ldots a_M$) be the attributes in Q
Complement the query log ($\sim Q$) // convert 1 to 0 and 0 to 1, also convert conjunctive form to disjunctive form
For (int $i = 1$ to M)
  If $\sim Q$ not empty
    Count # of queries satisfied both for $a_i = 1$ and $a_i = 0$.
    Assign the value of a_i that gives the minimum count
    Remove queries from $\sim Q$ satisfied by the value of a_i
Return the attributes assignment

Figure 3. Pseudocode of Approximate Algorithm Based on *MINSAT* Heuristic, *HeuristicPD*

The above heuristic has an approximation ratio equal to the maximum number of attributes (literals) in any query (clause). Note that this ratio does not hold for *PD* since in *PD* the solution is complemented, that is the number of satisfied queries is *S* minus the number of satisfied queries in *MINSAT*. Nevertheless, our experimental results in Section 6 show that the algorithm has a very small approximation error in practice.

6. EXPERIMENTS

Our main performance indicators are (a) the time cost of optimal and approximate algorithms, and (b) the approximation quality of approximate algorithm.

System Configuration: We used Microsoft SQL Server 2000 RDBMS on a P4 3.2-GHZ PC with 1 GB of RAM and 100 GB HDD for our experiments. Algorithms are implemented in C#.

Datasets: We used datasets of products and product queries. Note that products are just one of the possible instantiations of the more general packages of this paper. We used real and syn-

thetic datasets (query logs). In specific, we use two datasets: (i) *REAL*: real query log, and (ii) *REAL+*: synthetic query log generated from the real query log.

Real query log (REAL): We collected 240 queries for cell phones from university users of and friends through an online survey. The survey was designed with 30 Boolean features such as *Bluetooth*, *Wi-Fi*, *Camera*, *Speakerphone* and so on. Users were asked to select the features they prefer to have (positive) and most likely not to have (negative) in their cell phones. Users selected 3-6 positive and 1-2 negative features on average. Hard disk was a popular negative feature.

Synthetic query log generated from real query log (REAL+): As the real query log is very small, it is inappropriate for scalability experiments. So we generated larger datasets from the real query log. A total of 200,000 queries were generated as follows: at each step we randomly select a query from the *REAL* query log, randomly select two of its attributes and swap their values. We also generate datasets for a fixed size of query log for varying number of attributes (10, 15, 20, 25, and 30).

Table 1 summarizes the query logs or datasets.

Query log	# of attributes	Query log size
REAL	30	237
REAL+_30	30	25K, 50K, …, 200K
REAL+_1000	10, 15, …, 30	1000

Table 1. Summary of Query Logs (Datasets)

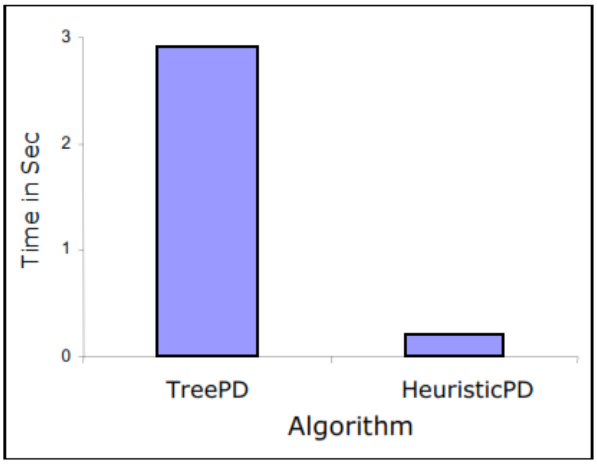

Figure 4. Time cost for *REAL* dataset

Figures 4 and 5 show the performance and quality of the algorithms for the real query log (*REAL*). Here, by quality we mean how many queries are satisfied by a newly designed package. Note that *HeuristicPD* has almost optimal quality.

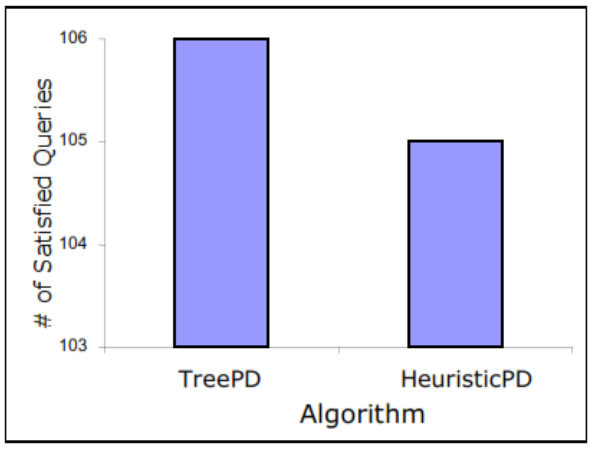

Figure 5. Quality for *REAL* dataset

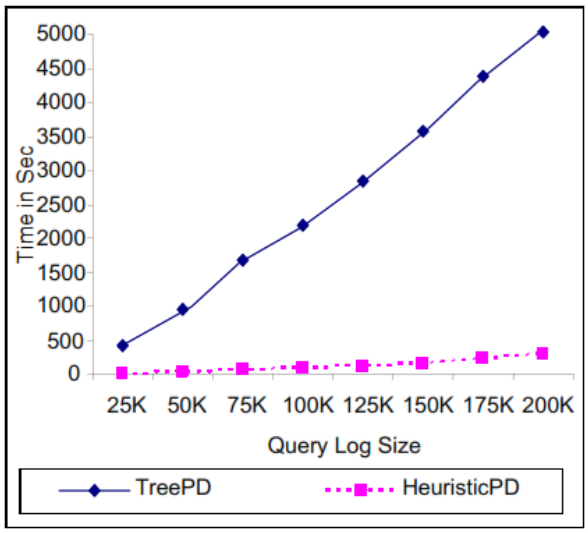

Figure 6. Time cost for varying query log size for *REAL+_30*

Figures 6 and 7 show the performance of the algorithms for varying query log size and number of attributes respectively, for *REAL+* dataset. For varying query log size, we want to see how our algorithms perform when query log sizes (datasets) increase. For varying number of attributes, we again want to see how the algo-

rithms perform when number of attributes increases for a fixed number of queries. We randomly select a subset (of size 10, 15, ..., 30) of the attributes of the dataset. As we can see from the graphs, the approximate algorithm is much more efficient than optimal algorithm.

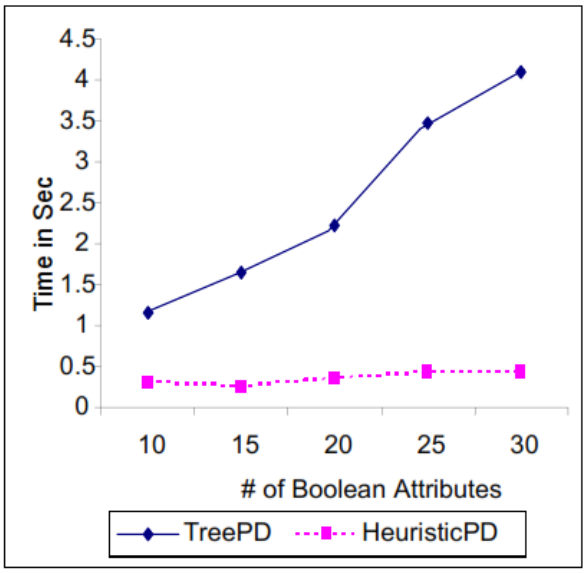

Figure 7. Time cost for varying # of attributes for *REAL+_1000*

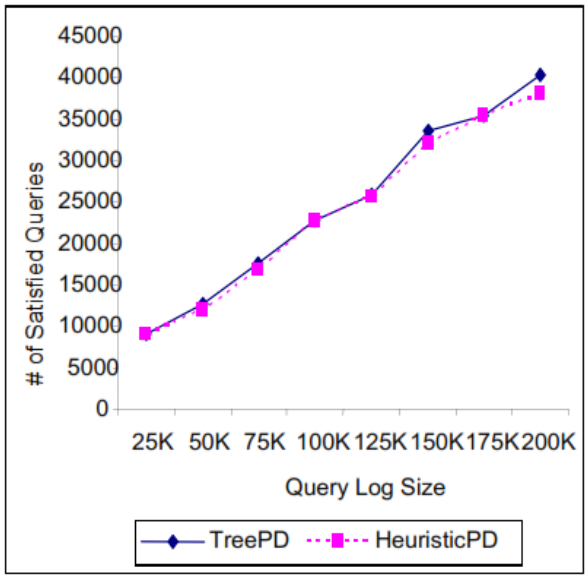

Figure 8. Quality for varying query log size for *REAL+_30*

Figures 8 and 9 show the quality (number of queries satisfied in the query log) of the approximate algorithm for varying query log size and number of attributes respectively for *REAL+* dataset. As we can see from the graphs, the approximate algorithm performs well. As we see in Figure 9, the number of satisfied queries decreases as the total number of attributes increases. The number decreases because as more attributes are added, the queries become more selective and harder to be satisfied. The approximate algorithm has quality close to the optimal algorithm.

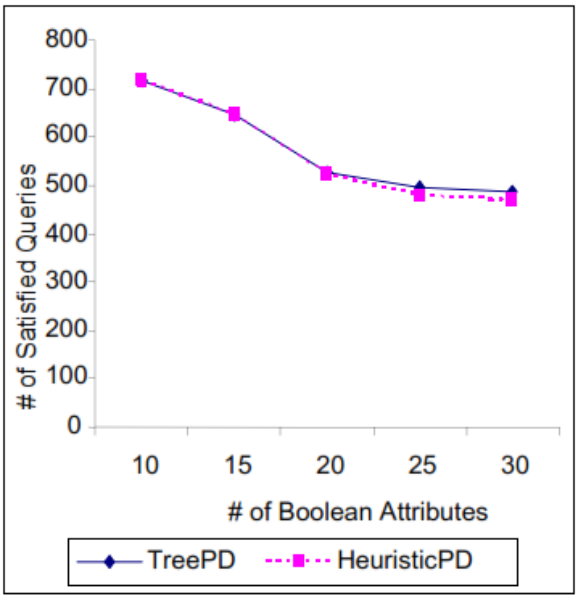

Figure 9. Quality for varying # of attributes for *REAL+_1000*

7. CONCLUSIONS AND FUTURE WORK

In this work we investigated the problem of designing a package, such that, given a query log, this package will be returned by the maximum number of queries in the query log where a query can have negations. We proposed an innovative optimal algorithm and showed the algorithm is feasible for moderate inputs. Furthermore, we present approximate algorithm, which are experimentally shown to produce good approximation ratios for large databases. A future direction is to extend the problem to other data types, such as categorical, text and numeric and different query semantics like top-*k* and skyline retrieval.

8. REFERENCES

Albers, S., Brockhoff, K. (1977). A procedure for new product positioning in an attribute space. *European Journal of Operational Research*. 1, 4 (Jul 1977), 230-238.

Albers, S., Brockhoff, K. (1980). Optimal Product Attributes in Single Choice Models. *Journal of the Operational Research Society*. (1980) 31, 647–655.

Albritton, D. M., McMullen, P. R. (2007). Optimal product design using a colony of virtual ants. *European Journal of Operational Research*. 176, 1 (Jan 2007), 498-520.

Gavish, B., Horsky, D. Srikanth, K. (1983). An Approach to the Optimal Positioning of a New Product. *Management Science*, 29, 11 (Nov 1983), 1277-1297.

Gruca, T. S., Klemz, B. R. (2003). Optimal new product positioning: A genetic algorithm approach. *European Journal of Operational Research*. 146, 3, 2003, 621-633.

Kohli, R., Krishnamurti, R., Mirchandani, P. (1994). The Minimum Satisfiability Problem. *Siam J. Discrete Math*.

Kohli, R., Krishnamurti, R. (1989). Optimal product design using conjoint analysis: Computational complexity and algorithms. *European Journal of Operational Research*. 40,2, 1989.

Li, C., Tung, A. K. H., Jin, W., Ester, M. (2007). On Dominating Your Neighborhood Profitably. *VLDB* 2007, 818-829.

Li, C., Ooi, B. C., Tung, A. K. H., Wang, H. (2006). DADA: a Data Cube for Dominant Relationship Analysis. *SIGMOD* 2006.

Miah, M. Das, G., Hristidis, V., Mannila, H. (2008). Standing Out in a Crowd: Selecting Attributes for Maximum Visibility. *ICDE* 2008: 356-365.

Shocker, A. D., Shrinivasan, V. A consumer-based methodology for the identification of new product ideas. *Management Science*. 20, 6 (Feb 1974), 921-937.