An Efficient Preference Elicitation Algorithm Using Group Generalized Binary Search

Yi Ren, Clayton Scott, and Panos Papalambros

Abstract

We examine how to efficiently elicit the most preferred products of a user out of a finite product set by asking the user to make a sequence of pairwise comparisons. The key challenge is to minimize the number of "queries" - the product pairs presented to the user, in order for such elicitation interactions to be practical. Previous work formulated such an elicitation task as a blackbox optimization problem with comparison (binary) outputs, and a heuristic search algorithm similar to Efficient Global Optimization (EGO) was used. In this paper, we propose a query algorithm that directly minimizes the expected number of queries, assuming that products are embedded in a known feature space and user preference is a linear function of product features. Besides its theoretical foundation, the proposed algorithm shows empirical performance better than the EGO approach in both simulated and real-user experiments. A novel approximation scheme is also introduced to alleviate the scalability issue of the proposed algorithm, making its computation tractable for a large number of product features or of candidate products.

Index Terms

Active learning, group identification, preference elicitation

I. Introduction

A typical online shopping platform takes in keywords and outputs a list of relevant products to the user. In most cases when the list is too long to read through, users resort to sorting and filtering tools to narrow down the search. In addition, the introduction of recommender systems allows users to navigate through only products they are likely to be interested in [3]. While a combination of these tools helps to locate preferred products for users, this work investigates

Y. Ren and P.Y. Papalambros are with the Department of Mechanical Engineering, University of Michigan, Ann Arbor.

C. Scott is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor.

a human-computer interaction that actively elicits user preference and thus has the potential to further speed up the search of preferred products.

Consider a user looking for a new laptop. The user may prefer products within a certain budget. Yet products beyond the budget limit may not be rejected if they offer much better performance. In this situation, the user may need to explore a variety of price ranges, and search within each resulting sub-list. One potential way to reduce search effort is to introduce an interaction that actively asks user questions in order to understand his preference. For instance, the computer could present a pair of laptops to the user and ask which one he or she prefers more. The binary response to such a query, i.e., selecting one from the pair, will be learned by the computer where the next query can be created based on the learned knowledge. The interaction terminates when one of the laptops is estimated to be the most preferred for this user with probability one. Such an interaction will help users to find what they want efficiently without the hassle of browsing all candidate products.

The challenge in implementing such a preference elicitation interaction is that the interaction needs to be short and efficient, leading to the question how to minimize the expected number of queries. In previous research, interactive genetic algorithms were introduced to address this issue by finding preferred product shapes based on evolutionary operations [11], [12]. More recently an idea based on ranking Support Vector Machine [6], [9] was proposed that iteratively refines a user preference model according to binary choice responses and creates queries with products predicted to be preferred according to the model [14].

This paper aims to improve the heuristic nature of previous query strategies. We present a strategy that minimizes directly the expected number of queries, provided that the product set is finite. This approach is inspired by the Group Generalized Binary Search (GGBS) algorithm, originally introduced as a solution to the "group identification" problem.

The paper is organized as follows: We first introduce the problem of group identification in Section II and show its similarity to preference elicitation. Some background and notations are given in Section III before we introduce the GGBS algorithm and its implementation in Section IV. We test the algorithm using simulations in Section V and conduct a real-user experiment in Section VI. Section VII discusses the relationship between this work and existing work. Section VIII concludes this paper and discusses future directions.

Group	**Object**	Q1: On ground?	Q2: Takes many people?	Q3: Moves vertically?
Ground vehicle	Motorcycle	Yes	No	No
	Bus	Yes	Yes	No
Aircraft	Helicopter	No	No	Yes
	Plane	No	Yes	No

Fig. 1. An example of the group identification problem. Among the three candidate questions, "Q1" is the best to query.

II. Preference elicitation as group identification

Group identification is a variation of object identification. A well-known example of the latter is the "twenty-questions" game. This is a game between two players: Player A first thinks about an object that both players are familiar with. Player B then asks yes/no questions to guess which object it is. In group identification, Player B is not required to identify exactly the object, but only the "group" that object belongs to.

How questions are asked will affect how quickly Player B can identify the correct group. We use a trivial case in Figure 1 to illustrate. Consider that both player A and player B are aware of but limited to the information presented in the figure, where two groups, four objects and three questions exist. Player A first picks an object from the four and Player B will start to raise questions among the three. One can see that the best question to raise in this case is "Q1", the answer of which will directly determine (with probability one) which group the object is from. The strategy of asking questions based on available information is in general called "active learning". Bellala et al. looked into this topic within the context of object and group identification and proposed the GGBS algorithm that adaptively chooses queries and minimizes the expected number of queries needed for an interaction [4]. We will revisit this example and explain theoretically why "Q1" is the best question to ask once we introduce the GGBS algorithm in detail in Subsection IV-A.

To see the similarity between preference elicitation and group identification, consider that we want to identify from products A, B and C, the most preferred product for a user. Here, the

counterpart of an object is a ranking of products, e.g., product A is better than B and B is better than C, or more concisely $A \succ B \succ C$; the counterpart of a group is a set of rankings with the same most preferred product, e.g., the group called "product A is the best" contains two objects: $A \succ B \succ C$ and $A \succ C \succ B$. The counterpart of a yes/no question is a pairwise comparsion, e.g., "from products A and B, which one do you prefer more?" Using this analogy, we can see that finding the most preferred product of a user is equivalent to identifying a group. Therefore, the GGBS algorithm developed for group identification would be applicable to preference elicitation.

III. Background and Notations

Before we explain the GGBS algorithm in the context of preference elicitation, this section gives the definitions of "preference", and introduces a binary decision tree representation for "interactions". We show that the expected number of queries of an interaction is derived as a function of this tree.

A. Product, utility and ranking

Consider a product set $\{\mathbf{x}_k : k = 1, \ldots, K\}$ where each product is represented by a product feature vector $\mathbf{x}_k \in \mathbb{R}^D$. Here K is the total number of products and D is the number of product features. The utility of product k to a user is modeled as $u_k = \mathbf{w}^T\mathbf{x}_k$, where $\mathbf{w}$ is called the "partworth" vector representing the unknown user preference, and $||\mathbf{w}||_2 = 1$. In this study, we assume that user responses are noiseless, i.e., when $u_{k_1} > u_{k_2}$ for any $k_1,\ k_2 \in 1, \ldots, K$, the user will always choose $\mathbf{x}_{k_1} \succ \mathbf{x}_{k_2}$. In addition, we assume that users will not be indifferent to any presented pairs, i.e., for a pair with $\mathbf{x}_{k_1}$ and $\mathbf{x}_{k_2}$, we have $\mathbf{w}^T(\mathbf{x}_{k_1} - \mathbf{x}_{k_2}) \neq 0$. The entire candidate query set consists of all $N = K(K-1)/2$ product pairs.

Given a product set, each user can have a ranking of products based on his or her preference $\mathbf{w}$. Let the total number of rankings be M. We denote by θ a ranking, for example, $\theta = \mathbf{x}_1 \succ \mathbf{x}_2 \succ ... \succ \mathbf{x}_K$, and by $\Theta = \{\theta_1, \theta_2, ..., \theta_M\}$ the ranking set. Note that M is usually much smaller than $K!$ when a large number of products is embedded in a low-dimensional space and utility is a linear function of product features [8].

Each ranking is then labeled according to its top-ranked product, e.g., if $\theta_m = \mathbf{x}_1 \succ ...$, then the label $y_m = 1$ is assigned. We denote by $\Theta_k = \{\theta_m \in \Theta : y_m = k\}$ the rankings for which

product k is the most preferred.

For each ranking θ_m, we use $\pi_{\theta_m} = \Pr(\theta = \theta_m)$ to represent its probability to be the correct ranking of the current user, and the set

$$\Pi_\theta := (\pi_{\theta_1}, ..., \pi_{\theta_M}) \tag{1}$$

for the set of probabilities of all M rankings (objects), with $\sum_{m=1}^{M} \pi_{\theta_m} = 1$.

Similarly, we use $\pi_{\Theta_k} = \sum_{\theta \in \Theta_k} \pi_\theta$ for the probability of product k being the most preferred of the current user, and the set

$$\Pi := (\pi_{\Theta_1}, ..., \pi_{\Theta_K}) \tag{2}$$

for the set of probabilities of all K products (groups).

Note that neither Π_θ nor Π is usually known. Nonetheless, the GGBS algorithm will require Π as an input. Therefore in the next section we will discuss how Π can be initialized and updated when most preferred products are collected from user interactions.

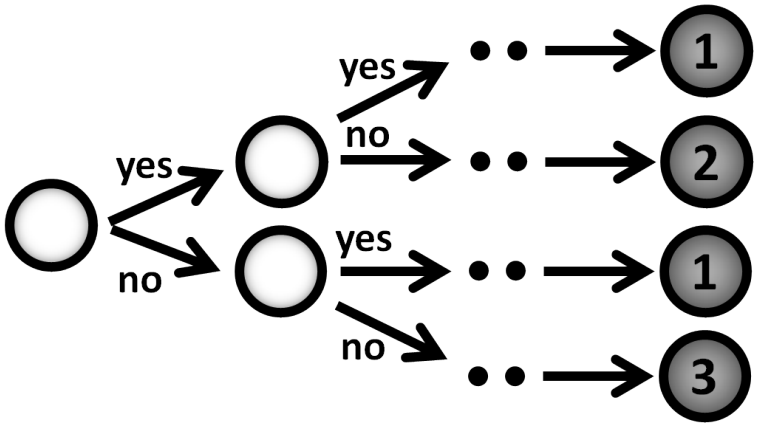
B. User-computer interaction

Each user-computer interaction consists of a sequence of queries, i.e., pairwise comparison tasks generated by the computer and completed by the user. The interaction can be considered as a "path", as shown in Figure 2(a) where a set of nodes are connected by edges. We call the last node of the path the "leaf" node, which is labeled by the most preferred product of the user. All other nodes are called "internal" nodes. For each internal node "a", it contains (1) a set of rankings $\Theta^a \subseteq \Theta$ that reaches the node based on previous query responses, and (2) a new query made at this node, the response to which will lead to the next node.

A query strategy refers to how queries are assigned to internal nodes. For a user with fixed preference $\mathbf{w}$, different query strategies could lead to paths with different lengths. For a fixed query strategy, the combination of all possible paths will form a "binary decision tree", where each internal node leads to two child nodes, see Figure 2(b).

The binary decision tree can have multiple paths with leaf nodes labeled by the same product. This is because while people may rank products differently, they can share the same most preferred product. Given the probabilities Π_θ for all rankings (and Π for all products) and given a query strategy, we can calculate the expected path length of the tree, i.e., the number of queries needed to identify the most preferred product, and use it to represent the efficiency of the chosen

(a)



(b)

Fig. 2. (a) An individual interaction as a path; (b) Combined interactions as a binary decision tree

query strategy. Formally, consider d_k as the expected number of queries needed to reach a leaf node labeled by product k, then the expected number of queries of a binary decision tree will be:

$$L = \sum_{k=1}^{K} \pi_{\Theta_k} d_k. \tag{3}$$

Minimization of L requires: (1) accurate approximation of the probabilities of each product being the most preferred ($\Pi = (\pi_{\Theta_1}, \ldots, \pi_{\Theta_k})$), and (2) a good query strategy that minimizes L for a given approximation of Π. Both will be discussed in the following section.

IV. The Algorithm

To recap, the efficiency of a query strategy, i.e., which query to raise at each internal node, can be measured by the expected number of queries, L, of the binary decision tree. Optimizing L over all possible arrangements of queries is shown to be NP complete [7]. As an affordable

alternative, the GGBS algorithm adaptively finds the best query at the current node during an interaction, based on previous responses of the current user, therefore reducing the problem from finding the optimal binary decision tree to finding only the best next query along the path for the current user.

This section first introduces the GGBS algorithm, and then discuss how Π shall be initialized and updated based on previous interactions. We then elaborate on an efficient implementation of the algorithm to allow real-time human-computer interactions.

A. Group Generalized Binary Search (GGBS)

Specifically, an important proof from Bellala et at. [4] showed that for any query strategy, L can be decomposed into a set of additive terms L^a with respect to each internal node "a":

$$L = \sum_{a \in \text{all internal nodes}} L^a + \text{constant}. \tag{4}$$

Thus minimization of L can be achieved by greedily minimizing L^a at each node with respect to the choice of query. This local objective L^a takes the following form and requires some explanation:

$$L^a = 1 - H(\rho^a) + \sum_{k=1}^{K} \frac{\pi_{\Theta_k^a}}{\pi_{\Theta^a}} H(\rho_k^a). \tag{5}$$

For a given query (pairwise comparison) with binary response, the current node "a" will lead to the "left" and "right" child nodes denoted as "$l(a)$" and "$r(a)$", respectively. The sets $\Theta^{l(a)}$ and $\Theta^{r(a)} \subseteq \Theta^a$ contain rankings that fall into these two child nodes. The symbol ρ^a in Equation (5) is called the "reduction factor" and is defined as

$$\rho^a = \max\{\pi_{\Theta^{l(a)}}, \pi_{\Theta^{r(a)}}\}/\pi_{\Theta^a}, \tag{6}$$

where $\pi_{\Theta^a} := \sum_{\{i:\theta_i \in \Theta^a\}} \pi_{\theta_i}$ is the total probability mass of rankings at node "a", which then splits into $\pi_{\Theta^{l(a)}}$ and $\pi_{\Theta^{r(a)}}$ for the "left" and "right" child nodes. Similarly, the "group reduction factor" ρ_k^a is defined as

$$\rho_k^a = \max\{\pi_{\Theta_k^{l(a)}}, \pi_{\Theta_k^{r(a)}}\}/\pi_{\Theta_k^a}, \tag{7}$$

where $\pi_{\Theta_k^a} := \sum_{\{i:\theta_i \in \Theta_k^a\}} \pi_{\theta_i}$ is the total probability mass of a group labeled by product k at node "a", which is then separated into $\pi_{\Theta_k^{l(a)}}$ and $\pi_{\Theta_k^{r(a)}}$ for the given query.

The term $\pi_{\Theta_k^a}/\pi_{\Theta^a}$ represents the conditional probability of product k being the most preferred product at node "a". Finally, we denote by $H(\rho) := -\rho \log_2 \rho$ the Shannon entropy of some scalar ρ, and define $0 \log_2 0 = 0$.

Note that in Equation (5), the values of both the reduction factor and the group reduction factor depend on the choice of query. Since the entropy $H(\rho)$ is a concave function maximized at $\rho = 0.5$, in order to minimize L^a, a desired query would both have ρ^a close to 0.5 and ρ_k^a close to 1 for all ks. The following explanation provides reasons behind the math of why such a query is desired: By definition, when a query achieves $\rho^a = 0.5$, it indicates that the query is most "uncertain", meaning that both "yes" and "no" (or equivalently, one product versus the other product in pairwise comparison) having 50 percent of chance to be selected as the answer. For such a query, half of the remaining objects (rankings) will be eliminated, regardless of the user's response to the query. On the other hand, when $\rho_k^a = 1$ for all ks, it indicates that grouped rankings will together go into either the left or the right child node, therefore reducing the number of candidate groups, i.e., most preferred products, in the child nodes.

To further clarify the idea, let us revisit the group identification problem in Figure 1. Once Player A picked one object from the four, Player B needs to pick a query. To start, it is reasonable for Player B to believe that the four candidate objects have equal chances to be the correct one, i.e., $\Pi_\theta = (\pi_{\text{Motorcycle}}, \pi_{\text{Bus}}, \pi_{\text{Helicopter}}, \pi_{\text{Plane}}) = (0.25, 0.25, 0.25, 0.25)$ and therefore $\Pi = (\pi_{\Theta_{\text{Ground vehicle}}}, \pi_{\Theta_{\text{Aircraft}}}) = (0.5, 0.5)$ for the two groups. Player B will now calculate Equation (5) for each query and pick one with minimal L^a. The calculation is performed in Table I, where the left (right) child node corresponds to the answer "yes" ("no"). Therefore to minimize L^a, "Q1" should be chosen.

B. Update of Π based on previous interactions

From the example, we see that how queries are chosen using GGBS depends on the given Π, the probabilities of products being the most preferred one. In this study, we use the assumption that prior to any user interaction, the probabilities are equal, i.e., $\pi_{\Theta_k} = 1/K$ for all $k = 1, \ldots, K$.

Once some user interactions are performed and the most preferred products of these users are collected, Π will be adjusted according to the observations. Consider a total of s interactions being performed. Let $c_k^{(s)}$ be the count of product k being the most preferred product. We can update π_{Θ_k} by $\tilde{\pi}_{\Theta_k}^{(s)} = 1 + tc_k^{(s)}$ and $\pi_{\Theta_k}^{(s)} = \tilde{\pi}_{\Theta_k}^{(s)} / \sum_{k=1}^K \tilde{\pi}_{\Theta_k}^{(s)}$, where t is an algorithmic parameter.

TABLE I

QUERY SELECTION FOR THE CASE IN FIGURE 1

Query	$\pi_{\Theta^{l(a)}}$ ($\pi_{\Theta^{r(a)}}$)	ρ^a	$\pi_{\Theta^{l(a)}_{\text{Ground vehicle}}}$ ($\pi_{\Theta^{r(a)}_{\text{Ground vehicle}}}$)	$\pi_{\Theta^{l(a)}_{\text{Aircraft}}}$ ($\pi_{\Theta^{r(a)}_{\text{Aircraft}}}$)	$\rho^a_{\text{Ground vehicle}}$	ρ^a_{Aircraft}	L^a
Q1	0.5 (0.5)	0.5	1 (0)	0 (1)	1	1	0.5
Q2	0.5 (0.5)	0.5	0.5 (0.5)	0.5 (0.5)	0.5	0.5	1
Q3	0.25 (0.75)	0.75	0 (1)	0.5 (0.5)	1	0.5	1.16

A larger t represents a stronger bias towards observations. We will demonstrate the effect of this update scheme in Subsection V-B.

C. Calculation of L^a during an interaction

From Equation (5), picking the best query at node "a" to minimize L^a involves calculation of the reduction factor ρ^a, group reduction factor ρ^a_k and conditional probabilities $\pi_{\Theta^a_k}/\pi_{\Theta^a}$ for all K products and all candidate queries. As demonstrated earlier, calculating these proportions in the group identification example from Figure 1 and Table I was straight forward. However, the same calculation task is more involving in preference elicitation, as the calculation of probabilities requires the probability density of the partworth vector $\mathbf{w}$ as an input, denoted as $p(\mathbf{w})$ henceforth.

Below we first explain how $p(\mathbf{w})$ can be approximated from a given Π. We then demonstrate how the conditional probability $\pi_{\Theta^a_k}/\pi_{\Theta^a}$ is calculated based on $p(\mathbf{w})$ and all query responses prior to node "a". The calculation of ρ^a and ρ^a_k follows the same method and thus will not be reiterated. We then introduce an efficient way to perform these calculations, in order for GGBS to be computationally affordable for real-time human-computer interactions.

For clear exposition, we start with Figure 3. Here, three products A, B and C are embedded in a two-dimensional space. The unit circle centered at the origin represents the feasible space of the partworth vector $\mathbf{w}$. Each colored segment of the circle is a sub-space for $\mathbf{w}$ within which all $\mathbf{w}$s share the same most preferred product. For example, a user will prefer product A the

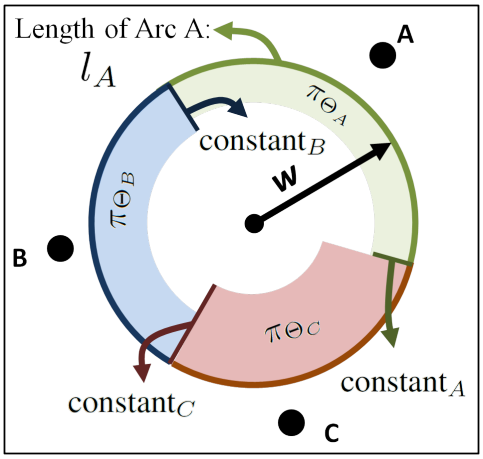


Fig. 3. Sample case with three products in $\mathbb{R}^2$. Best viewed in color.

most when $\mathbf{w}$ is on "Arc A", within which the utility $\mathbf{w}^T\mathbf{x}_A$ is higher than $\mathbf{w}^T\mathbf{x}_B$ and $\mathbf{w}^T\mathbf{x}_C$. We denote by l_A the length of "Arc A" and the same applies to products B and C. Note that when the number of product features is greater than two ($D > 2$), l_A (and its counterparts for B and C) will be the area of a segment of the unit hypersphere. Formally we have

$$l_A = \int_{||\mathbf{w}||=1} \mathbb{1}\{\mathbf{w} \text{ on "Arc A"}\} d\mathbf{w}, \tag{8}$$

where $\mathbb{1}\{\text{conditions}\}$ is an indicator function that takes 1 when the conditions are true and 0 otherwise.

Let $p(\mathbf{w})$ be the probability density of $\mathbf{w}$ so that $\int_{||\mathbf{w}||=1} p(\mathbf{w}) d\mathbf{w} = 1$, representing how the partworth vector is distributed. While $p(\mathbf{w})$ is unknown, it can be approximated by a given Π. More specifically, we assume $p(\mathbf{w})$ to be piecewise uniform with parameters constant$_k$ for $k = A, B, C$ as marked in Figure 3:

$$\begin{aligned} p(\mathbf{w}) = \quad & \mathbb{1}\{\mathbf{w} \text{ on "Arc A"}\}\text{constant}_A \\ & + \mathbb{1}\{\mathbf{w} \text{ on "Arc B"}\}\text{constant}_B \\ & + \mathbb{1}\{\mathbf{w} \text{ on "Arc C"}\}\text{constant}_C. \end{aligned} \tag{9}$$

In order to determine constant$_k$ for $k = A, B, C$, first notice that π_{Θ^A}, the area shaded in light green in Figure 3, can be calculated as:

$$\begin{aligned} \pi_{\Theta^A} &= \int_{||\mathbf{w}||=1} p(\mathbf{w}) \mathbb{1}\{\mathbf{w} \text{ on "Arc A"}\} d\mathbf{w} \\ &= \text{constant}_A l_A. \end{aligned} \tag{10}$$

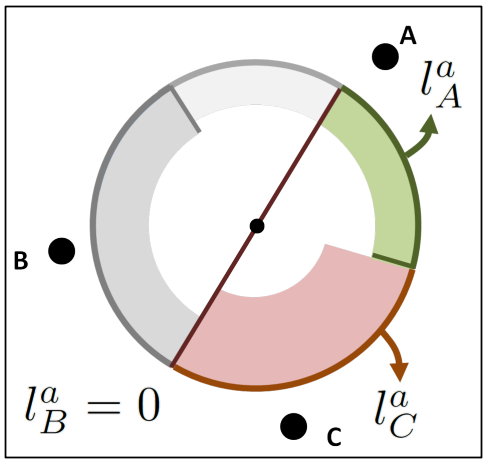


Fig. 4. The remaining space of $\mathbf{w}$ when the user prefers product C more than B.

When π_{Θ_A} is initialized or updated by the method described in Subsection IV-B and l_A is calculated by Equation (8), we can determine constant_A as

$$\text{constant}_A = \pi_{\Theta_A}/l_A. \tag{11}$$

The same applies to constant_B and constant_C. These parameters of $p(\mathbf{w})$ will remain fixed during an interaction and will be updated along with Π.

The probability masses $\pi_{\Theta_k^a}$ for $k = 1, \ldots, K$ at node "a" can then be calculated based on the given $p(\mathbf{w})$. Following the example in Figure 3, let us assume that at the first internal node "a", the pair of products B and C has been queried and the user has chosen product C over B. Under this response, the feasible space of $\mathbf{w}$ is halfed as shown in Figure 4, so that $\mathbf{w}^T\mathbf{x}_C > \mathbf{w}^T\mathbf{x}_B$ is ensured for any $\mathbf{w}$ drawn from the remaining feasible space. While Arc B becomes infeasible, Arc C and part of Arc A still remain. We denote by l_k^a, $k = A, B, C$, the lengths of the remainders of the three arcs, where $l_B^a = 0$. The conditional probability of product A being the most preferred at this node is:

$$\frac{\pi_{\Theta_A^a}}{\pi_{\Theta^a}} = \frac{\text{constant}_A l_A^a}{\sum_{k=A,B,C} \text{constant}_k l_k^a}. \tag{12}$$

From Equations (11) and (12), we see that calculation of constant_k and $\pi_{\Theta_k^a}$ depends on that of the arc lengths or in general surface areas of a hypersphere. Numerical integration over irregular surfaces in a high dimensional space is costly, and will lead to undesirable delayed responses in user-computer interactions. To this end, we introduce a method inspired by Tong and Koller [19]

that efficiently approximates these arc lengths. Also note that, instead of directly approximating the arc lengths $\gamma_k := l_k^a / \sum_{k'=A,B,C} l_{k'}^a$ since they are unitless. For example, Equation (12) can be rewritten as:

$$\frac{\pi_{\Theta_A^a}}{\pi_{\Theta^a}} \approx \frac{\text{constant}_A \gamma_A}{\sum_{k=A,B,C} \text{constant}_k \gamma_k}. \tag{13}$$

Let us take the approximation of γ_A as an example. As shown in Figure 5, we first find the largest circle that centers on the unit circle at $\mathbf{w}$, and is tangent to one of the boundaries of "Arc A". We then use the radius of the found circle to approximate the length of the corresponding arc. To implement this idea, consider the center of this circle to be some partworth vector $\mathbf{w}$, the radius of the circle will be $r_A = \min\{\mathbf{w}^T \mathbf{d}_{AB}, \mathbf{w}^T \mathbf{d}_{AC}\}$, where $\mathbf{d}_{AB}$ and $\mathbf{d}_{AC}$ are normalized vectors of product feature differences:

$$\begin{aligned} \mathbf{d}_{AB} &= \frac{\mathbf{x}_A - \mathbf{x}_B}{||\mathbf{x}_A - \mathbf{x}_B||_2} \\ \mathbf{d}_{AC} &= \frac{\mathbf{x}_A - \mathbf{x}_C}{||\mathbf{x}_A - \mathbf{x}_C||_2}. \end{aligned} \tag{14}$$

The center of the largest circle can then be found by solving the following convex problem:

$$\begin{aligned} \underset{\mathbf{w}:\ ||\mathbf{w}||=1}{\text{maximize}} \quad & \min\{\mathbf{w}^T \mathbf{d}_{AB}, \mathbf{w}^T \mathbf{d}_{AC}\} \\ \text{subject to:} \quad & \mathbf{w}^T \mathbf{d}_{AB} \geq 0 \\ & \mathbf{w}^T \mathbf{d}_{AC} \geq 0 \end{aligned} \quad . \tag{15}$$

The solution to Problem (15), $\mathbf{w}^*$, can be efficiently searched using a solver for linear support vector machines such as [5] and [17], provided that the utility function is linear. Since the feasible space of the partworth vector $\mathbf{w}$ is in general of $D-1$ dimensions, we use the following approximation:

$$\gamma_A \approx \frac{{r_A}^{D-1}}{\sum_{k=A,B,C} {r_k}^{D-1}}. \tag{16}$$

Equations (13) and (16), together with Problem (15) are used to approximate conditional probabilities $\pi_{\Theta_k^a}/\pi_{\Theta^a}$. Since the values of ρ^a and ρ_k^a are also based on some proportions of arc lengths, their approximation shares the same procedure as presented. We will demonstrate in Subsection V-C that this method leads to a close approximation of $\pi_{\Theta_k^a}/\pi_{\Theta^a}$ and has practical computational cost.

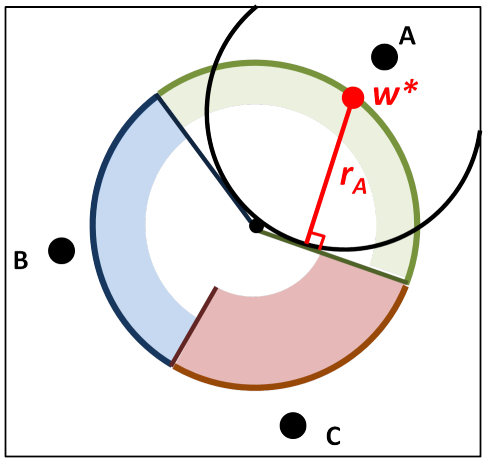


Fig. 5. Approximation of $l_A/\sum_{k=A,B,C} l_k$: The radius of the circle (highlighted as the red solid line perpendicular to the tangential boundary) is used for approximation.

D. Summary of the algorithm

Figure 6 summarizes the proposed GGBS algorithm with efficient approximation for conditional probabilities. The algorithm will be denoted as "appGGBS" hereafter. In order to further speed up the calculation, the algorithm only examines candidate queries related to the four products with the highest conditional probabilities in each turn. In addition, for each query, the calculation of ρ_k^a is only performed on products with conditional probabilities greater than $10^{-3}/K$. In theory, the algorithm should terminate when the conditional probability of a product reaches 1, or equivalently, when the Shannon entropy reaches 0. In practice, it terminates when the conditional probabilities of any $K-1$ products are less than $10^{-3}/K$.

V. Simulated Experiments

A. Performance comparison: appGGBS versus EGO

Prior to experiments with real users we explore the behavior of the algorithm through simulations. We compare the performance of the appGGBS algorithm against a heuristic method previously proposed in [14]. This existing algorithm, referred to as "EGO", is inspired by the Efficient Global Optimization method widely adopted in solving black-box optimization problems [10]. In the same interative pairwise comparison setting, EGO finds a candidate product with high predicted utility and high variance in prediction, and pairs this product with the current

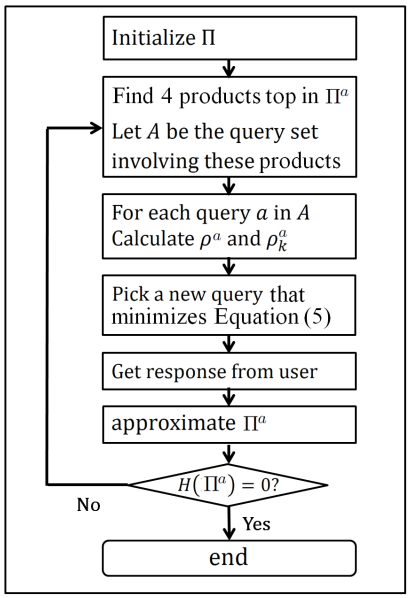


Fig. 6. appGGBS flowchart. $\Pi_a := (\pi_{\Theta_k^a})_{k=1}^K$.

preferred one in the next query. Previous studies on EGO has looked into various schemes for selecting the next sample (product) by weighing differently the predicted utility and the variance of prediction [15] in the context of non-convex optimization. In order to exclude the effect of weighting, here we simplify EGO to use only predicted utility as the query criterion, i.e., the new product to be queried will be the one that has the highest predicted utility among all remaining candidates. This is a reasonable heuristic when the utility function to be maximized is linear. In all experiments, the EGO algorithm is set to have the same termination criterion as appGGBS. Figure 7 summarizes the flow of EGO. The performance of appGGBS and EGO is tested under settings with various numbers of candidate products embedded in the feature space $\mathbb{R}^D$, with $D = 10, 15$ and 20. For each case, we generate products with features randomized through i.i.d. standard normal distribution and projected to the unit sphere. This allows all product feature vectors to have the same norm and thus non-zero probability to be the preferred one, i.e., $l_k > 0$ for all $k = 1, ..., K$. This can be considered as the worst case scenario for a given number of products as none of them is dominated by others and thus cannot be eliminated from the candidate set. For each case given K and D, 100 random partworth vectors are drawn uniformly from the unit sphere to represent 100 users' preferences, and interactions using appGGBS and

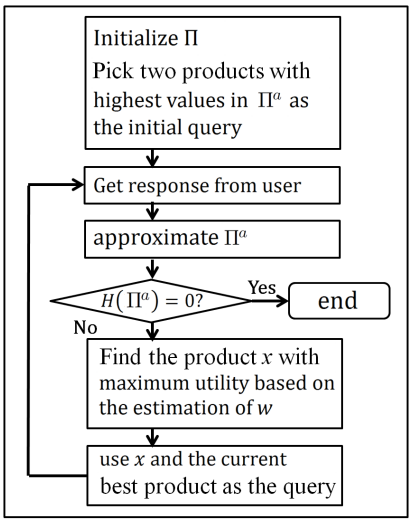


Fig. 7. EGO flowchart. $\Pi_a := (\pi_{\Theta_k^a})_{k=1}^K$.

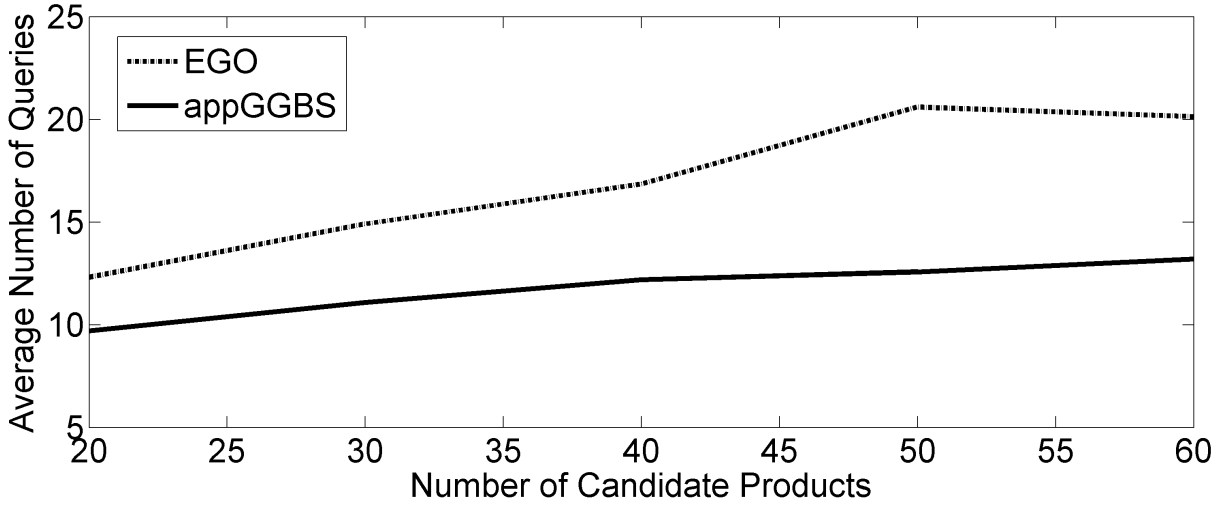
EGO are simulated using these partworth vectors. Figure 8 compares the average numbers of queries needed for each K and D using appGGBS and EGO.

Under these test conditions, appGGBS has overall better performance than EGO, especially when the number of candidate products is large. Nonetheless, with increasing D, we also observe the performance of the two algorithms under small K becomes similar. In fact, EGO outperforms appGGBS under large D and small K scenarios, as is shown in the case of $D = 15$. This suggests that EGO can be a good heuristic method to minimize the expected number of queries when the number of products is relatively small given the dimensionality of the feature space.

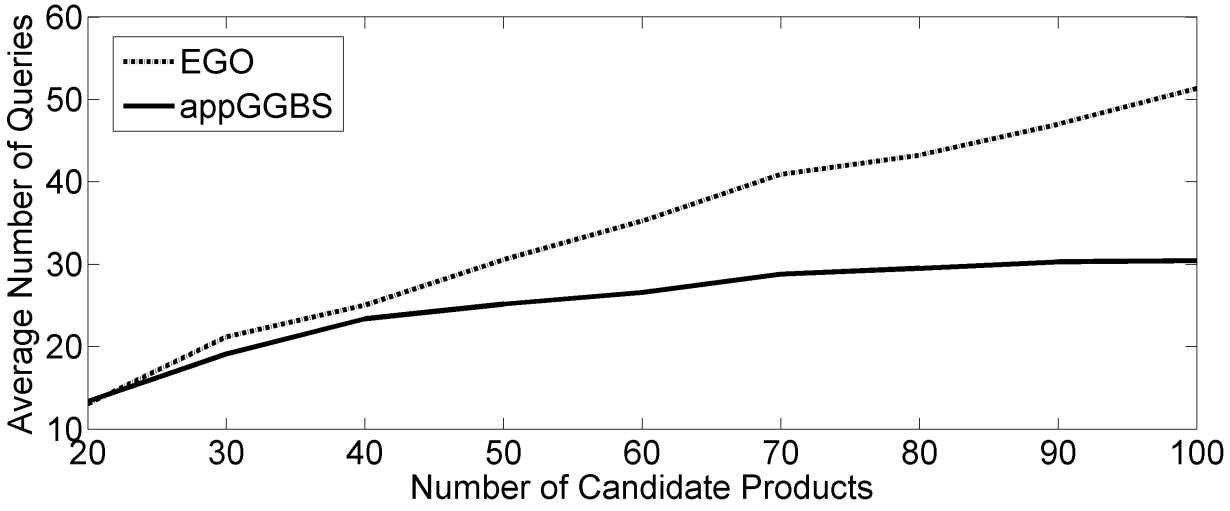
Let us also check the response time of both algorithms, as it is essential for human-computer interactions. Figure 9 shows the average response time for generating a new query using appGGBS and EGO, using the liblinear package for solving Problem (15). While appGGBS is computationally more expensive than EGO, its cost is almost linear with respect to K. Considering that the query size for human-computer interaction is often limited to a small number, the response time of appGGBS is practical for such interactions.

B. Performance of the update scheme on Π

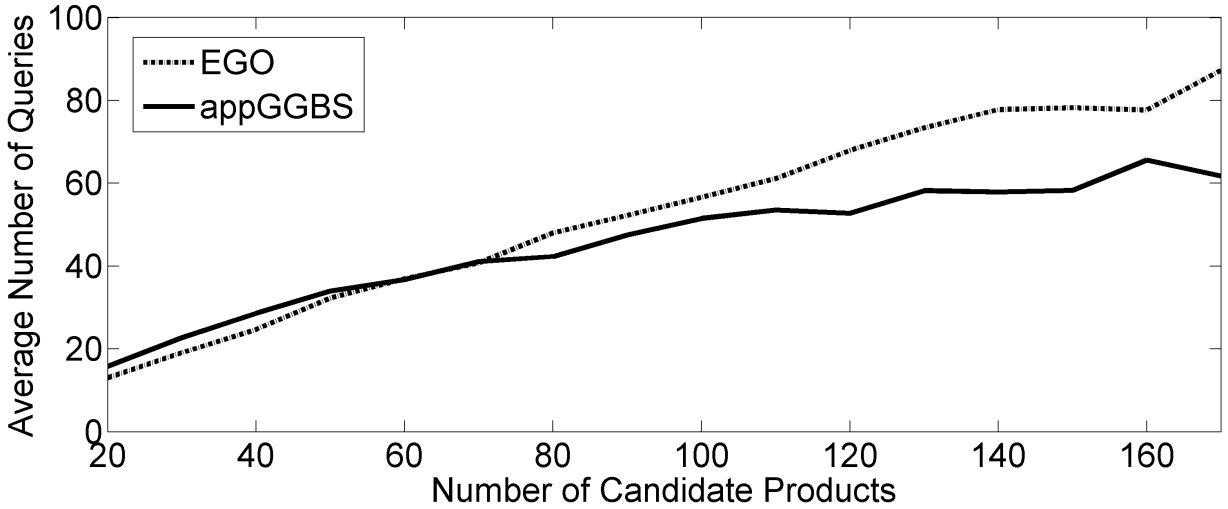
We demonstrate now that the performance of appGGBS and EGO can be improved if the prior knowledge Π can be updated according to the observed preferred products of users. To do



(a) D=5

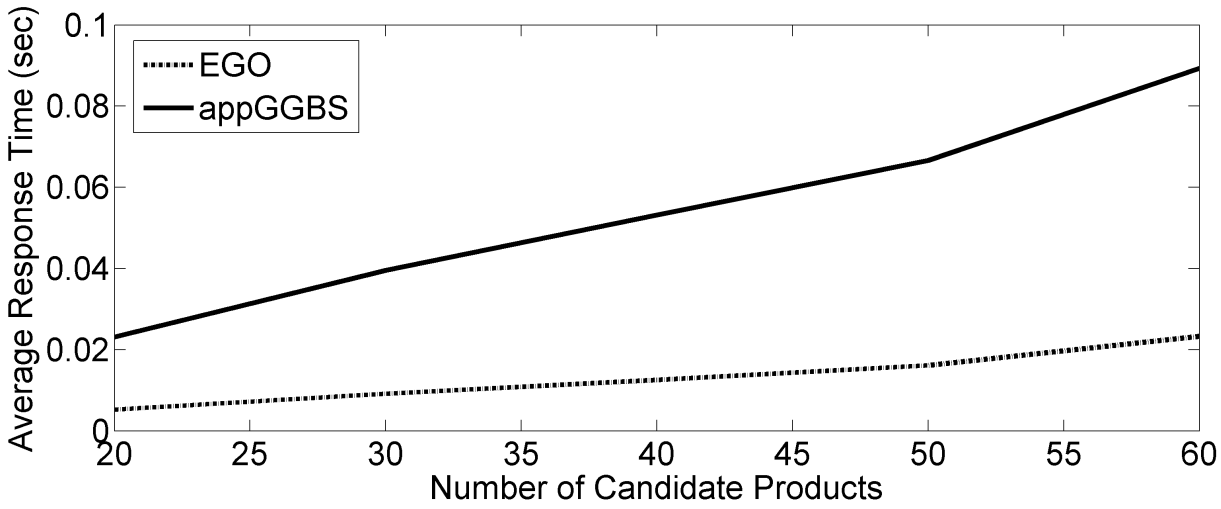


(b) D=10

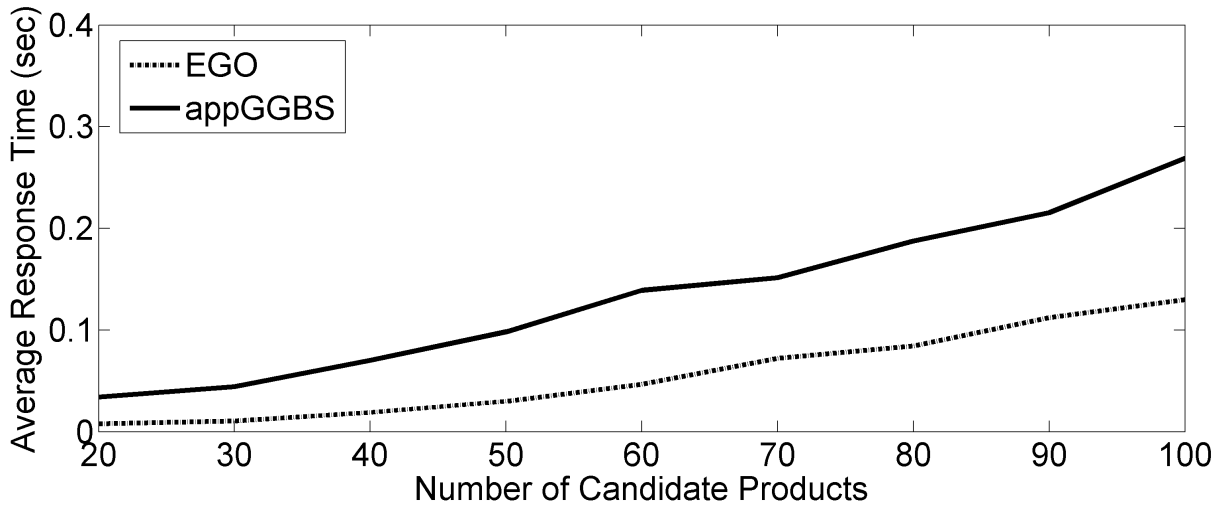


(c) D=15

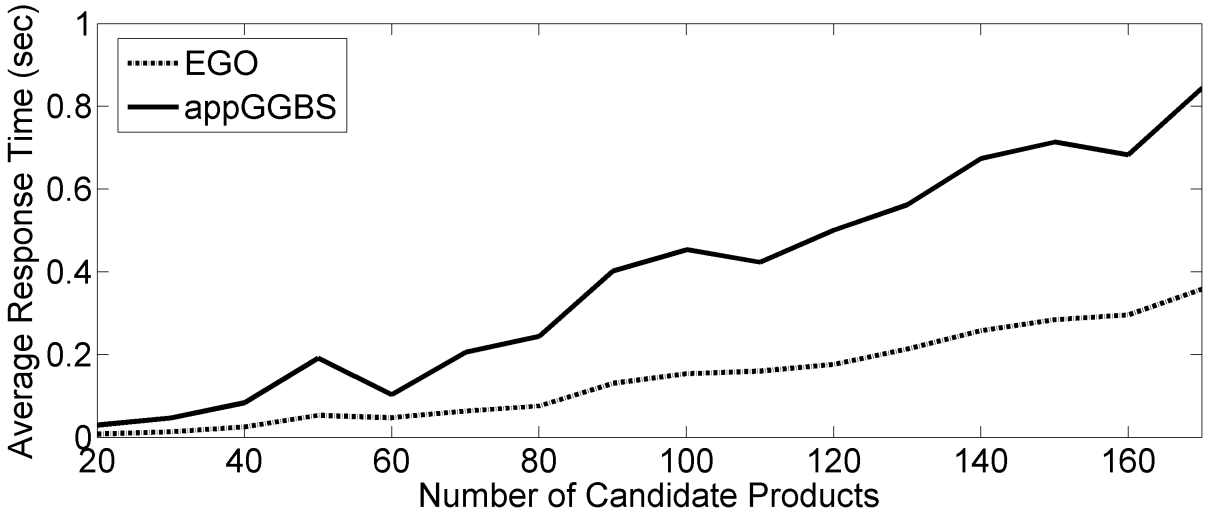
Fig. 8. Performance of appGGBS and EGO under $D = 5, 10$ and 15



(a) D=5



(b) D=10



(c) D=15

Fig. 9. Response time per query of appGGBS and EGO under $D = 5, 10, 15$

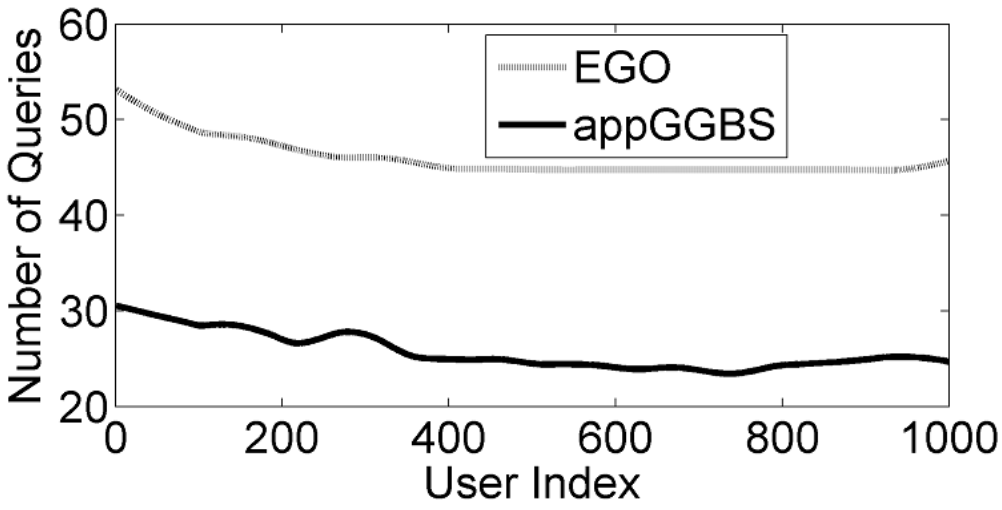
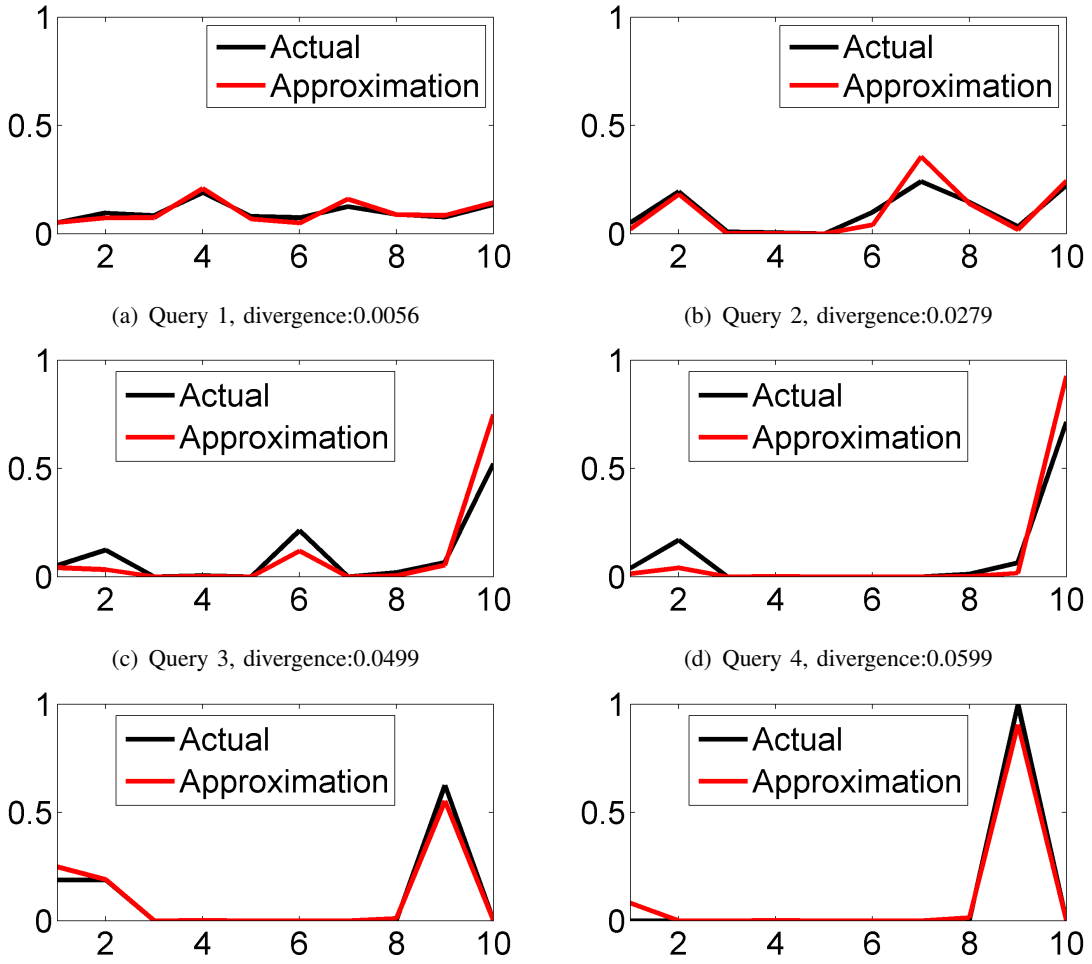


Fig. 10. Updating Π lowers the number of queries needed in a long run.

so, we use a simulation with $D = 10$ and $K = 100$. We set up a sequence of 1000 simulated users whose preferred products are among a fixed set of 20 products out of the total of 100 candidates. Both appGGBS and EGO start with the same initial guess that each product shares the same probability. This distribution is then updated using the method described in Subsection IV-B with parameter $t = 1$. Figure 10 shows the resulting number of queries needed for each user coming into the interaction. Note that when the set of preferred products from the user population is small, e.g., the entire population prefers either product A or B, EGO can perform better than appGGBS since the algorithm starts by querying the two products with the highest probabilities.

C. Performance of the approximation method

For completeness, we now examine the accuracy of the proposed approximation method for calculating conditional probabilities $\pi_{\Theta_k^a}/\pi_{\Theta^a}$. To illustrate, we run appGGBS on a simulated experiment with $D = 5$ and $K = 10$. During the query process, we compare the approximation of $\pi_{\Theta_k^a}/\pi_{\Theta^a}$, for $k = 1, ..., 10$, with their "actual" values. These "actual" values are calculated using the following steps: We uniformly scatter a large amount (10^4) of points on the unit hypersphere where w resides; considering each point as a realization of the random vector w, we label the point by its induced top-ranked product; to calculate $\pi_{\Theta_k^a}/\pi_{\Theta^a}$, we count the number



(a) Query 1, divergence:0.0056

(b) Query 2, divergence:0.0279

(c) Query 3, divergence:0.0499

(d) Query 4, divergence:0.0599

(e) Query 5, divergence:0.0093

(f) Query 6, divergence:0.0492

Fig. 11. The actual and approximated $\pi_{\Theta_k^a}/\pi_{\Theta^a}$ for $k = 1, ..., 10$ during an appGGBS run in a simulated experiment with $D = 5$ and $K = 10$. The x-axis represents indices for the 10 candidate products, and the y-axis for conditional probabilities $\pi_{\Theta_k^a}/\pi_{\Theta^a}$.

of points labeled by product k within the feasible spherical space under the cumulative responses up to node "a". Next, the count is divided by the total sample size (10^4) to represent the value of $\pi_{\Theta_k^a}/\pi_{\Theta^a}$. We use Jensen-Shannon divergence to measure the difference between the "actual" and the approximated distributions of $\pi_{\Theta_k^a}/\pi_{\Theta^a}$. Figure 11 compares the two distributions in the first six queries during the simulated experiment. This result demonstrates the accuracy of the proposed approximation method.

VI. Real-user Experiment

A. Experiment setup

In the real-user experiment, we examine identifying the most preferred laptops. To this end, a list of candidate laptops is created based on four product features: Screen size, storage size, CPU and battery. We use three levels of screen size: 11, 13 or 14 inch; two levels of storage: 128G or 256G; three levels of CPU: i3, i5 or i7; and two levels of battery: half or full day capacity. The laptops are then priced based on their features according to market data. The feature vector is encoded as:

$$\begin{aligned} x = \quad & (\delta_{13in}, \delta_{14in}, \delta_{11in,256G}, \delta_{13in,256G}, \\ & \delta_{14in,256G}, \delta_{i5}, \delta_{i7}, \delta_{full}, \text{price}), \end{aligned} \tag{17}$$

while the linear utility model follow as:

$$\begin{aligned} u = & w_1\delta_{13in} + w_2\delta_{14in} + w_3\delta_{11in,256G} \\ & + w_4\delta_{13in,256G} + w_5\delta_{14in,256G} + w_6\delta_{i5} \\ & + w_7\delta_{i7} + w_8\delta_{full} + w_9\text{price}. \end{aligned} \tag{18}$$

Here all δs are binary variables: When the laptop is 13 inch, we set $\delta_{13in} = 1$ and when it is 11 inch, we set both δ_{13in} and δ_{14in} to zero. Similarly, $\delta_{i5} = 1$ when the laptop has an i5 CPU, and $\delta_{full} = 1$ when the battery lasts a full day. Notice that we set the partworths on storage levels separately for different screen sizes to increase flexibility of the utility model. For instance, an 11 inch laptop with 256G storage will have $\delta_{11in,256G}$ set to 1. In other words, the partworths $w_{3,4,5}$ represent increases in the utility when the storage upgrades from 128G to 256G for a laptop size of 11, 13 and 14 inches, respectively.

To summarize, a laptop is represented by 8 binary digits and one real value for price. Thus the experiment will have 36 candidate products ($K = 36$) and a feature space of nine dimensions ($D = 9$). The prices are normalized by first dividing by the highest price and then subtracting by the mean. This treatment is necessary as otherwise some laptops can be dominated with 0 prior probability to be the most preferred product. The full laptop set is listed in the Appendix.

The online experiment is set up on the Google application engine (see Figure 12 for the interface and [13] for the living web application) and announced through Amazon Mechanical Turk [1]. The two algorithms, appGGBS and EGO, are drawn with an equal chance when a participant initializes an interaction.

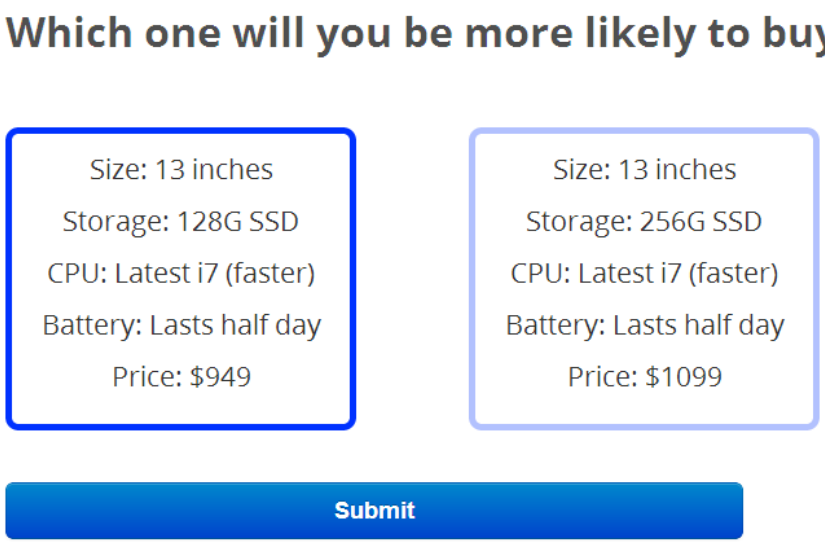


Fig. 12. The interface for the real-user experiment

B. *Results*

A total of 50 responses were collected, among which 22 valid responses are from appGGBS and 19 from EGO. The average number of queries needed for appGGBS is 11.0 with a standard deviation of 1.7, while these numbers for EGO are 15.5 and 3.2, respectively. A simple t-test shows that appGGBS requires less queries than EGO in this experiment (with a p-value less than 0.001).

There are also nine invalid responses generated: During these interactions, conditional probabilities of all candidate products went to zero, leading to termination of the interaction without identifying a preferred laptop. There could be two reasons for this to happen: (1) User responses could be inconsistent in a way that no partworth vector could be found to be consistent with the given set of user comparisons; or (2) the encoded utility model is not flexible enough to represent the preferences of these users.

VII. Discussion

A. *Relationship with recommender systems*

This research is closely related to the topic of recommender systems, as the two share the similar motivation of finding the most preferred product of a user in real time.

In the problem setting of a recommender system, it is usually assumed that a matrix spanned by users and products is given, where the elements represent "scores" of products by users. Since the matrix is sparse with only a few elements filled in, the challenge of building a recommender system is to estimate scores of each product for each user [3], so that the predicted best product can be recommended to a specific user.

The critical difference between the goal of this paper and that of a recommender system is that we pursue a fast convergence to the most preferred product. Therefore while recommender systems are designed to passively garner user data, we designed the interaction to actively collect user responses. In practice, a recommender system is more suitable for people to explore products while an active query strategy as proposed in this paper is more applicable to users who have an idea of what they want and would like to quickly find a product that fits their preference. In addition to its usage in online shopping, the proposed query method could also be useful to effectively collect user feedback on new product alternatives through a crowd.

A recommender systems can be incorporated to better approximate Π, the prior distribution of the most preferred product of each user, which in turn helps to make our query algorithm more effective.

B. Relationship with adaptive conjoint analysis

Another related research area is adaptive choice-based conjoint analysis (ACBC) [2], [20] or, more generally, active learning [16], [18]. The difference of the work here from ACBC is that we are interested in finding the most preferred product, while the goal of conjoint analysis is to estimate the distribution of partworth w, or equivalently, the distribution of full rankings Π_Θ. It can be shown that, by considering each ranking as a single-element "group", the proposed GGBS algorithm can be transformed into the classic Generalized Binary Search algorithm equivalent to the uncertainty sampling strategy (also called "utility balance" in [2]) commonly used in active learning tasks [16].

C. Nonlinear utility

So far we assumed a linear utility function and that all product features are known. In some applications, this assumption may not hold. Notice, however, that in the appGGBS algorithm, the linearity assumption is only used in Problem (15) where conditional probabilities $\pi_{\Theta_a^k}$

are approximated. While kernelization of Problem (15) can help to circumvent the linearity assumption, it could also prolong the interaction significantly. In the extreme case where infinite dimensions exist in the kernel space, each product will have a non-zero probability to be the best one as long as it has not been explicitly labeled as the inferior product from a pairwise comparison. Indeed, there could always be some reason that one product is the most preferred, regardless of how comparisons are made on other products. Therefore, for the appGGBS algorithm to be effective, it is recommended that a reasonable utility function, with product features, is derived first.

VIII. Conclusions and Future Work

We examined the problem of identifying users' most preferred products from a candidate product set through pairwise comparisons. Unlike earlier heuristic approaches to this problem, the appGGBS algorithm proposed in this work directly minimizes the expected number of queries and has a tractable computational cost. Besides its theoretical foundation, the algorithm also showed better performance both in simulated and real-user experiments than the EGO algorithm in [14].

The appGGBS algorithm relies on the existence of known product features and a utility model that captures preferences of different users. As we observed from the real-user experiment, a manually tuned utility model could lead to a failed interaction due to its inability to capture the nonlinearity of the user's preference. On the other hand, the proposed algorithm should also be improved so that it performs robustly under inconsistent user responses.

IX. Acknowledgement

This research was partially supported by the National Science Foundation Grant CMMI No. 1266184 and Award No. 0953135, and by the Automotive Research Center, a U.S. Army Center of Excellence in Modeling and Simulation of Ground Vehicle Systems headquartered at the University of Michigan. This support is gratefully acknowledged.

References

[1] Amazon mechanical turk. http://www. mturk. com, recent access: 10/21/2013.

[2] J. Abernethy, T. Evgeniou, O. Toubia, and J. Vert. Eliciting consumer preferences using robust adaptive choice questionnaires. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):145–155, 2008.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[4] G. Bellala, S. Bhavnani, and C. Scott. Group-based active query selection for rapid diagnosis in time-critical situations. *IEEE Transactions on Information Theory*, 58(1):459–478, 2012.

[5] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[6] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, volume 1, pages 97–102, 1999.

[7] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[8] K. Jamieson and R. Nowak. Active ranking using pairwise comparisons. *arXiv preprint arXiv:1109.3701*, 2011.

[9] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

[10] D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[11] J. Kelly and P. Papalambros. Use of shape preference information in product design. In *International Conference on Engineering Design, Paris, France*, 2007.

[12] H. Kim and S. Cho. Application of interactive genetic algorithm to fashion design. *Engineering Applications of Artificial Intelligence*, 13(6):635–644, 2000.

[13] Y. Ren. Web implementation of the appGGBS algorithm. Online at: http://ggbsweb.appspot.com, recent access: 10/21/2013.

[14] Y. Ren and P. Papalambros. A design preference elicitation query as an optimization process. *Journal of Mechanical Design*, 133(11):111004–111004–9, 2011.

[15] M. Sasena. *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 2002.

[16] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[17] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[18] S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *Proceedings of the 9th ACM International Conference on Multimedia*, pages 107–118, 2001.

[19] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[20] O. Toubia, J. Hauser, and D. Simester. Polyhedral methods for adaptive choice-based conjoint analysis. *Journal of Marketing Research*, 41(1):116–131, 2004.

APPENDIX

APPENDIX: CANDIDATE PRODUCTS FOR SECTION VI

Screen (inch)	Storage (G)	CPU	Battery	Price (USD)
11	128	i3	half-day	449
11	128	i3	full-day	649
11	128	i5	half-day	699
11	128	i5	full-day	899
11	128	i7	half-day	949
11	128	i7	full-day	1149
11	256	i3	half-day	599
11	256	i3	full-day	799
11	256	i5	half-day	849
11	256	i5	full-day	1049
11	256	i7	half-day	1099
11	256	i7	full-day	1299
13	128	i3	half-day	449
13	128	i3	full-day	649
13	128	i5	half-day	699
13	128	i5	full-day	899
13	128	i7	half-day	949
13	128	i7	full-day	1149
13	256	i3	half-day	599
13	256	i3	full-day	799
13	256	i5	half-day	849
13	256	i5	full-day	1049
13	256	i7	half-day	1099
13	256	i7	full-day	1299
14	128	i3	half-day	449
14	128	i3	full-day	649
14	128	i5	half-day	699
14	128	i5	full-day	899
14	128	i7	half-day	949
14	128	i7	full-day	1149
14	256	i3	half-day	599
14	256	i3	full-day	799
14	256	i5	half-day	849
14	256	i5	full-day	1049
14	256	i7	half-day	1099
14	256	i7	full-day	1299

Fig. 13. All candidate products for the real-user experiment