Proceedings of the ASME 2013 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2013
August 4-7, 2013, Portland, United States

DETC2013-13059

A SCALABLE PREFERENCE ELICITATION ALGORITHM USING GROUP GENERALIZED BINARY SEARCH

Yi Ren*
Mechanical Engineering
University of Michigan
Ann Arbor, Michigan, 48109
Email: yiren@umich.edu

Clayton Scott
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan, 48109
Email: clayscot@umich.edu

Panos Y. Papalambros
Mechanical Engineering
University of Michigan
Ann Arbor, Michigan, 48109
Email: pyp@umich.edu

ABSTRACT

We examine the problem of eliciting the most preferred designs of a user from a finite set of designs through iterative pairwise comparisons presented to the user. The key challenge is to select proper queries (i.e., presentations of design pairs to the user) in order to minimize the number of queries. Previous work formulated elicitation as a blackbox optimization problem with comparison (binary) outputs, and a heuristic search algorithm similar to Efficient Global Optimization (EGO) was used to solve it. In this paper, we propose a query algorithm that minimizes the expected number of queries directly, assuming that designs are embedded in a known space and user preference is a linear function of design variables. Besides its theoretical foundation, the proposed algorithm shows empirical performance better than the EGO search algorithm in both simulated and real-user experiments. A novel approximation scheme is also introduced to alleviate the scalability issue of the proposed algorithm, making it tractable for a large number of design variables or of candidate designs.

1 Introduction

A typical online shopping platform takes in a user query and outputs a list of relevant designs (products). 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, a recommender system can further guide users to products they might be interested in by comparing their browsing behaviour and that of other users [2]. While a combination of these methods helps to locate preferred designs of users, better shopping experience could be achieved if we can shorten the search list by incorporating users' preferences better. Consider a user looking for a new laptop as a running example. The user may prefer designs within a certain budget. Yet a price tag beyond the budget limit is not necessarily rejected if the laptop offers 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 shorten the search is to introduce comparison questions before generating the search list. For instance, one may show a few laptop models and ask (query) the user which design(s) he prefers more. Based on the response, another query can be adaptively created with designs more relevant to the user. In the running example, a set of laptops with the most preferred price-performance combinations can be queried. Such an interaction helps down-scale the search list to include only the most preferred designs of the user. In addition, unlike filtering and sorting tools, the interaction encourages users to provide their detailed preference information in an indirect manner. The collected crowd preference can then be used to facilitate better design decision making. Like any human-computer interactions, the queries we propose need to be short and efficient, leading to the key question: How can queries be adaptively designed based on user responses? We show in this paper that this problem can be formulated as a group identification task [4] and that its solution shares the theoretical foundation of the classic algorithm

*Address all correspondence to this author.

used in 20-question games. We then propose and test a novel treatment to address the computational issue of the algorithm for large design space and large design sets.

To be specific, we investigate a human-computer interaction that identifies designs from a fixed set that are most preferred by an individual. The interaction comprises a sequence of queries, in the form of pairwise comparisons. Each query contains two designs from the given design set $\mathcal{X} = \{x_1, x_2, ..., x_K\} \in \mathbb{R}^D$, K being the total number of designs and D the number of design variables (attributes). The individual user responds to each query by evaluating and choosing one design according to his/her preference. The interaction terminates when one design is estimated to be the most preferred design with probability one.

The challenge here is to present as few queries as possible in eliciting the most preferred design. Interactive genetic algorithms were previously introduced that use heuristics to find near-optimal designs based on evolutionary operations [12, 11]. More recently a response-surface method was proposed that iteratively refines a user preference model based on choice responses [6, 9] and creates queries predicted to be preferred according to the model [13]. This paper presents an algorithm that minimizes directly the actual expectation of the query size, under a finite design set. This approach is inspired by the Group Generalized Binary Search (GGBS) algorithm for group identification, where the objective is to identify the group label (e.g., category of disease) of an object (e.g., a patient) by querying "yes-no" questions (e.g., "Do you have fever or not?") [3, 4]. We consider rankings of designs as "objects". The rankings with design i being top ranked constitute the ith "group". Under this setting, finding the most preferred design of a user is equivalent to identifying a group. We show that GGBS can be directly applied and that it outperforms the heuristic algorithm previously proposed in [13] in both simulated and real-user experiments.

The paper is organized as follows: We define the problem formally in the remainder of this introductory section. In Section 2, we discuss in detail a scalable GGBS algorithm to solved the defined problem and then test the algorithm using simulations in Section 3. A real-user experiment is conducted and results reported in Section 4. Section 5 offers a discussion and conclusions.

Problem statement

The individual preference on design x_j is modeled as a linear utility $u_j = w^T x_j$, where w is the unknown "partworth" vector lying on a unit sphere: $||w||_2 = 1$. As a note, the norm of w can be interpreted as the randomness in user responses: The smaller the norm, the noisier user responses are. In this study, we will assume that user responses are noiseless, making a variable norm of w unnecessary. We will discuss how this assumption can be removed in future work.

The interaction consists of a sequence of pairwise comparison queries. A query with its response is represented by $q = (x^{(1)}, x^{(2)})$, where design $x^{(1)}$ is more preferred than design $x^{(2)}$, i.e, $u_1 > u_2$ (we can always flip the two designs if the second design from the query is more preferred). We also assume non-existence of indifferent pairs, i.e., $w^T(x^{(1)} - x^{(2)}) \neq 0$ for any query. The candidate query set is denoted as $\mathcal{Q}$, containing all possible $N = K(K-1)/2$ pairwise comparisons, where K is the number of candidate designs. The individual partworth vector w can be estimated based on query responses, as will be shown in Problem (8) from Section 2.

We define a set of objects $\Theta = \{\theta_1, \theta_2, ..., \theta_M\}$ where θ is a ranking of designs, e.g., $\theta := x_1 \succ x_2 \succ ... \succ x_K$, induced by some w. Here M is the number of possible rankings and M is usually much less than $K!$ when a large amount of designs are embedded in a low-dimensional space and utility is a linear function of the design variables [8]. Each object is then labeled with by its top ranked design, e.g., $y_m = 1$ if $\theta_m = x_1 \succ ....$ We denote by $\{\Theta^k\}_{k=1}^K$ the grouping of Θ, where a "group" $\Theta^k = \{\theta_m \in \Theta : y_m = k\}$ represents the rankings for which design k is the top ranked. The set $\Pi_\Theta = \{\pi_{\theta_1}, ..., \pi_{\theta_M}\}$, containing $\pi_{\theta_m} = \Pr(\theta = \theta_m)$, represents the probabilities of the true ranking being θ_m. Following this, we can use $\Pi = \{\pi_{\Theta^1}, ..., \pi_{\Theta^K}\}$ to represent the probability masses of each design being the most preferred one, i.e., $\pi_{\Theta^k} = \sum_{\theta \in \Theta^k} \pi_\theta$. The true Π, however, is induced by the true distribution of w of the population and is usually unknown. We discuss in Section 2 how it can be approximated and updated when preferred designs from users are collected.

We can now use a binary decision tree to represent the query process. On this tree, each internal node has a query from $\mathcal{Q}$, the binary response to which leads to one of the two child nodes and eventually to a leaf node labeled with a design. Each interaction will start with the initial probabilities Π, representing our current belief of how likely each design is to be the most preferred without knowing anything about the current user. At any internal node "a", let $\Theta_a \subseteq \Theta$ be the rankings that reach node "a". Further, we use $\Theta_a^k \subseteq \Theta_a$ for the set of rankings belonging to group k that reach "a". We denote by $\pi_{\Theta_a} := \sum_{\{i:\theta_i \in \Theta_a\}} \pi_{\theta_i}$ the probability mass of the rankings in Θ_a, and likewise $\pi_{\Theta_a^k} := \sum_{\{i:\theta_i \in \Theta_a^k\}} \pi_{\theta_i}$ the probability mass of rankings in Θ_a^k. The distribution Π will then be updated as $\Pi_a = \{\pi_{\Theta_a^1}, ..., \pi_{\Theta_a^K}\}$, based on query responses from the user along the path to "a". By reaching a leaf node l labeled with design k, we have the conditional probability $\frac{\pi_{\Theta_l^k}}{\sum_{k'=1}^K \pi_{\Theta_l^{k'}}} = 1$ and zeros otherwise. The interaction terminates at this point since the most preferred design is confirmed. Notice that multiple leaf nodes can be labeled with the same design. For a given decision tree, let us denote by d_k the expected number of queries needed to reach design k. Then the expected number of queries needed

to identify a preferred design for some initial believe Π will be:

$$L(\Pi) = \sum_{k=1}^{K} \pi_{\Theta^k} d_k. \quad (1)$$

In the next section, we introduce the GGBS algorithm to minimize L in Equation (1) for a given approximation of Π, and update this approximation based on collected preferred designs to further lower L.

2 The Algorithm

2.1 Group Generalized Binary Search (GGBS)

From [3, 4], optimizing Equation (1) over all possible decision trees is NP complete [7]. GGBS greedily minimizes this objective at each internal node. Minimizing the expected query size from Equation (1) is shown to be equivalent to minimizing the following merit at each node of the decision tree:

$$1 - H(\rho_a) + \sum_{k=1}^{K} \frac{\pi_{\Theta_a^k}}{\pi_{\Theta_a}} H(\rho_a^k). \quad (2)$$

Below we first introduce all elements in Equation (2) and then explain its underlying intuition. First let $\{x_a^{(1)}, x_a^{(2)}\}$ be the design pair raised at node "a", and $l(a)$, $r(a)$ be the "left" and "right" child nodes corresponding to $x_a^{(1)} \succ x_a^{(2)}$ and $x_a^{(2)} \succ x_a^{(1)}$. Thus $\Theta_{l(a)}$ and $\Theta_{r(a)} \subseteq \Theta_a$ are the two sets of rankings that fall into the two child nodes from "a". Based on these, the "reduction factor" ρ_a in Equation (2) is defined as

$$\rho_a = \max\{\pi_{\Theta_{l(a)}}, \pi_{\Theta_{r(a)}}\}/\pi_{\Theta_a}, \quad (3)$$

which is the maximum of the two probabilities that the response to the next query is 1 and 0. The "group reduction factor" ρ_a^k is defined by:

$$\rho_a^k = \max\{\pi_{\Theta^k_{l(a)}}, \pi_{\Theta^k_{r(a)}}\}/\pi_{\Theta_a^k}. \quad (4)$$

Finally, we denote the Shannon entropy of a proportion $\pi \in [0, 1]$ by $H(\pi) := -\pi \log_2 \pi - (1-\pi)\log_2(1-\pi)$ and that of a vector Π by $H(\Pi) := -\sum_i \pi_i \log_2 \pi_i$, where we use the limit, $\lim_{\pi \to 0} \pi \log_2 \pi = 0$ to define the value of $0 \log_2 0$.

We will explain in detail in the next subsection how ρ_a, ρ_a^k, π_{Θ_a} and $\pi_{\Theta_a^k}$ can be calculated depending on (i) the embedding of designs in the design space, (ii) the cumulative queries and responses up to node "a", denoted as $\{q\}_a$, and (3) the choice of the current query $\{x_a^{(1)}, x_a^{(2)}\}$.

With Equation (2) explained, we now notice that (i) only the reduction factors ρ_a and ρ_a^k are affected by the choice of query and (ii) the entropy $H(\rho)$ is a concave function maximized at $\rho = 0.5$. Therefore in order to minimize the expected query size, we seek a query that pushes not only ρ_a to 0.5 but also every ρ_a^k to either 0 or 1. This approach is intuitively appealing as such a query will (i) split the remaining probability masses equally, and (ii) put rankings from the same group all in one child.

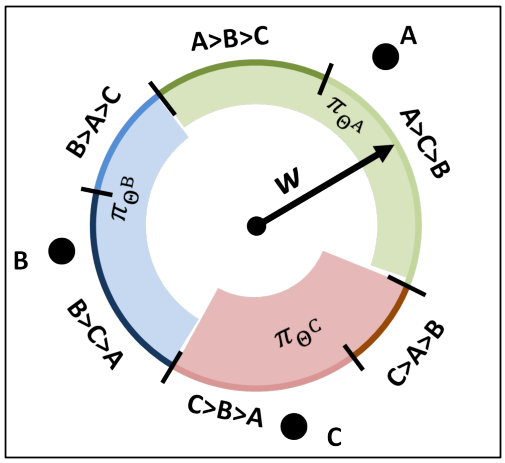


FIGURE 1. Sample case with three designs in $\mathbb{R}^2$

2.2 Geometry interpretation of GGBS

From the above discussion, the GGBS algorithm requires an approximation of Π to start and the values of ρ_a and ρ_a^k to determine which query to make during an interaction. We explain below using an example how Π is calculated and how one query is selected. We then provide efficient approximations of Π, ρ_a and ρ_a^k, in order to alleviate the scalability issue that arises due to a large number of design variables and a large design set.

2.2.1 Calculation of $\pi_{\Theta_a^k}$ We use the simple example in Figure 1 to explain the idea. Three designs are embedded in $\mathbb{R}^2$. The unit circle centered at the origin represents the feasible space of the partworth vector w, while the six arcs in different colors represent segments of w for unique rankings. The cardinality of a group, for example $|\Theta^A|$, is the length of the two combined green arcs, or formally

$$|\Theta^A| = \int_{||w||=1} I_{\{w^T(x_A - x_B)>0, w^T(x_A - x_C)>0\}} dw. \quad (5)$$

At this initial node $a = 0$, $\pi_{\Theta_0^A}$ is the area shaded by light green in the figure, or

$$\pi_{\Theta_0^A} = \int_{||w||=1} p(w) I_{\{w^T(x_A - x_B)>0, w^T(x_A - x_C)>0\}} dw, \quad (6)$$

where $p(w)$ is the density of w. Without any interaction being conducted, we assume w to be uniformly distributed, i.e., $p(w) =$ constant. Thus the three shaded areas should have the same height. From $\sum_{k=A,B,C} \pi_k = 1$, we then have for any node "a":

$$\pi_{\Theta_a^k} = |\Theta_a^k| / \sum_{k=1}^{K} |\Theta_a^k|. \quad (7)$$

2.2.2 Calculation of ρ_a and ρ_a^k We now move on to explain how a query from $\{x_A, x_B\}$, $\{x_A, x_C\}$ and $\{x_B, x_C\}$

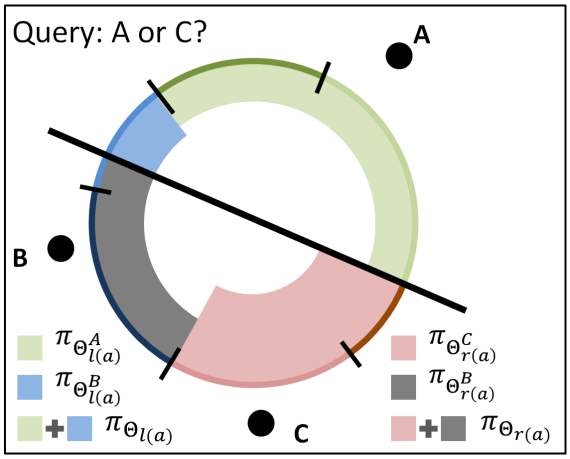


FIGURE 2. Visualization of parameters that contribute to the calculation of ρ_a and ρ_a^k

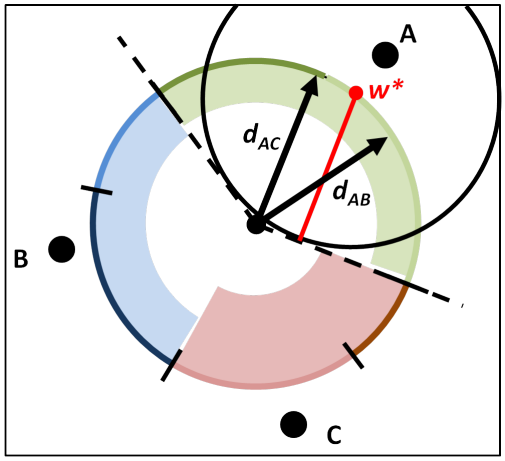


FIGURE 3. Approximation of π_{Θ^A} based on an SVM solution

can be selected based on the criteria from Equation (2). The key is the calculation of ρ_a and ρ_a^k for a given query. To this end, we use Figure 2 to illustrate parameters needed for calculating ρ_a and ρ_a^k, using query $\{x_A, x_C\}$ as an example. Here $l(a)$ is the child node for $A \succ C$ and $r(a)$ for $C \succ A$. Once the parameters $\pi_{\Theta^k_{l(a)}}$ and $\pi_{\Theta^k_{r(a)}}$ are calculated for each design k and a candidate query, the query that yields the minimum value of Equation (2) will be chosen.

2.3 Scalability and approximation of π_{Θ^k} and $\pi_{\Theta^k_a}$

From Equation (7), we see that calculations of probability masses depend on calculating arc lengths in the two-dimensional example or surface areas of a hypersphere in general. Since direct integration over an irregular surface in a high dimensional space as shown in Equation (5) can be costly, we introduce an approximation method following the interpretation of version space for support vector machines (SVM) from [17]. Let us take the calculation of $|\Theta^A|$ as an example. In Figure 3, we show the feasible space of w for A to be the most preferred design, which are the dark and light green arcs combined and bounded by the two doted lines. On this arc, a point w^* can be estimated as the partworth vector for a user who prefers A the most:

$$\begin{aligned} \underset{w:\ ||w||=1}{\text{maximize}} \quad & \min\{w^T d_{AB}, w^T d_{AC}\} \\ \text{subject to:} \quad & w^T d_{AB} \geq 0 \\ & w^T d_{AC} \geq 0 \end{aligned} \quad , \tag{8}$$

where d_{AB} (d_{AC}) is a normalized vector from $B(C)$ to A. Problem (8) is a form of SVM [17] and its solution can be efficiently searched using a linear SVM solver such as [5], provided the utility function is linear. Geometrically, the solution w^* from Figure 3 corresponds to the largest circle centered on the feasible arc and tangent to the boundaries (dotted lines). The radius of the resulting circle, the black solid circle in the figure, is indicated by the red solid line perpendicular to the tangential boundary. We use this radius to approximate the feasible space of w. Specifically for this example and assuming $w^T d_{AB} > w^T d_{AC}$, the radius is calculated as $r = w^T d_{AC}$. We then use r^2 to approximate the proportion of the arc $|\Theta^A|$ over the circumference of the circle. In a more general case, we approximate the proportion of the surface area $|\Theta^A|$ to that of the hypersphere with $(w^T d_{AC})^{D-1}$. We demonstrate in Subsection 3.3 that this approximation method has promising performance.

2.4 Update Π based on observations

The probability masses for designs to be the most preferred ones can be updated as Π^s when s users have completed interaction. With an increase in s, the approximation Π^s will be closer to the true probability masses, which helps to improve the efficiency of the interaction. Therefore, we propose the following simple scheme for updating Π^s: Let $c_k^{(s)}$ be the count of design x_k being the most preferred design from a sequence of s users, then π_{Θ^k} is updated by $\tilde{\pi}^{(s)}_{\Theta^k} = 1 + tc_k^{(s)}$ and $\pi^{(s)}_{\Theta^k} = \tilde{\pi}^{(s)}_{\Theta^k} / \sum_{k=1}^K \tilde{\pi}^{(s)}_{\Theta^k}$, where t is an algorithmic parameter. A larger t allocates probability masses heavier on those observed preferred designs. We demonstrate the effect of this update scheme in Subsection 3.2.

2.5 Summary of the algorithm

Figure 4 summarizes the proposed GGBS algorithm (denoted as "appGGBS" hereafter). In order to speed up the calculations in an iteration, the algorithm only examines queries related to the four designs with the highest probability masses at the moment. In addition, for each query, the calculation of ρ_a^k is only performed on such design ks with probability mass $\pi_{\Theta^k} > 10^{-3}/K$. In theory, the algorithm should terminate

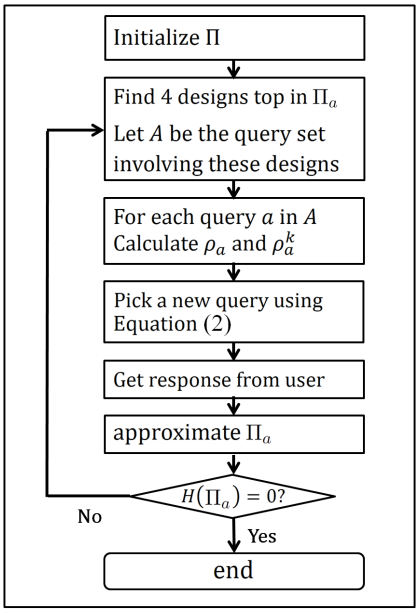


FIGURE 4. appGGBS flowchart

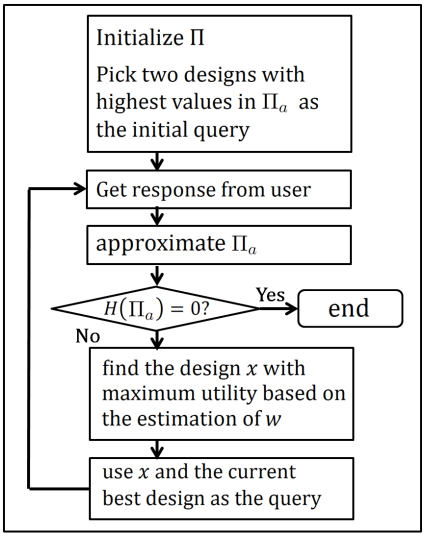


FIGURE 5. EGO flowchart

when the probability mass of a design reaches 1, or equivalently, when the Shannon entropy reaches 0. In practice, it terminates when the probability masses of any $K-1$ designs are less than $10^{-3}/K$.

3 Simulated experiments

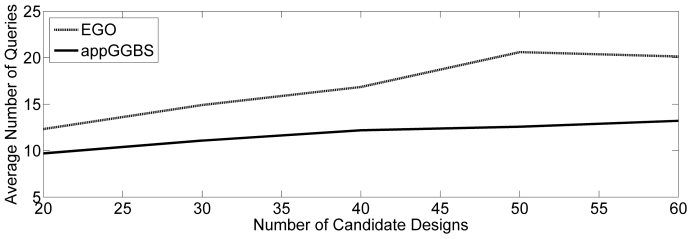
3.1 Performance comparison: appGGBS versus EGO

Prior to experiments with real users we explore the behavior of the algorithm in simulation. We compare the performance of appGGBS against a heuristic algorithm previously proposed in [13]. This existing algorithm, referred to as "EGO", is inspired by the Efficient Global Optimization (EGO) method widely adopted in solving black-box optimization problems [10]. In the same interative pairwise comparison setting, EGO finds a candidate design with high predicted utility and high variance in prediction, and uses this design alongside the current preferred one as the next query. Previous research on EGO has looked into various schemes for selecting the next sample (design) by weighing differently the predicted utility and the variance of prediction [14], in the context of non-convex optimization. To exclude the effect of the algorithmic parameter of EGO from the conclusions in the present work, we simplify EGO to use only predicted utility as a query criterion, i.e., the new design 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 5 summarizes the flow of EGO.

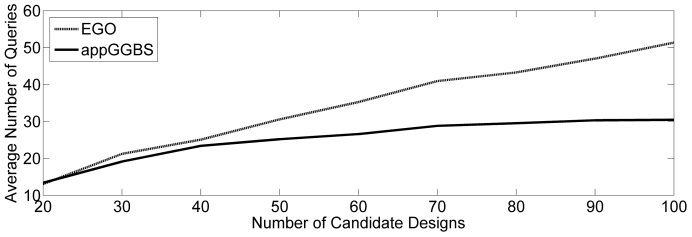
The performance of appGGBS and EGO is tested under settings with various numbers of candidate designs embedded in $\mathbb{R}^D$, with $D = 10, 15$ and 20. We randomize designs through i.i.d. standard normal distribution and project them to the unit sphere in order for each design to have non-zero probability to be the preferred design, i.e., $\pi_{\Theta^k} > 0,\ \forall k = 1, ..., K$. This can be considered as the worst case scenario for a given number of designs as none of them is dominated by others and thus cannot be eliminated from the candidate set. For each K and D, 100 random partworth vectors are drawn uniformly from the unit sphere to represent users. The uniform draw makes the approximated $\Pi_{\mathbf{y}}$ a good estimation of the true probability masses. Figure 6 shows the average performance of appGGBS and EGO.

It can be seen that under these test conditions, appGGBS has overall better performance than EGO, especially when the number of candidate designs is large. Nevertheless, we also observe that, with the increase in D, the performance of the two algorithms for a small number of candidate designs becomes hard to differentiate. In fact, EGO can outperform appGGBS for large D and small K, as is shown in the case of $D = 15$. This suggests that EGO is a good heuristic algorithm for minimizing the expected number of queries for large D and small K.

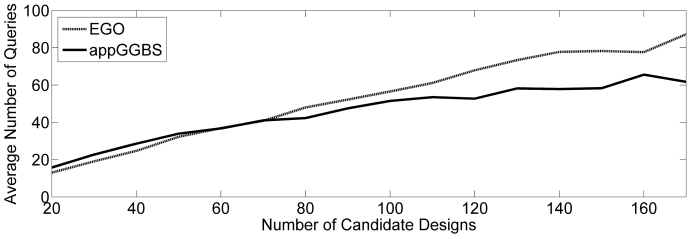
Let us also check the response time of both algorithms, as it is essential for human-computer interactions. Figure 7 shows the average response time for generating a new query using appGGBS and EGO. While appGGBS is computationally more expensive than EGO, its cost is almost linear with respect to the number of candidate designs. Considering that the query size for human-computer interaction is often limited to a small number, the response time of appGGBS is viable for such interactions. We note that with either appGGBS or EGO, the number of candidate designs that we can query from is rather limited for a practical query size.



(a) D=5

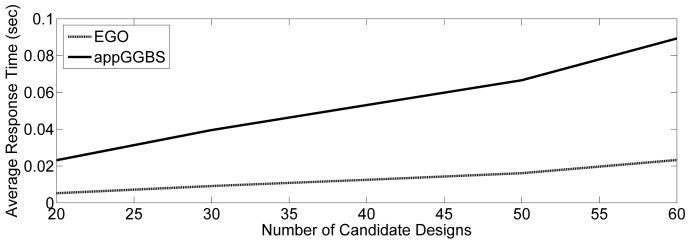


(b) D=10

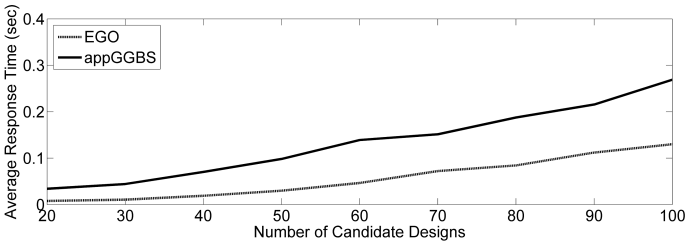


(c) D=15

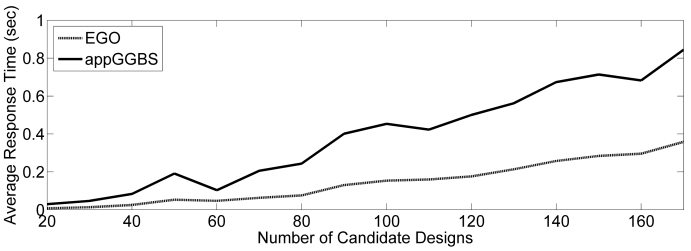
FIGURE 6. Performance of appGGBS and EGO under $D = 5, 10$ and 15



(a) D=5



(b) D=10



(c) D=15

FIGURE 7. Response time per query of appGGBS and EGO under $D = 5, 10, 15$

3.2 Performance of the update scheme on Π

We demonstrate now that the efficiency of appGGBS and EGO can be improved by updating Π according to the observed preferred designs from users. To do so, we use a simulation with $D = 10$ and $K = 100$. We set up a sequence of 1000 simulated users whose preferred designs are among a fixed set of 20 designs out of the total of 100 candidate designs. Both appGGBS and EGO start with the initial approximation Π^0. This probability distribution is updated using the method described in Subsection 2.4 with $t = 1$. Figure 8 shows the resulting number of queries needed for each user coming into the interaction. When the set of preferred designs from the user population is small, e.g., two designs, EGO can perform better than appGGBS since the algorithm starts by querying the two designs with the highest probabilities.

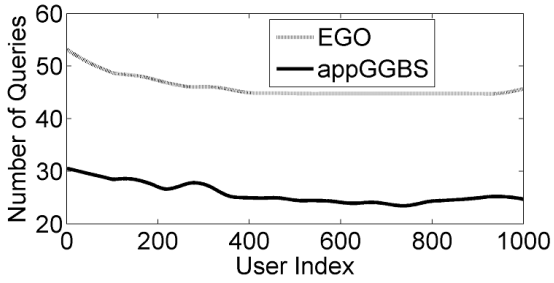
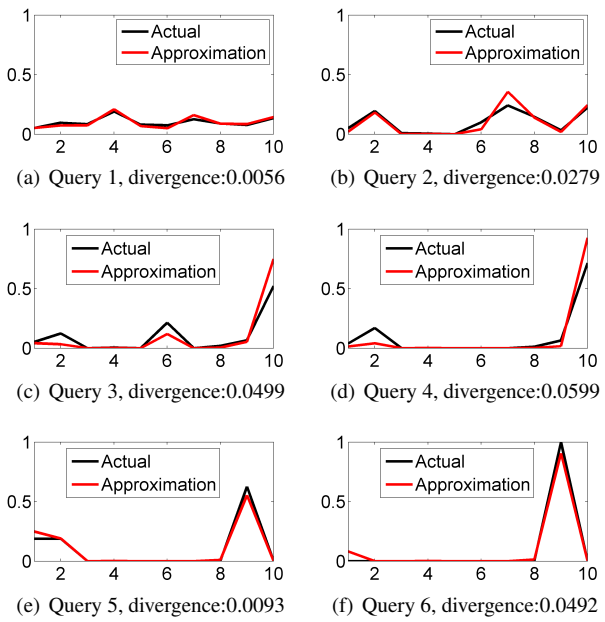


FIGURE 8. Updating Π lowers the number of queries needed in a long run.

3.3 Performance of the approximation method

For completeness, we examine the accuracy of the proposed approximation method. To illustrate, we run appGGBS on a simulated experiment with $D = 5$ and $K = 10$. During the query process, we compare the approximated probability masses π_{Θ^k}, 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



(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

FIGURE 9. The actual and approximated $\pi_{\Theta_a^k}$ 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 designs, and the y-axis for conditional probabilities $\pi_{\Theta_a^k} / \sum_{k'=1}^{K} \pi_{\Theta_a^{k'}}$.

design; to calculate $\pi_{\Theta_a^k}$, we count the number of points labeled by design k within the feasible spherical space under the cumulated responses up to node "a". Next, the count is divided by the total sample size (10^4) to represent the value of $\pi_{\Theta_a^k}$. We use Jensen-Shannon divergence to measure the difference between the actual and the approximated distributions of π_{Θ^k}. Figure 9 compares the two distributions in the first six queries during the simulated experiment. This result demonstrates the accuracy of the proposed approximation method.

4 Real-user Experiment

4.1 Experiment setup

The experiment seeks to identify laptop designs that are most preferred by a group of people.

A list of 48 candidate laptops are created based on five design features: Screen size, storage size, input type, CPU and battery. We use two levels of screen size: 11 and 13 inch; three levels of storage: 64, 128 and 256G; two levels of input: touchpad and touchpad plus keyboard; two levels of CPU: average and high-performance; and two levels of battery: half and full day duration. The designs are then labeled with realistic prices according to their feature levels. The prices also ensure that none of the designs is dominated by others under a linear utility model. Each design is then encoded using one real value for the price and six binary digits for design features (one digit for the two-level features and two for the three-level one). In order to incorporate a linear preference model in this experiment, interactions (screen size $\times$ input type), (screen size $\times$ CPU) and (screen size $\times$ battery) are added along side all the design features. This decision was made based on pilot tests with users. To summarize, a design x will contain 9 binary digits and one real value for price. Thus the experiment will have $K = 48$ and $D = 10$. The prices are normalized by first dividing by the highest price and then subtracting by the mean. This treatment is necessary as otherwise the design with the highest price will have a dominating approximated probability mass.

We conducted experiments with people in the form of interviews. Each experiment contained three stages: The validation test and two interactive sessions with pairwise comparisons using appGGBS and EGO correspondingly. The subject was first shown the full list of candidate designs. We explained the purpose of the experiment and asked the subject to pick one preferred design from the list. This test serves two purposes: First, the choice will be used to check if the interactions will converge successfully to the correct design; and second, it allows subjects to form their preference before they enter the interactions. Once the validation test is done, the interactions are conducted. The outputs from each experiment are the most preferred design and the number of queries used in the corresponding algorithms. In all experiments conducted, both algorithms correctly terminate when the most preferred design reaches probability close to 1 (see Subsection 2.5 for the termination criterion).

4.2 Results and analysis

Figure 10 shows the results of the eleven experiments conducted. On average appGGBS shows better performance than EGO under the described experiment settings. In addition, appGGBS and EGO suggest that both algorithms are promising in locating preferred designs for subjects among a fairly large amount of candidates, provided the preference of people on the product of interest can be captured by a relatively low-dimensional linear model.

5 Discussion and Conclusion

This study is related to recent developments in adaptive choice-based conjoint analysis [18, 1], and more generally the domain of active learning [16, 15]. The difference is that in this work, we are only interested in finding the most preferred design, while in conjoint analysis, the goal 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 "group", the proposed GGBS algorithm can be transformed into the classic Generalized Binary Search algorithm which is equivalent to the uncertainty sampling strategy (also called "utility balance" in [1]) commonly used in active learning practices [15].

We examined the problem of identifying users' most pre-

User ID	bestID	appGGBS	EGO
1	46	6	10
2	46	6	10
3	6	10	9
4	8	7	13
5	24	6	6
6	34	9	14
7	40	9	13
8	24	6	6
9	22	6	7
10	12	7	9
11	46	6	10
Average		7.1	9.7

FIGURE 10. Experiment results

ferred designs from a candidate design 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 computation cost. Besides its theoretical foundation, the algorithm also showed better performance both in simulated and real-user experiments than the algorithm in [13].

It should be noted, however, that the appGGBS algorithm relies on the assumption that the underlying utility function is linear with respect to known design features. While we showed in the real-user experiment that a remedy to this is to manually add interactions to the linear model, the algorithm may conclude incorrectly the subjects' preferred design when its built-in model fails to capture the nonlinearity in the preference. This limitation can be relaxed by incorporating a nonlinear kernel in Problem (8). Future investigation in this direction is needed.

The interaction we investigated in this paper utilizes only individual preferences. In future work we can leverage this interaction with a collaborative filter, allowing the algorithm to refine the probability updates by taking into account preferences of other subjects with similar responses as the current one.

6 Acknowledgement

This research was partially supported by the Automotive Research Center, a US 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] 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.

[2] 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.

[3] G. Bellala, S. Bhavnani, and C. Scott. Extensions of generalized binary search to group identification and exponential costs. *Advances in Neural Information Processing Systems (NIPS)*, 23, 2010.

[4] G. Bellala, S. Bhavnani, and C. Scott. Group-based active query selection for rapid diagnosis in time-critical situations. *Information Theory, IEEE Transactions on*, 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 and P. Papalambros. A design preference elicitation query as an optimization process. *Journal of Mechanical Design*, 133:111004, 2011.

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

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

[16] 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.

[17] 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.

[18] 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: Candidate Designs For Section 4

ID	screen	storage	keyboard	cpu	battery	price
1	11inch	64G	no	Average	full	600
2	11inch	64G	yes	Average	full	700
3	11inch	128G	no	Average	full	700
4	11inch	128G	yes	Average	full	800
5	11inch	256G	no	Average	full	800
6	11inch	256G	yes	Average	full	900
7	13inch	64G	no	Average	full	750
8	13inch	64G	yes	Average	full	850
9	13inch	128G	no	Average	full	850
10	13inch	128G	yes	Average	full	950
11	13inch	256G	no	Average	full	950
12	13inch	256G	yes	Average	full	1050
13	11inch	64G	no	High	full	900
14	11inch	64G	yes	High	full	1000
15	11inch	128G	no	High	full	1000
16	11inch	128G	yes	High	full	1100
17	11inch	256G	no	High	full	1100
18	11inch	256G	yes	High	full	1200
19	13inch	64G	no	High	full	1050
20	13inch	64G	yes	High	full	1150
21	13inch	128G	no	High	full	1150
22	13inch	128G	yes	High	full	1250
23	13inch	256G	no	High	full	1250
24	13inch	256G	yes	High	full	1350
25	11inch	64G	no	Average	half	450
26	11inch	64G	yes	Average	half	550
27	11inch	128G	no	Average	half	550
28	11inch	128G	yes	Average	half	650
29	11inch	256G	no	Average	half	650
30	11inch	256G	yes	Average	half	750
31	13inch	64G	no	Average	half	600
32	13inch	64G	yes	Average	half	700
33	13inch	128G	no	Average	half	700
34	13inch	128G	yes	Average	half	800
35	13inch	256G	no	Average	half	800
36	13inch	256G	yes	Average	half	900
37	11inch	64G	no	High	half	750
38	11inch	64G	yes	High	half	850
39	11inch	128G	no	High	half	850
40	11inch	128G	yes	High	half	950
41	11inch	256G	no	High	half	950
42	11inch	256G	yes	High	half	1050
43	13inch	64G	no	High	half	900
44	13inch	64G	yes	High	half	1000
45	13inch	128G	no	High	half	1000
46	13inch	128G	yes	High	half	1100
47	13inch	256G	no	High	half	1100
48	13inch	256G	yes	High	half	1200

FIGURE 11. All candidate designs for the real-user experiment