

# Design Preference Elicitation, Identification and Estimation

by

Yi Ren

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Mechanical Engineering)  
in The University of Michigan  
2012

Doctoral Committee:

Professor Panos Y. Papalambros, Chair  
Professor Richard D. Gonzalez  
Professor Noboru Kikuchi  
Professor George Michailidis

© Yi Ren 2012  
All Rights Reserved

For my parents

## ACKNOWLEDGEMENTS

I would like to gratefully and sincerely thank Dr. Panos Papalambros for his guidance, understanding, and friendship during my graduate studies. The Chinese maxim says “It’s better to teach fishing rather than offering fish”. That is exactly what you did, with great patience. It is your consistent intellectual and financial support that allowed me to freely enjoy my research and my life in general. For everything you have done for me, Dr. Papalambros, I thank you. I would also like to acknowledge my committee members: Dr. Richard Gonzalez, Dr. Noboru Kikuchi and Dr. George Michailidis for their valuable inputs. In particular, Dr. Gonzalez has been generously sharing his expertise to enrich my research throughout years of my graduate study.

I would like to give my special thanks to Dr. Yilun Chen. You not only broadened my knowledge but more importantly, showed me the devotion, enthusiasm and honesty a researcher should have. I also want to thank all of the members of the Optimal Design Laboratory for their support and participation in my research.

Finally and most importantly, I would like to thank my parents for their understanding, patience and faith in me. They have always been supportive to my life path and I truly appreciate their challenge on my research to keep me connected with the real world. My last and special thank goes to my girlfriend Zhuqing. I want to thank you for all your efforts to make me more matured and responsible in and outside of research. Your heartfelt company during this special and busy year was an essential force driving me forward.

# TABLE OF CONTENTS

<b>DEDICATION</b> . . . . .	ii
<b>ACKNOWLEDGEMENTS</b> . . . . .	iii
<b>LIST OF FIGURES</b> . . . . .	viii
<b>LIST OF TABLES</b> . . . . .	xii
<b>ABSTRACT</b> . . . . .	xiii
<b>CHAPTER</b>	
<b>I. Introduction</b> . . . . .	1
1.1 Motivation . . . . .	1
1.2 Problem Definition . . . . .	3
1.2.1 Design space and features . . . . .	3
1.2.2 Utility: A measure of preference . . . . .	4
1.2.3 Interaction . . . . .	5
1.2.4 Subject response model . . . . .	5
1.2.5 Preference identification . . . . .	7
1.2.6 Preference elicitation . . . . .	8
1.2.7 Preference estimation . . . . .	8
1.3 Related Work . . . . .	9
1.3.1 Interactive evolutionary computation . . . . .	10
1.3.2 Recommender systems . . . . .	15
1.3.3 Conjoint analysis . . . . .	19
1.3.4 Summary . . . . .	21
1.4 Dissertation Overview . . . . .	22
<b>II. Literature Review</b> . . . . .	24
2.1 Conjoint Analysis . . . . .	25
2.1.1 Utility and choice probability . . . . .	25

2.1.2	Likelihood models . . . . .	27
2.1.3	Parameter estimation and its variance . . . . .	30
2.2	Support Vector Machine (SVM) . . . . .	31
2.2.1	Basic concepts . . . . .	32
2.2.2	Primal and dual problems . . . . .	34
2.2.3	Kernel trick . . . . .	36
2.2.4	Soft margin formulation and complexity control . . . . .	39
2.3	Conclusion . . . . .	41
<b>III. Preference Elicitation . . . . .</b>		<b>42</b>
3.1	Efficient Global Optimization (EGO) . . . . .	43
3.1.1	Kriging modeling . . . . .	45
3.1.2	Mean-squared error of the predictor . . . . .	48
3.1.3	Expected improvement . . . . .	48
3.1.4	Properties of the expected improvement . . . . .	49
3.1.5	Optimization on the expected improvement . . . . .	50
3.2	SVM Search . . . . .	51
3.2.1	The algorithm . . . . .	51
3.2.2	SVM Search Simulated Test Results . . . . .	53
3.3	EGO Search . . . . .	57
3.3.1	The algorithm . . . . .	57
3.3.2	EGO Search versus SVM Search in simulated tests . . . . .	60
3.4	Discussion on the EGO Search Algorithm . . . . .	60
3.4.1	Parameters used in the merit function . . . . .	62
3.4.2	Dimensionality and user sensitivity . . . . .	64
3.4.3	Computational cost . . . . .	65
3.4.4	EGO Search versus GA plus SVM . . . . .	65
3.5	Vehicle Exterior Styling Design Elicitation . . . . .	67
3.5.1	Software development . . . . .	67
3.5.2	Convergence test setup . . . . .	68
3.5.3	User data analysis . . . . .	70
3.5.4	Observed issues in user interactions . . . . .	74
3.6	Concluding Remarks . . . . .	77
<b>IV. Augmented Preference Elicitation . . . . .</b>		<b>79</b>
4.1	Preference Modeling with Comparison Tree . . . . .	80
4.1.1	Terminology and definition . . . . .	80
4.1.2	Learning based on pairwise comparison . . . . .	81
4.1.3	Simulated test results and discussion . . . . .	81
4.2	Variations of the Merit Function . . . . .	83
4.2.1	Kushner's criterion . . . . .	86
4.2.2	Generalized Expected Improvement (GEI) . . . . .	86
4.2.3	Lower Confidence Bounding function (LCB) . . . . .	87

4.2.4	Locating the Regional Extreme (LRE)	87
4.2.5	Switching criterion	88
4.2.6	Computational difficulty	88
4.3	Fast EGO and Its Simulated Test Results	89
4.3.1	The geometric meaning of $\hat{\sigma}$	90
4.3.2	A computationally inexpensive merit function for EGO	90
4.3.3	Simulated test results	93
4.3.4	Discussion	94
4.4	Concluding Remarks	98

**V. Preference Estimation and Identification** . . . . . 99

5.1	Active Learning Background	102
5.1.1	Non-adaptive query	102
5.1.2	Active learning in conjoint analysis	103
5.1.3	Active learning in machine learning	104
5.2	Preference Estimation	111
5.2.1	Problem definition	111
5.2.2	Design features	112
5.2.3	Estimation of $\mathbf{w}$	113
5.2.4	Active learning on preference estimation	115
5.2.5	Test setup	118
5.2.6	D-optimal design setup	119
5.2.7	Performance of active learning, $D$ -optimal and random designs on 2D problems	120
5.2.8	Limitation of the linearity assumption	122
5.2.9	Identification of preference features	123
5.2.10	Robustness of active learning	124
5.3	Preference Identification	126
5.3.1	Justification of the loss function	126
5.3.2	Application of active learning	129
5.3.3	Active learning on preference identification	132
5.3.4	Simulated tests and results	132
5.4	Concluding Remarks	135

**VI. Collaborative Filtering in Preference Elicitation** . . . . . 137

6.1	Problem Formulation	137
6.2	Heuristics	138
6.2.1	Simplification of a search tree (JUMP0)	139
6.2.2	Exploration of more efficient trees (JUMP1)	139
6.2.3	2D Demonstration of the exploration heuristic	142
6.2.4	Enhancement to JUMP1 (JUMP2)	145
6.2.5	2D Demonstration of the enhanced heuristic	148
6.2.6	Update of initial guess (JUMP3)	149

6.2.7	2D Demonstration of initial guess update . . . . .	151
6.3	Discussion . . . . .	152
6.4	Relationship With Collaborative Filtering . . . . .	156
6.5	Concluding Remarks . . . . .	157
<b>VII.</b>	<b>Conclusions . . . . .</b>	<b>158</b>
7.1	Summary . . . . .	158
7.2	Contributions . . . . .	161
7.3	Assumptions and Limitations . . . . .	162
7.4	Future Work . . . . .	163
7.4.1	Preference modeling with enriched user interaction data . . . . .	163
7.4.2	Calibration on weighted search . . . . .	164
7.4.3	Feature addition and subtraction . . . . .	164
7.4.4	Linking user preferences and designer decisions . . . . .	165
7.4.5	Search towards a mutual target . . . . .	166
<b>BIBLIOGRAPHY</b>	<b>. . . . .</b>	<b>167</b>



## LIST OF FIGURES

### Figure

1.1	General framework of an IGA . . . . .	11
1.2	Multi-armed roulette wheel parent selection . . . . .	13
1.3	An example of crossover. Here the two grayed designs are parents and the white one is a child. The child acquires the hood design of the first parent and the silhouette design of the second parent. . . . .	13
2.1	A separation hyperplane ( $\hat{\mathbf{w}}^T \mathbf{x}$ ) for a two dimensional training set. Figure from <i>Cristianini and Shawe-Taylor (2000)</i> . . . . .	33
2.2	$\gamma$ is the margin between two classes. Figure from <i>Cristianini and Shawe-Taylor (2000)</i> . . . . .	34
2.3	Mapping data to a different space where they can be linearly separated. Figure from <i>Cristianini and Shawe-Taylor (2000)</i> . . . . .	36
2.4	Two classes separated by a hyperplane in the feature space (Gaussian kernel). Figure from <i>Cristianini and Shawe-Taylor (2000)</i> . . . . .	38
3.1	EGO operation concept. . . . .	44
3.2	SVM Search scattering and classification. . . . .	52
3.3	SVM Search design space reduction. . . . .	53
3.4	Proposed SVM Search algorithm . . . . .	54
3.5	Proposed EGO Search algorithm . . . . .	58
3.6	Comparison between EGO Search and SVM Search. . . . .	61
3.7	Comparison between EGO Search and GA plus SVM. . . . .	66
3.8	Online human-computer interaction interface and data visualization. (a): The interactive environment at “/convergencetest.html” allows the user to zoom, pan, rotate each design and updates the guesses once the user hits the “Next” button; (b): The data visualization window at “/log.html” has all user tests listed at the top, number of iterations in the middle, and the cumulated data at the bottom. The red curve represents the target design, the highlighted dark curve(s) represent the preferred design at this point and the rest all not preferred. . . . .	69
3.9	Normalized Euclidean distance from each sampled design to the target in each test. The circled design is the one submitted by the user, while the triangle design has the lowest Euclidean distance to the target. . . . .	72

3.10	Visual comparison between user test results and the targets from side and perspective views. . . . .	73
3.11	Visual comparison between samples in the last iteration and the target. Data generated from Test0 on “log.html”. Euclidean distances to the target are listed under the designs. . . . .	74
3.12	Designs labeled as preferred in the first four iterations, compared with the target and the one with the minimum Euclidean distance within all samples. Euclidean distances to the target are listed under the designs. . . . .	75
3.13	Features that capture the roof design. . . . .	75
3.14	Samples from data “Test0” in the feature and Euclidean space. Multidimensional scaling is applied to both measures to create 2D visualization of the data. . . . .	76
4.1	2D Six-Hump Camelback . . . . .	83
4.2	2D Branin . . . . .	83
4.3	10 dimensional Gaussian, $\lambda^{\text{user}} = 5$ . . . . .	84
4.4	20 dimensional Gaussian, $\lambda^{\text{user}} = 5$ . . . . .	84
4.5	31 dimensional Gaussian, $\lambda^{\text{user}} = 5$ . . . . .	84
4.6	Violation rates at each iteration in each test . . . . .	85
4.7	The computational costs at the first 30 iterations when optimizing the 31-dimensional Gaussian in Equation (3.19). . . . .	89
4.8	Merit functions during the search on a Branin function (Equation (3.17)). Bright areas indicate high function values. . . . .	91
4.9	Comparison between the mean-squared error function and the minimum distance function under a set of sampled designs (represented as white crosses). Bright areas indicate high function values. . . . .	92
4.10	Convergence performance of searches with $f_{\text{fast}}$ and $f_{\text{lcb}}$ as their merit functions. . . . .	95
4.11	Computational cost of searches with $f_{\text{fast}}$ and $f_{\text{lcb}}$ as their merit functions. . . . .	96
4.12	Contour plot of the merit function at different $r^2$ values. . . . .	97
5.1	Geometrical representation of the version space. In this 2D case, the version space is the highlighted arc of the circle. Each normal vector of a constraining hyperplane represents a sample point $\mathbf{v}_i$ and the label $y_i$ determines which side of the hyperplane is feasible for $\mathbf{w}$ . The solution $\hat{\mathbf{w}}$ of a classification problem is the center of the largest hypersphere within the gray cone $y_i(\mathbf{w}^T \mathbf{v}) > 0$ , $i = 1 \dots n$ . The bisection of the version space $\mathcal{V}$ can be approximated by cutting through the current solution $\hat{\mathbf{w}}$ , i.e., $\hat{\mathbf{w}}^T \mathbf{v} = 0$ . . . . .	106
5.2	Penalty <sub>ML</sub> and Penalty <sub>SL</sub> in the range $u \in [-5, 5]$ for preference realization; $\theta = 1$ , $h = 1$ . . . . .	115
5.3	Penalty <sub>ML</sub> and Penalty <sub>SL</sub> in the range $u \in [-2, 2]$ for preference identification; $\sigma_\varepsilon = 1$ , $h = 1$ . . . . .	128

5.4	Demonstration of active learning for preference identification. Each block on the 2D space represents a design. The black blocks represent designs such that $f(\mathbf{x}) < 0$ and the white ones represent $f(\mathbf{x}) > 0$ . The circled blocks are designs with $y = -1$ and dotted ones with $y = 1$ .	134
6.1	We jump from vertex $n_s$ to vertex $n_{s+\delta}$ if there is no alternative path from $n_s$ and through $n_{s+1}$ .	140
6.2	Concept of JUMP1: Jumping to a conclusion (leaf vertex) based on previous knowledge, and search from there.	141
6.3	JUMP1 algorithm: From the current status $n_s$ , find a set $\mathbf{N}_s$ from $T_t$ that is most similar to $n_s$ . Find the leaf vertex set $\mathbf{N}_s^*$ accessible from $\mathbf{N}_s$ . Pick the next query design as the one from $\mathbf{N}_s^*$ that has the highest average path length $\bar{P}$ .	142
6.4	Setup of the 2D demonstration for JUMP1: The test optimal solutions are uniformly distributed on the shaded locations. The indices are of ascendant order starting from the left down corner as 1 and ending at the top right corner as 100.	143
6.5	Average path lengths for 10 batches of random tests with and without JUMP1. This comparison shows that JUMP1 can effectively reduce the search cost with different $d$ .	144
6.6	Histogram of path lengths from 200 random tests with and without the search tree generated from the experiment in Figure 6.5.	144
6.7	The evolution of search path for a preference function optimized at index 86. JUMP1 explores different paths and finds better ones than from the original search algorithm.	146
6.8	The average query size of the two clusters (optimal solutions at $\{\mathbf{x}_{15,24,25,33,34,35,36,44,45,55}\}$ and in $\{\mathbf{x}_{68,77,78,86,87,88,89,97,98}\}$ ) along number of batches of tests.	146
6.9	JUMP2 algorithm: From the current status $n_s$ , find a set $\mathbf{N}_s$ from $T_t$ that is most similar to $n_s$ . Find the leaf vertex set $\mathbf{N}_s^*$ accessible from $\mathbf{N}_s$ . Pick the next query design as the one from $\mathbf{N}_s^*$ that has the lowest average path length to all other leaf vertices accessible from it.	147
6.10	Performance of JUMP1 and JUMP2 on 1000 random tests with update frequency: 1, 10, 100 tests per update. Both algorithms switch to heuristic search after every 2 iterations. For visualization purpose, the 1000 test results are grouped and we show the average query size of each group.	149
6.11	Performance of JUMP2 and JUMP3 on 1000 random tests. Both algorithms update their search strategy after every test and switch to heuristic search after every 2 iterations. The figure is segmented to show performance of JUMP3 with different initial guesses. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group.	150

6.12	Performance of JUMP3 with fixed and adaptive initial guess change frequency on 1000 random tests. Both algorithms update their search strategy after every test and switch to heuristic search after every 2 iterations. The figure is segmented to show performance with different initial guesses. For visualization purpose, the 1000 test results are grouped and we show the average query size of each group. . . .	152
6.13	Densities of the optimal designs along time. The five figures show how the optimal designs are distributed at different stages of the experiment. The grey scale indicates the frequency of occurrence. . .	153
6.14	Performance of JUMP2, JUMP3 and the default algorithm on 1000 random tests. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 2 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group. . . . .	153
6.15	A situation where optimal designs are not clustered. The grey scale indicates the frequency of occurrence. . . . .	154
6.16	Performance of JUMP2, JUMP3 and the default algorithm on the 1000 random tests with optimal designs located in Figure 6.15. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 2 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group. . . . .	155
6.17	Performance of JUMP2, JUMP3 and the default algorithm on the 1000 random tests on a design space with 5 dimensions and 3 levels for each. The optimal designs of these random tests are uniformly generated on the entire space. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 7 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group. . . .	155
6.18	Performance of JUMP2, JUMP3 and the default algorithm on the 1000 random tests on a design space with 5 dimensions and 3 levels for each. The optimal designs of these random tests are uniformly generated on a small set of indices of the space. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 7 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group. . . . .	156

## LIST OF TABLES

### Table

1.1	Overview of related work . . . . .	23
3.1	Means and standard deviations of the final error from SVM Search, GA and Random Search (Lower values are better) . . . . .	57
3.2	Impact of the weights of the weighted-sum merit function (Lower values are better) . . . . .	63
3.3	Impact of the spread of the expected improvement merit function (Lower values are better) . . . . .	64
3.4	Impact of dimensionality and spread of the utility (Lower values are better) . . . . .	65
5.1	Generalization errors of active learning, $D$ -optimal and random (Lower values are better) . . . . .	121
5.2	Generalization errors under the linearity assumption . . . . .	123
5.3	$p$ -values of $H_0 : \hat{w}_i = 0$ for estimators on linear and polynomial preferences (Highlighted are significantly lower $p$ -values) . . . . .	124
5.4	Robustness of active learning (Lower values are better) . . . . .	125
5.5	Preference identification test settings . . . . .	135
5.6	Preference identification test results . . . . .	135

## ABSTRACT

Understanding user preference has long been a challenging topic in the design research community. Econometric methods have been adopted to link design and market, achieving design solutions sound from both engineering and business perspectives. This approach, however, only refines existing designs from revealed or stated preference data. What is needed for generating new designs is an environment for concept exploration and a channel to collect and analyze preferences on newly-explored concepts. This dissertation focuses on the development of querying techniques that learn and extract individual preferences efficiently. Throughout the dissertation, we work in the context of a human-computer interaction where in each iteration the subject is asked to choose preferred designs out of a set. The computer learns from the subject and creates the next query set so that the responses from the subject will yield the most information on the subject's preferences. The challenges of this research are: (1) To learn subject preferences within short interactions with enormous candidate designs; (2) To facilitate real-time interactions with efficient computation.

Three problems are discussed surrounding how information-rich queries can be made. The major effort is devoted to *preference elicitation*, where we discuss how to locate the most preferred design of a subject. Using efficient global optimization, we develop search algorithms that combine exploration of new concepts and exploitation of existing knowledge, achieving near-optimal solutions with a small number of queries. For design demonstration, the elicitation algorithm is incorporated with an online 3D car modeler. The effectiveness of the algorithm is confirmed by real user

tests on finding car models close to the users' targets. In *preference identification*, we consider designs as binary labeled, and the objective is to classify preferred designs from not-preferred ones. We show that this classification problem can be formulated and solved by the same active learning technique used for *preference estimation*, where the objective is to estimate a preference function. Conceptually, this dissertation discusses how to extract preference information effectively by asking relevant but not redundant questions during an interaction.

# CHAPTER I

## Introduction

### 1.1 Motivation

Design is a heterogeneous topic, across diverse disciplines including art, engineering, psychology, marketing and many others. We call an artefact well designed when it is not only well engineered but also calibrated to have a considerate interaction interface, to fulfil the emotional needs of its user, to improve the social behaviour of a population and more. In the design community, we have witnessed the developments in simulation, optimization and system engineering technologies, and from there we have adopted the concept of building multidisciplinary design models and obtaining optimal solutions that not only are viable from an engineering point of view but also maximize their market shares (*Michalek et al. (2005)*). We have gone even beyond a single design to find balanced solutions in a competitive environment (*Frischknecht (2009)*) and in the presence of policy changes (*Whitefoot et al. (2011)*). The power of mathematical abstraction of the real-world surroundings has allowed us to design with better functionality, for improved social welfare and profit, and most importantly, with rigorous justification of compromises on these objectives when they compete.

Design is rooted in human preference. As a matter of fact, positive preference on a design can improve the functionality of the design through enhanced brain activity



of its user (*Norman (2002)*). Incorporating human preference in the design process is an ill-defined problem. The topic of extracting preference from humans has long been studied in psychology and econometrics to understand the perceptual trade-off among design features, e.g., price vs. performance metrics, and this line of research has been adopted in the design community to complete the connection between engineering design and market performance (for example, see *Wassenaar et al. (2005)*; *Kumar et al. (2007)*; *Li and Azarm (2000)*; *Michalek et al. (2005)*). This research, however, can only “refine” existing products since the features under preference test are controlled by a relatively fixed product architecture. In other words, the framework we have adopted from the econometrics provides only the best compromise within the testers’ vision rather than an opportunity to explore user preferences outside the prescribed boundaries. Exploring possibilities to meet user preferences is at least as important as understanding the trade-offs users make. This is what can make design research unique and differentiated from econometric studies.

The computer science community has made one step forward in this direction. Computer scientists share the desire to create attractive artefacts, but in most cases virtually. Evolutionary algorithms have been applied to creation of images, music and virtual creatures. Design researchers shall find this line of research appealing, as it provides an entirely different and much more entertaining environment for obtaining user input, while getting rid of the possibly dreadful interfaces grafted from economists. The defect of an evolutionary algorithm when applied to design, however, is that it evolves too slowly, and may not reach anything interesting before the user gets tired and quits. This is especially the case when the search space has many dimensions.

The major goal of the present research, therefore, is to develop an interaction and its underlying search algorithm, that will be able to locate some near-optimally preferred conceptual designs of a subject, within a small amount of time. The search

algorithm shall rely on statistical understanding of the subject’s responses instead of “free” evolution. The interaction shall be accessible by a large population and previous interactions by others shall enhance future searches. The computational cost of each search shall be reasonable for real-time interactions. For completeness, we should also understand how traditional preference modeling can be enhanced by new developments from statistics and computer science. From a high-level viewpoint, the theme throughout this dissertation is the balance between exploration for new knowledge and exploitation of existing knowledge during a querying process. Conceptually, this work is about how to ask relevant but not redundant questions during a user-computer interaction and get close to a good answer quickly.

## 1.2 Problem Definition

This section transforms the real-life problems we try to solve into mathematical ones. Inevitably, such a transformation introduces assumptions that may detract from the value of solving these problems. Nonetheless, these assumptions are either commonly adopted in related research or acceptable enough for the solutions to have practical utility. We start with basic definitions and terminologies used throughout the dissertation and then we introduce three fundamental problems we try to address, namely, preference elicitation, identification and estimation.

### 1.2.1 Design space and features

Let  $\mathcal{D} \subset \mathbb{R}^p$  be the design space where any  $\mathbf{x} \in \mathcal{D}$  represents a design described with  $p$  variables. The variables are usually defined by designers to represent features of a design that they can control. For example, in a vehicle study, fundamental engineering variables such as engine size and gear ratio form a design space, and together they allow control of performance features such as miles per gallon and acceleration. In a shape design problem, coordinates of the control points of a shape

form a design space, and together they control qualitative shape features such as curvature, sportiness or environmentally friendly “looks” (see [Reid \(2010\)](#)). From common sense, it is safe to say that subjects, i.e., consumers in a market or users of a human-computer interaction, are usually unaware of the underlying design variables and form their preference based on the design features. Therefore it is necessary to introduce a mapping (to be more strict, a set of functions defined on  $\mathcal{D}$ ) that maps the design space to the feature space  $\mathcal{F}: \mathbf{v} : \mathcal{D} \mapsto \mathcal{F}$ . The tricky part here is that usually this mapping is unknown to researchers or even to subjects, leaving  $\mathbf{v}$  arbitrary. We will show how this difficulty can be technically addressed in [Chapter III](#) as well as in [Chapter V](#). Throughout this dissertation we assume  $\mathcal{D}$  to be rectangular, i.e.,  $\mathcal{D}$  has simple bounds on each dimension but there is no constraint imposed on  $\mathcal{D}$ .

### 1.2.2 Utility: A measure of preference

The classical measure of subjects’ preferences according to the econometrics and marketing literature is the utility function. The utility of a design according to a specific subject is a function of the design features  $\mathbf{v}(\mathbf{x})$ , parameterized by subject-related parameters  $\mathbf{w}$ :

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{v}(\mathbf{x}) + \varepsilon. \tag{1.1}$$

The error term  $\varepsilon$  is added to model the stochastic nature of the preference. With this model, we have already assumed the existence of a constant  $\mathbf{w}$ , meaning that the subject has a fixed preference in mind before any questionnaire occurs. Literature, however, showed that people may construct preference on an as-needed basis and the underlying preference model may change along with the change of the designs ([MacDonald et al. \(2009\)](#)). While acknowledging the flaw of this assumption, its adoption here is supported by the reported success in econometrics and marketing research. Notice that transitivity of preferences will also be assumed when the error term is ignored, which, according to [Petiot and Grognet \(2006\)](#) may not hold in

reality, i.e., one can prefer A over B and B over C, but also C over A. We shall also mention that the usage of  $f$  for utility is not common in the utility literature. We do this to keep consistency of notation throughout the present writing.

### 1.2.3 Interaction

All of the problems discussed in this dissertation are set in the context of a human-computer interaction described here. The interaction is between an individual subject and the search algorithm. The purpose of the search may vary but from the subject's perspective, she is always assigned the same task: Choosing some preferred designs from a set presented by the algorithm and submitting the choice result. Once the result is submitted, the algorithm analyzes the accumulated information from this subject and creates a new query with another set of designs. The iterations continue until some termination criterion is met.

### 1.2.4 Subject response model

Assume that the subject is presented with  $q$  designs in each iteration and is asked to select any number of designs as preferred and the rest as not preferred. Therefore the response from the subject is always a binary vector with each element, 0 or 1, being the choice decision on the corresponding design. However, the interpretation of this binary vector can vary. In one case, the subject may only select those designs that she prefers, and the implication is that she wants to see more designs with similar features as the ones she chooses, although such features may not be explicitly described; In the other case, the subject may select designs that are relatively better than the others but not necessarily indicate that the chosen ones are preferred. The difference may be subtle in words but as we discuss below, variation on the subject response model behind its binary outcome leads to different mathematical problems.

We consider the process of generating a choice decision to start with utility cal-

ulation. Denote the utilities of the set as  $\{f_i\}_{i=1}^q$ , two different labeling processes could follow:

- In the *definite model*, only such  $f_i$ s that exceed a threshold will be labeled as ones, and the rest zeros. This is corresponding to the situation where the subject only labels the preferred designs as ones.
- In the *comparison model*,  $\{f_i\}_{i=1}^q$  is grouped into two clusters. The cluster with high values will be labeled as ones, and the other cluster as zeros. Therefore the binary outcome represents the comparison within the set.

In both cases, the subject can assign multiple ones and zeros. It is also possible for the subject to reject (all zeros) or accept (all ones) all designs. Therefore, these two definitions largely extend the restriction of choosing only one design from a set in the traditional choice-based conjoint analysis setup, and is more natural as a human-computer interaction.

In practice, it is important to make sure that we use the right interpretation for the user choice. For that purpose, we can remind the subject to use a fixed response model by presenting a question along with the design set: For the definite model, the question is:

“Are there any preferred designs in the presented set?”;

While for the comparison model, the question will be:

“Which design(s) in the set do you prefer more?”.

The underlying objective of asking these two questions is different: Under the definite model, we would like to identify and describe the domain of preferred designs (preference identification); under the comparison model, the objective is to either find the most preferred design through the interaction (preference elicitation) or realize the unknown utility parameters  $\mathbf{w}$  (preference estimation). The following subsections explain these problem definitions in detail.

### 1.2.5 Preference identification

In preference identification, the user is assumed to pick only preferred designs rather than compare the designs in the set. Therefore each design within  $\mathcal{D}$  is associated with a definite label and is independent of other alternatives. We can introduce an indicator function on  $\mathcal{D}$  that takes ones in a domain  $\mathcal{D}_+$  and zeros otherwise:

$$f(\mathbf{x}) \triangleq 1_{\mathcal{D}_+}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \mathcal{D}_+, \\ 0, & \text{if } \mathbf{x} \notin \mathcal{D}_+. \end{cases} \quad (1.2)$$

The objective of preference identification thereafter is to identify the domain  $\mathcal{D}_+$  using queries and their labels, which is in fact a classification problem with two classes. However, to accurately identify a classifier, one would usually need a large amount of queries, which is not affordable in human-computer interaction when taking subject fatigue into account. Therefore the underlying problem of preference identification is to find a sequence of queries that would effectively help to identify the classifier. The methodology for solving this problem exists in statistical learning. The term “active learning” was first coined by *Tong and Koller (2002)* who showed that faster classification can be achieved by querying samples iteratively. By formulating the classification task using Support Vector Machine (SVM) (see *Vapnik (1998)*), they proved that efficient queries can be made by cutting the space of the classifier coefficients into equal halves. Further research discussed the balance between exploration (querying without using knowledge gained) and exploitation (like Tong and Koller, querying with all knowledge gained) to achieve more stable performance on different problems (*Osugi et al. (2005)* and *Baram et al. (2004)*). A “locking” problem, i.e., a slow convergence in the classifier estimation, when using active learning in a continuous space was reported by *Basudhar and Missoum (2010)* and a heuristic sampling method was proposed to overcome it.

### 1.2.6 Preference elicitation

One practical flaw of the preference identification setup is that the active learning mechanism requires to know at least one design from each class to create a classifier. However, in a usually high-dimensional design space, it is difficult to find a preferred design without thoroughly querying the space. Thus comes the need to introduce the comparison model. Formally, let  $f_{\text{cluster}}(f_i, \{f_j\}_{j \neq i}) : \mathbb{R} \mapsto \{0, 1\}$  be a function of the utility  $f_i$  parameterized by the alternative utilities from the set. In the comparison model we assume that the subject first evaluate the utilities of the design set and use  $f_{\text{cluster}}$  to label the designs.

The objective of preference elicitation is to find a design  $\mathbf{x}^*$  that optimizes the deterministic part of the utility  $\mathbf{w}^T \mathbf{v}(\mathbf{x})$  using the binary output under the comparison response model. Indeed, when the stochastic part  $\varepsilon$  of the utility is ignored,  $\mathbf{x}^*$  is such a design that for any set of alternatives from  $\mathcal{D}$  that have utilities  $\mathbf{u}$ ,  $f_{\text{cluster}}(u^*, \mathbf{u}) = 1$ , or verbally,  $\mathbf{x}^*$  is a design that will always be chosen regardless of other alternatives. When  $\varepsilon$  exists,  $\mathbf{x}^*$  has the least probability to be labeled as zero. Therefore preference elicitation corresponds to a “black-box” optimization problem with binary outputs. We discuss the basic and extended search strategies for preference elicitation in Chapter III and IV. Further, we explore how search history from different subjects can help to enhance future searches in Chapter VI.

### 1.2.7 Preference estimation

The objective of preference estimation is to estimate the utility parameter  $\mathbf{w}$  (or “part-worths” in econometrics), which is exactly the same objective of choice-based conjoint analysis. However, instead of using non-adaptive queries designed before a questionnaire, we investigate how effective queries should be created during the interaction to achieve good estimation within a short query. As will be shown, this problem can be transformed to the active learning problem as well. Therefore we

discuss preference identification and estimation together in Chapter V.

### 1.3 Related Work

The problems we defined above have been investigated in a variety of disciplines, including econometrics, computer science and statistics. It is therefore essential for us to review related work. In this section, we introduce, at a conceptual level, three research areas that partially address the problems we defined.

- In computer science, *interactive evolutionary computation* refers to a broad range of methods and applications that allow a population of virtual creatures or artefacts to evolve according to a simulated environment which can be subject ratings. One classic realization of interactive evolutionary computation is the Galapagos interactive exhibit created by *Sims (1997)* which evolves 3D creatures based on user-computer interaction. This line of research partially solves the preference elicitation problem we proposed.
- Another related branch of research mainly rooted in computer science and statistics is called *recommender system* (*Adomavicius and Tuzhilin (2005)*; *Balanović and Shoham (1997)*). A typical recommender system proposes products or services to the user based on specific user preference (content-based) and collaborated preferences from other users (collaborative filtering). The underlying research question of a recommender system is similar to all of the three problems we proposed: To understand the user preference. Nonetheless, the objective of a recommender system is to propose items that a subject may prefer rather than searching for the most preferred item of that subject. From a practical perspective, the recommender system is usually applied on existing products with a few semantic attributes while we are interested in extracting preferences on non-existing concepts.



- The preference realization problem we proposed is identical to conjoint analysis in econometrics and marketing (*McFadden (1973)*; *Green and Srinivasan (1978)*; *Green et al. (2001)*). With the development in human-computer interaction, researches have been conducted to explore new forms of conjoint analysis (*Toubia et al. (2004, 2007a)*; *Netzer et al. (2008)*). Our focus is on how questionnaire can be created adaptively during an interaction based on subject inputs.

### 1.3.1 Interactive evolutionary computation

A prevalent approach to tackle the preference elicitation problem is Interactive Evolutionary Computation (IEC), an umbrella term for evolutionary algorithms that obtain fitness function values from subjects' inputs through interactions rather than through functions or simulations. Two major branches of IEC are interactive genetic algorithm (IGA) and genetic programming (IEC). The former, which is more convenient to employ and more prevalent, represents an artefact using a vector of values, or in its own term, a chromosome, while the latter uses a structure as a representation. The idea of IEC was originated from *Dawkins and Pyle (1991)* and was then propagated to many disciplines where human knowledge is required to enable or enhance an artefact of interest. Notable work include *Sims (1991)* for using interactive genetic algorithm (IGA) to create virtual creatures that evolve according to human evaluation or simulated environments, *Johanson and Poli (1998)* for interactive genetic programming (IGP) on evolutionary music design, *Kim and Cho (2000)* for interactive dress design, and *Tokui and Iba (2000)* for combined IGA and IGP in music design, to name a few. In the engineering design field, *Kelly and Papalambros (2007)* introduced an IGA to elicit user preference on car silhouette design. The same author also employed IGA to show the trade-off between shape preference and engineering attributes of bottle design (*Kelly (2008)*).

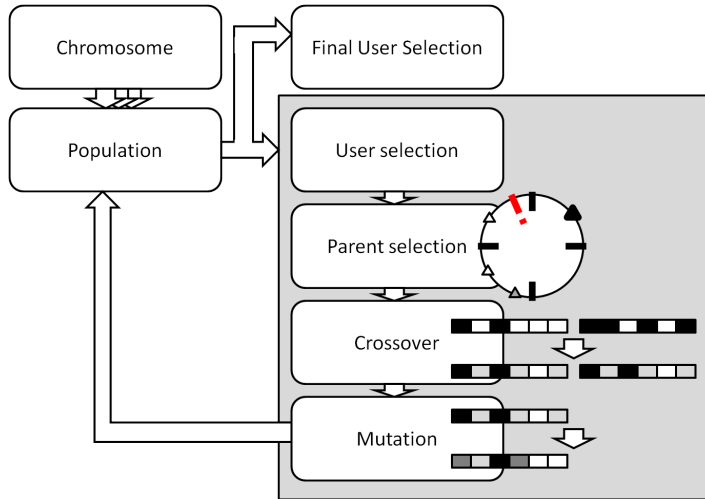


Figure 1.1: General framework of an IGA

Below we briefly go over the most commonly accepted IGA framework and a vehicle exterior design algorithm we developed using IGA. We then summarize the pros and cons of the IGA approach.

### 1.3.1.1 General IGA framework

An IGA consists of the following elements: Chromosome is referred to as the fundamental element of the object under evolution. In practice, a chromosome is a binary or real-valued vector that is used to generate a phenotype, i.e., the artefact under study. The one-to-one mapping between chromosomes and phenotypes is pre-defined. A population is a group of chromosomes that is used to carry out parent selection, crossover and mutation operations. The overall flow of IGA is illustrated in Figure 1.1: A random population is first presented to the subject. Based on subject response, fitness of the phenotypes is determined and those with high fitness will have higher probability to be selected as parents. The chromosomes of these parents are copied to the children and modified by crossover and mutation schemes. The phenotypes of the offspring chromosomes are presented to the subject in the next iteration for evaluation. The algorithm continues until terminated by the subject.

### 1.3.1.2 Vehicle exterior design study

We introduce the key operations of a genetic algorithm, namely, fitness calculation, parent selection, crossover and mutation within the context of an earlier study on interactive vehicle exterior design *Ren (2009)*. The motivation of this study was to evaluate how suitable IGA might be in interactive geometric design which usually involves a large number of design variables. For this purpose a system was built that takes real-valued vectors of 33 dimensions and outputs 3D vehicle models for each of these vectors. Subjects are asked to evaluate these models based on their own preference and select the preferred ones. The interaction is set up in the way that subjects are able to see all sampled designs at once, so that a preferred design can be re-emphasized in later iterations. The fitness of the population is the normalized selection count. The program requires binary choices rather than ranking or rating inputs based on the belief that this is a natural way for subjects to express their preferences. The disadvantage of the scheme, however, is that it assumes a higher count to correspond to a more preferred design. This may not be always the case. As an example, Design A that appears early can be picked repeatedly because it is a design relatively better than the others present; Once a better Design B comes into play, the user may drop A and pick B which will have only one count.

Once fitness is determined, a multi-armed roulette wheel is used for parent selection. Figure 1.2 illustrates an example of this selection scheme where four parents, represented by the four arms, are chosen from a candidate set of four. The size of the pie segment represents the relative fitness of these candidates. According to the figure, the parent set consists of one copy of Designs 1 and 2 and two copies of Design 4.

These parents are then paired in the pattern of (1, 3) and (2, 4) for crossover. We use two random parameters  $j$  and  $\alpha$  to perform the crossover on the parent pair  $\mathbf{a}$

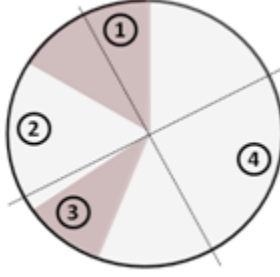


Figure 1.2: Multi-armed roulette wheel parent selection

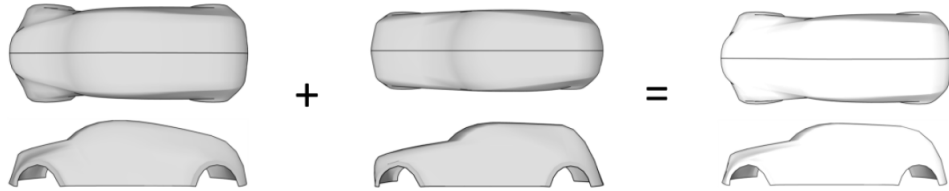


Figure 1.3: An example of crossover. Here the two grayed designs are parents and the white one is a child. The child acquires the hood design of the first parent and the silhouette design of the second parent.

and  $\mathbf{b}$ , to create the children  $\mathbf{a}'$  and  $\mathbf{b}'$  in the form:

$$\begin{aligned}\mathbf{a}' &= [\alpha\mathbf{a}_1 + (1 - \alpha)\mathbf{b}_1, \alpha\mathbf{b}_2 + (1 - \alpha)\mathbf{a}_2], \\ \mathbf{b}' &= [\alpha\mathbf{b}_1 + (1 - \alpha)\mathbf{a}_1, \alpha\mathbf{a}_2 + (1 - \alpha)\mathbf{b}_2].\end{aligned}\tag{1.3}$$

Here  $\mathbf{a}_1, \mathbf{b}_1$  are vectors of the first  $j$  elements in the parents, and  $\mathbf{a}_2, \mathbf{b}_2$  are the remaining part of the parents. This split and recombination scheme allows fusion of design features from parents as illustrated in Figure 1.3. Mutation then takes place by changing every variable in each child chromosome by a random amount while maintaining all variable values within their bounds.

### 1.3.1.3 IGA drawbacks

Our experiment with the vehicle exterior design interaction shows that this IGA setup is helpful at exploring the design space but is not efficient enough at converging to a preferred design. Although appealing, the IGA framework has been criticized for

its value in practice. Some drawbacks of IGA are included below and they lead to our motivation for developing more efficient search algorithms for preference elicitation.

- Convergence of IGAs is usually not guaranteed. This is mainly due to the inevitable fluctuation in the subject’s preference during the interaction. Therefore the fitness of designs will not be accurately obtained based on ranking, rating or binary choice input from the subject. The optimal solution of the IGA is considered to be a domain rather than an exact point due to the fact that subjects cannot distinguish the difference between one design and another when both fall below the level of least distinguishable difference. One attempt to improve the convergence in practice is to allow human intervention during the interaction. As *Kosorukoff (2001)* has shown, allowing manual modification of chromosomes can essentially speed up the convergence of IGA in a color matching experiment. This amendment however, does not prevent human fatigue from fluctuating the preference, which is believed to be one of the major factors frustrates IGA practices. Based on the review of a large variety of IEC applications and developments in domains including art and design, data mining, education and social systems, *Takagi (2001)* concludes that the growth in the population size, number of generations and the dimensionality of the design space will all cause the onset of human fatigue, thus limiting the performance of an IEC.
- The IGA framework does not prevent designs close to the not-preferred ones from reappearing in future iterations. Although not-preferred designs have low probability to be selected as candidate parents, the stochastic nature of the parent selection, crossover and mutation schemes will still produce unfavorable designs regardless of user inputs, leading to a frustrating experience with IGA. In fact, there is no theoretically “correct” schemes for crossover and mutation, which makes the search almost pure random except for the parent selection

stage.

### 1.3.2 Recommender systems

Applications that utilize consumer preferences and provide recommendations are abundant these days. The term “recommender system” refers to all intelligent systems including the Amazon product recommender (*Linden et al. (2003)*), the Google personalized news feeder (*Das et al. (2007)*) and many others. A thorough review of state-of-art recommender systems and researches is given by *Adomavicius and Tuzhilin (2005)*. In its most common formulation, the recommendation problem is a problem of estimating the utilities of items that have not yet been seen by a subject, based on the utilities the subject assigned to some other items, the personal profile of the subject or even the utilities assign by other subjects. Formally, let  $\mathcal{V}$  be an item (feature) space where each item is represented as a set of features  $\mathbf{v}$ . Let  $c$  be a subject and  $\mathcal{C}$  be the entire subject set. Let  $f(c, \mathbf{v})$  be the utility function of subject  $c$ , i.e.,  $f : \mathcal{C} \times \mathcal{V} \mapsto \mathbb{R}$ . Then, for each subject  $c$ , we want to find such items that maximize the user’s utility:

$$\mathbf{v}^* = \arg \max_{\mathbf{v}^* \in \mathcal{V}} f(c, \mathbf{v}). \quad (1.4)$$

The central problem of recommender systems lies in that utility  $f$  is usually not defined on the whole space  $\mathcal{C} \times \mathcal{V}$ , but only on those items that have been evaluated. Therefore a prediction of  $f$  is required based on the limited observations. Many different ways can be employed for this purpose based on the structure of user evaluation data, including machine learning and statistical approaches. Recommender systems can be classified into the following categories based on how recommendations are made (*Balabanović and Shoham (1997)*):

- Content-based recommendations: The user will be recommended items similar

to the ones the user preferred in the past;

- Collaborative recommendations: The user will be recommended items that people with similar tastes and preferences liked in the past;
- Hybrid approaches: These methods combine collaborative and content-based methods.

### 1.3.2.1 Content-based recommender systems

The content-based recommender system is the most similar to the preference elicitation problem we defined above. Although rooted in text-based applications, such as documents, websites and news messages retrieval, the techniques developed for content-based recommender systems can be applied to many other artefacts or services, including movies, books, restaurants and other commercial products. In practice, for a certain subject, various candidate items that have not been seen by this subject are compared to a previous set that has been viewed and utility measures are calculated. More formally, a subject  $c$  is represented by a profile vector  $\mathbf{w}_c$  based on the subject's previous ratings where each element  $w_{ic}$  represents how much the subject prefers the  $i$ th feature of the item. The utility of a candidate item is then calculated as a function of the subject's profile and the features of this item. One commonly used form of utility in information retrieval literature (*Baeza-Yates and Ribeiro-Neto (1999)*; *Salton (1989)*) is the cosine similarity measure:

$$f(\mathbf{w}_c, \mathbf{v}) \triangleq \cos(\mathbf{w}_c, \mathbf{v}) = \frac{\mathbf{w}_c^T \mathbf{v}}{\|\mathbf{w}_c\|_2 \|\mathbf{v}\|_2}. \quad (1.5)$$

The subject profile  $\mathbf{w}_c$  based on previous subject behaviour can be realized using different techniques based on the type of the behaviour, e.g., labelling, rating, ranking and others. For example, classifiers using Bayesian (*Mooney et al. (1998)*; *Pazzani and Billsus (1997)*), machine learning, decision trees and neural network (*Pazzani*

and *Billsus (1997)*) techniques can be used to realize the profile and thus the utility.

As reviewed in *Adomavicius and Tuzhilin (2005)*, one of the major limitations of content-based recommender systems is “overspecialization”, referring to the situation where the system recommend only the items that are similar to those that have already been rated. Therefore the subject will not be given recommendations of possibly favorable items that are far away from those exposed in the feature space. As an example given in *Adomavicius and Tuzhilin (2005)*, a person with no experience with Greek cuisine would never receive a recommendation for even the greatest Greek restaurant in town. This problem is usually addressed by introducing randomness to the system, e.g., genetic algorithm (*Sheth and Maes (1993)*). The other aspect of the overspecialization problem is that recommended items can be too similar to things that the subject has already seen. To address this, researchers proposed to not only filter out the dissimilar items but also the most similar items (*Billsus and Pazzani (2000)*) or only recommend items that have both high similarity and novel information (*Zhang et al. (2002)*). As a conclusion, the diversity of recommendations is often desirable and the subject should be presented with a range of options rather than replicates of the previously viewed items. The other difficulty of a content-based recommender system is that a newcomer to the system is not likely to get useful information since the system has very limited information to capture his profile.

### 1.3.2.2 Collaborative filtering

Collaborative filtering is the opposite way of doing recommendation from the content-based approach. In collaborative filtering, the subject receives recommendations that are rated well by other subjects similar to her. In memory-based collaborative recommendations, the similarity between two people is measured as the cosine or correlation between their profiles, which contain rating, browsing or purchasing histories and demographic information (*Pazzani (1999)*). Taking a slightly different



approach, a model-based collaborative filter will first update a cluster model or a Bayesian network of the cumulative subject rating data and predict ratings based on that model (see *Breese et al. (1998)*). *Billsus and Pazzani (1998)* show that the same modeling purpose can also be fulfilled with machine learning techniques. The difference between memory-based and model-based collaborative filters is that while the former rely on subject inputs as they are, the latter summarize these inputs with a model before prediction. Instead of using similarities between people, *Linden et al. (2003)* proposed to use item-to-item collaborative filtering in order to speed up a recommendation. The idea is to have an item-to-item similarity table computed offline and find from this table only the correlated items with respect to the few items a subject purchased or rated, by which means the computational cost is reduced and the online shopping experience is enhanced.

By its nature, a collaborative filter can recommend products even when they are not similar to the subjects' profile, circumventing the overspecialization problem of a content-based mechanism. However, the system will still need to get enough ratings from a new subject before its recommendation can be accurate for that subject. Similarly, a new product can be recommended only after being rated by a substantial number of users.

### 1.3.2.3 Hybrid methods

Mixing features of both content-based and collaborative recommender systems can avoid limitations of each of these methods. One simple hybrid approach is to make recommendations with a linear sum of ratings from the two systems *Claypool et al. (1999)*. Another way is to add content-based characteristics to collaborative models. Here, the similarity between two people is calculated not using their ratings, but features of items which both subjects have rated. On the other hand, one can also enhance a content-based algorithm by taking collaborative information into account.

For example, *Soboroff and Nicholas (1999)* shows that dimension reduction can be used on a population of subject profiles and latent features can be derived, which result in performance improvement.

### 1.3.3 Conjoint analysis

The last related research area we shall introduce is “conjoint analysis”, widely adopted in econometric and marketing researches and practices. Conjoint analysis and its variants have been the mainstream techniques for preference measurement since its first appearance (*GREEN and RAO (1971)*). The general idea of conjoint analysis is to collect preference data, e.g., ratings, rankings or choices on products, from a population and summarize the data with a regression model. An introduction in depth of its technical details is presented in the next chapter. Here we focus on the conceptual properties of this method and more importantly, pointing out what is missing in the current literature of conjoint analysis. As reviewed in *Netzer et al. (2008)*, a conjoint analysis study consists of three major components from a high-level viewpoint: (1) the problem that the study is intended to address; (2) the design of the preference measurement task and the data collection approach; (3) the specification and estimation of a preference model. In the traditional conjoint analysis formulation, the problem is to help firms predict market share and from there design products to maximize profit; the data are usually collected through questionnaires online or offline using consumer rating, ranking or choosing among hypothetical profiles designed according to traditional statistical efficiency measures; and, the output prediction models for a population or for individuals are commonly estimated using hierarchical Bayes techniques under the assumption of a linearly additive utility model.

A well-acknowledged dilemma in conjoint analysis is the trade-off between better estimating the preference and reducing the number of responses needed from subjects. Therefore to understand the preference of a population on a complex prod-

uct, the researcher will need to conduct a survey that either asks subjects a large number of questions or has a large pool of subjects. Some theoretical research has been developed to alleviate this problem. Utilizing the concept of complexity control from machine learning, *Cui and Curry (2005)* and *Evgeniou et al. (2005)* used a support vector machine (see Chapter II) approach to handle complex preference measurement problems. The general idea here is to form a better estimation of the preference by using some features which are nonlinear functions of the original design variables. Another promising direction is to create questions adaptively according to the previous responses from subjects. Investigations from *Toubia et al. (2007b, 2004, 2003)* show both theoretically and empirically that optimal questions with respect to some statistical criteria can be produced adaptively to achieve better estimation performance.

One divergent topic in conjoint analysis is preference modeling. The linearly additive utility model has been dominant, due mainly to its mathematical simplicity and intuitive interpretation. However, the linear additive model assumes features are compensatory, meaning that lack in one feature can be made up by an increase in other features. In addition, researchers have reported that subjects may very likely resort to simplified heuristics instead of undertaking complex decision-making processes before making a choice (see *Lloyd (2003)*). *Todd and Gigerenzer (2000)* show that in reality, people adopt fast heuristics in decision-making that involve little computation rather than trading between attributes. For example, in a pairwise choice scenario, a subject will compare the two choices by one attribute at a time. If the attribute can distinguish between the two choices then a decision is made on that basis, ignoring other present information. Research has been conducted to incorporate these non-compensatory heuristics into conjoint analysis. *Hauser et al. (2010)* proposed a generalization of non-compensatory decision rules called disjunction of conjunctions, e.g., one would like either a heavy low-mileage car for safety considera-

tions or a light high-mileage car for fuel efficiency. The authors show that results of a conjoint analysis using this consideration set model explain the user response data better.

#### 1.3.4 Summary

We reviewed work related to the questions raised in this dissertation across disciplines at a conceptual level. To summarize, each of these research directions has its own practical motivation and usage; nonetheless, the underlying research questions and developed techniques are shared in spite of the diversity in terminology. Both conjoint analysis and recommender systems deal with preference modeling. The recommender system can be considered as a real-time application of conjoint analysis, focusing more on scalability and real-time response of the algorithm, while in conjoint analysis, the major difficulty is in reducing the number of questions needed and make the interaction more adaptive. In terms of interaction, IGA has the most flexibility and is arguably the most appealing algorithm to subjects, with its capability to create virtual artefacts in real time. Also, some of the preference modeling techniques can be applied to fitness calculation in IGA as well. However, unlike the other two research areas, IGAs have not been successful in real-world applications due to slow convergence to an optimal solution and human fatigue resulting from that. The preference elicitation problem we posed in this chapter is in a sense an attempt to enhance the usability of IGA by relying more on rigorously modeling the preference than using an arbitrary evolutionary scheme. This idea shares the IGA challenge of finding an optimal solution in a small amount of iterations. The problem is similar to a recommender system since the general goal is to guess what artefacts people may like. However, a preference elicitation algorithm should be able to explore non-existing designs of artefacts and thus create new opportunities for a design study rather than promoting existing products as in a recommender system. We use Table

1.1 to summarize this section.

## 1.4 Dissertation Overview

The remainder of this dissertation is organized as follows: Chapter II provides necessary background on traditional conjoint analysis and support vector machine. Both are ways of creating models to explain data. We emphasize the key concepts behind their derivation which will be used throughout this work. In Chapter III we propose solutions to the preference elicitation problem. Both simulated tests and real subject tests are conducted and their results analyzed. Chapter IV continues this topic and provides two major improvements to the elicitation algorithm. We switch topic in Chapter V to address both the preference identification and estimation problems. Chapter VI discusses how collaborative filtering can be used in the preference elicitation framework to improve search efficiency of one subject by using search history of others. We conclude and discuss future directions in Chapter VII.

Table 1.1: Overview of related work

	IGA	Recommender system	Conjoint analysis (Preference identification and estimation)	Preference elicitation
Practical usage	Find the best design or to explore the design space	Find relevant items	Explain and predict stated or revealed preferences	Find the best design
Underlying problem	Search the optimal solution in a given space	Establish rating prediction models	Establish a preference model, and estimate its parameters	Search the optimal solution in a given space
Key difficulties	Slow convergence in high dimensions	Scalability in real-time applications	Limited design variables and levels	Slow convergence in high dimensions
Artefact type	Virtual artefacts	Existing artefacts	Existing artefacts	Virtual artefacts
Response type	Explicit or implicit fitness	Ratings, rankings or labels	Ratings, rankings or binary choices	Binary choices
Interaction type	Short term	Long term	Short term	Short term

## CHAPTER II

### Literature Review

In this chapter, we explore existing work related to preference elicitation from different disciplines. A major established technique for understanding consumer preferences is conjoint analysis (or discrete-choice modeling) from econometrics, which is rooted to research as early as in the 1970's (*GREEN and RAO (1971)*; *Green and Srinivasan (1978)*) and has maintained its influence throughout the past few decades (*Green et al. (2001)*). Conjoint analysis uses survey responses from a large population to create preference models for that population. The method has been broadly applied in marketing-related business practices to guide product and service design and thus enlarge a firm's market share. To the author's knowledge, the usability of any conjoint analysis mainly depends on the form of two core elements: Preference model and survey observation. Throughout its history, conjoint analysis has adopted models of preference to different degrees of complexity, as well as different types of survey observations, which mainly includes ranking, rating and discrete-choice. In a ranking or rating test, the subject assigns orders or rates to a set of products of different attribute levels; while in a discrete-choice setting, the subject directly chooses the most acceptable product (implying one) from the given set. It is widely acknowledged that discrete-choice is more successful than other survey types since it closely simulates human decision making in real life. Therefore we focus mainly on the discrete-choice

survey both in this review and in the dissertation work.

The other important background knowledge domain we shall cover in this chapter is support vector machine (SVM). Originally introduced to the machine learning community by [Vapnik \(1982\)](#), SVM became a state-of-art supervised-learning tool broadly applied in industry (see [Tong and Chang \(2001\)](#); [Tong and Koller \(2002\)](#); [Chang et al. \(2005\)](#); [Joachims \(2002\)](#) for example and references from [Cristianini and Shawe-Taylor \(2000\)](#)) as well as various research disciplines including preference-related studies (see [Evgeniou et al. \(2007\)](#); [Chapelle and Harchaoui \(2005\)](#)). In this chapter, we will go over the theories behind SVM and derive its primal and dual problem formulations. We will cover its application to preference elicitation in [Chapter III](#) and a deeper understanding of its geometric meaning in [Chapter V](#).

## 2.1 Conjoint Analysis

Let us now consider designs as vectors of attribute levels. For example, one can represent different car models with their 0-60 acceleration time, miles per gallon, and safety performance. Hence we denote a design as a vector  $\mathbf{x}$ , where its components  $x_i$ , for  $i = 1, \dots, p$ , are levels for the  $p$  attributes. For a given set of  $n$  designs:  $\mathbf{X}_n \triangleq \{\mathbf{x}_i\}_{i=1}^n$ , a discrete-choice model provides the probability of each design being chosen. To achieve this purpose, one usually requires observations from a population of subjects, each of which represents a decision-making scenario where a certain design is chosen from a discrete choice set  $\mathbf{X}_n$  by a subject. We shall now derive the mathematical model for a choice decision and list assumptions used during the derivation.

### 2.1.1 Utility and choice probability

To interpret a choice decision, we introduce the utility  $f_i$  as a real value assigned to design  $i$ . For a pair of designs  $\mathbf{x}_1$  and  $\mathbf{x}_2$ ,  $f_1 > f_2$  if and only if  $\mathbf{x}_1$  is chosen against



$\mathbf{x}_2$ . Therefore the utility is a way to measure preference, and by using the utility we assume that preference differences always exist and are not introduced as responses to choice sets. We acknowledge that a controversy for this assumption exists and is discussed, for example, in [MacDonald et al. \(2009\)](#). For a design  $\mathbf{x}$ , its utility can be further written as the following:

$$f = V(\mathbf{x}) + \varepsilon, \tag{2.1}$$

where  $V(\mathbf{x})$  is a non-stochastic value and reflects the “representative” taste of the population on the design  $\mathbf{x}$ , and  $\varepsilon$  is stochastic and reflects the idiosyncracies of subjects in tastes for the alternative with  $\mathbf{x}$  ([McFadden \(1973\)](#)).  $V(\mathbf{x})$  is usually considered a weighted summation of design features:

$$V(\mathbf{x}) = \sum_{i=1}^k w_i v_i(\mathbf{x}), \tag{2.2}$$

where  $\mathbf{v}(\mathbf{x})$  are the  $k$  design features and  $w_i$  are model parameters (usually called “part-worths” in econometrics literature). We introduce the concept of design features to be different from design variables because assuming variables to have a linear effect on utility is often controversial. For example, the size of a MP3 player shall be considered as a design variable but the preference is not likely to be linearly dependant on this variable. Rather, a concave function of the size can be used as a design feature that has linear relationship with the preference. We leave the discussion on how features can be identified to Chapter [V](#) where we deal with the nonlinearity of preference.

Let us now look into the probability of choosing one design from a pair. Denote the conditional distribution of  $\varepsilon_2$  for a given  $\varepsilon_1$  be  $p(\varepsilon_2|\varepsilon_1)$ . The probability of  $\mathbf{x}_1$

being chosen against  $\mathbf{x}_2$  has the following form:

$$\begin{aligned}
\Pr(f_1 > f_2) &= \Pr(V_1 + \varepsilon_1 > V_2 + \varepsilon_2) \\
&= \Pr(\varepsilon_2 < V_1 - V_2 + \varepsilon_1) \\
&= \int_{\varepsilon_1=-\infty}^{\infty} \int_{\varepsilon_2=-\infty}^{V_1-V_2+\varepsilon_1} p(\varepsilon_2|\varepsilon_1)d\varepsilon_2d\varepsilon_1.
\end{aligned} \tag{2.3}$$

Extending to a general case with  $q$  designs in one query, the probability of choosing design  $i$  will be

$$\begin{aligned}
\Pr(f_i > f_j, \forall j \neq i) &= \Pr(\varepsilon_j < V_i - V_j + \varepsilon_i, \forall j \neq i) \\
&= \int_{\varepsilon_i=-\infty}^{\infty} \left( \int_{\varepsilon_1=-\infty}^{V_i-V_1+\varepsilon_i} \int_{\varepsilon_2=-\infty}^{V_i-V_2+\varepsilon_i} \cdots \int_{\varepsilon_q=-\infty}^{V_i-V_q+\varepsilon_i} p(\hat{\boldsymbol{\varepsilon}}|\varepsilon_i)d\hat{\boldsymbol{\varepsilon}} \right) d\varepsilon_i,
\end{aligned} \tag{2.4}$$

where  $\hat{\boldsymbol{\varepsilon}}$  represents the set  $\varepsilon_1, \dots, \varepsilon_{i-1}, \varepsilon_{i+1}, \dots, \varepsilon_q$ . Several different likelihood models we discuss below are direct consequences of this choice interpretation.

### 2.1.2 Likelihood models

For a sequence of  $n$  queries each with a set of  $q$  designs  $\mathbf{x}_{j_i}$  for  $i = 1, \dots, n$  and  $j = 1, \dots, q$ , let the chosen design in the  $i$ th query have index  $j_i^*$ . Following Equation 2.4, the likelihood of such an observation is

$$\begin{aligned}
L &= \Pr(f_{j_i^*i} > f_{j'i}, \forall i = 1, \dots, n, j' \neq j_i^*) \\
&= \prod_{i=1}^n \Pr(f_{j_i^*i} > f_{j'i}, \forall j' \neq j_i^*).
\end{aligned} \tag{2.5}$$

Substituting utilities as a function of design features using Equation (2.2), the likelihood can be rewritten as a function of the unknown feature parameters  $\mathbf{w}$ :

$$L(\mathbf{w}) = \prod_{i=1}^n \Pr(\varepsilon_{j'i} < \mathbf{w}^T(\mathbf{v}_{j_i^*} - \mathbf{v}_{j'i}) + \varepsilon_{j_i^*i}, \forall j' \neq j_i^*). \quad (2.6)$$

When  $p(\hat{\boldsymbol{\varepsilon}}|\varepsilon_i)$  is a multivariate normal distribution, Equation 2.6 is called the multinomial probit model. Clearly this model is not a closed-form function of  $\mathbf{w}$ . To circumvent this difficulty, it is further assumed that  $\varepsilon_{ji} \forall i, j$  are independently and identically distributed following the extreme value (Gumbel) distribution:

$$p(\varepsilon|\theta) = \frac{1}{\theta} e^{-\varepsilon/\theta} e^{-e^{-\varepsilon/\theta}}, \quad (2.7)$$

from where we can derive the simplified choice probability:

$$\Pr(f_i > f_j, \forall j \neq i) = \frac{e^{V_i/\theta}}{\sum_{j=1}^q e^{V_j/\theta}}. \quad (2.8)$$

It is usually assumed that the the Gumbel distribution used here is standard, i.e.,  $\theta = 1$ , which leads to

$$\Pr(f_i > f_j, \forall j \neq i) = \frac{e^{V_i}}{\sum_{j=1}^q e^{V_j}}. \quad (2.9)$$

Inserting this into Equation (2.4), we arrive at the logit model for likelihood:

$$\begin{aligned} L(\mathbf{w}) &= \prod_{i=1}^n \frac{e^{V_{j_i^*i}}}{\sum_{j=1}^q e^{V_{ji}}} \\ &= \prod_{i=1}^n \frac{e^{\mathbf{w}^T \mathbf{v}_{j_i^*i}}}{\sum_{j=1}^q e^{\mathbf{w}^T \mathbf{v}_{ji}}} \end{aligned} \quad (2.10)$$

which now has a closed form. The advantage of this formula is that it gives an intuitive interpretation of the choice probability in terms of the relative utilities of alternatives. It also provides amenable computation for the likelihood. Nonetheless,

the logit model requires the assumption that a choice is “independent from irrelevant alternatives” (IIA) and could be problematic in practice. The IIA assumption implies that the ratio between choice probabilities of two designs, as shown in Equation (2.11), will not be affected by the introduction of a new alternative.

$$\frac{\Pr(f_1 > f_j, \forall j \neq 1)}{\Pr(f_2 > f_j, \forall j \neq 2)} = \frac{e^{V_1}}{e^{V_2}}. \quad (2.11)$$

A classical example illustrates this problem (*McFadden (1973)*). Suppose we have a (red) bus and a personal automobile as the two commuting options, and two-thirds of the population choose to use the automobile. If we introduce a second (blue) bus that is identical to the first one, intuitively, we expect that auto users will not switch to a bus because of this change and bus users will split between the two buses. However, if we calculate the share of automobile and bus usage according to the logit model, we will find half of the population using the personal automobile and the other half split between the two buses. The reason this is counter-intuitive is that we expect individuals to make a choice between auto and bus before deciding which color bus to choose (if they cared). Therefore the logit model is properly used only when designs are distinct and evaluated independently in a query.

Variations of the multinomial logit model have been proposed to mitigate criticisms related to the assumed IIA property, among which mixed logit is prevalent. In fact *McFadden and Train (2000)* proved that the mixed logit model can approximate any choice model including multinomial probit. The reverse, however, is not true since multinomial probit model requires the error term to be normal. Mixed logit assumes  $\mathbf{w}$  to be random following a joint distribution  $p(\mathbf{w}|\boldsymbol{\theta})$  where  $\boldsymbol{\theta}$  are the fixed parameters characterizing the distribution and  $\mathbf{w}$  is defined on a set  $\mathcal{V}$ . The likelihood

corresponding to Equation (2.10) under  $\boldsymbol{\theta}$  becomes

$$L(\mathbf{w}, \boldsymbol{\theta}) = \prod_{i=1}^n \int_{\mathbf{w} \in \mathcal{V}} \frac{e^{\mathbf{w}^T \mathbf{v}_{j_i^* i}}}{\sum_{j=1}^q e^{\mathbf{w}^T \mathbf{v}_{ji}}} p(\mathbf{w} | \boldsymbol{\theta}) d\mathbf{w}. \quad (2.12)$$

Examination of Equation (2.12) reveals that the choice probability is a linear mixture of logit probabilities, with the weight of each logit probability determined by the mixing distribution  $p(\mathbf{w} | \boldsymbol{\theta})$ . The introduction of this mixing distribution relaxes the IIA assumption since one can easily examine the following inequality:

$$\begin{aligned} \frac{\Pr(f_1 > f_j, \forall j \neq 1)}{\Pr(f_2 > f_j, \forall j \neq 2)} &= \frac{\int \exp(\mathbf{w}^T \mathbf{v}_1) / \sum_{j=1}^q \exp(\mathbf{w}^T \mathbf{v}_j) p(\mathbf{w} | \boldsymbol{\theta}) d\mathbf{w}}{\int \exp(\mathbf{w}^T \mathbf{v}_2) / \sum_{j=1}^q \exp(\mathbf{w}^T \mathbf{v}_j) p(\mathbf{w} | \boldsymbol{\theta}) d\mathbf{w}} \\ &\neq \int \frac{\exp(\mathbf{w}^T \mathbf{v}_1)}{\exp(\mathbf{w}^T \mathbf{v}_2)} p(\mathbf{w} | \boldsymbol{\theta}) d\mathbf{w}. \end{aligned} \quad (2.13)$$

We only briefly introduced relaxations on the IIA property because the dissertation mainly focuses on individual tests rather than the likelihood model for a population. Readers may refer to *McFadden and Train (2000)* for details on the mixed logit model and *McFadden (1980)* for a review on different likelihood models. Nonetheless, the idea that the heterogeneous taste of a population can be described by a distribution is important. In Chapter VI, we will investigate how preference elicitation on individuals can be enhanced by previous search results.

### 2.1.3 Parameter estimation and its variance

Above we went through the derivation of several forms of the likelihood model. We now turn our view to the estimation of the model parameters from observations. For multinomial logit, the estimation of  $\mathbf{w}$  can be found by maximizing the likelihood in Equation (2.10) or, equivalently, minimizing the negative log-likelihood loss:

$$L_{ML}(\mathbf{w}) \triangleq - \sum_{i=1}^n \log \left( \frac{e^{\mathbf{w}^T \mathbf{v}_{j_i^* i}}}{\sum_{j=1}^q e^{\mathbf{w}^T \mathbf{v}_{ji}}} \right). \quad (2.14)$$

The Hessian matrix  $\Omega$  of this loss is

$$\Omega \triangleq \frac{\partial L_{ML}(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^T} = \sum_{i=1}^n \sum_{j=1}^q (\mathbf{v}_{ji} - \bar{\mathbf{v}}_i) P_{ji} (\mathbf{v}_{ji} - \bar{\mathbf{v}}_i)^T, \quad (2.15)$$

where  $\bar{\mathbf{v}}_i = \sum_{j=1}^q \mathbf{v}_{ji} P_{ji}$  and  $P_{ji} = \exp(\mathbf{w}^T \mathbf{v}_{ji}) / \sum_{j'=1}^q \exp(\mathbf{w}^T \mathbf{v}_{j'i})$ . Since  $P_{ji}$  are probabilities and thus always positive, the Hessian of the loss function is positive semi-definite. Thus we can use existing convex programming techniques to find the optimal solution  $\hat{\mathbf{w}}$ . One shall however notice that since the choice observations are probabilistic, i.e., replicates on the same choice set and the same subject may result in contradictory observations, the outcome  $\hat{\mathbf{w}}$  also has a variance around the true parameter  $\mathbf{w}$ . [McFadden \(1973\)](#) shows that the estimator is asymptotically normally distributed with the true parameter as its mean and  $\Omega^{-1}$  as its covariance. This is an important statistical property of the maximum likelihood estimation for the multinomial logit model since  $\Omega$  indicates how much we shall trust the estimation, i.e., smaller determinant of  $\Omega$  indicates larger variance of  $\hat{\mathbf{w}}$ . Examination of  $\Omega$  shows that its determinant is determined by (1) the number of queries made, (2) the arrangement of  $\mathbf{v} - \bar{\mathbf{v}}$ , and (3) the probability  $P_{ji}$  which is a function of both  $\mathbf{v}$ s and  $\hat{\mathbf{w}}$ .

Parameters of a mixed logit model can also be estimated using maximum likelihood. The extra difficulty comes from the numerical integral in Equation (2.12). The details on how numerical methods address the estimation on a mixed logit model is not elaborated here. Readers may refer to [Train \(2001\)](#); [McFadden and Train \(2000\)](#) for detailed techniques on calculating simulated choice probability.

## 2.2 Support Vector Machine (SVM)

This introduction aims to explain the motivation and development of SVM. SVM was originally introduced by Vapnik ([Vapnik \(1982, 1998\)](#)) for the supervised learning

problem: Given a query set  $\{\mathbf{x}_i\}_{i=1}^n$  sampled from a space  $\mathcal{D}$  with the corresponding label set  $\{y_i\}_{i=1}^n$ , find a classifier  $f(\mathbf{x})$  such that it has both good testing performance (i.e.,  $f(\mathbf{x}_i)$  close to  $y_i$ ) and prediction power (i.e., for  $\mathbf{x} \in \mathcal{D}$  not yet queried,  $f(\mathbf{x})$  shall be close to the true label  $y$ ). When the  $y$ 's are categorical, the learning is called “classification”; when they are real-valued, it is called “regression”. We mainly focus on binary  $y$ 's and readers may refer to [Drucker et al. \(1997\)](#) for the regression case. [Cristianini and Shawe-Taylor \(2000\)](#) provide a complete introduction of SVM.

### 2.2.1 Basic concepts

Formally, SVM finds a separation hyperplane of the two (or multiple) classes of data  $\mathbf{X}$  and  $\mathbf{y}$  where  $\mathbf{X} \triangleq \{\mathbf{x}_i\}_{i=1}^n$  is a data matrix and  $\mathbf{y} \triangleq \{y_i\}_{i=1}^n$  are the labels of  $\mathbf{X}$ , with  $y_i \in \{1, -1\}$ . If these data are separable in  $\mathcal{D}$ , we can derive a separation hyperplane as shown in [Figure 2.1](#) which has the following form:

$$f(\mathbf{x}) = \hat{\mathbf{w}}^T \mathbf{x}, \tag{2.16}$$

where  $\hat{\mathbf{w}}$  is the estimator of the unknown true model. We also introduce the definition of a “decision function”:

$$\hat{h}(\mathbf{x}) = \text{sgn}(f(\mathbf{x})). \tag{2.17}$$

We call  $\{\mathbf{x} \in \mathcal{D} | \hat{h}(\mathbf{x}) = 0\}$  a “decision boundary”.

The problem of finding such a hyperplane is ill-defined since there is an infinite number of qualified hyperplanes if the data are separable. Nonetheless, it is proved that the hyperplane with the maximal “margin” will minimize the risk of overfitting (Chapter 4 in [Cristianini and Shawe-Taylor \(2000\)](#)) and is a good choice for the hyperplane. Here the “margin” is defined as the minimal distance from any query  $\mathbf{x}$  to the decision boundary, as is defined in [Equation \(2.18\)](#) and illustrated in [Figure](#)

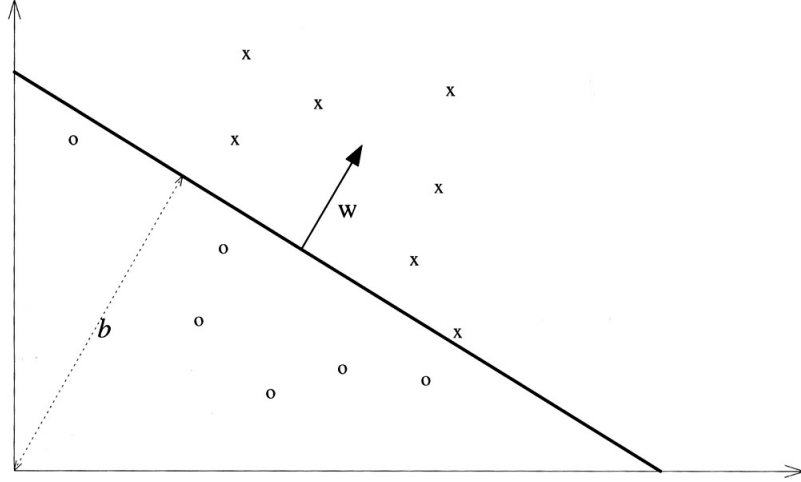


Figure 2.1: A separation hyperplane ( $\hat{\mathbf{w}}^T \mathbf{x}$ ) for a two dimensional training set. Figure from *Cristianini and Shawe-Taylor (2000)*.

2.2:

$$\gamma \triangleq \min_i \left\{ \frac{y_i \hat{\mathbf{w}}^T \mathbf{x}_i}{\|\hat{\mathbf{w}}\|_2} \right\}. \quad (2.18)$$

The samples that minimize  $\gamma$  (lie closest to the decision boundary) are called support vectors, hence the name support vector machine. The physical meaning of the phrase “support vector” can be understood intuitively as the decision boundary is “supported” only by the support vectors. This property of SVM will be explained in more depth shortly. One shall also notice that with any nonzero scaling factor  $\alpha \in \mathbb{R}$ , the margin will stay the same with any parameter set  $\{\alpha \hat{\mathbf{w}}, \alpha b\}$ . Therefore it is necessary but also safe to normalize the hyperplane so that all support vectors satisfy the equation set

$$y_i \hat{\mathbf{w}}^T \mathbf{x}_i = 1, \quad \forall i \in I. \quad (2.19)$$

Here  $I$  is the index set of support vectors and has a length of  $m$ . Maximizing the margin is then equivalent to minimizing the norm of the estimator  $\hat{\mathbf{w}}$ , leading to the primal and dual problems discussed next.



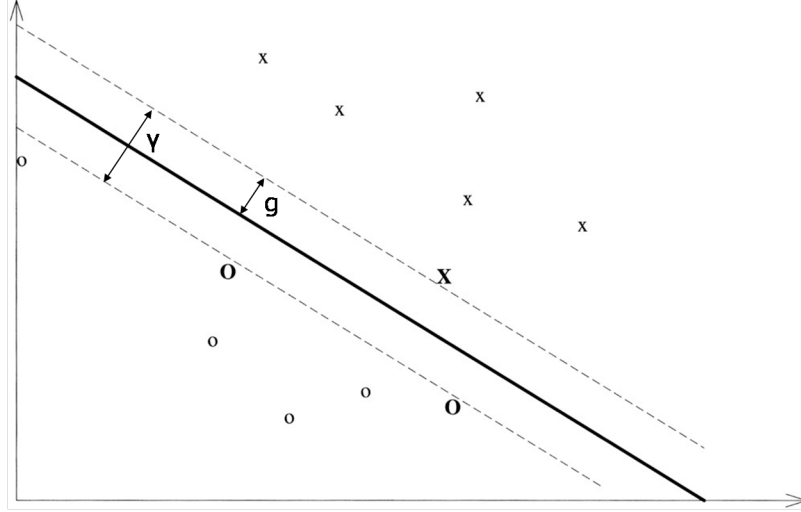


Figure 2.2:  $\gamma$  is the margin between two classes. Figure from *Cristianini and Shawe-Taylor (2000)*.

### 2.2.2 Primal and dual problems

We turn the problem of finding a classifier into the following quadratic problem:

$$\begin{aligned} \min_{\hat{\mathbf{w}}} \quad & \hat{\mathbf{w}}^T \mathbf{w}, \\ \text{subject to} \quad & y_i \hat{\mathbf{w}}^T \mathbf{x}_i \geq 1, \quad i = 1, \dots, n, \end{aligned} \tag{2.20}$$

which has a Lagrangian:

$$L(\hat{\mathbf{w}}, \boldsymbol{\alpha}) = \frac{1}{2} \hat{\mathbf{w}}^T \hat{\mathbf{w}} - \sum_{i=1}^n \alpha_i (1 - y_i \hat{\mathbf{w}}^T \mathbf{x}_i). \tag{2.21}$$

From the KKT conditions we have:

$$\hat{\mathbf{w}} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i, \tag{2.22}$$

$$\sum_{i=1}^n y_i \alpha_i = 0.$$

Inserting (2.22) into (2.21) and including the other KKT conditions, we arrive at the dual problem:

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j & (2.23) \\
\text{subject to} \quad & \sum_{i=1}^n \alpha_i y_i = 0 \\
& \alpha_i \geq 0, \quad i = 1, \dots, n.
\end{aligned}$$

Since the primal is convex and Slater’s condition holds (intuitively, one can always find a classifier that separates the classes, see Subsection 2.2.3), this problem has strong duality (no duality gap exists) and therefore solving the dual problem is equivalent to solving the primal one. From the KKT conditions we also know that the optimal solutions  $\alpha_i^*$  are only nonzero when  $y_i \mathbf{w}^T \mathbf{x}_i = 1$ . This means that only support vectors will have nonzero  $\alpha^*$ s and the estimator  $\hat{\mathbf{w}}$  is a linear combination of the support vectors. This justifies our intuition that the decision boundary is only “supported” by the support vectors. Another important observation from the dual formulation is that  $\mathbf{x}$  appears only in an vector product form. Therefore the solution  $\boldsymbol{\alpha}^*$  is affected only by the vector product matrix  $\{\mathbf{x}_i^T \mathbf{x}_j\}_{i,j=1}^n$ . This property of SVM leads to the so-called “kernel trick” which enables the above primal and dual formulations for linear classifiers tackle linearly non-separable data.

We shall also mention that in some of the SVM literature, instead of having an affine classifier  $f(\mathbf{x}) = \hat{\mathbf{w}}^T \mathbf{x}$ , one would have a linear classifier with bias  $b$ :  $f(\mathbf{x}) = \hat{\mathbf{w}}^T \mathbf{x} + b$ . The inclusion of  $b$  does not change the derivation of any algorithm but it does change the performance of an SVM algorithm in practice. When it is included, the optimal  $b$  can be computed once we have  $\boldsymbol{\alpha}^*$ :

$$b = \frac{1}{m} \sum_{i=1}^m y_i \left( 1 - \sum_{j=1}^m \alpha_j^* \mathbf{x}_{I_j}^T \mathbf{x}_{I_i} \right). \quad (2.24)$$

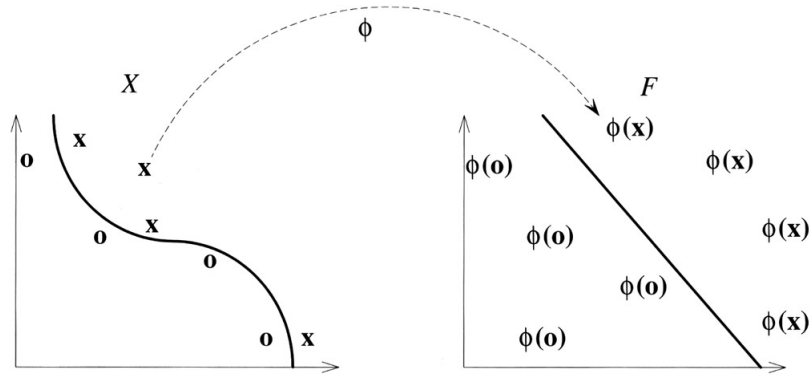


Figure 2.3: Mapping data to a different space where they can be linearly separated. Figure from *Cristianini and Shawe-Taylor (2000)*.

### 2.2.3 Kernel trick

In the case where  $\{\mathbf{x}_i\}_{i=1}^n$  are not linearly non-separable, we introduce a feature space  $\mathcal{F}$  with  $k$  dimensions and the mapping  $\phi(\mathbf{x}) : \mathcal{D} \mapsto \mathcal{F}$ :

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_k(\mathbf{x})), \quad (2.25)$$

so that the mapped data can be separated by a linear classifier (hyperplane), see Figure 2.3. Finding such a feature space and mapping is much easier than one might think at first glance. As a matter of fact, the feature space can be of any high dimensionality to ensure the linear separability of the data. To understand this conceptually, we can acknowledge that humans can always accomplish a classification task, with the least efficient but working strategy being circling every sample out and defining the union of circles as the decision boundary. *Cristianini and Shawe-Taylor (2000)* provides the numerical treatment that guarantees the existence of the mapping, which is equivalent to the soft-margin SVM formulation in Subsection 2.2.4.

Once the data are mapped, the dual problem can be rewritten in the feature space  $\mathcal{F}$  with the matrix  $\{\mathbf{x}_i^T \mathbf{x}_j\}_{i,j=1}^n$  being substituted by the “kernel” matrix (also called Gram matrix):  $\{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)\}_{i,j=1}^n$ . Since  $\phi$  are arbitrary, the kernel matrix is indeed a matrix of generalized inner products of samples  $\mathbf{x}$ . Therefore estimators of a linear

classifier in the feature space require solutions from the following augmented dual problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K_{ij}, \\ \text{subject to} \quad & \sum_{i=1}^n \alpha_i y_i = 0, \\ & \alpha_i \geq 0, \quad i = 1, \dots, n, \end{aligned} \tag{2.26}$$

where

$$\begin{aligned} K_{ij} &= \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j) \\ &\triangleq \langle \mathbf{x}_i, \mathbf{x}_j \rangle \end{aligned} \tag{2.27}$$

The exact formulation of  $K_{ij}$  can vary but the kernel matrix must acquire the following properties:

- Be symmetric:  $K_{ij} = K_{ji}$ ;
- Satisfy the Cauchy-Shwarz inequality:  $K_{ij}^2 \leq K_{ii}K_{jj}$ ;
- Obey Mercer's Theorem:  $\mathbf{K}$  must be positive semi-definite since it represents a vector product of samples in the feature space; this property also ensures the concavity of the dual problem.

Some commonly used kernels satisfying the above properties are the following:

- Polynomial (homogeneous):  $K_{ij} = (\mathbf{x}_i^T \mathbf{x}_j)^\lambda$ ;
- Polynomial (inhomogeneous):  $K_{ij} = (\mathbf{x}_i^T \mathbf{x}_j + 1)^\lambda$ ;
- Radial basis (Gaussian):  $K_{ij} = \exp(-\lambda \|\mathbf{x}_i - \mathbf{x}_j\|_2^2)$ ,  $\lambda > 0$ ;
- Hyperbolic tangent:  $K_{ij} = \tanh(\lambda_1 \mathbf{x}_i^T \mathbf{x}_j + \lambda_2)$ ,  $\lambda_1 > 0$  and  $\lambda_2 < 0$ .

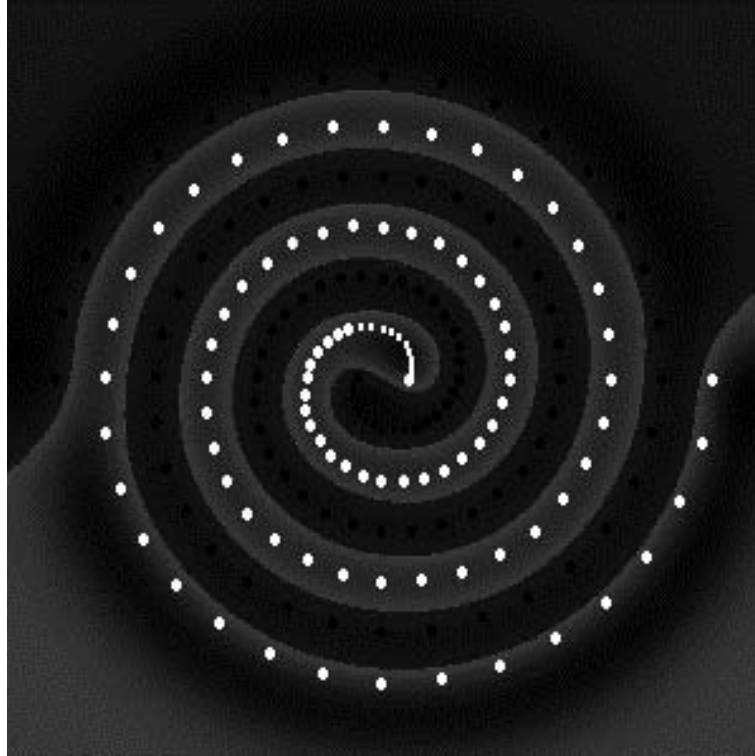


Figure 2.4: Two classes separated by a hyperplane in the feature space (Gaussian kernel). Figure from *Cristianini and Shawe-Taylor (2000)*.

Figure 2.4 illustrates how a linear separation in the feature space induced by a Gaussian kernel can work for linearly non-separable data in the original design space.

The choice of kernel is usually case dependent although it is reported that the Gaussian kernel usually performs well (*Chang and Lin (2011)*). This is also confirmed by most SVM application literature. The kernel parameters need to be tuned before one can get close to optimal performance of the classifier. Theoretically, since the optimal kernel parameters are solutions to the same maximum margin problem we solve for  $\hat{\mathbf{w}}$ , we could have a nested optimization where the optimal classifier

parameters and kernel parameters are solved iteratively:

$$\begin{aligned}
& \max_{\lambda} && \sum_{i=1}^n \alpha_i^* - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i^* \alpha_j^* y_i y_j K_{ij}(\lambda) \\
\text{subject to} &&& \boldsymbol{\alpha}^* = \arg \max_{\boldsymbol{\alpha}} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K_{ij}(\lambda) \\
&&& \sum_{i=1}^n \alpha_i y_i = 0 \\
&&& \alpha_i \geq 0, \quad i = 1, \dots, n.
\end{aligned} \tag{2.28}$$

While the maximum margin problem is concave with regard to the classifier parameters ( $\boldsymbol{\alpha}$ ), it is not so when it comes to the kernel parameters  $\lambda$ . Therefore in practice it is suggested that a near-optimal  $\lambda$  can be found using a grid search ([Chang and Lin \(2011\)](#)).

#### 2.2.4 Soft margin formulation and complexity control

So far we assumed that there exists a hyperplane that perfectly separates the two classes in the feature space. We argued that such a space and mapping can always be found theoretically. In practice however, one would like to find a classifier that not only performs well on the training data, but also does well in prediction. In other words, a classifier should be preferred if it enlarges the margin a lot by only violating a small amount of training observations, so that it avoids overfitting the data and provides better prediction accuracy. The theoretical support for this intuition is the proof given by Vapnik that the prediction error is bounded above by the following inequality with probability  $1 - \eta$ , provided that test data are independent and identically distributed as the training data:

$$\text{error}_{\text{predict}} \leq \text{error}_{\text{train}} + \sqrt{\frac{h(\log(2n/h) + 1) - \log(\eta/4)}{n}}. \tag{2.29}$$

Here  $h$  is a measure of model complexity, and  $n$  is the training size. In the context of SVM, the model complexity can be represented by the norm  $\|\hat{\mathbf{w}}\|_2$ . To elaborate, first recall that the decision boundary is determined by the support vectors and the boundary is more complex when more support vectors are involved. It can also be shown that adding data to the support vector set will increase the norm  $\|\hat{\mathbf{w}}\|_2$  which has the formula

$$\|\hat{\mathbf{w}}\|_2^2 = \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \quad (2.30)$$

from the KKT conditions. This explains the connection between a classifier margin and its complexity. Examination of the inequality (2.29) reveals that decrease in the prediction error requires smaller training error but also a less complex model, which leads to the trade-off between model complexity and training performance because increasing the complexity will usually lower the training error. To balance this trade-off, we rewrite the maximum margin problem to include violations to observations:

$$\begin{aligned} \min_{\hat{\mathbf{w}}} \quad & \frac{1}{2} \hat{\mathbf{w}}^T \hat{\mathbf{w}} + C \sum_i^n \xi_i \\ \text{subject to} \quad & y_i \hat{\mathbf{w}}^T \mathbf{x}_i \geq 1 - \xi_i, \quad i = 1, \dots, n \\ & \xi_i \geq 0, \quad i = 1, \dots, n. \end{aligned} \quad (2.31)$$

Here the parameter  $C$  controls the weights of complexity and training error, which can also be tuned using a grid search or other methods (for example, see [Chapelle et al. \(2002\)](#)). This formulation of a classification problem is called “soft margin SVM” and was proposed by [Cortes and Vapnik \(1995\)](#). The difference between this formulation and traditional training formulations, such as ordinary least square regression and maximum likelihood estimation, is that while the latter optimize the training error, soft margin SVM tries to optimize the overall performance of training and prediction, or the so-called “generalization error”. The success of soft margin SVM shows that when training a model, what matters is not just the training error but the model

complexity as well. Researches have applied complexity control to conjoint analysis (*Toubia et al. (2007a)*; *Evgeniou et al. (2007)*), and we will use the same technique in Chapter V when dealing with pairwise choices.

## 2.3 Conclusion

Two technical reviews are provided in this chapter. In the first part we covered elementary concepts in conjoint analysis (discrete-choice modeling). Most of the derivation techniques and terminologies will be repeatedly used in the following chapters. A notable aspect of this topic we did not discuss is the hierarchical Bayes (HB) method used in mixed logit estimation (*Train (2001)*). This is because the studies in this dissertation deal only with individual data where mixed logit is rarely needed. It is also worth to mention that machine learning methods have been recently introduced to deal with the aggregated subject choice data, and have been reported to have overall better performance than HB (*Evgeniou et al. (2007)*).

One important idea to take away from this review is that the variance of a preference model estimation is associated with the query size as well as the arrangement of the queries (what question to ask). This leads to the practically important trade-off between the quality of a questionnaire versus its length, as mentioned in the introduction. We investigate this problem in Chapter V.

The second part of this review focused on the essential concepts of SVM: quadratic maximum margin algorithm, kernel trick from the dual problem and most importantly, complexity control. Fundamentals such as the Vapnik-Chervonenkis theory are avoided since they are beyond the scope and needs of the present work. References we cited in this section (e.g. *Vapnik (1998)*; *Cristianini and Shawe-Taylor (2000)*) provide more insights into these topics.



## CHAPTER III

### Preference Elicitation

This chapter is motivated by the observation that while consumer preference plays an essential role in product design, its effective integration within design optimization remains challenging. Research in linking preference models to engineering optimization models for consumer product design (e.g., *Wassenaar et al. (2005)*; *Kumar et al. (2007)*; *Li and Azarm (2000)*; *Michalek et al. (2005)*) has demonstrated how these different disciplines can be integrated. Most of these demonstrations utilize preference models with only a small number of variables since the variance in coefficient estimation of these models can be large and unfavorable when the dimensionality of the design space becomes high. Yet, to capture the right design variables in an essentially qualitative or holistic design problem (vehicle styling is an example) we must have sufficient design freedom and thus a high-dimensional design space. This “curse of dimensionality” is alleviated by static design of experiments (e.g., *Kuhfeld et al. (1994)*; *Hoyle et al. (2009)* on D-efficient sampling) as well as adaptive methods (e.g. *Toubia et al. (2004)*; *Abernethy et al. (2008)* on adaptive sampling). However, these methods are all model-based, i.e., tests are designed so that they can estimate effectively the preference models. Utility models are prevalent in preference research, but the validity of such models is still under discussion, see *Netzer et al. (2008)*.

An alternative way to capture preferences without a preference model, as we

discussed in Chapter I, is through Interactive Evolutionary Computation (IEC) where the computer gradually refines its assessment of people’s preferences based on user feedback. Two drawbacks of IEC mentioned previously (Subsection 1.3.1.3) are:

- The fitness requests force users to assign values for each individual design, which is not a natural way for people to express preferences and can cause user fatigue and hamper convergence (*Takagi (2001)*);
- Designs close to not-preferred designs presented earlier are likely to appear in later iterations due to the stochastic nature of IEC schemes.

This chapter introduces an interaction similar to that in IEC but relieving the burden on users by requiring only binary feedback. Users assign only “preferred” and “not-preferred” labels to designs. The goal is to investigate how a search algorithm should be designed to elicit effectively the most preferred designs using this binary information with a tolerable interaction effort, namely, how to converge with a small number of iterations. Like with IEC, the focus here is not preference modeling and estimation but structuring the interaction to speed up convergence with limited elicitation (data collection). Two algorithms are proposed: 1) SVM Search is a heuristic method that shrinks the search space by combining knowledge at different stages of a questionnaire; 2) EGO search is a modification of the well-established Efficient Global Optimization (EGO) algorithm that samples the design space based on the response surface of current observations. Results from simulated tests will show that with only binary feedback available, the SVM search algorithm is more effective than a genetic algorithm, while EGO search is better than both.

### 3.1 Efficient Global Optimization (EGO)

EGO (*Jones et al. (1998)*) is a global optimization method designed for black-box objectives and constraints. In a minimization problem, EGO queries a point that

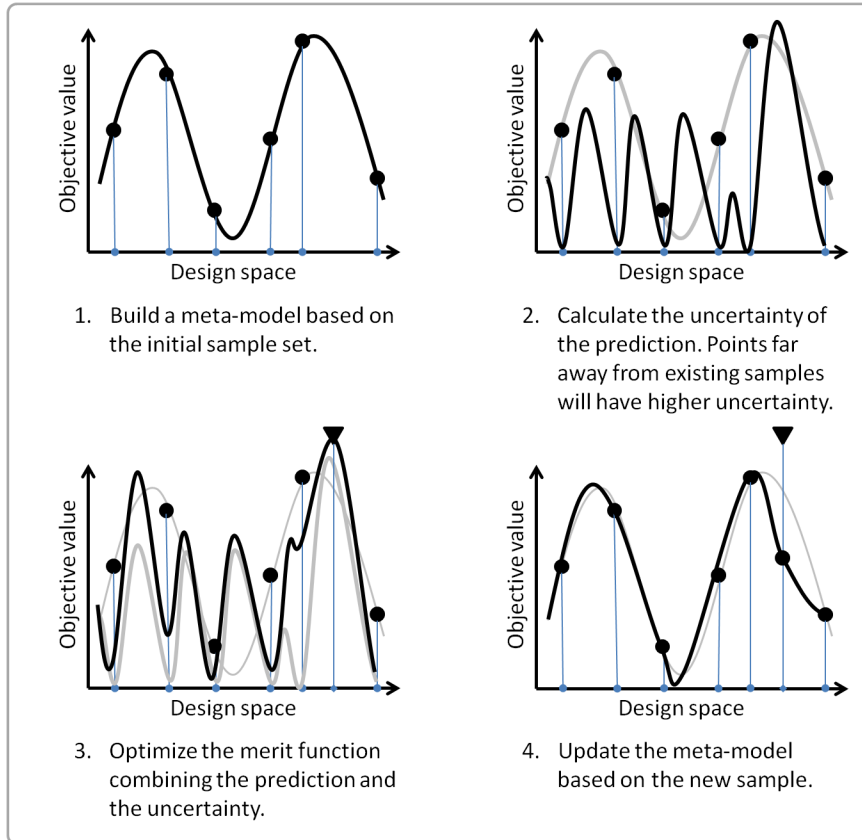


Figure 3.1: EGO operation concept.

has low predicted objective value and high variance in the prediction. It is shown that such a point has the highest expected improvement in minimizing the objective. In this section, we provide an overview of the EGO algorithm followed by technical details. The major steps that EGO takes are the following, as is also shown in Figure 3.1:

1. Build a kriging model for the given data.
2. Build the uncertainty function of the prediction using the kriging parameters.
3. Build and optimize the expected improvement function based on the kriging and uncertainty model.
4. Query the optimal solution derived above and go back to Step 1.

Below we discuss the technical details of these steps.

### 3.1.1 Kriging modeling

The kriging modeling technique is usually more appropriate than ordinary linear regression in applications related to deterministic computer simulations, where measurement error or noise does not exist. Here, since kriging is not directly used in our algorithms, we only lay out its derivation for completeness of the review and will not go into its conceptual development. Readers may refer to [Jones et al. \(1998\)](#) for a more detailed review on kriging. We shall also mention that this technique is identical to the generalized least square model in statistics, with a simple set of regressors (usually a constant term).

Starting with a general model for real-valued responses:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{v}(\mathbf{x}) + \varepsilon, \quad (3.1)$$

where  $f(\mathbf{x})$  is the response to  $\mathbf{x}$ ,  $\mathbf{w}$  is the unknown regression parameter vector and  $\mathbf{v}(\mathbf{x})$  are functions of  $\mathbf{x}$  and will be called features of designs throughout this work.  $\varepsilon$  is the error term representing the unexplained part of the response. The kriging model is a stochastic process model following these assumptions:

- The error term is a function of  $\mathbf{x}$  with no measurement error or noise;
- Any two errors  $\varepsilon(\mathbf{x}_i)$  and  $\varepsilon(\mathbf{x}_j)$  have a correlation so that when  $\varepsilon(\mathbf{x}_i)$  is large,  $\varepsilon(\mathbf{x}_j)$  is also likely to be large.

Following the second assumption, the correlation among errors can be represented in the Gaussian form:

$$\text{Corr}[\varepsilon(\mathbf{x}_i), \varepsilon(\mathbf{x}_j)] = \exp(-\lambda^{\text{kriging}} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2), \quad (\lambda^{\text{kriging}} > 0). \quad (3.2)$$

Here  $\lambda^{\text{kriging}}$  is a parameter that controls the Gaussian spread of the correlation: when  $\lambda^{\text{kriging}}$  is large, the correlation between  $\varepsilon(\mathbf{x}_i)$  and  $\varepsilon(\mathbf{x}_j)$  gets weak more rapidly as the

$\mathbf{x}_i$  and  $\mathbf{x}_i$  depart from each other. The kriging model predicts the response at some  $\mathbf{x}$  as an interpolation using the existing responses:

$$\begin{aligned}\hat{f}(\mathbf{x}) &= \sum_{i=1}^n a_i(\mathbf{x})f(\mathbf{x}_i) \\ &= \mathbf{a}^T \mathbf{f}(\mathbf{x}_i)\end{aligned}\tag{3.3}$$

where  $\mathbf{a}$  are interpolation parameters which can be estimated by minimizing the kriging variance, defined as the variance of the difference between the interpolation and the random variable  $f(\mathbf{x})$ . In the following derivation, we simplify the notation of  $\mathbf{f}(\mathbf{x}_i)$  as  $\mathbf{f}$ .

$$\begin{aligned}\text{Var}(f(\mathbf{x}) - \hat{f}(\mathbf{x})) &= E[(\mathbf{a}^T \mathbf{f} - f - E(\mathbf{a}^T \mathbf{f} - f))^2] \\ &= E[(\mathbf{a}^T \mathbf{f} - f)^2] \\ &= E[\mathbf{a}^T \mathbf{f} \mathbf{f}^T \mathbf{a} + f^2 - 2f \mathbf{a}^T \mathbf{f}] \\ &= E[\mathbf{a}^T \mathbf{f} \mathbf{f}^T \mathbf{a}] + \text{Var}(f) - 2E[f \mathbf{a}^T \mathbf{f}] \\ &= \mathbf{a}^T E[\mathbf{f} \mathbf{f}^T] \mathbf{a} + \sigma_\varepsilon^2 - 2\mathbf{a}^T E[f \mathbf{f}] \\ &= \mathbf{a}^T \begin{bmatrix} \text{cov}(f_1, f_1) & \text{cov}(f_1, f_2) & \cdots & \text{cov}(f_1, f_n) \\ \text{cov}(f_2, f_1) & \text{cov}(f_2, f_2) & \cdots & \text{cov}(f_2, f_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(f_n, f_1) & \text{cov}(f_n, f_n) & \cdots & \text{cov}(f_n, f_n) \end{bmatrix} \mathbf{a} \\ &\quad - 2\mathbf{a}^T \begin{bmatrix} \text{cov}(f_1, f) \\ \text{cov}(f_2, f) \\ \vdots \\ \text{cov}(f_n, f) \end{bmatrix} + \sigma_\varepsilon^2 \\ &= \sigma_\varepsilon^2 (\mathbf{a}^T \mathbf{R} \mathbf{a} - 2\mathbf{a}^T \mathbf{r}(\mathbf{x}) + 1).\end{aligned}\tag{3.4}$$

Here we used the vector  $\mathbf{r}(\mathbf{x})$  to represent the correlation between the errors at every

sampled  $\mathbf{x}_i$  and the error at  $\mathbf{x}$  and the matrix  $\mathbf{R}$  to represent the correlation of the errors among the sampled designs. The variance of  $\varepsilon$  is denoted as  $\sigma_\varepsilon$  and can be estimated by maximizing the likelihood of the current observation

$$L(b, \sigma_\varepsilon^2, \lambda^{\text{kriging}}) = \frac{1}{(2\pi)^{n/2} (\sigma_\varepsilon^2)^{n/2} |\mathbf{R}|^{1/2}} \exp\left(-\frac{(\mathbf{f} - \mathbf{V}\mathbf{w})^T \mathbf{R}^{-1} (\mathbf{f} - \mathbf{V}\mathbf{w})}{2\sigma_\varepsilon^2}\right), \quad (3.5)$$

where  $\mathbf{V}$  is a matrix with each row being a feature vector for a sampled design. This likelihood formulation is convex in both  $\sigma_\varepsilon$  and  $\mathbf{w}$  and can be maximized by its estimators:

$$\hat{\sigma}_\varepsilon = \frac{(\mathbf{f} - \mathbf{V}\mathbf{w})^T \mathbf{R}^{-1} (\mathbf{f} - \mathbf{V}\mathbf{w})}{n}, \quad (3.6)$$

and

$$\hat{\mathbf{w}} = (\mathbf{V}^T \mathbf{R}^{-1} \mathbf{V})^{-1} \mathbf{V}^T \mathbf{R}^{-1} \mathbf{f}. \quad (3.7)$$

Notice also that  $E[f(\mathbf{x})] = E[\hat{f}(\mathbf{x})] = \mathbf{w}^T \mathbf{v}(\mathbf{x})$  leads to  $\mathbf{w}^T \mathbf{V}^T \mathbf{a} = \mathbf{w}^T \mathbf{v}(\mathbf{x})$ . With this constraint, minimizing the kriging variance in Equation (3.4) provides the optimal estimator for  $\mathbf{a}$ , as a function of  $\mathbf{x}$ :

$$\hat{\mathbf{a}}(\mathbf{x}) = \mathbf{R}^{-1} \left( \mathbf{r}(\mathbf{x}) + \frac{\hat{\mathbf{w}}^T \mathbf{v}(\mathbf{x}) - \mathbf{f}^T \mathbf{R}^{-1} \mathbf{r}}{\mathbf{f}^T \mathbf{R}^{-1} \mathbf{f}} \mathbf{f} \right). \quad (3.8)$$

Inserting Equation (3.8) into Equation (3.3) we derive the kriging prediction:

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{w}}^T \mathbf{v}(\mathbf{x}) + \mathbf{r}(\mathbf{x})^T \mathbf{R}^{-1} (\mathbf{f} - \mathbf{V}\hat{\mathbf{w}}). \quad (3.9)$$

One shall notice that the kriging parameter  $\lambda^{\text{kriging}}$  can also be estimated using maximum likelihood. However, since the likelihood function is not convex in  $\lambda^{\text{kriging}}$ , the estimation of  $\lambda^{\text{kriging}}$  is usually time consuming and is therefore pre-set in this study.

### 3.1.2 Mean-squared error of the predictor

With the above derivation, we can now measure the uncertainty of prediction by the kriging variance.  $\mathbf{r}(\mathbf{x})$ ,  $\hat{\mathbf{a}}(\mathbf{x})$ ,  $f(\mathbf{x})$  and  $\hat{f}(\mathbf{x})$  are simplified as  $\mathbf{r}$ ,  $\hat{\mathbf{a}}$ ,  $f$  and  $\hat{f}$ , correspondingly.

$$\begin{aligned} \text{Var}(f - \hat{f}) &= \sigma_\varepsilon^2 (\hat{\mathbf{a}}^T \mathbf{R} \hat{\mathbf{a}} - 2\hat{\mathbf{a}}^T \mathbf{r} + 1) \\ &= \sigma_\varepsilon^2 \left( 1 - \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r} + \frac{(\hat{\mathbf{w}}^T \mathbf{v} - \mathbf{f}^T \mathbf{R}^{-1} \mathbf{r})^2}{\mathbf{f}^T \mathbf{R}^{-1} \mathbf{f}} \right). \end{aligned} \quad (3.10)$$

The kriging variance here is also called the mean-squared error of the predictor, and will be denoted as  $\hat{\sigma}^2$  hereafter. [Jones et al. \(1998\)](#) explains the meaning of this variance: The term  $-\mathbf{r}^T \mathbf{R}^{-1} \mathbf{r}$  represents the reduction in prediction error due to the fact that  $\mathbf{x}$  is correlated with the sampled points. With no correlation, i.e.,  $\mathbf{r} = 0$ , this adjustment would be zero; the term  $(\hat{\mathbf{w}}^T \mathbf{v} - \mathbf{f}^T \mathbf{R}^{-1} \mathbf{r})^2 / \mathbf{f}^T \mathbf{R}^{-1} \mathbf{f}$  reflects the uncertainty that stems from our not knowing  $\mathbf{w}$  exactly, but rather having to estimate it from the data. Geometrically, one can consider  $\hat{\sigma}^2$  as a continuous minimum distance function of  $\mathbf{x}$ , that is,  $\hat{\sigma}^2(\mathbf{x})$  is high when  $\mathbf{x}$  is not surrounded by existing samples and it is low when  $\mathbf{x}$  is close to an existing sample.

### 3.1.3 Expected improvement

The simplest way to use a response surface for “black-box” optimization is to directly query the optimal solution of the response surface, i.e.,  $\hat{f}$ . However, this approach exploits the observation and can easily lead to a local optimum. The problem of this approach is that it does not consider the uncertainty of the response surface, or in other words, it jumps to a conclusion before a thorough exploration of the design space. The EGO approach tries to balance exploitation and exploration by combining the prediction and uncertainty together when deciding where to query next. With  $\hat{f}$  and  $\hat{\sigma}^2$  available, EGO optimizes a merit function that evaluates the

expected improvement in the objective. The optimal solution to this merit function will be chosen as the next query point, in other words, the algorithm picks the point that is most likely to decrease the observed objective value further.

The expected improvement criterion is computed as follows. Let the current best observed function value be  $f_{\min}$ , assuming we face a minimization problem. Recall that the response we try to predict at some query  $\mathbf{x}$  is a random variable  $f$ . The expected improvement of this new query is defined as follows:

$$E[\text{Improvement}(\mathbf{x})] = E[\max\{f_{\min} - f, 0\}]. \quad (3.11)$$

To compute this expectation, let  $f$  follow a normal distribution  $N(\hat{f}, \hat{\sigma}^2)$  and also denote  $\Phi(\cdot)$  and  $\phi(\cdot)$  as the cumulative distribution function and the probability density function of a standard normal distribution. The expected improvement can then be represented analytically as a function of  $\mathbf{x}$  when  $\hat{\sigma} \neq 0$ :

$$\begin{aligned} f_{\text{EI}} \triangleq E[\text{Improvement}(\mathbf{x})] &= \int_{-\infty}^{f_{\min} - \hat{f}} (f_{\min} - \hat{f} - \varepsilon) p(\varepsilon) d\varepsilon \\ &= (f_{\min} - \hat{f}) \int_{-\infty}^{f_{\min} - \hat{f}} p(\varepsilon) d\varepsilon - \int_{-\infty}^{f_{\min} - \hat{f}} \varepsilon p(\varepsilon) d\varepsilon \\ &= (f_{\min} - \hat{f}) \Phi\left(\frac{f_{\min} - \hat{f}}{\hat{\sigma}}\right) + \hat{\sigma} \phi\left(\frac{f_{\min} - \hat{f}}{\hat{\sigma}}\right) \end{aligned} \quad (3.12)$$

When  $\hat{\sigma} = 0$ , by its definition,  $f_{\text{EI}} = 0$ .

### 3.1.4 Properties of the expected improvement

One property of  $f_{\text{EI}}$  is that it is monotonic with regard to  $\hat{f}$  and  $\hat{\sigma}$ . To be more specific, this expected improvement increases when  $\hat{f}$  decreases or  $\hat{\sigma}$  increases. This property is consistent with the purpose of creating the merit function: to find such



queries that either have lower predicted function values or are away from current samples.

The other important property is that  $f_{\text{EI}}$  takes zeros in most places of the design space except for a few promising regions. This is because  $f_{\min} - \hat{f}$  is usually negative since, by prediction, most  $\hat{f}$  will not be lower than the current lowest. Thus when  $\hat{\sigma}$  is small, the entire term  $(f_{\min} - \hat{f})/\hat{\sigma}$  is a large negative number, resulting in almost zero  $\Phi\left((f_{\min} - \hat{f})/\hat{\sigma}\right)$  and  $\phi\left((f_{\min} - \hat{f})/\hat{\sigma}\right)$  values. In case where  $\hat{\sigma}$  takes small values compared to  $(f_{\min} - \hat{f})/\hat{\sigma}$  on the entire design space,  $f_{\text{EI}}$  will have zeros almost everywhere in the design space except a few peaks.

### 3.1.5 Optimization on the expected improvement

Optimization on Equation (3.12) deserves special attention because (1) the function is nonlinear and usually has multiple local minima, and (2) the dimensionality of the function can be high for design problems of interesting complexity. *Jones et al. (1998)* proposed to tackle the nonlinearity with a branch-and-bound strategy. The strategy divides the design space into subregions and finds their upper bounds. Due to the monotonicity of  $f_{\text{EI}}$  with respect to  $\hat{f}$  and  $\hat{\sigma}$ , finding the upper bound of a subregion is equivalent to finding the upper bound for  $\hat{f}$  and the lower bound for  $\hat{\sigma}$  in that subregion, which involves relaxing the nonlinear problem to a linear one within that subregion. A similar approach proposed by *Siah et al. (2004)* is to apply the DIRECT search algorithm on  $f_{\text{EI}}$ . Here DIRECT, an acronym for DIvided RECTangles, is by itself a “black-box” search algorithm (see *Jones (2001)*). It divides the design space into subspaces and chooses to further divide the subspace that has the most “potential” to contain the optimal solution. The potential is usually measured jointly by the objective value at the center of the subspace and the size of the subspace.

As reported in *Jones et al. (1998)*, these methods can enhance the search efficiency for the nonlinear  $f_{\text{EI}}$ . Nonetheless, the reported problems usually have a fairly small

number of dimensions, e.g., *Jones et al. (1998)* uses 2-dimensional examples and *Siah et al. (2004)* uses applications with 3 and 4 variables. The reason is simple: When the dimensionality goes up, the number of branches or subregions becomes larger and eventually intractable. Therefore, in this chapter we resort to genetic algorithm to optimize the expected improvement.

## 3.2 SVM Search

SVM Search is a heuristic algorithm that aims to shrink the design space during the user interaction and thus force it to converge in a few iterations. The algorithm attempts to address the problem that, when the dimensionality of the design space becomes high, search convergence can become too slow for practical user-computer interaction.

### 3.2.1 The algorithm

The proposed algorithm works as follows. Let the initial feasible design space be  $\mathcal{D}_0$  and that of iteration  $i$  be  $\mathcal{D}_i$ , where  $i = 1, \dots, s$ . The algorithm starts with  $q$  random samples in  $\mathcal{D}_0$ . An SVM decision boundary  $g_1 = 0$  is generated from the data set  $\{\mathbf{x}_i, y_i\}_{i=1}^q$  once user feedback is collected. Suppose that  $q - q_2$  out of  $q$  samples are labeled as  $y = 1$ ; then for the second iteration,  $q_2$  samples are drawn from the region  $\mathcal{D}_1 \triangleq g_1 > 0$ . Each sample maximizes the minimum distance from all existing samples. The user is asked to label the combination of these new samples and the  $q - q_2$  preferred designs from the previous iteration. Once new labels are assigned and old labels are updated, a new decision function  $g_2$  is generated upon the current total data set  $\{\mathbf{x}_i, y_i\}_{i=1}^{2q-q_0}$ . We illustrate this process in Figure 3.2. Notice that not preferred designs will not be presented in the future query again and thus require no update.

To speed up convergence, the decision boundary is recorded after every  $s^*$  iter-

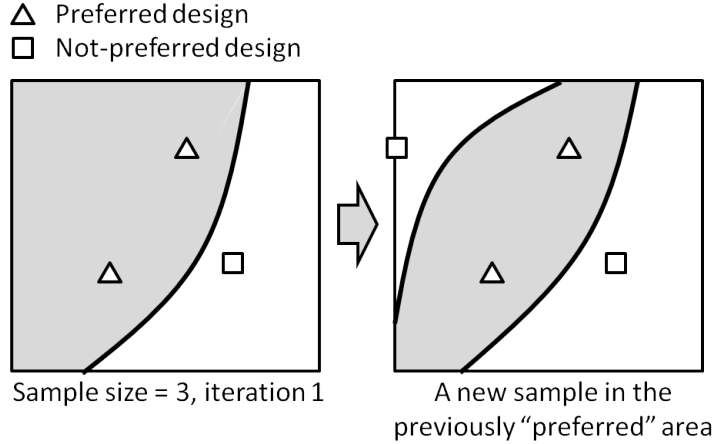


Figure 3.2: SVM Search scattering and classification.

ations, where  $s^*$  is an adjustable algorithmic parameter. Denote the design space corresponding to a recorded decision boundary as  $\mathcal{D}_r$ , the feasible design space at any iteration  $s'$  is the intersection of the design space defined by the current decision boundary  $g_{s'} > 0$  and  $\mathcal{D}_r$ , i.e.,  $\mathcal{D}_{s'} = \{\mathbf{x} \in \mathcal{D}_0, g_{s'}(\mathbf{x}) > 0\} \cap \mathcal{D}_r$ . When  $s^*$  is small, the forced shrinkage of the feasible space occurs frequently and thus the algorithm converges after a small number of iterations although it may miss the optimal solution due to the shrinkage. On the other side, when  $s^*$  has a large value, the algorithm has more places to sample and thus has more exploration capability. Therefore  $s^*$  balances exploitation and exploration of the algorithm. Figure 3.3 illustrates this space reduction procedure. The algorithm terminates when a solution is found close to the optimal solution within the tolerance set by a user or the maximum number of iterations is reached. In either situation, the best design found will be taken as the most preferred design  $\mathbf{x}^*$ . The pseudo code of this algorithm is shown as Algorithm 1 in Figure 3.4.

The “Scatter” subroutine generates designs constrained by  $\mathcal{D}_r \cap \mathcal{D}_c$ . Each design is generated so that its minimum distance to all of the other labeled data, including the designs that have just been created, is maximized. As an exception, the first call of “Scatter” carries out a Latin Hypercube design on  $\mathcal{D}_0$ . The subroutine “Label”

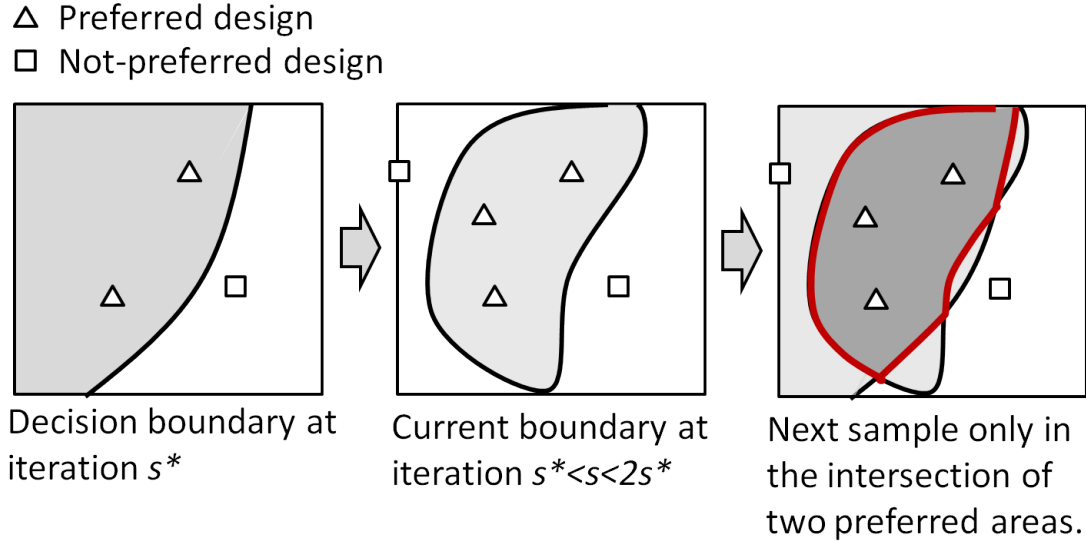


Figure 3.3: SVM Search design space reduction.

refers to human evaluations in real subject tests and is treated as a function call in the simulated tests; Once function values are obtained for each design, these values are grouped into two clusters using the K-means algorithm in Matlab. The subroutine “SVM” calls the LIBSVM package ([Chang and Lin \(2011\)](#)), and *svm\_options* stores parameters required by the package. In this study, the soft margin SVM weight  $C$  is set at  $1e6$  and no kernel parameter tuning is performed.

### 3.2.2 SVM Search Simulated Test Results

We compare results from simulated tests using SVM Search and the Matlab GA toolbox with binary fitness. Results from a random sampling scheme are also presented as a baseline performance. This latter algorithm merely samples points randomly during the process, keeping better points and dropping the rest. The comparison is conducted on the standard test functions including the 2D-Rosenbrock ([Rosenbrock \(1960\)](#)), Six-hump Camelback ([Hardy \(1975\)](#)), and Branin ([Dixon and Szegö \(1978\)](#)). These functions are used to simulate user preferences where the optimal solutions represent the most preferred designs. Notice that since our objective is

---

**Algorithm 1:** SVM Search algorithm

---

```
input :  $\mathcal{D}_0, s, q, s^*$ 
output:  $\mathbf{x}^*$ 

Initialization;
// initialize the recorded feasible space
 $\mathcal{D}_r = \mathcal{D}_0$ ;

// initialize the current feasible space
 $\mathcal{D}_c = \mathcal{D}_0$ ;

// initialize the number of preferred designs
 $q_1 = q$ ;

// initialize the preferred design set
 $\mathbf{X}_p = \emptyset$ ;

for  $i = 1$  to  $s$  do
    // scatter  $q_i$  samples in  $\mathcal{D}_0$  constrained by  $\mathcal{D}_r$ 
     $\mathbf{X}_i \triangleq \{\mathbf{x}_j\}_{j=1}^{q_i} = \text{Scatter}(\mathcal{D}_r, \mathcal{D}_c, q_i)$ ;

    // label samples
     $\{y_j\}_{j=1}^q = \text{Label}([\mathbf{X}_i, \mathbf{X}_p])$ ;

    check termination criteria;
    if Satisfied ( $\{\mathbf{x}\}$ ) then
        | break;
    end

    // train with accumulated data
     $g_i = \text{SVM}(\text{all } \mathbf{x}, \text{all } y, \text{svm\_options})$ ;

    // set the current feasible space
     $\mathcal{D}_c = \{\mathbf{x} \in \mathcal{D}_0, g_i(\mathbf{x}) > 0\}$ ;

    // set the preferred designs set
     $\mathbf{X}_p = \{\mathbf{x}_j \in \mathbf{X}_i, y_j == 1\}$ ;

    // set the number of preferred designs
     $q_{i+1} = q - |\mathbf{X}_p|$ ;

    Check if it is time to record current decision boundaries;
    if  $\text{mod}(i, s^*) == 0$  then  $\mathcal{D}_r = \mathcal{D}_c$ ;

end
return the best design found as  $\mathbf{x}^*$ ;
```

---

Figure 3.4: Proposed SVM Search algorithm

to “maximize” the preference, we setup tests in the maximization context. Therefore the tests maximize the negative of the original functions.

### 3.2.2.1 Test Function 1: Negative Rosenbrock function

The negative of the Rosenbrock function is defined as

$$f(x_1, x_2) = - \left( (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \right), \quad (3.13)$$

with

$$\mathcal{D} = \{(x_1, x_2), x_1 \in [-1.5, 1.5], x_2 \in [-0.5, 1.5]\}, \quad (3.14)$$

which has a unique global optimum at  $\mathbf{x}^* = [0, 0]$ .

### 3.2.2.2 Test Function 2: Negative Six-hump Camelback function

The second function is called Six hump camelback function, the negative of which is defined as:

$$f(x_1, x_2) = - \left( (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \right), \quad (3.15)$$

defined on the design space:

$$\mathcal{D} = \{(x_1, x_2), x_1 \in [-3, 2], x_2 \in [-3, 2]\}. \quad (3.16)$$

The negative Camelback function has two globally optimal solutions located at  $\mathbf{x}_1^* = [-0.0898, 0.7127]$  and  $\mathbf{x}_2^* = [0.0898, -0.7127]$  with the same value of  $f = 1.0316$ . It has another four local optima within this region and thus the name “six hump”.

### 3.2.2.3 Test Function 3: Negative Branin function

The third test function is the negative of the Branin function:

$$f(x_1, x_2) = - \left( (x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6)^2 + 10(1 - \frac{1}{8\pi}) \cos(x_1) + 10 \right), \quad (3.17)$$

defined on the design space:

$$\mathcal{D} = \{(x_1, x_2), x_1 \in [-5, 10], x_2 \in [0, 15]\}. \quad (3.18)$$

The three globally optimal solutions are at  $\mathbf{x}_1^* = [3.1416, 2.2750]$ ,  $\mathbf{x}_2^* = [9.4248, 2.4750]$  and  $\mathbf{x}_3^* = [-3.1416, 12.2750]$  and all have the identical optimum  $f = -0.3979$ .

### 3.2.2.4 Performance metric and results

The performance metric used here is the absolute minimum gap between the (known) global optimum of the test function and the best sampled function values at the end of each iteration. This metric is selected among others, such as the minimum Euclidean distance from the optimal solutions, because all we care about in practice is whether the search algorithm will lead us to a design with high preference. Therefore, a design with near-optimal preference but away from the optimal solution in the design space shall still be considered as close to the optimum and the corresponding search as successful.

Due to the stochastic nature of the search algorithms, we run ten tests on each combination of algorithms and test functions. Twenty iterations are conducted in each test and within each iteration three designs are queried. Means and standard deviations of the performances are reported in Table 3.1. The results suggest that the proposed SVM search algorithm has the best overall performance.

Table 3.1: Means and standard deviations of the final error from SVM Search, GA and Random Search  
(Lower values are better)

Mean(Std)	Rosenbrock	Camelback	Branin
SVM Search	<b>4.12</b> (6.77)	<b>0.19</b> ( <b>0.33</b> )	<b>1.40</b> ( <b>2.53</b> )
GA	4.31 ( <b>3.76</b> )	1.02 (0.83)	31.0 (7.45)
Random	19.8 (13.5)	18.3 (18.4)	23.0 (21.7)

### 3.3 EGO Search

The conceptual drawback of the SVM Search algorithm is the usage of the intersection of spaces, which has no physical meaning. In fact, it is also numerically difficult to find points within an intersection if the intersected space is small or has irregular shape, which is often the case when SVM is used. By observing these flaws in practice, we move on to explore alternative search algorithms.

#### 3.3.1 The algorithm

It turns out that such a search algorithm that deals with binary subject choice feedback can be built in the same fashion as EGO: In the first iteration, the computer presents a random design set (based on a Latin Hypercube design). The user then evaluates these designs and chooses the relatively preferred ones. The computer takes in these binary choices as labels on the sample set. SVM is run to find the optimal decision function. A merit function combining this decision function and the mean square error function is then optimized, and its solutions along with the preferred designs from the previous round are used as the samples for the next iteration. The algorithm continues until the design set becomes identical to the user selection or the maximum number of iteration is reached. Algorithm 2 provides a summary of the steps for EGO Search, Figure 3.5.



---

**Algorithm 2:** EGO Search Algorithm

---

```
input :  $\mathcal{D}$ ,  $s$ ,  $q$ 
output:  $\mathbf{x}^*$ 

Initialization;
// initialize the merit function
 $f_{\text{merit}} = 0$ ;
// initialize the number of preferred designs
 $q_1 = q$ ;
// initialize the preferred design set
 $\mathbf{X}_p = \emptyset$ ;
for  $i = 1$  to  $s$  do
    // scatter  $q_i$  samples in  $\mathcal{D}$  using the merit function  $f_{\text{merit}}$ 
     $\mathbf{X}_i \triangleq \{\mathbf{x}_j\}_{j=1}^{q_i} = \text{Scatter}(f_{\text{merit}}, \mathcal{D}_c, q_i)$ ;
    // label samples
     $\{y_j\}_{j=1}^q = \text{Label}([\mathbf{X}_i, \mathbf{X}_p])$ ;
    check termination criteria;
    if Satisfied ( $\{\mathbf{x}\}$ ) then
        | break;
    end
    // train with accumulated data
     $g_i = \text{SVM}(\{\mathbf{x}\}, \{y\}, \text{svm\_options})$ ;
    // update the merit function
     $f_{\text{merit}} = f_{\text{merit}}(g_i, \text{all samples})$ ;
    // set the preferred designs set
     $\mathbf{X}_p = \{\mathbf{x}_j \in \mathbf{X}_i, y_j == 1\}$ ;
    // set the number of preferred designs
     $q_{i+1} = q - |\mathbf{X}_p|$ ;
end
return the best design found as  $\mathbf{x}^*$ ;
```

---

Figure 3.5: Proposed EGO Search algorithm

Some details of this algorithm need to be clarified:

- The merit function we used is a weighted sum of the predicted utility  $\hat{f}$  and the mean square error  $\hat{\sigma}^2$ , i.e.,  $f_{\text{merit}}(\mathbf{x}) = -(w_1\hat{f} + w_2\hat{\sigma}^2)$ . Although physically not meaningful, this is the simplest form of a merit that satisfies the goal of balancing exploitation and exploration. The weight  $w_1$  is set to 1 and  $w_2$  decreases as  $s/i - 1$ , where  $s$  is the maximum iteration number and  $i$  is the current iteration number. These weight values are empirically set to make the two function values be of the same scale. A decreasing  $w_2$  is needed so that the search starts with an emphasis on exploration and then shifts to exploitation towards the end.
- When calculating  $\hat{\sigma}^2$ , the Gaussian spread parameter  $\lambda^{\text{kriging}}$  is set to 10; also recall that the calculation of  $\hat{\sigma}^2$  requires the estimation of the error variance  $\sigma_\varepsilon$  which involves the observed function values  $\mathbf{f}$  at all sampled designs. Here since we only have binary  $ys$ , we substitute  $\mathbf{f}$  with the decision function values provided by the SVM model.
- The weight  $C$  in the soft-margin SVM is set to a large value  $10^6$  since we consider all user input to be correct and little training error is allowed; in real user tests, this value shall be lowered to incorporate the stochastic error  $\varepsilon$  in user preference and user choice mistakes.
- As we mentioned in the introduction of EGO, both  $\hat{f}$  and  $\hat{\sigma}^2$  are non-convex functions of  $\mathbf{x}$  and thus optimizing the merit function can be costly. Also, a locally optimal solution shall be avoided since such a query will lead to low quality estimations of the objective function and further undesired queries. Therefore, an inevitable trade-off exists between finding the global solution of the merit function and reducing the computational cost of the algorithm. Here we focus mainly on the quality of queries and thus we use the Matlab GA toolbox (*Chip-*

*perfield and Fleming (1995)*) to optimize the merit function. The maximum generation size is set at 1000 and the maximum population size at 100.

### 3.3.2 EGO Search versus SVM Search in simulated tests

Figure 3.6 shows results on test functions using EGO Search and SVM Search following the same test setup from Section 3.2. In order to compare the performances of these algorithms on high dimensions, we add to the test function set a multivariate Gaussian function as defined in Equation (3.19).

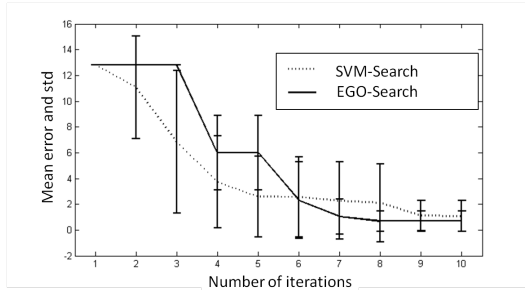
$$f(\mathbf{x}) = \exp(-\lambda^{\text{user}} \sum_{i=1}^p (x_i - 0.9)^2), \quad x_i \in [-1, 1], \quad \forall i = 1, \dots, p. \quad (3.19)$$

The optimal solution of this function is set near a corner of the design space since this is one of the most difficult cases for the search algorithm, considering the fact that there are  $2^p$  corners in total. For each algorithm and each function, ten tests are executed and the error is calculated as the current minimum gap from the known optimum. The solid line and bars show the mean error and its standard deviation for the EGO Search respectively, while the dotted ones show those for the SVM Search algorithm. We can see from this comparison that EGO Search has more reliable overall performance than SVM Search especially in higher dimensions.

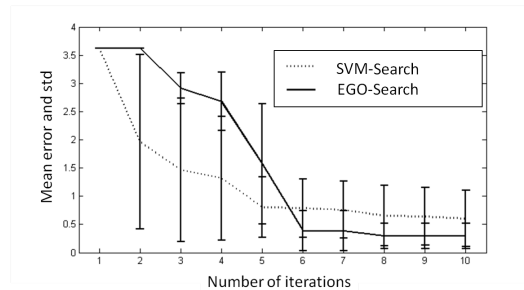
## 3.4 Discussion on the EGO Search Algorithm

The pilot test results from the previous subsection leads to the following observations:

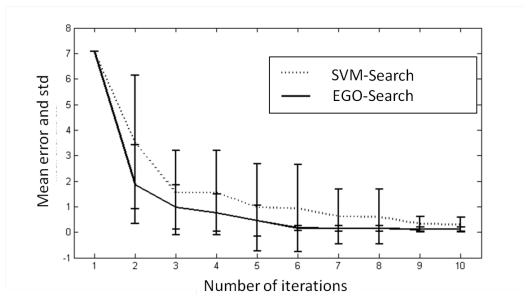
- The EGO algorithm relies on properly selecting the merit function and its parameters. How will the forms of the merit and their parameters affect the performance of the algorithm in actual user interactions? Is there a set of “correct” parameters?



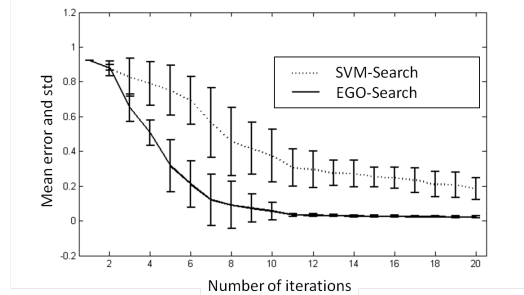
(a) Rosenbrock (2 dimensions)



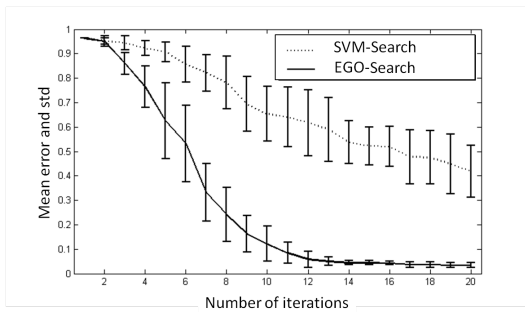
(b) Six-Hump Camelback (2 dimensions)



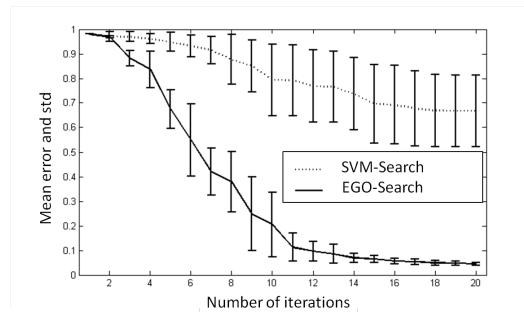
(c) Branin (2 dimensions)



(d) Gaussian (10 dimensions),  $\lambda_{user} = 0.5$



(e) Gaussian (13 dimensions),  $\lambda_{user} = 0.5$



(f) Gaussian (15 dimensions),  $\lambda_{user} = 0.5$

Figure 3.6: Comparison between EGO Search and SVM Search.

- How many dimensions can the algorithm handle effectively? This question is essential since its answer determines the maximum number of variables we can use in a user-computer interaction.
- Besides the dimensionality issue, it is also important to understand how user preference sensitivity will affect the search. To explain, some users may have strong preferences only in a small region of the design space and will not prefer designs elsewhere. Some other users, on the contrary, may have broader range of preferred designs. We will investigate the search efficiency on simplified preferences using the multivariate Gaussian model from Equation (3.19), where the preference sensitivity is controlled by the Gaussian spread  $\lambda^{\text{user}}$ .

### 3.4.1 Parameters used in the merit function

We investigate the influence of the form of the merit function on the search result. Two types of merit function are of interest. One form of merit function, as we used in the previous section, is a weighted sum of the utility and the mean square error:  $w_1\hat{f} + w_2\hat{\sigma}^2$ . The other form, as we introduced in the review of EGO, evaluates the expected improvement of a new query. In a maximization context, given  $f_{\max}$  as the best design sampled so far, this expected improvement has the following form:

$$E[\max(F - f_{\max}, 0)] = (\hat{f} - f_{\max})\Phi\left(\frac{\hat{f} - f_{\max}}{\hat{\sigma}}\right) + \hat{\sigma}\phi\left(\frac{\hat{f} - f_{\max}}{\hat{\sigma}}\right). \quad (3.20)$$

Notice that  $\hat{\sigma}$  is a function of  $\mathbf{x}$  through the correlation  $\mathbf{R}$  and  $\mathbf{r}$  which are parameterized by  $\lambda^{\text{kriging}}$ .

We now investigate the influence of parameters of these two merit functions. For the weighted sum function, we test a set of weights:  $w_1 = 1, w_2 = (s/i)^r - 1, r = 1, 2, 3$ , where  $s$  is the maximum number of iterations and  $i$  is the current iteration number. The weights are set this way to examine how much exploration is needed during

Table 3.2: Impact of the weights of the weighted-sum merit function  
(Lower values are better)

error $\mu(\sigma)$	$k = 1$	2	3
Branin	<b>0.140 (0.147)</b>	0.233 (0.333)	0.427 (0.468)
Camel	<b>0.578 (0.298)</b>	0.588 (0.329)	0.584 (0.424)
Hartman3	0.167 (0.053)	0.173 (0.038)	<b>0.149 (0.035)</b>
Rosenbrock	2.582 (1.314)	<b>1.175 (1.282)</b>	3.777 (2.712)

the search and diminish the exploration emphasis towards the end of the process. For the expected improvement merit function, we are interested in the Gaussian spread  $\lambda^{\text{kriging}}$  used in evaluating the mean-squared error function. We test a set of  $\lambda^{\text{kriging}} = 10/i, 100/i, 1000/i, 10000/i$  to see what scale of  $\lambda^{\text{kriging}}$  should be used. Also, we notice that the correlation matrix  $\mathbf{R}$  may become ill-conditioned when the spread  $\lambda$  is small. This is because all values off the diagonal in  $\mathbf{R}$  will be closer to 1 with a small  $\lambda^{\text{kriging}}$  while the diagonal elements are always 1's. Thus the minimum  $\lambda^{\text{kriging}}$  during the process is empirically set to 1 (when the maximum iteration number is set to 10). Again,  $\lambda^{\text{kriging}}$  decreases along the iteration to shift the priority from exploration to exploitation.

Tables 3.2 and 3.3 show simulation results from various test functions using the proposed merit functions. We run ten tests for each combination of function and spread; each test contains ten iterations; within each iteration eight samples are labeled. The reported numbers are the mean errors of the searches after the last iteration and their standard deviations. From the results it can be concluded that although some parameter values achieve better and more stable performance over others, there is no universally recommended parameter values for the merit functions at this point. Also, the sensitivity of the resulting error with regard to the parameters is different for different test functions.

Table 3.3: Impact of the spread of the expected improvement merit function (Lower values are better)

error $\mu(\sigma)$	$\lambda^{kriging} = 10/i$	$100/i$	$1000/i$
Branin	0.300 (0.122)	0.126 (0.102)	<b>0.051 (0.039)</b>
Camel	1.057 (0.554)	<b>0.559 (0.356)</b>	1.356 (0.704)
Hartman3	<b>0.208 (0.065)</b>	0.232 ( <b>0.038</b> )	0.226 (0.067)
Rosenbrock	5.844 (5.686)	<b>0.642 (0.135)</b>	1.074 (0.771)

### 3.4.2 Dimensionality and user sensitivity

It is intuitive that when a design has many dimensions, the search will become less effective. This is also the case if the utility function of a user takes high values only within a small region. We are interested to see how the EGO Search algorithm works for different dimensions and utilities. The test uses Gaussian functions with 10, 15 and 20 dimensions as the utilities with a set of spreads  $\lambda^{\text{user}} \in [0.5, 1, 2, 4]$ . We run ten tests for each combination of dimensionality and spread; each test contains twenty iterations; within each iteration eight samples are labelled. The merit function used here is  $\hat{f} + (s/i - 1)\hat{\sigma}^2$ , where  $s$  is the maximum iteration number and  $i$  is the current iteration number. Table 3.4 shows the mean error that the algorithm reaches at the 20th iteration and its standard deviation.

Although simulations are not direct reflection of how the algorithm will perform in real situations, there are a few observations we can derive from these results: (1) Noticing that the Gaussian function ranges from 0 to 1, the algorithm achieves an error less than 10% only when the spread is large (low sensitivity of user preference) or the dimensionality is low; the algorithm has very limited capability when dimensionality increases beyond twenty; (2) the algorithm has fairly consistent performance since the standard deviation of the error at the 20th iteration is low.

Table 3.4: Impact of dimensionality and spread of the utility  
(Lower values are better)

error $\mu(\sigma)$	dim = 10	15	20
$\lambda^{\text{user}} = 0.5$	0.026 (0.005)	0.041 (0.007)	0.066 (0.018)
1	0.046 (0.016)	0.074 (0.010)	0.221 (0.139)
2	0.068 (0.026)	0.149 (0.017)	0.305 (0.094)
4	0.147 (0.039)	0.265 (0.036)	0.481 (0.125)

### 3.4.3 Computational cost

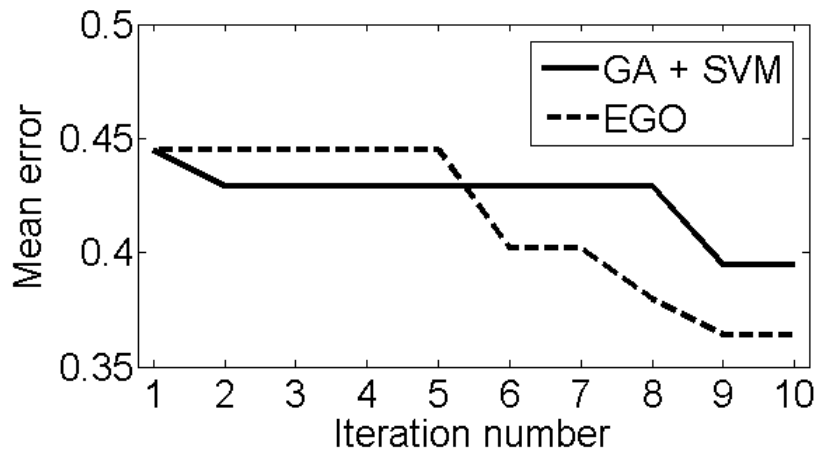
For the proposed algorithm to work effectively, a solution close to the global optimum must be found in each optimization of the merit function. Since the merit function is non-convex, finding such a solution is difficult. Although the Matlab GA toolbox provides a satisfactory solution for the tested problem dimensions, the computational cost of such an implementation is less than acceptable for real human-computer interactions. This problem will be discussed in detail in Section 4.3.

### 3.4.4 EGO Search versus GA plus SVM

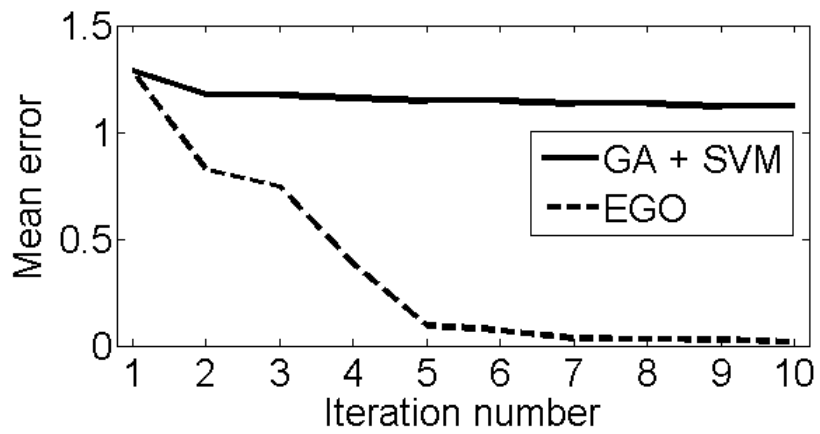
Recall that one of the reasons that we abandoned GA for the purpose of preference elicitation is the absence of fitness in binary choice data. However, with SVM transforming binary choices into a decision function, this seems no longer a barrier, and naturally one may wonder whether GA plus SVM will be competitive against EGO. The answer to this is negative, as can be confirmed by the simulated test results shown in Figure 3.7 for comparisons between GA plus SVM and EGO with the same SVM setup. The way we transform the decision function into fitness will be described in the following paragraph.

The reason behind this result is that the decision function generated from SVM is not suitable to represent the fitness of designs. Recall that all support vectors will have decision function value either 1 or -1, while other non-support vectors could take values much greater than 1 or much smaller than -1. Therefore using the decision

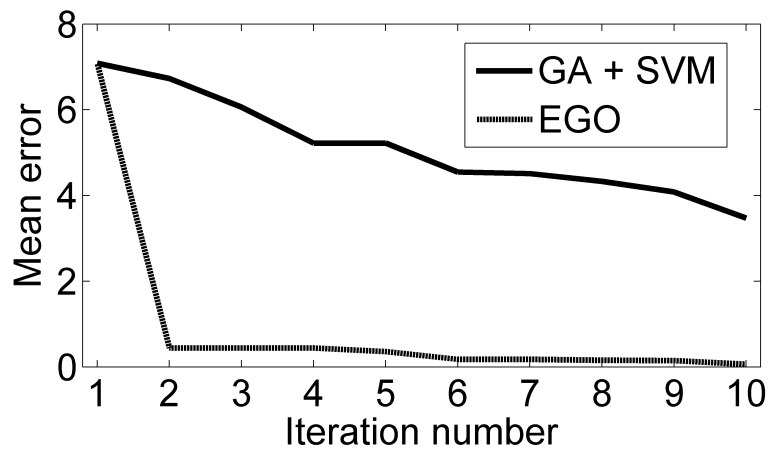




(a) Rosenbrock (2 dimensions)



(b) Six-Hump Camelback (2 dimensions)



(c) Branin (2 dimensions)

Figure 3.7: Comparison between EGO Search and GA plus SVM.

function values directly as fitness will be problematic and the resulting parents will only be replicates of designs with high decision function values. On the other side, if we use the ranking of the decision function values, which is what we did in the above tests, the parent selection scheme will still not work since the support vectors with 1s and -1s will now have similar fitness, making the algorithm very likely to choose those not-preferred designs as parents. A third attempt, which is to choose only parents from the preferred designs also has its own problem: With very limited number of designs, only a few will be non-support vectors, leaving the fitness values of the selection pool all 1s. This is especially the case when the dimensionality of the design space is high. As a conclusion, the idea of using GA for the purpose of preference elicitation is not effective because of the difficulty in getting fitness information, in validating its crossover and mutation operations, and thereafter in converging to a solution within a small number of queries.

### **3.5 Vehicle Exterior Styling Design Elicitation**

Due to its observed performance in simulation, EGO Search is chosen for implementation in an actual user study involving vehicle exterior styling design. The purpose of the study is to investigate whether the algorithm can successfully help the subjects to search for a design that is close to a given target.

#### **3.5.1 Software development**

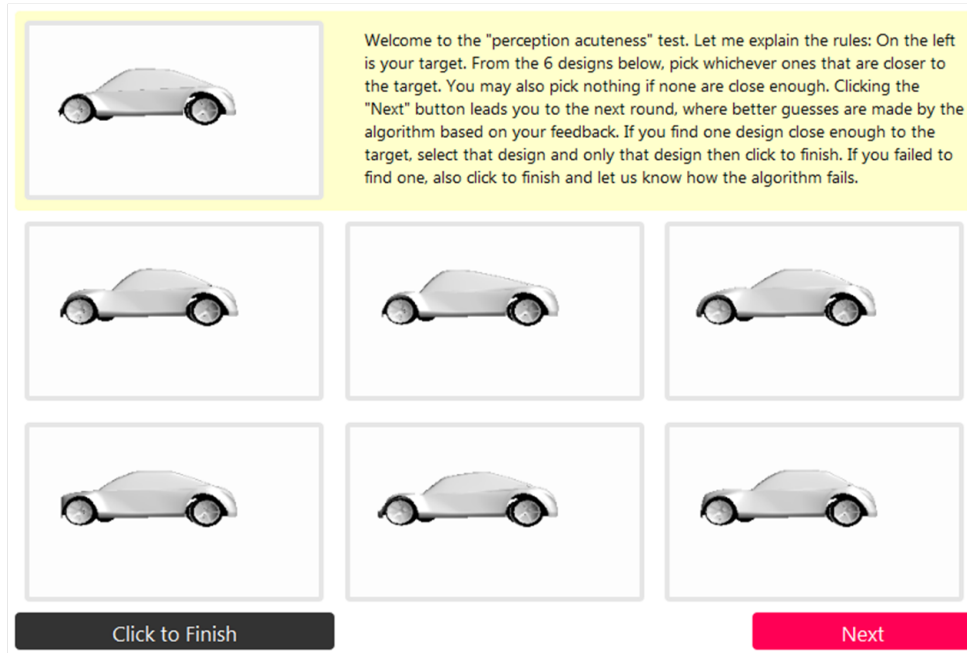
To enable real-time online human-computer interactions on designing vehicle styles, a parametric three-dimensional (3D) model incorporating 20 variables within the range of  $[0, 1]$  is programmed in WebGL, which is a context of the canvas HTML element that provides a 3D graphics API implemented in a web browser without the use of plug-ins. A Java servlet implementing the proposed search algorithm runs on the server side to parse user feedback and provide queries for the next iteration.

At the time of this writing, readers may access “<http://yirenumich.appspot.com>” for the EGO Search implementation. A latest browser, e.g., Mozilla Firefox or Google Chrome, is required to run these applications.

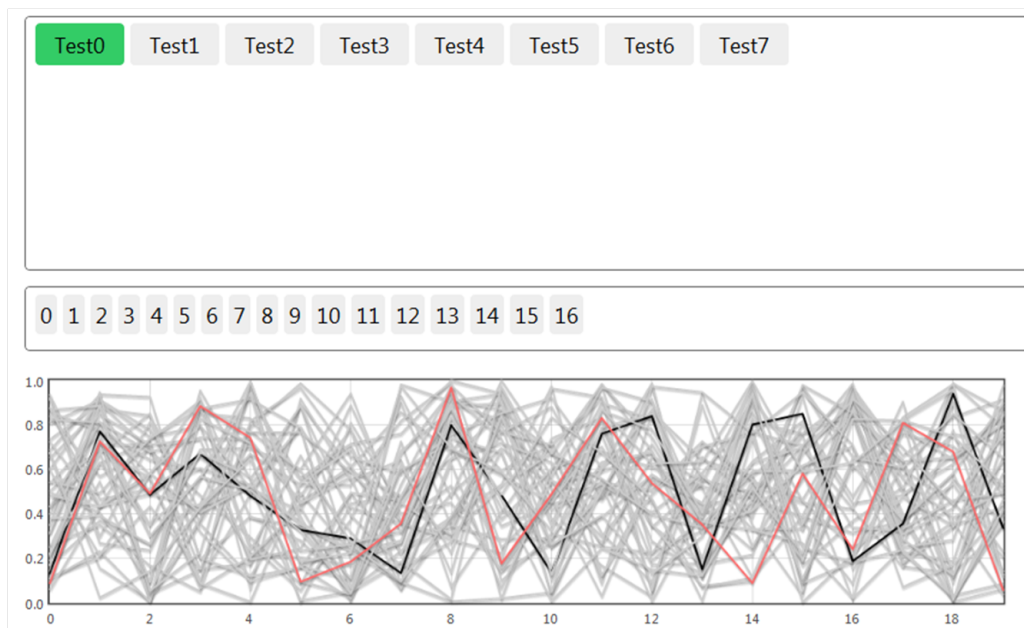
### 3.5.2 Convergence test setup

The test pages are set up at “/convergencetest.html” on the website and all user data are submitted and can be retrieved at “/log.html”. In the test environment, a random and a fixed target design are shown at the top of the web page. Six initial random guesses are shown below the target, see Figure 3.8(a). The data retrieval page shows in the top division how many tests have been conducted, followed by the accumulated data from each iteration in that test. Each design sampled thus far is visualized in the bottom division where the preferred ones are highlighted. The red highlighted curve represents the target in that test. Figure 3.8(b) contains the data visualization. The parameters set up in the implementation are:  $\lambda^{\text{SVM}} = \lambda^{\text{kriging}} = 0.05$  and  $[w_1, w_2] = [1, 1000]$ . These parameters are set empirically and their value should be further studied. Since Google Appengine limits the request time at 30 seconds, we set the population limit to 10 and the number of generations to 500 in the GA implementation for the global search inside the EGO Search iterations. This is a fairly low-cost setup for a 20-dimensional space.

This online environment was developed for mass data collection. As of this writing, more than six hundred anonymous interactions have been recorded. However, looking at the collected data, we surmised that the anonymous users do not necessarily follow the test instructions and thus, while interesting, the collected results cannot be analyzed with confidence. Consequently, we conducted a pilot test in an interview setting with eight subjects from the University of Michigan. Prior to initiating the test, the subjects were instructed on how they can view, rotate, zoom, pan, synchronize and reset the viewpoint of each design rendering. They were also made clearly



(a) Test environment



(b) Test data visualization

Figure 3.8: Online human-computer interaction interface and data visualization. (a): The interactive environment at `/convergencectest.html` allows the user to zoom, pan, rotate each design and updates the guesses once the user hits the “Next” button; (b): The data visualization window at `/log.html` has all user tests listed at the top, number of iterations in the middle, and the cumulated data at the bottom. The red curve represents the target design, the highlighted dark curve(s) represent the preferred design at this point and the rest all not preferred.

aware of their task which was to pick a set of designs that were relatively closer to the given target in each iteration. Moreover, the subjects understood that they could pick up to five designs as “preferred” and that they had the option to pick nothing if none was relatively close to the target. After the subjects submitted their final design, we asked them to state whether they were “satisfied” or “not satisfied” with the test result. This information was also sent back to the server.

### 3.5.3 User data analysis

While all users were satisfied with the test result according to their responses, we need to examine whether the interactions converged. We start by measuring the Euclidean distance between each sampled design and the target in each test, as illustrated in Figure 3.9, since no explicit objective function is available and the gap between the optimum and the converged objective value at each sample cannot be measured. In the figure, samples towards the end of each test curve are those generated in later iterations. The plot shows that later samples do not necessarily acquire lower distance values, and thus convergence is not observed.

Nonetheless, visual comparison between targets and user interaction results, as shown in Figure 3.10, indicates that users did submit designs close to their targets. One possible reason for this contradiction is that, when people compare two different shapes, the difference is not measured in the design (variable) space but in some feature space of the shape, i.e., the perceived difference of a pair can be small when its distance in the design space is large. We can try to verify this reasoning by examining one user’s test data (data “Test0” on the website). Figure 3.11 compares the target design and samples in the last iteration (iteration 17) which hint that the algorithm understands that the user prefers a roof curve design that has a steep front wind shield and an extended hatchback. In fact, although distance-wise most of these designs are off target, the first five have roof fashions visually close to that

of the target while the last one still meets the description but has some variation of its own.

One then surmises that the user focuses mainly on matching the roof style, and the algorithm indeed understands what the user is searching for. In Figure 3.12, we show what the user selects in the first four iterations, and from these selections we see a consistent user pursuit and convergence of the search. In the same figure we also show that the design with minimum Euclidean distance does not acquire the roof style the user is looking for and is in fact not picked in the iteration where it appears.

From the above observations, it appears that convergence cannot be shown by the Euclidean distance to the target. Since the user focuses mainly on the roof style design, we formulate a measure that differentiates the roof style as follows. We define style features as seven lengths between key control points of the roof silhouette. The lengths are denoted in Figure 3.13 and the measure of difference is defined as the Euclidean distance in the feature space spanned by the seven lengths. With this new measure definition, we examine our earlier conclusion quantitatively. Figure 3.14(a) and 3.14(b) show scatter plots of the sampled designs in a reduced space corresponding to the feature measure and the Euclidean measure accordingly. Each circle represents a sampled design and the square represents the target. The inner radius of a circle indicates when it first comes into existence while the outer radius shows when it is abandoned, in both cases the smaller the radius the earlier the event occurs.

Two observations are made: Firstly, most of the thick circles, representing repeatedly selected designs, are close to the target. This verifies the assumption that the user mainly focuses on matching the roof design; secondly, the feature map has most of the larger circles around the target while smaller ones are away, while in the Euclidean map some of the larger circles are scattered away from the target. The implication is that while no knowledge about the user is posed on the algorithm *a priori*, it “learns” what the user is looking for and generates designs close to the target

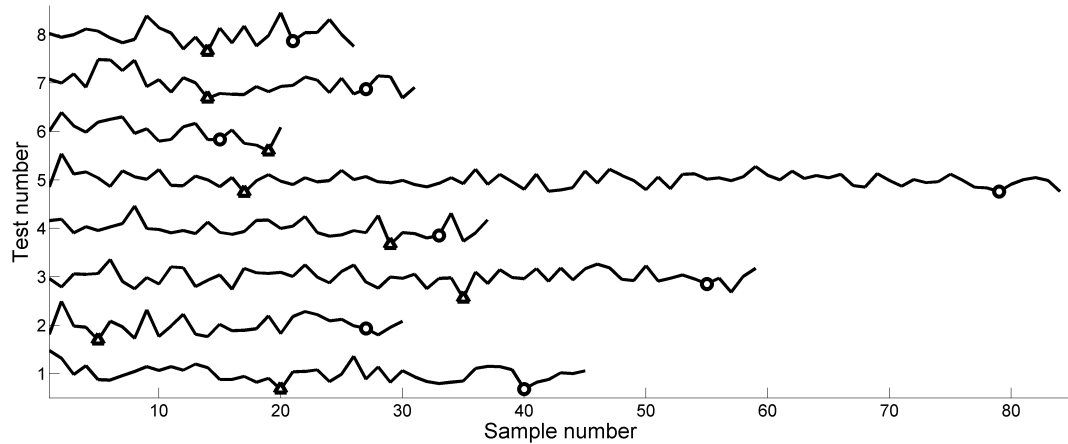


Figure 3.9: Normalized Euclidean distance from each sampled design to the target in each test. The circled design is the one submitted by the user, while the triangle design has the lowest Euclidean distance to the target.

in the unknown feature space, rather than in the explicit design space.

It should be noted that there are two factors that may affect convergence. Recall that in EGO Search we have a balance between exploration and exploitation. Every new sample is forced to be away from existing ones, and therefore it may not be close to the target in the feature or the Euclidean space. In fact, in the highlighted test data shown above, the final submitted design (the one with Euclidean distance to the target of 1.55, as shown in Figure 3.11), appears as early as the fourth iteration but nothing better is found afterwards. One can verify this from Figure 3.14(a) where the final submission is shown by the thickest circle close to the target. Therefore, while a balance clearly should be made, how the weights shall be tuned needs more study. Further, the GA implementation in each EGO Search iteration uses a fairly low number of generations with a small population. Therefore, the sampled point may not be the optimum of the merit function and this can lead to an ineffective search.

































User	Side view		Perspective view	
	Result	Target	Result	Target
1				
2				
3				
4				
5				
6				
7				
8				

Figure 3.10: Visual comparison between user test results and the targets from side and perspective views.



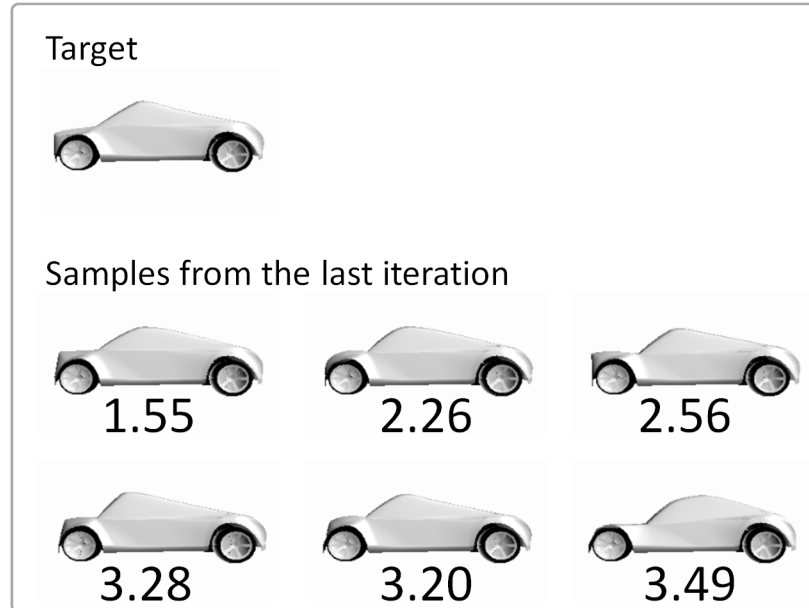


Figure 3.11: Visual comparison between samples in the last iteration and the target. Data generated from Test0 on “log.html”. Euclidean distances to the target are listed under the designs.

### 3.5.4 Observed issues in user interactions

There are a few issues we observed from the users in the pilot test. First, even with a target given, the user can still exhibit inconsistent preference during the interaction. For example, some of the users do not realize that they should rotate the designs when evaluating them, though they have been told they can do it. Once they realize in the middle of the interaction process that they can rotate the image, their perception of the shapes changes, and so does their measure of difference between shapes. Second, some users appeared to be rather insensitive to shape changes and would miss good designs (following their own measure) during the interaction, while the interaction does not allow them to make changes to already submitted choices. Both of these “human mistakes” can confuse the algorithm and make it less effective. In addition, a practical issue of collecting online interaction data from anonymous users is that it is difficult to discriminate well-thought interactions against randomly-submitted ones.

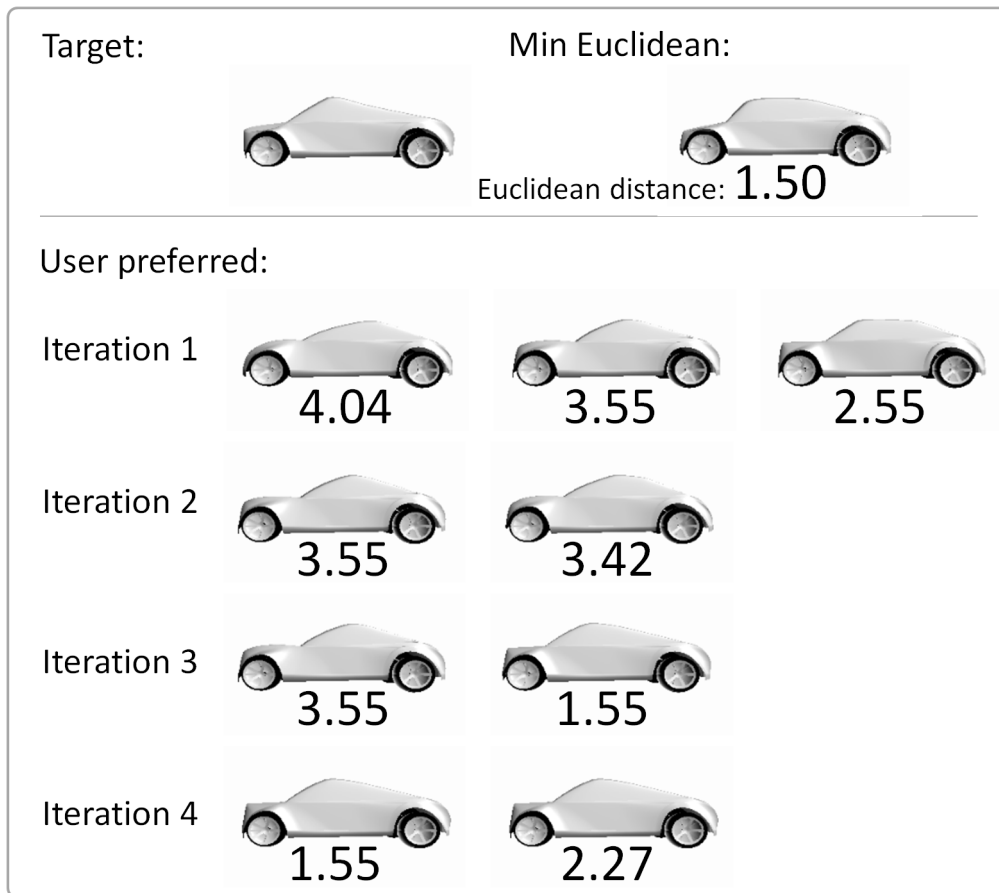


Figure 3.12: Designs labeled as preferred in the first four iterations, compared with the target and the one with the minimum Euclidean distance within all samples. Euclidean distances to the target are listed under the designs.

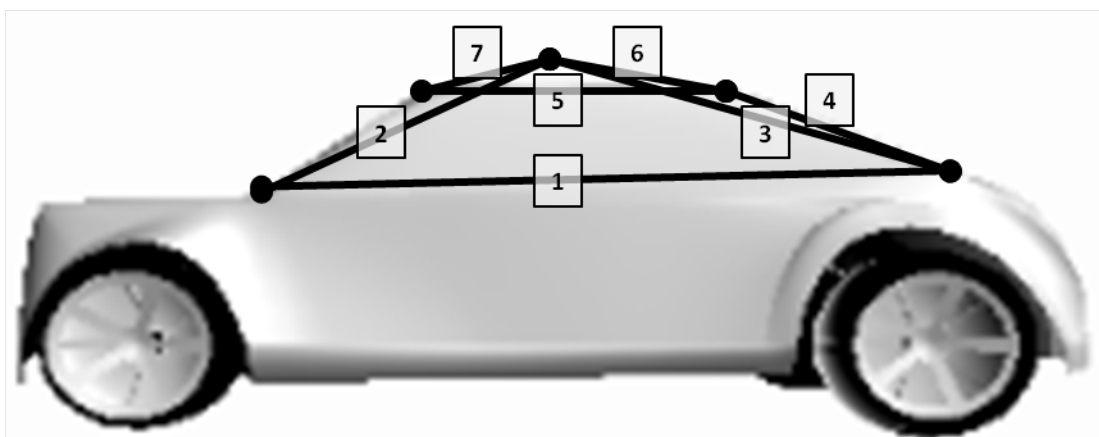
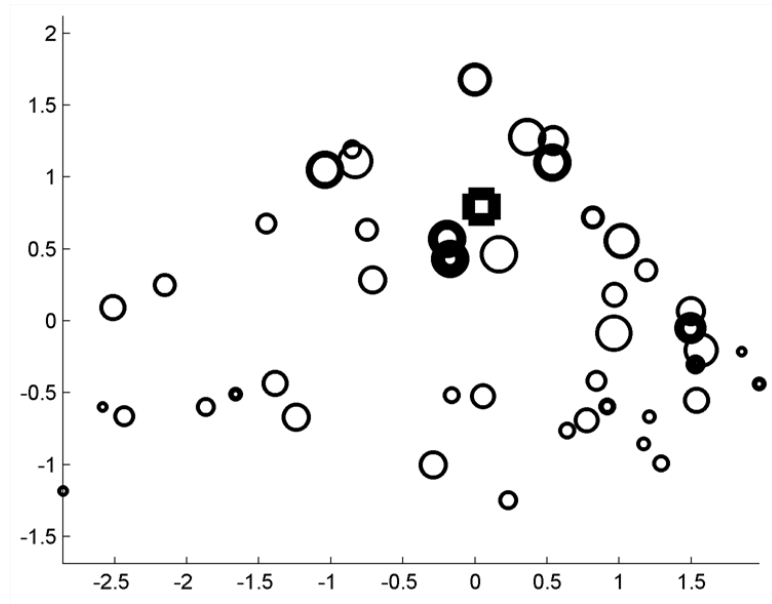
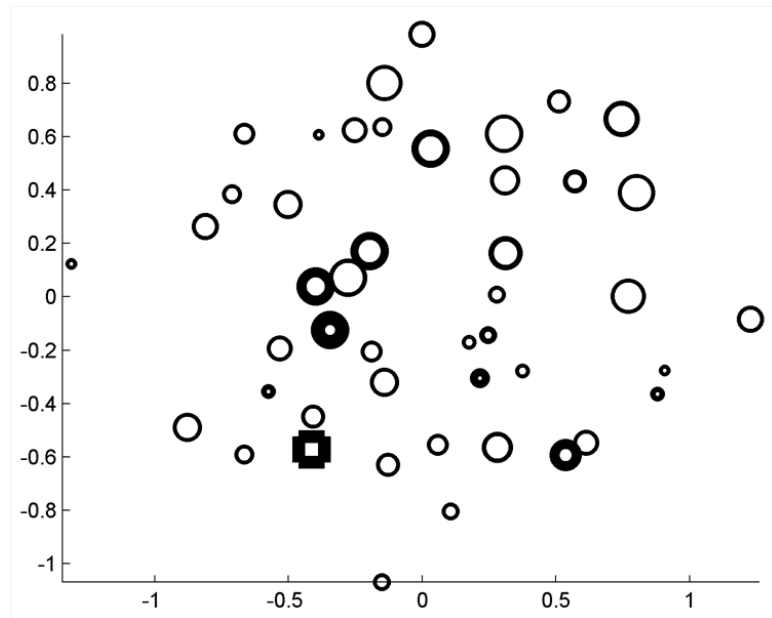


Figure 3.13: Features that capture the roof design.



(a) Feature map



(b) Euclidean map

Figure 3.14: Samples from data “Test0” in the feature and Euclidean space. Multidimensional scaling is applied to both measures to create 2D visualization of the data.

### 3.6 Concluding Remarks

This chapter investigated search algorithms behind an iterative human-computer interaction to fulfil the purpose of preference elicitation. The SVM Search reduces the design space and forces convergence. The EGO Search combines exploitation of the observed comparisons and exploration of the unsampled space. The simulated test results showed that both proposed methods are better than GA when only binary feedback is available and that EGO Search outperforms SVM Search when the dimensionality of the design space is high. The EGO Search algorithm was implemented for a 3D vehicle exterior design application and the pilot real subject tests showed that subjects can successfully locate designs close to their targets.

There are several possibilities for improving and expanding this work. Proper adaptive parameter setting within the merit function in EGO Search is desirable and requires further testing. The computational cost of global optimization of the merit function needs to be reduced further before the proposed algorithm can be used effectively for extensive user interactions. It is also interesting to investigate whether more accurate decision functions can be estimated by using more than two classes. In fact, designs that are switched from preferred to not-preferred should be treated differently from designs that are not preferred from the beginning. The rotation matrices recorded during the interaction can also provide insights on what features the user is sensitive to when she evaluates the difference between a target and the samples. Such insights should be incorporated into the learning process to create more accurate merit functions. We continue exploration in these aspects in Chapter [IV](#).

Another important direction of future investigation is how to utilize preferences collected from different individuals. Investigation in Chapter [VI](#) shows that the search can be made more efficient if the algorithm incorporates history from previous users into a current session.

Finally, the data collected from an interactive session can be treated as revealed choices from a simulated market, and so combining search process records from many users may provide insights and possibly lead to preference models for groups of users or market segments.

## CHAPTER IV

# Augmented Preference Elicitation

The EGO Search algorithm developed in the previous Chapter has three elements: 1) the decision function (prediction)  $\hat{f}$ , 2) the mean-squared error function  $\hat{\sigma}$  and 3) the merit function that combines the first two. We devote this chapter to investigating how the EGO Search algorithm can be enhanced. Two areas of improvement are documented:

- The first improvement in Section 4.1 is modeling a preference function using pairwise relationship of sampled designs rather than treating the entire data as binary labelled. We will show that this modification in  $\hat{f}$  results in better search performance especially for problems with high dimensions.
- The second improvement is about reducing the computational cost of optimizing the merit function. Variations of the merit function are reviewed in Section 4.2. We show that these functions have multiple local optima around the global one, and this property causes the optimization on them to be slow. We then propose new merit function in Section 4.3 and show empirically that this smoother function can significant reduce the computational cost of an EGO Search in high-dimensional spaces.

## 4.1 Preference Modeling with Comparison Tree

In previous studies, we treated the accumulated user feedback as binary labelled data, i.e., a design may either be “preferred” or “not-preferred”. This is a rather simplified interpretation of the subject response since a “once-preferred” query is not differentiated from those that are “not-preferred on sight”. As we will show later in the section, treating subject responses as binary labelled data will cause the decision function  $\hat{f}$  to be inconsistent with the response, i.e., design A is considered better than design B but  $\hat{f}(\mathbf{x}_A)$  may be less than  $\hat{f}(\mathbf{x}_B)$ . The easy fix we propose here is to create a complete comparison tree and formulate a learning problem that incorporates this tree as constraints rather than treating data as binary. We elaborate below.

### 4.1.1 Terminology and definition

Let  $n$  and  $p$  be the number of sampled designs and their dimensionality. Denote  $\mathbf{x}_i : p \times 1$  as the  $i$ th sample, and  $\mathbf{X} : n \times p$  as the total sample set so far. Let  $\mathbf{G} : n \times n$  be the unidirectional graph representing the pairwise comparison relationship between samples, where  $\mathbf{G}_{ij} = 1$  if  $\hat{f}(\mathbf{x}_i) > \hat{f}(\mathbf{x}_j)$ . Introduce a decision function

$$\hat{f}(\mathbf{x}) = \mathbf{w}^T \mathbf{v}(\mathbf{x}), \quad (4.1)$$

where  $\mathbf{w} : n \times 1$  is the parameter vector and  $v_i(\mathbf{x})$  is defined as the correlation between  $\mathbf{x}$  and the  $i$ th sampled design  $\mathbf{x}_i$ :

$$v_i(\mathbf{x}) = \exp(-\lambda \|\mathbf{x}_i - \mathbf{x}\|_2^2). \quad (4.2)$$

The parameter  $\lambda$  is the spread of the Gaussian kernel and is set to  $1/p$  as suggested in [Chang and Lin \(2011\)](#).

### 4.1.2 Learning based on pairwise comparison

Infinite number of decision functions exist that are consistent with  $\mathbf{G}$ . Similar to the philosophy of SVM, here we form an optimization problem by finding a decision function with the least complexity. In other words, the optimal decision function shall have the parameter vector  $\mathbf{w}^*$  with the least possible norm. Therefore, the optimal parameters can be found by solving the following quadratic problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{subject to} \quad & \mathbf{w}^T (\mathbf{v}(\mathbf{x}_j) - \mathbf{v}(\mathbf{x}_i)) \leq a, \quad \forall i, j \text{ such that } \mathbf{G}_{ij} = 1, \end{aligned} \quad (4.3)$$

where  $a \leq 0$  is a user-defined parameter that controls the steepness of the decision function. In the special case when  $a = 0$ , we have  $\mathbf{w}^* = \mathbf{0}$  and thus the search becomes a farthest-first search, i.e., the search depends only on the spatial arrangement of  $\mathbf{X}$ . The parameter  $a$  plays a role of scaling the norm of feasible  $\mathbf{w}^*$  and is empirically set to  $-1$  in this study.

### 4.1.3 Simulated test results and discussion

Below we test three different forms of decision functions: In the first case, we treat subject responses as binary labelled data and train the data with SVM with  $C = 10^6$  (indicating hard-margin). We call this method “binary classes”. In the second case, we train the data using a comparison tree and solve the problem in Equation (4.3). We call this method “comparison tree”. In the last case, we use the binary labels for the first half of the search and switch to using the comparison tree in the second half. This method is denoted as “hybrid”.

The test functions we used are 2D Camel, 2D Branin and multidimensional Gaussian with 10, 20, 31 dimensions, as defined in Equations (3.15), (3.17) and (3.19), respectively. To understand how these algorithm setups will work in a difficult situa-



tion, the spread in the Gaussian tests is set at  $\lambda^{\text{user}} = 5$ , creating a function that has only non-zero values close to the unique optimal solution. All tests are repeated 10 times due to the stochastic nature of the global search inside each iteration. Mean errors from the corresponding known optimal objective values are reported for each test as shown in Figures 4.1 to 4.5.

Results show that the comparison tree method is not appealing in a short run and it in fact slows down the convergence speed for the first few iterations. However, when the dimensionality of the search space is high, this method will eventually outperform the binary classes method in a long run. Recalling that a good search strategy balances the exploration and exploitation, we can argue that the observation is resulted due to the fact that while the binary classes method keeps the decision function fuzzy enough to enhance exploration, the comparison tree method exploits the limited observation and may lead to an inefficient search in early stages. The hybrid approach is fact inspired by this observation, and indeed has superior performances across all test functions.

In addition to the above qualitative explanation of the simulated test results, we present in Figure 4.6 the violation rates of the binary classes method in each test for readers to have a better sense of its “fuzzyness”. The violation rate at some iteration  $i$  is defined as follows:

$$\text{violation}_i = \frac{\# \text{ violations in constraints}_i}{\# \text{ constraints}_i}. \quad (4.4)$$

The violation rates data here show that (1) high violation rate at early stage of a search will not hamper the searching efficiency; this is because exploration has more weight than exploitation during that period and thus the accuracy of the decision function does not contribute too much to the effectiveness of the search; (2) high violation rate at late stage of the search will slow down the search and this is when

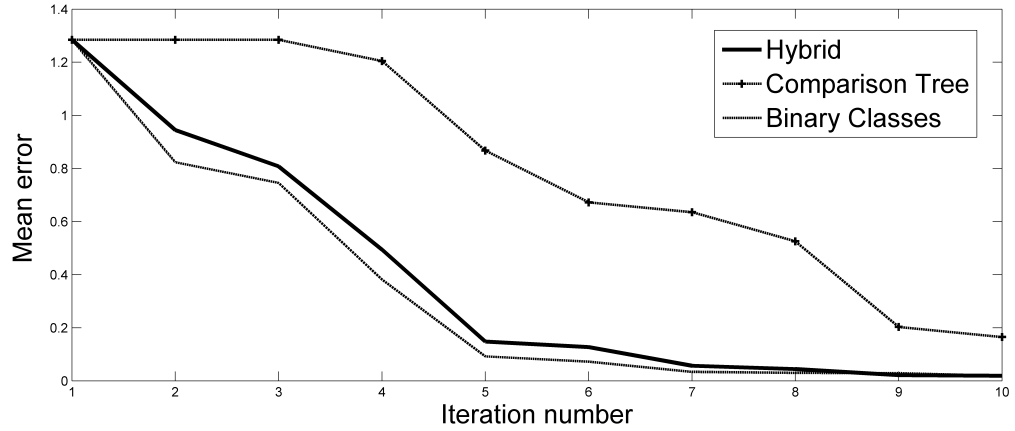


Figure 4.1: 2D Six-Hump Camelback

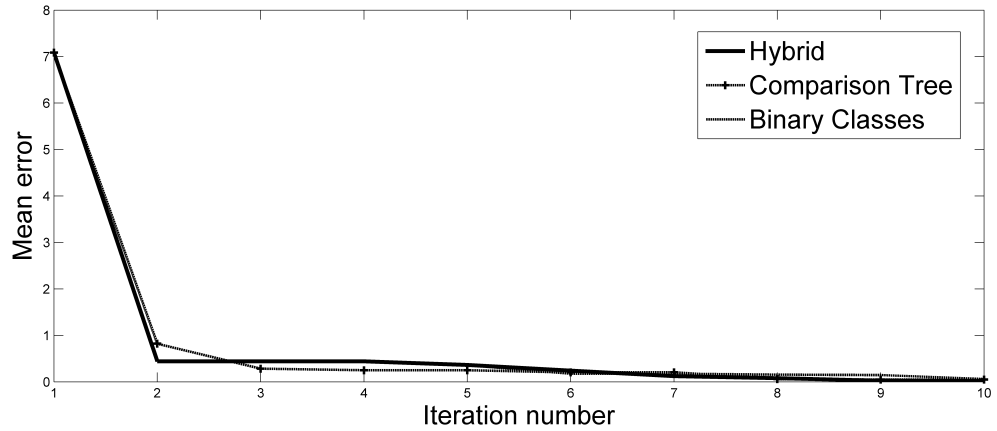


Figure 4.2: 2D Branin

comparison tree can be useful.

## 4.2 Variations of the Merit Function

In Chapter III we introduced two merit functions in the EGO algorithm to balance exploration and exploitation, namely, the linear weighted sum and the expected improvement. In this section we review variations of the merit function from the literature (*Sanena (2002)*; *Kushner (1964)*; *Cox and John (1992)*; *Watson and Barnes (1995)*; *Locatelli (1997)*; *Žilinskas (1981)*) and discuss how they can help to improve algorithmic performance on simulated tests. Although derived from different concepts

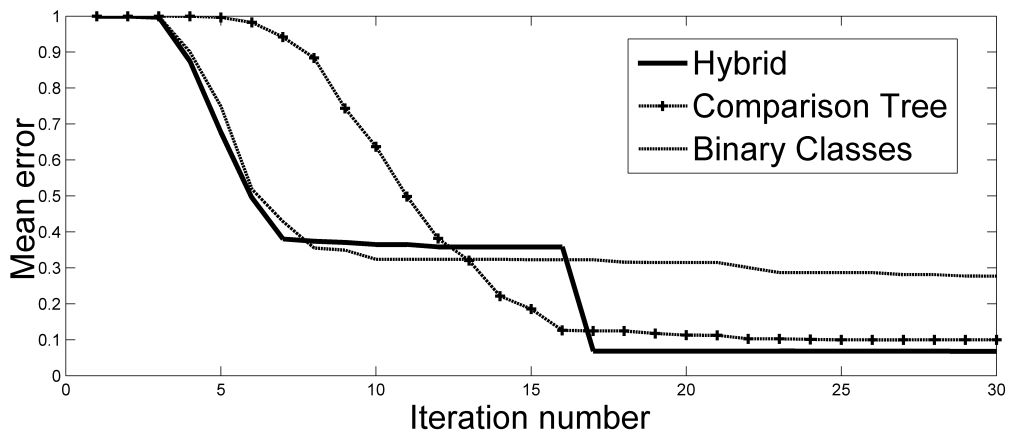


Figure 4.3: 10 dimensional Gaussian,  $\lambda^{\text{user}} = 5$

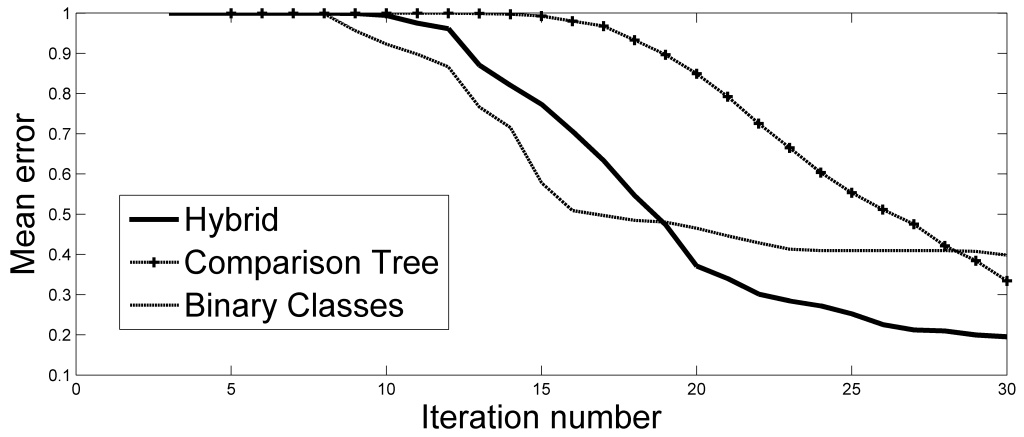


Figure 4.4: 20 dimensional Gaussian,  $\lambda^{\text{user}} = 5$

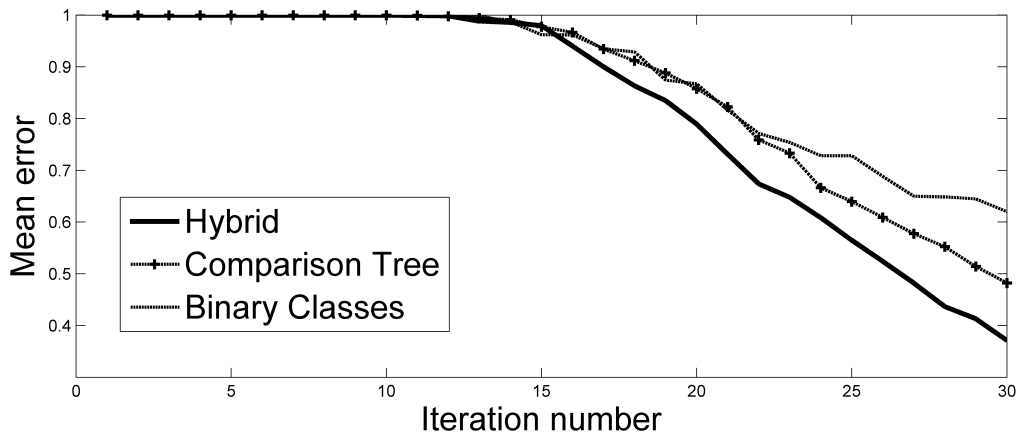


Figure 4.5: 31 dimensional Gaussian,  $\lambda^{\text{user}} = 5$

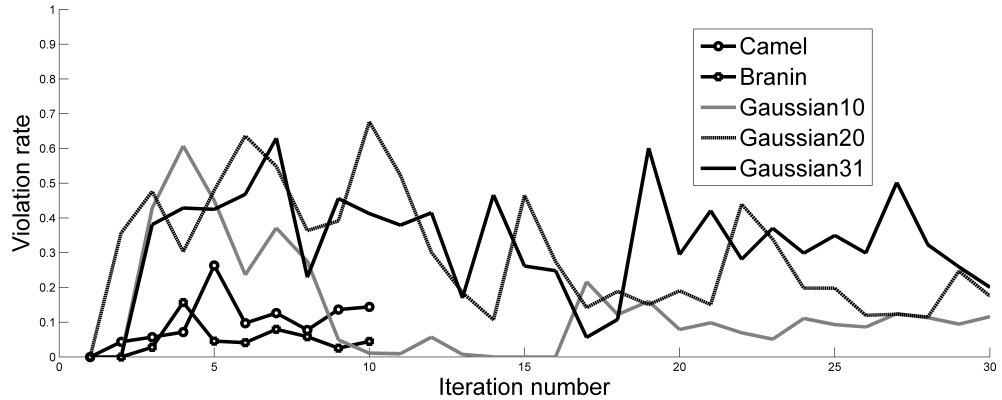


Figure 4.6: Violation rates at each iteration in each test

and disciplines, these variations all share the same functionality which is to balance exploration and exploitation, and thus all require parameter tuning. To simplify the notation, we use  $\hat{f}$  instead of  $\hat{f}(\mathbf{x})$  and  $z$  instead of  $(f_{\min} - \hat{f}/\hat{\sigma})$ , where  $f_{\min}$  is the current best sampled  $\hat{f}$ . Readers are reminded that  $z$  here shall not be confused with  $\mathbf{Z}$  and  $\mathbf{z}$  in other chapters. Also we denote  $\phi(\cdot)$  and  $\Phi(\cdot)$  as the Gaussian probability density function and the Gaussian cumulative distribution function, respectively.

To recall the context of the present discussion, the objective of creating and optimizing the merit function is to find a design (query) that will be potentially close to the optimal solution. When creating this merit function, we have at hand the decision function  $\hat{f}$  representing the predicted preference of a subject, and the mean-squared error of the prediction  $\hat{\sigma}^2$ . Since most optimization literature deals with minimization problems, we follow this protocol in this section and denote the best queried design to have value  $f_{\min}$ , i.e., all merit functions introduced here are applied on minimization problems but will be modified to fit maximization problems in the implementation.

### 4.2.1 Kushner's criterion

Originally proposed by *Kushner* (1964), this criterion is to maximize the probability for the new query to be improved from  $f_{\min}$  by an amount  $\epsilon$ :

$$\begin{aligned} f_{\text{Kushner}} &= P(\hat{f} < f_{\min} - \epsilon) \\ &= \Phi\left(\frac{f_{\min} - \epsilon - \hat{f}}{\hat{\sigma}}\right), \end{aligned} \quad (4.5)$$

where  $\epsilon$  is a user-defined parameter controlling the balance between exploration and exploitation. To explain, when  $\epsilon$  is large, the effect of the term  $-\frac{1}{\hat{\sigma}}$  in the cumulative distribution function is amplified and thus a larger variance  $\hat{\sigma}$  is favorable; on the other hand, when  $\epsilon$  is small compared to  $f_{\min} - \hat{f}$ , the merit function will favour an  $\mathbf{x}$  that has a small  $\hat{\sigma}$  and a large  $f_{\min} - \hat{f}$  which is pure exploitation.

### 4.2.2 Generalized Expected Improvement (GEI)

The definition of generalized expected improvement directly comes from the expected improvement formula in Equation (3.12). Introducing a non-negative integer parameter  $r$ , the expected improvement is now defined as:

$$E[\text{Improvement}(\mathbf{x})] = E[\max\{(f_{\min} - \hat{f})^r, 0\}]. \quad (4.6)$$

Expanding  $(f_{\min} - \hat{f})^r$  to its polynomials, one can get the recursive formula for the generalized expected improvement

$$f_{\text{GEI}} \triangleq E[\text{Improvement}(\mathbf{x})] = \hat{\sigma}^r \sum_{i=0}^r (-1)^i \left( \frac{r!}{i!(r-i)!} \right) z^{r-i} T_i, \quad (4.7)$$

where

$$T_i = -\phi(z)z^{i-1} + (i-1)T_{i-2}, \quad (4.8)$$

starting with  $T_0 = \Phi(z)$  and  $T_1 = -\phi(z)$ . The expected improvement function is a special case of Equation (4.7) when  $r = 1$ . Although not obvious, here  $r$  plays the role of balancing exploitation and exploration. To be specific, the merit function favors more exploitation when  $r$  is low and more exploration when  $g$  is high. Notice that the special case where  $r = 0$  represents a pure exploitation equivalent to  $\epsilon = 0$  in Kushner's criterion.

### 4.2.3 Lower Confidence Bounding function (LCB)

The lower confidence bounding function employed in *Cox and John (1992)* is essentially the weighted sum formula we used in Chapter III. In a minimization context, this merit function can be written as:

$$f_{\text{LCB}} \triangleq \hat{f} - b\hat{\sigma}, \quad (4.9)$$

where  $b$  is a weighting factor that controls the relative importance of  $\hat{f}$  and  $\hat{\sigma}$ .

### 4.2.4 Locating the Regional Extreme (LRE)

This merit function proposed by *Watson and Barnes (1995)* is very similar to the expected improvement merit function, except the additional  $\hat{f}$  term:

$$f_{\text{LRE}} = \begin{cases} \hat{f} + (f_{\min} - \hat{f})\Phi\left(\frac{f_{\min} - \hat{f}}{\hat{\sigma}}\right) + \hat{\sigma}\phi\left(\frac{f_{\min} - \hat{f}}{\hat{\sigma}}\right), & \text{if } \hat{\sigma} > 0 \\ 0, & \text{if } \hat{\sigma} = 0. \end{cases} \quad (4.10)$$

Recalling that the expected improvement function usually has zeros in most places, we see that this additional  $\hat{f}$  acts as a smoothing factor in the merit function.

#### 4.2.5 Switching criterion

Instead of balancing exploitation and exploration, *Sanena* (2002) proposed a hybrid method that purely exploits or explores during the search process. The algorithm starts with a few iterations of pure exploration, regardless of the observations, and then switches to pure exploitation until the queries are close to each other in a sequence, at which point the algorithm goes back to exploration. Similar schemes can also be found from *Locatelli* (1997) and *Žilinskas* (1981).

#### 4.2.6 Computational difficulty

We reviewed different forms of the merit function used that can be in the EGO framework. All of these formulations have the same practical purpose: Search by balancing exploration and exploitation. Also, all merit functions rely on the prediction  $\hat{f}$  and error  $\hat{\sigma}$ . A performance comparison on two-dimensional functions including Branin, Six-hump Camelback and others is presented in *Sanena* (2002) and will not be repeated here. As we briefly mentioned in Chapter III, the major concern we have about using these merit functions is their computational cost in high dimensional problems. To illustrate, Figure 4.7 provides the recorded computational cost at each iteration when finding the optimal solution of the 31-dimensional Gaussian function in Equation (3.19). The test was run on an i5-460M CPU with 4G RAM computer.

The nonlinearity of the merit function and the use of a GA to optimize it are major contributors to the computational cost, which becomes a great obstacle for employing the algorithm in a user-computer interaction. In the following section, we investigate how the computational cost can be reduced while maintaining the quality of the search.

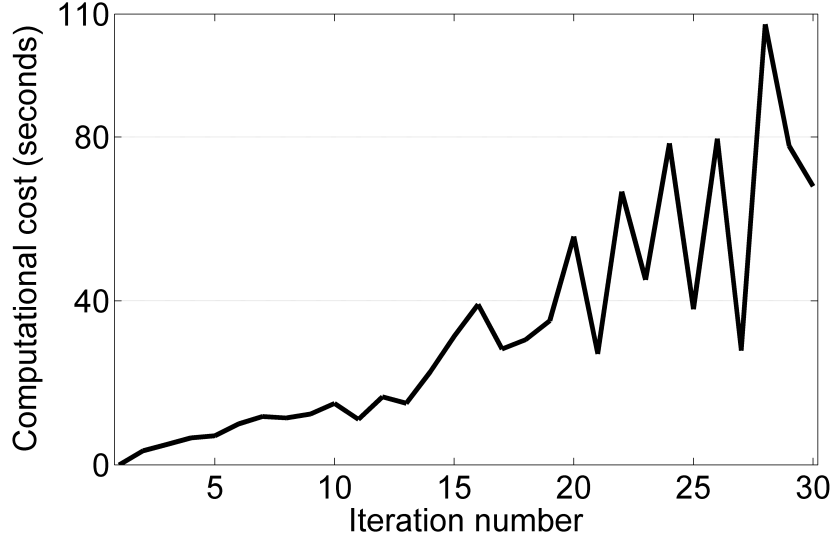


Figure 4.7: The computational costs at the first 30 iterations when optimizing the 31-dimensional Gaussian in Equation (3.19).

### 4.3 Fast EGO and Its Simulated Test Results

Our experiments on simulated tests with EGO show that the nonlinearity of the merit function mainly comes from the error function  $\hat{\sigma}$  which takes low function values (close to zero) around sampled designs and high values elsewhere. This property of the error function causes the expected improvement function and other variations of the merit function to have usually one or a few global optima and many local ones. To illustrate this, Figure 4.8 shows the contour plots of variances of the merit function during the search on a Branin function, where brighter areas indicate higher function values. The crosses represent sampled designs at this moment, and they are colored only to be differentiated from the background contour. In each sub-figure, the bright stripe from the top left corner to the bottom right corner represents a zone of high prediction value. The Kushner’s criterion in this case is pure exploitation and is not preferable. In figures on the other merit functions, the scattered dark areas represent regions close to the sampled designs and have lower merit values than their surroundings. Notice that regions with the highest merit function values (bright areas circled out) are irregular due to the low merit values around the sampled



designs, causing the region to have multiple local optima. Considering that the search algorithm will gradually shift its focus from exploration to exploitation, a fairly large amount of designs will be scattered in a small region close to the optimizer in later iterations. We believe it is this phenomenon that deteriorates the efficiency of the genetic algorithm searching on the merit function.

### 4.3.1 The geometric meaning of $\hat{\sigma}$

With the above observation, and bearing in mind that the goal of a merit function is to balance exploration and exploitation, it is natural to discuss alternatives for the error function  $\hat{\sigma}$  that will potentially make the merit function to be easier to optimize. Although not obvious from its definition in Equation (3.10), the geometric meaning of  $\hat{\sigma}$  at a certain location  $\mathbf{x}$  is nothing other than the minimum distance from  $\mathbf{x}$  to all sampled designs. Figure 4.9 compares the error function  $\hat{\sigma}$  and the minimum distance function defined as:

$$f_{\min \text{ dist}}(\mathbf{x}) = \max_i \{ \exp(-\lambda^{\text{distance}} \|\mathbf{x} - \mathbf{x}_i\|_2) \}, \quad (4.11)$$

where  $\mathbf{x}_i$  for  $i = 1, \dots, n$  are sampled designs and  $\lambda^{\text{distance}}$  plays the same scaling role as  $\lambda^{\text{kriging}}$ . From this comparison, we see that these two ways to describe the geometric arrangement of the samples are almost equivalent under some proper settings of  $\lambda^{\text{distance}}$  and  $\lambda^{\text{kriging}}$ . This observation leads to the new merit function we propose in the next subsection.

### 4.3.2 A computationally inexpensive merit function for EGO

The new merit function we propose has the following form:

$$f_{\text{fast}}(r^2, \mathbf{x}) = w_1 \hat{f}(\mathbf{x}) + w_2 r^2 - \sum_{i=1}^n \exp(p_i(r^2 - \|\mathbf{x} - \mathbf{x}_i\|_2^2)). \quad (4.12)$$

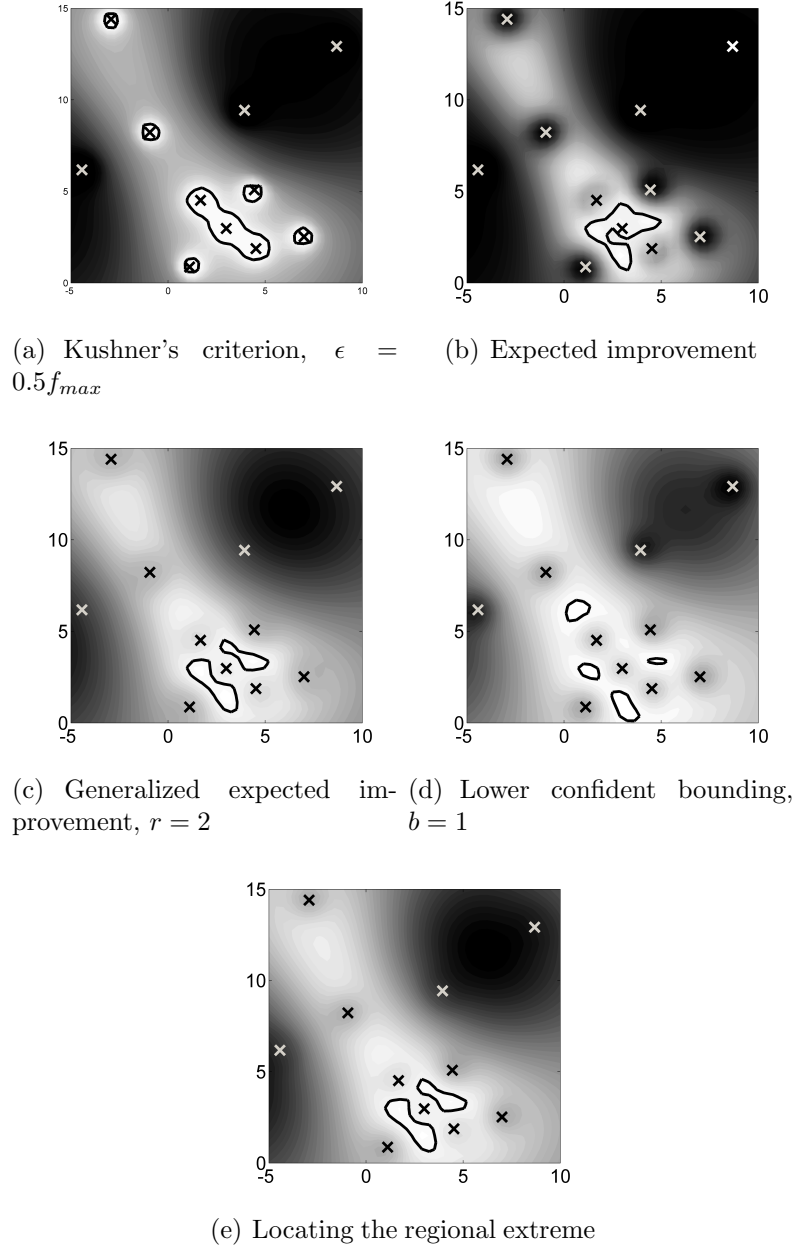


Figure 4.8: Merit functions during the search on a Branin function (Equation (3.17)). Bright areas indicate high function values.

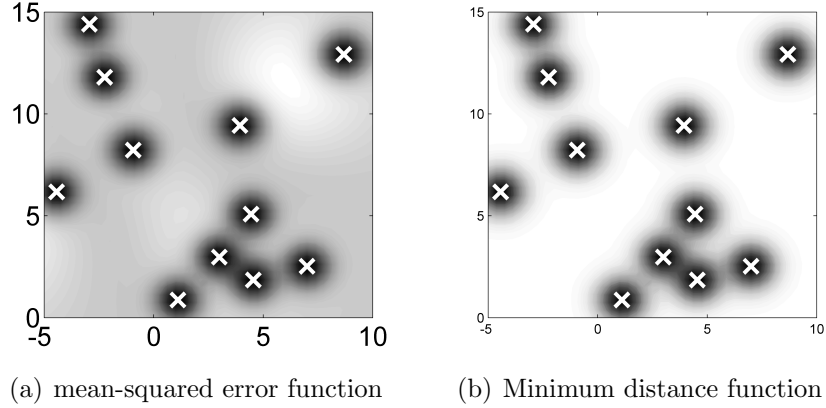


Figure 4.9: Comparison between the mean-squared error function and the minimum distance function under a set of sampled designs (represented as white crosses). Bright areas indicate high function values.

Instead of defining the merit function on  $\mathbf{x}$ , we introduce a new variable  $R^2$  to represent the squared minimum radius from  $\mathbf{x}$  to all existing samples  $\mathbf{x}_i$  for  $i = 1, \dots, n$ . Following the goal of balancing exploration and exploitation, this merit function preserves the two objectives: Maximizing the prediction  $\hat{f}(\mathbf{x})$  and the prediction error which can be represented by  $r^2$ . The setting of the weights  $w_1$  and  $w_2$  determines the focus on either exploration or exploitation and shall be calibrated according to the scale of the two objectives. Given the geometrical meaning of  $r^2$ , we set

$$w_1 = \frac{\sum_{i=1}^p (x_{\min}^i - x_{\max}^i)^2}{\hat{f}_{\max}}, \quad (4.13)$$

where  $x_{\min}^i$  and  $x_{\max}^i$  are the lower and upper bounds of the  $i$ th dimension of  $\mathcal{D}$ . The numerator of  $w_1$  represents the longest distance in  $\mathcal{D}$  and its denominator  $\hat{f}_{\max}$  is the optimum of  $\hat{f}$ . Then  $w_2$  is set at 1 for the first 10 iterations of an interaction, and is reduced to 0.01 afterwards. This setting ensures the two objectives have values of the same scale in the early stages of the search and shifts the focus to exploitation to improve convergence in the later stages.

The last term in Equation (4.12) serves as a penalty scaled by  $p_i$ s. The exponential

function shapes the penalty so that it comes into play when  $r^2 - \|\mathbf{x} - \mathbf{x}_i\|_2^2$  is greater than zero. For  $r^2$  to precisely fulfil its physical meaning, one is required to repeat optimization of the merit function by increasing  $p_i$  if the corresponding term  $r^2 - \|\mathbf{x} - \mathbf{x}_i\|_2^2$  is greater than zero, reducing the value of the penalty term to be less than  $n$ . However, from our experiments, such an iteration is not necessary and the approximated solution with  $p_i = 1$  for  $i = 1, \dots, n$  is enough to provide satisfactory search performances on high dimensional Gaussian functions.

We summarize the algorithm for optimizing this merit function below.

1. Optimize  $\hat{f}$  using sequential quadratic programming starting with a support vector with label 1. This will not find the global optimum of  $\hat{f}$  but the local solution is good enough for the purpose of weighting the two objectives. Also, using a gradient-based algorithm with an initial guess that is close to a local solution reduces the cost of this step.
2. Apply GA on the merit function with  $w_1$  and  $w_2$  set as described. The variable  $\mathbf{x}$  is bounded in  $\mathcal{D}$  and the variable  $r^2$  is bounded below by

$$r_l^2 = \frac{\sum_{i=1}^p (x_{\min}^i - x_{\max}^i)^2}{n^2}, \quad (4.14)$$

where  $n$  is the number of samples so far. This is set to force the solution  $\mathbf{x}$  to be sufficiently away from sampled designs.

3. If multiple designs are required for a query, the above two steps are repeated for each new design needed. Once a new design is found, it is placed into the sampled set so that the next new design will not replicate the previous ones.

### 4.3.3 Simulated test results

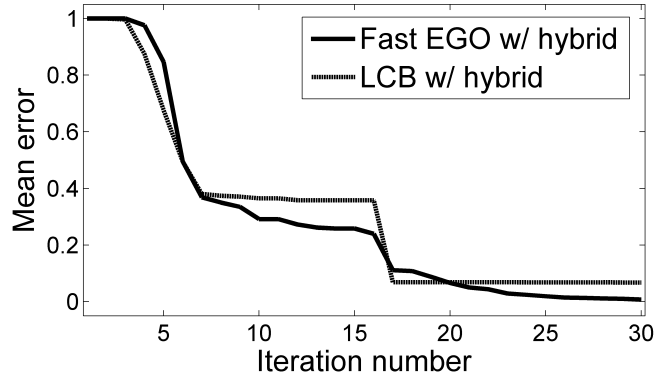
We compare the convergence performance and corresponding computational costs of searches using  $f_{\text{fast}}$  and  $f_{\text{icb}}$  as their merit functions. The experiment is conducted

using the Gaussian functions from Equation (3.19) with 10, 20 and 31 dimensions and  $\lambda^{\text{utility}} = 5$ . Throughout the experiment, the hybrid decision function introduced in Section 4.1 is employed. The MATLAB genetic algorithm toolbox is used to solve the merit function and the maximum generation number is set at 500. Figure 4.10 compares the convergence performance and Figure 4.11 compares the corresponding computational costs. The result is encouraging, as the performance from using  $f_{\text{fast}}$  is not compromised and the computational cost is reduced significantly. In fact, when using  $f_{\text{fast}}$ , the computational cost does not grow with the growth of the sampled set, which is the case when using  $f_{\text{lcb}}$ .

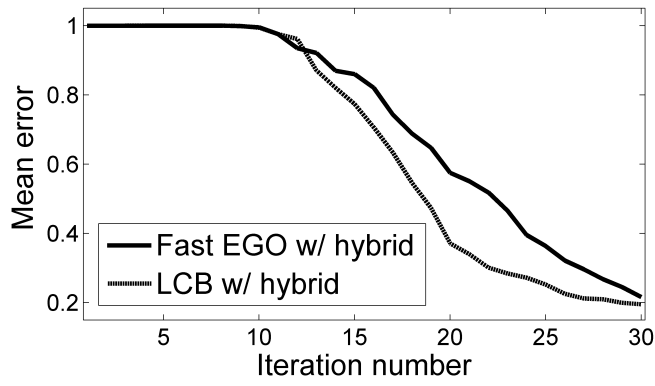
#### 4.3.4 Discussion

Although we do not have a theoretical proof for the superior results of the proposed Fast EGO algorithm, the following empirical discussion may provide some explanation. The general argument is that Fast EGO can alleviate the nonlinearity of the merit function by introducing the new variable  $r^2$  to the search space. At each realization of  $r^2$ , the merit function is smoother than the original one and a solution can be obtained in fewer iterations. We illustrate an example in Figure 4.12.

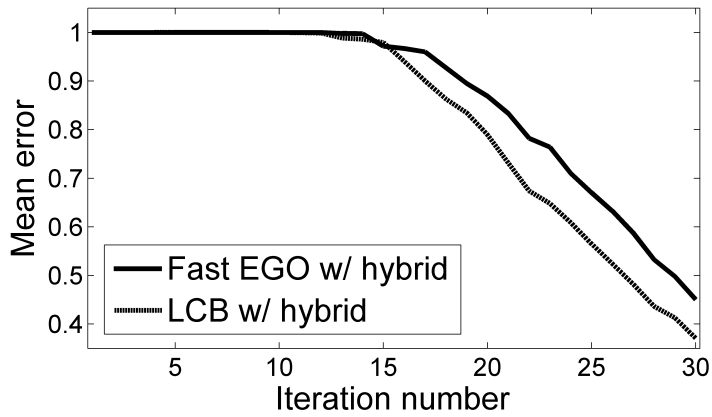
Here we show several slices of the Fast EGO merit function with  $r^2 = 0.1, 1, 1.2$  and 1.5 during a search on the Branin function, Equation 3.17. Crosses in each figure represent sampled designs at that moment and are colored only to be differentiated from the background contours. Brighter areas indicate higher merit function values. Recalling the physical meaning of  $r^2$ , we can see that the change in the merit along  $r^2$  is intuitive: When  $r^2 = 0.1$ , the merit function almost coincides with the decision function  $\hat{f}$ . When  $r^2$  increases, the optima gradually move to the regions that have not been sampled. In general, we conclude that the genetic algorithm works better in this expanded search space  $r^2 \times \mathcal{D}$  where the function is smoother than in the original design space  $\mathcal{D}$  where the function is highly nonlinear.



(a) 10-dimensional Gaussian,  $\lambda^{\text{utility}} = 5$

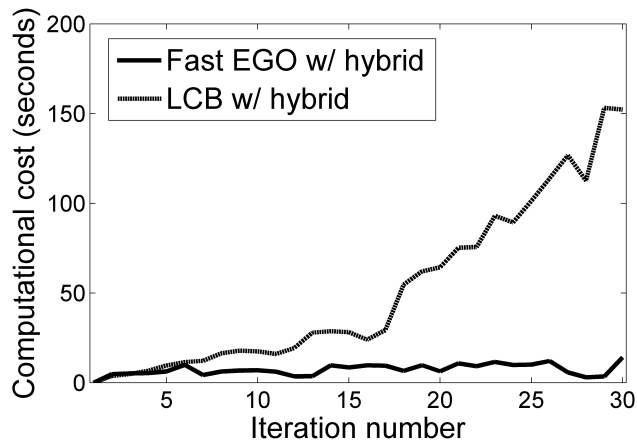


(b) 20-dimensional Gaussian,  $\lambda^{\text{utility}} = 5$

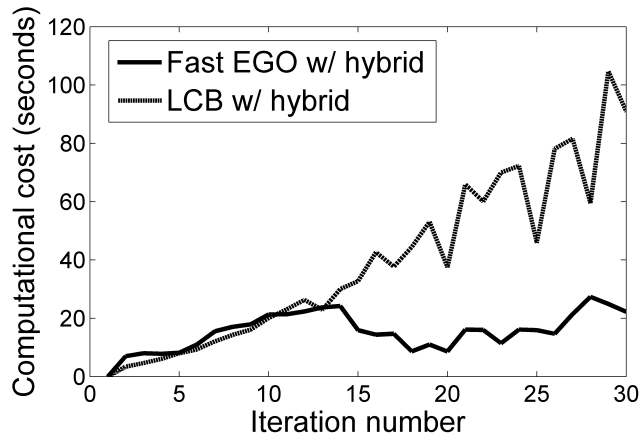


(c) 31-dimensional Gaussian,  $\lambda^{\text{utility}} = 5$

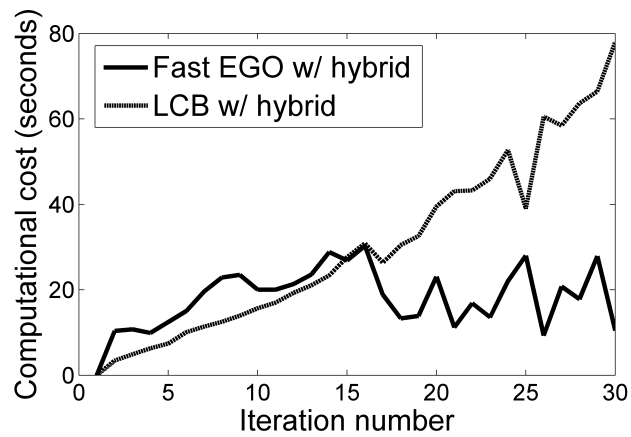
Figure 4.10: Convergence performance of searches with  $f_{\text{fast}}$  and  $f_{\text{lcb}}$  as their merit functions.



(a) 10 dimensional Gaussian,  $\lambda^{\text{utility}} = 5$

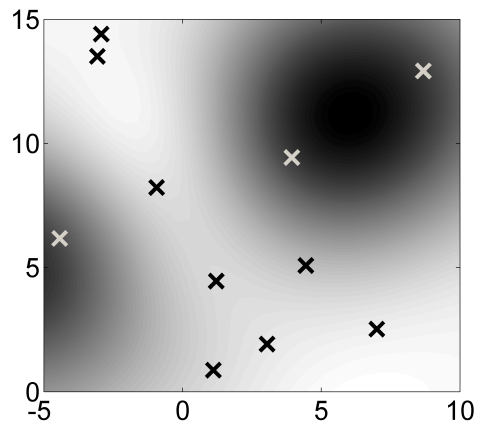


(b) 20 dimensional Gaussian,  $\lambda^{\text{utility}} = 5$

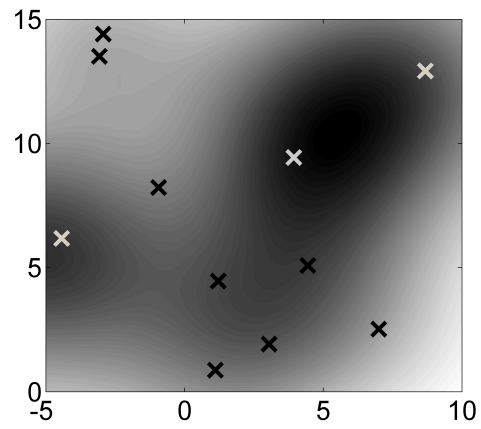


(c) 31 dimensional Gaussian,  $\lambda^{\text{utility}} = 5$

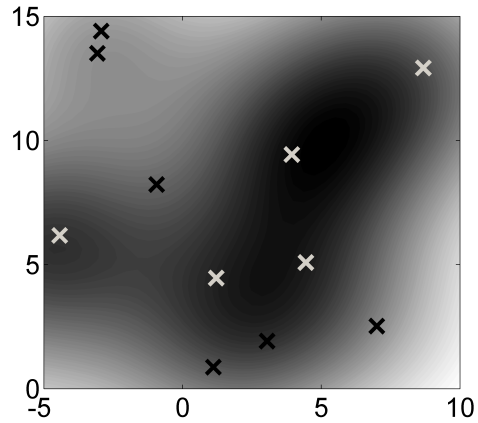
Figure 4.11: Computational cost of searches with  $f_{\text{fast}}$  and  $f_{\text{lcb}}$  as their merit functions.



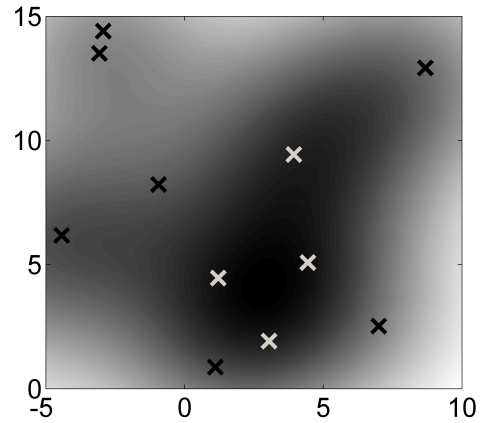
(a)  $r^2 = 0.1$ , min: -11.39, max: 1.33



(b)  $r^2 = 1$ , min: -14.85, max: 0.92



(c)  $r^2 = 1.2$ , min: -16.63, max: 0.60



(d)  $r^2 = 1.5$ , min: -23.60, max: -0.19

Figure 4.12: Contour plot of the merit function at different  $r^2$  values.



## 4.4 Concluding Remarks

This chapter focused on improvements of the EGO Search algorithm from two aspects:

- We considered the accumulative user binary choice response as a tree structure and from there reformulated the estimation problem of the utility function. The estimation from this method can avoid violations to the observed responses and is thus more accurate. We showed by simulation that a hybrid approach where rough estimations of the utility is used during the first leg of querying and the improved estimation during the second leg will improve the search performance, and this improvement is significant in high dimensions.
- We proposed a new form of merit function by adding one more dimension to the search space of the merit. The computational cost on this new Fast EGO merit function is largely reduced using the same genetic algorithm as before due to the smoothness of the new function. This idea has the potential to improve also the efficiency of EGO in traditional “black-box” optimization problems.

There are two possible directions we still need to explore: (1) The chosen designs from a set during any iteration shall have similar utilities compared to the rest. This is so because we assume the subject clusters the utilities of the set into two groups where the between-distance is larger than the within-distance. Incorporating this information in utility estimation may further improve its accuracy. (2) The weights in the proposed merit function, i.e.,  $w_1$  and  $w_2$  are empirically set to tune the focus of a search between exploration and exploitation. An automatic rule to achieve this would be a useful further development.

## CHAPTER V

### Preference Estimation and Identification

This chapter discuss the solutions of the preference estimation and identification problems defined in Chapter I. We reiterate the problem definitions below and will show that they share the same solution.

1. In “preference estimation”, we still use comparative preference tests during the interaction, i.e., given a choice set, the subject assigns “1” to the relatively preferred designs and “-1” to the rest. Instead of optimizing the utility, the goal here is to estimate the utility function such that the order of any design pair  $\mathbf{x}_1$  and  $\mathbf{x}_2 \in \mathcal{D}$  will be consistent with the subjects’ true preference, i.e.,  $f(\mathbf{x}_1) > f(\mathbf{x}_2)$  when  $\mathbf{x}_1$  is more preferred than  $\mathbf{x}_2$ .
2. The “preference identification” problem uses definite preference tests where the subject assigns “1”s only to those designs that they truly prefer and “-1”s otherwise. The goal of this problem is to identify the decision boundary in  $\mathcal{D}$  between the preferred and not-preferred classes.

The preference estimation problem is usually known as parameter (part-worth) estimation in conjoint analysis. Recall from Section 2.1 that the parameters  $\mathbf{w}$  can be estimated and used to model the utility

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{v}(\mathbf{x}) + \varepsilon, \tag{5.1}$$

where  $f$  is the utility for design features  $\mathbf{v}(\mathbf{x})$  and  $\varepsilon$  is the unexplained error. The probability for a subject to choose a certain design is modeled as a function of the utility differences between the chosen design and all alternatives. An estimator  $\hat{\mathbf{w}}$  of the true parameters is then determined by maximizing the likelihood of all test responses from a population of subjects. Overall, the quality of a preference estimation procedure is measured by the training and predictive performance of the estimator and the amount of effort spent on collecting the data.

Most of the research effort from econometrics and the engineering design community has been devoted to building specific models to explain the data better. Some significant contributions include the early development of multinomial logit (*McFadden (1973)*), nested logit (*Wen and Koppelman (2001)*), mixed logit (*McFadden and Train (2000)*; *Train (2001)*) models and also non-utility-based models such as *Hauser et al. (2010)*. With regard to data collection (or questionnaire design in conjoint analysis), the main idea is to design experiments (a set of queries) that will minimize the variance of the estimator. Although theoretically sounding, queries (or sometimes called questionnaires) for conjoint analysis are notoriously known for being prolonged and repetitive. It is not until recently that this (non-adaptive) method has been challenged by the concept of active learning (adaptive query) from both the econometrics and machine learning communities, but with very limited recognition from each community of the work in the other. To make terminology concise, we will use the term “active learning” throughout this writing. The idea of active learning is to create queries that maximize the information gain from the subject and also maintain the diversity of the query in the feature space. Independent research in econometrics and machine learning show that active learning can increase the convergence speed of an estimation (see *Toubia et al. (2004, 2003, 2007a)*; *Abernethy et al. (2008)*; *Tong and Chang (2001)*; *Chang et al. (2005)*; *Tong and Koller (2002)*) as well as enhance the accuracy of the estimator (see *Toubia et al. (2007a)*; *Abernethy et al. (2008)*).

The purpose of this chapter is to verify the usability of active learning in preference estimation and identification.

Notation-wise, we discretize the design space  $\mathcal{D}$  of  $p$  dimensions and  $l$  levels for each dimension to have  $N = l^p$  designs  $\mathbf{x}_i$  for  $i = 1, \dots, N$ . The maximum number of pairwise comparisons that can be used in preference estimation is  $M = (N^2 - N)/2$ . In preference identification, we use  $\mathbf{V}$  ( $n \times p$  matrix) to represent the sampled design set where each row contains a feature vector  $\mathbf{v}$  of a design  $\mathbf{x}$ . In preference estimation, we use  $\mathbf{Z}$  as the sampled feature difference set where each row is the feature difference between some design  $i$  and  $j$ :

$$\mathbf{z} = \mathbf{v}_i - \mathbf{v}_j. \tag{5.2}$$

We use  $\mathbf{V}_N$  in preference identification and  $\mathbf{Z}_M$  in preference estimation to represent the entire candidate set. The term “questionnaire” is used interchangeably with the term “interaction”, meaning a set of queries conducted on a subject.

The rest of the chapter is organized as follows: We review the traditional non-adaptive query methods (Section 5.1.1), the development of active learning techniques from econometrics (Section 5.1.2) and machine learning (Section 5.1.3), followed by a summary of their commonalities and differences. We then switch the discussion to how features shall be formulated in the active learning problem. Sections 5.2 and 5.3 on Preference Estimation and Identification, respectively, verify the application and elaborate implementations of active learning in the two specific problems we are interested in. We conclude with contributions and point out limitations of the current study in Section 5.4.

## 5.1 Active Learning Background

### 5.1.1 Non-adaptive query

Non-adaptive methods are commonly used in traditional (Choice-based Conjoint) questionnaires. Recall from the review in Section 2.1.3 that the maximum likelihood estimator of  $\mathbf{w}$  in a multinomial logit model is asymptotically normal with its mean equal to its true value and covariance matrix equal to the inverse of the information matrix  $\Omega$  given by:

$$\Omega = \sum_{i=1}^N R_i \sum_{j=1}^{J_i} (\mathbf{v}_{ji} - \bar{\mathbf{v}}_i)^T P_{ji} (\mathbf{v}_{ji} - \bar{\mathbf{v}}_i), \quad (5.3)$$

where  $R_i$ ,  $i = 1, \dots, N$  are the effective number of replicates (i.e., a query  $i$  answered by  $R_i$  subjects);  $J_i$  is the number of designs  $\mathbf{v}$  in each query and  $P_{ji}$  is the probability of design  $\mathbf{v}_{ji}$  being chosen in query  $i$ . Without loss generality but to simplify the theoretical development, we use  $J_i = 2$  (pairwise comparison) for each query. For individual questionnaire with  $n$  queries,  $\Omega$  becomes

$$\Omega = \mathbf{Z}^T \text{diag}(e_1, \dots, e_n) \mathbf{Z}, \quad (5.4)$$

where  $e_i$  for  $i = 1, \dots, n$  have the form

$$e_i = \frac{\exp(\mathbf{w}^T \mathbf{z}_i)}{(1 + \exp(\mathbf{w}^T \mathbf{z}_i))^2}. \quad (5.5)$$

The objective of designing queries is to minimize the volume of  $\Omega^{-1}$  or equivalently maximize the determinant (a measure of volume) of  $\Omega$ . Notice that  $e_i$  are functions of the true parameters  $\mathbf{w}$  which are yet to be estimated. Therefore a common practice is to consider  $\hat{\mathbf{w}} = \mathbf{0}$  a priori (hence the probabilities of choosing each design are equal) and reduce the problem to maximizing the determinant of  $\mathbf{Z}^T \mathbf{Z}$ . This method

is called  $D$ -optimal design, see [Arora and Huber \(2001\)](#); [Huber and Zwerina \(1996\)](#); [Kuhfeld et al. \(1994\)](#); [Kuhfeld \(2005\)](#) for example. Readers shall understand that the term “design” here in “ $D$ -optimal design” is not the design  $\mathbf{x}$ . Here “design” refers to the choice of the candidate set  $\mathbf{Z}$  to be used in a questionnaire (or an experiment in general). When the resulting information matrix is proportional to the identity matrix, the  $D$ -optimal designs are called orthogonal and balanced. Orthogonality refers to each design being normal to the others, and balance refers to the information matrix (or the covariance) being spherical. Such designs are denoted as  $D_0$ -optimal designs and are only available for specific combinations of the numbers of queries, designs per query, dimensions and levels per dimension ([Bunch et al. \(1996\)](#); [Huber and Zwerina \(1996\)](#); [Kuhfeld et al. \(1994\)](#); [Kuhfeld \(2005\)](#); [Mitchell \(1974\)](#)). Several authors have proposed to use a non-zero estimator of  $\mathbf{w}$  in  $D$ -optimal design with the estimator derived from a set of pre-test subjects or from prior belief ([Arora and Huber \(2001\)](#); [Huber and Zwerina \(1996\)](#); [Kanninen \(2002\)](#); [Sándor and Wedel \(2001\)](#)). With prior knowledge  $\hat{\mathbf{w}}_0$ , one can minimize the covariance volume by choosing a pair of designs that have balanced (equivalent) utilities. To elaborate, notice that in Equation (5.5),  $e_i$  achieves its maximum when the utilities of the pair are balanced (equal), i.e.,  $e_i \leq 1$  with the equivalence achieved if and only if  $\mathbf{w}^T \mathbf{v}_{1i} = \mathbf{w}^T \mathbf{v}_{2i}$ . In practice, when the a priori knowledge  $\hat{\mathbf{w}}_0$  is close to  $\mathbf{w}$ , subjects will find the pair of designs with balanced utility hard to differentiate. The queries generated this way are usually called aggregate customization designs. The active learning method introduced below is similar to aggregate customization, with the estimation updated in real time.

### 5.1.2 Active learning in conjoint analysis

Active learning (or adaptive query) was first introduced to choice-based conjoint analysis in [Toubia et al. \(2004\)](#) to challenge the traditional non-adaptive  $D$ -optimal design. Considering the constrained space containing all possible estimators  $\hat{\mathbf{w}}$  as

a  $k$ -dimensional space  $\mathcal{V}$  (see Equation (5.6)), and any further pairwise comparison  $\hat{\mathbf{w}}^T \mathbf{z} > 0$  to be a half-space in  $\mathcal{V}$ , the procedure of querying and estimating  $\mathbf{w}$  can be treated as cutting  $\mathcal{V}$  with hyperplanes in a sequence and using the center of the remaining polyhedral as the estimator ( $\hat{\mathbf{w}}$ ). Therefore, increasing the convergence of estimation depends on the cutting strategy and queries are more informative if they reduce the volume of  $\mathcal{V}$  more rapidly. *Toubia et al. (2004)* suggested that a query normal to the current  $\hat{\mathbf{w}}$  will approximately cut  $\mathcal{V}$  into equal halves and has the most expected volume reduction. One shall see that this is indeed utility balance, the motivation of which is more rigorously proved in *Tong and Koller (2002)*. *Toubia et al. (2003)* then extended this work to address the zero volume  $\mathcal{V}$  issue in practice when subjects are prone to fault choices.

$$\mathcal{V} = \{\mathbf{w} \mid \mathbf{w}^T \mathbf{w} = 1, y_i(\mathbf{w}^T \mathbf{z}) > 0, i = 1 \dots n\}. \quad (5.6)$$

The concept of adaptive choice-based conjoint was then further completed with the introduction of statistical learning. *Abernethy et al. (2008)* proposed a two-step algorithm for choosing the next query: (1) A regularized information matrix is projected onto the hyperplane of the current estimator  $\hat{\mathbf{w}}$ ; and (2) the eigenvector associated with the smallest eigenvalue of this projected matrix is chosen as the next query. One can see that this is a mixed strategy taking into consideration both utility balance (by step 1) and D-efficiency (by step 2, see Section 5.2.4 for more detail). It is shown that this query strategy has robust performance under different subject fault choice rates (*Abernethy et al. (2008)*). This is the main strategy we adopt in this chapter, as we elaborate below.

### 5.1.3 Active learning in machine learning

In supervised machine learning, it is often the case that querying labels  $\{y_i\}_{i=1}^n$  from all candidate samples  $\mathbf{V}_N$  is expensive. Active learning is referred to as the

algorithm starting with a subset of  $\mathbf{V}_N$  and deciding the next best query to present using the current classifier. This method potentially helps to reduce the number of queries needed for converging the estimation of the classifier. Active learning was first derived in the area of relevance feedback where the goal is to train a machine to understand a human concept using interaction. For example, using active learning, [Tong and Chang \(2001\)](#) and [Chang et al. \(2005\)](#) trained the search engine to generate more relevant images based on input keywords, and [Mandel et al. \(2006\)](#) created a music retrieval system. We provide the technical details of active learning mainly following the derivations and terminologies in [Tong and Koller \(2002\)](#), which is in turn rooted in SVM (see Section 2.2 for details on SVM).

To start with, we take a close look at the geometrical meaning of the space  $\mathcal{V}$  defined in Equation (5.6). In machine learning literature,  $\mathcal{V}$  is called the “version space”. It is part of a hypersphere in the feature space  $\mathcal{F}$  with radius 1, constrained by hyperplanes  $y_i(\mathbf{w}^T \mathbf{v}_i) \geq 0$ , as illustrated in Figure 5.1. Recall that the SVM objective can be considered as maximizing the margin  $\min_i \{y_i(\mathbf{w}^T \mathbf{v}_i)\}$ . Geometrically, this objective is equivalent to finding a  $\hat{\mathbf{w}}$  corresponding to the center of the largest hypersphere bounded in  $\mathcal{V}$ . An informative query is made when its corresponding hyperplane intersects with  $\mathcal{V}$ . [Tong and Koller \(2002\)](#) proved that a query that cuts  $\mathcal{V}$  into two equal-sized halves will minimize the maximum expected size of  $\mathcal{V}$ , where the maximum is taken over all possible  $\mathbf{w}$ . In other words, a query set that halves  $\mathcal{V}$  will never be the worst strategy when the true  $\mathbf{w}$  is unknown. Noticing the geometrical meaning of  $\hat{\mathbf{w}}$ , it is reasonable to argue that  $\mathbf{w}$  is approximately the center of  $\mathcal{V}$ . Therefore, a bisection can be approximated by cutting through  $\hat{\mathbf{w}}$ . In other words, the query  $\mathbf{v}$  should be preferably normal to  $\hat{\mathbf{w}}$  or at least minimize the angle  $|\hat{\mathbf{w}}^T \mathbf{v}|$ . Such a strategy that approximates  $\mathbf{w}$  as the center of  $\mathcal{V}$  is called the “simple algorithm” ([Tong and Koller \(2002\)](#)). This concludes the reasoning for utility balance from a machine learning perspective.



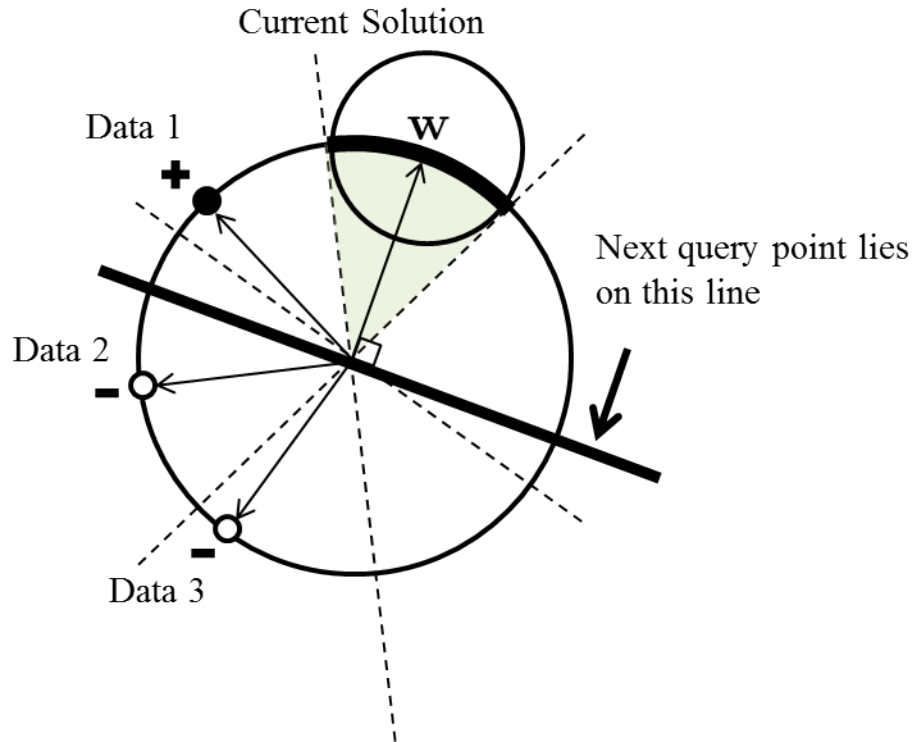


Figure 5.1: Geometrical representation of the version space. In this 2D case, the version space is the highlighted arc of the circle. Each normal vector of a constraining hyperplane represents a sample point  $\mathbf{v}_i$  and the label  $y_i$  determines which side of the hyperplane is feasible for  $\mathbf{w}$ . The solution  $\hat{\mathbf{w}}$  of a classification problem is the center of the largest hypersphere within the gray cone  $y_i(\mathbf{w}^T \mathbf{v}) > 0$ ,  $i = 1 \dots n$ . The bisection of the version space  $\mathcal{V}$  can be approximated by cutting through the current solution  $\hat{\mathbf{w}}$ , i.e.,  $\hat{\mathbf{w}}^T \mathbf{v} = 0$ .

Besides the simple algorithm, *Tong and Koller (2002)* provided another two algorithms rooted in the same motivation: (1) The “maxmin algorithm” approximates the area of two halves ( $m_+$  for  $y = 1$  and  $m_-$  for  $y = -1$ ) resulting from any query and then chooses a query that maximizes the minimum of the two, i.e.,  $\mathbf{v} = \arg \max_{\mathbf{v}} \min\{m_+(\mathbf{v}), m_-(\mathbf{v})\}$ . (2) The “ratio” algorithm takes a similar route as “maximin”. It chooses a query that maximizes the minimum of the area ratio and its inverse, i.e.,

$$\mathbf{v} = \arg \max_{\mathbf{v}} \min\{m_+(\mathbf{v})/m_-(\mathbf{z}), m_-(\mathbf{v})/m_+(\mathbf{v})\}. \quad (5.7)$$

*Tong and Koller (2002)* show that all three algorithms outperform random queries. While “simple” has the least computational cost, “maximin” and “ratio” have more stable performance based on test data cross different domains.

Since multiple candidates may exist satisfying the condition  $\hat{\mathbf{w}}^T \mathbf{v} = 0$  (indeed, an infinite number of candidates exists when the query space  $\mathbf{Z}$  is continuous), *Chang et al. (2005)* investigated four heuristics for choosing a query set, namely, speculative, batch-simple, error-reduction and angle-diversity.

The speculative heuristic recursively applies the simple algorithm on  $\mathcal{V}$ , then the resulting halves  $\mathcal{V}_+$  and  $\mathcal{V}_-$ , and so on; The batch-simple heuristic chooses a set of queries that are closest to the decision boundary and relatively apart from each other. The error-reduction heuristic (also see *Roy and McCallum (2001)*) attempts to reduce the expected error on future test examples. To this end, denote the current query set and associated labels as  $L \triangleq \{\mathbf{V}, \mathbf{y}\}$ . The distribution of output  $\hat{P}_L(y|\mathbf{v})$  for a given  $\mathbf{v}$  can be estimated following the training of  $\hat{\mathbf{w}}$ . If the unknown true distribution is  $P(y|\mathbf{v})$ , the expected error of the classifier can be written as

$$E[\text{Error}] = \int_{\mathbf{v}} \text{Loss}(P(y|\mathbf{v}), \hat{P}_L(y|\mathbf{v}))P(\mathbf{z})d\mathbf{v}, \quad (5.8)$$

where the loss function measures the difference between the estimation and its true distribution. [Chang et al. \(2005\)](#) and [Roy and McCallum \(2001\)](#) use the log-loss function which has the form

$$\text{Loss}(P(y|\mathbf{v}), \hat{P}_L(y|\mathbf{v})) = \sum_{y \in -1,1} P(y|\mathbf{v}) \log(\hat{P}_L(y|\mathbf{v})). \quad (5.9)$$

In practice, the unknown  $P(y|\mathbf{v})$  is replaced by  $\hat{P}_L(y|\mathbf{v})$  in Equations (5.8) and (5.9). The error-reduction heuristic selects a query  $\mathbf{v}^*$  if  $E[\text{Error}_{\hat{P}_{L \cup \mathbf{v}^*}}]$  is smaller than  $E[\text{Error}_{\hat{P}_{L \cup \mathbf{v}}}]$  for any other query  $\mathbf{v}$ . Finally, the angle-diversity heuristic has the best empirical performance according to [Chang et al. \(2005\)](#). The idea of angle-diversity is to select queries close to the decision boundary and also maintain their diversity, which is measured by the angle between queries. The angle between two queries  $\mathbf{v}_1$  and  $\mathbf{v}_2$  can be written as

$$|\cos(\angle(\mathbf{v}_1, \mathbf{v}_2))| = \frac{\mathbf{v}_1^T \mathbf{v}_2}{\|\mathbf{v}_1\|_2 \|\mathbf{v}_2\|_2}. \quad (5.10)$$

Introducing a weighting parameter  $C$ , the angle-diversity algorithm chooses a query that minimizes the merit function

$$\text{Merit} = |\hat{\mathbf{w}}^T \mathbf{v}^*| + C \left( \max_{\mathbf{v}^* \in \mathbf{V}} \frac{\mathbf{v}^T \mathbf{v}^*}{\|\mathbf{v}\|_2 \|\mathbf{v}^*\|_2} \right). \quad (5.11)$$

In Equation (5.11), the first term requires the next query  $\mathbf{v}^*$  to be close to the decision boundary and the second term maximizes the smallest angle between the next query and all existing queries  $\mathbf{V}$ . One can see that this strategy is consistent with the two-step approach proposed by [Abernethy et al. \(2008\)](#) with the only difference being the formulation of the diversity of queries.

The previous section reviewed the active learning literature from both the economic and the machine learning communities. One missing discussion is with regard

to the feature definition  $\mathbf{v}$  (or the feature difference  $\mathbf{z}$  in the preference estimation problem). Let us first define the two subtly different terms: “Feature identification” refers to the choice of a feature set that can be used for query design; “Feature Selection” refers to the determination of a subset from the feature set that has the best generalization error. The former is usually conducted by human expert knowledge before data collection while the latter governed by parameter estimation after data collection and provide more insights in the observations. Note that we intentionally discriminate the features  $\mathbf{v}$  from the design variables  $\mathbf{x}$ . The theoretical and practical motivation for this is as follows: First, it is often acknowledged in the econometrics literature that the linearity assumption on the utility function is not always valid, i.e., the utility is not a weighted sum of the design variables. To circumvent this difficulty, studies introduce design features as an arbitrary mapping of the variables to another space where the utility can be linearized; From a practical point of view, choosing the right features can improve the generalization error of a learning algorithm significantly, because the learning will be effective when the variables  $\mathbf{v}$  truly explain the outcome  $y$ . One can see that selecting  $\mathbf{v}$  requires both identification of what features are relevant and selection of features that provide the best generalization error.

In the machine learning literature, features are usually identified manually based on expert knowledge, e.g., *Chang et al. (2005)* and *Tong and Chang (2001)* use color and texture as features for images; *Mandel et al. (2006)* use a short-time spectral decomposition of audio signals (called Mel-frequency cepstral coefficients) as features for songs; *Tong and Koller (2002)* represent text documents as weighted word frequency vectors; *Joachims (2002)* trains a ranking mechanism of a search engine based on user click-through data and uses features such as the number of words that query and document share, or page-rank of the document. Compared with feature identification, feature selection gets more attention by the machine learning community. While a good set of features can help train the data, irrelevant features may deteriorate the

trainer (*Weston et al. (2001)*). *Hermes and Buhmann (2000)* propose to use gradients of a trained kernel machine as indications of the importance of features; *Neumann et al. (2005)* extend the SVM formulation with an extra zero-norm regularization on the estimator to systematically select features within training; *Liu and Zheng (2006)* introduce an importance measure of features and recursively update the feature set through training on the set.

In traditional conjoint analysis, features are identified from revealed (market) data. For example, a civilian vehicle market study (*Frischknecht et al. (2009)*) identifies features such as brands, demographic attributes (age, salary, height and gender), performance attributes (price, horsepower, mpg and footprint) and others from a private customer satisfaction survey. With regard to stated (survey) data, systematic identification of features has also been studied. *Lee and Bradlow (2007)* conducted research in text mining methods to identify features and levels from online customer reviews. Similar to machine learning studies, identification and selection of features will affect the training results in conjoint analysis as well. Regretably, use of linear utility is prevalent, while rarely justified in practice. To circumvent this issue, *Evgeniou et al. (2007)* suggest to treat the estimation problem of choice data in a machine learning fashion. By reformulating the maximum likelihood estimation problem as a regularized regression problem, *Evgeniou et al. (2007)* showed that kernel tricks can be deployed and the resulting estimation has overall better prediction power than a standard hierarchical Bayes approach with a linear utility model.

While there are established routines for feature selection, feature identification in active learning is less investigated. On one hand, one would like to expand the dimensionality of features so that training can be performed well (see *Evgeniou et al. (2007)*); on the other hand, one would also like to maintain a query space with a manageable size so that only limited feature parameters need to be estimated and

explained, and the determinant of the estimation covariance will be small. The angle-diversity algorithm proposed in [Chang et al. \(2005\)](#) is a viable solution which uses kernel SVM to deal with nonlinearity in training and enforces diversity in the kernel space. However, there is a fundamental drawback of applying SVM to preference estimation, as we will discuss in the next section.

## 5.2 Preference Estimation

### 5.2.1 Problem definition

Let us consider the problem of estimating a preference function using a sequence of pairwise comparisons. In pairwise comparison tests, two designs are shown to the user in turn and the user is asked to pick the better one according to her preference. It is also allowed for the user to state that both designs are equally preferred. In either case, a constraint is posed to limit the possible value of  $\mathbf{w}$ , i.e.,

$$h - y\hat{\mathbf{w}}^T(\mathbf{v}_1 - \mathbf{v}_2) \leq 0 \tag{5.12}$$

when a preference choice is made or

$$\hat{\mathbf{w}}^T(\mathbf{v}_1 - \mathbf{v}_2) = 0 \tag{5.13}$$

when designs are indifferent. Here  $h > 0$  is a scaling parameter and  $y = 1$  when  $\mathbf{v}_1$  is chosen over  $\mathbf{v}_2$  or  $y = -1$  in the opposite case. Since active learning methods developed for pairwise comparison can be easily extended to have queries with multiple designs by treating them as sets of pairwise comparisons, it is safe for us to focus on pairwise comparisons only. To reiterate the notations, we discretize the design space  $\mathcal{D}$  of  $p$  dimensions and  $l$  levels for each dimension to have  $N = l^p$  designs  $\mathbf{x}_i$  for  $i = 1, \dots, N$ . The maximum number of pairwise comparisons is  $M = (N^2 - N)/2$ .

### 5.2.2 Design features

As we discussed before, adopting kernel machines such as SVM let us use the design variables directly as features even when the function under estimation is non-linear. However, we show here that SVM is not suitable for preference estimation. Let  $\hat{\mathbf{w}} = \sum_{i=1}^m \alpha_i \mathbf{v}_i$  be the SVM solution, where  $m$  is the number of support vectors  $\mathbf{v}_i$ . The inequality constraint of a comparison requires

$$h - y \sum_{i=1}^m \alpha_i (K(\mathbf{v}_i, \mathbf{v}_1) - K(\mathbf{v}_i, \mathbf{v}_2)) \leq 0, \quad (5.14)$$

where  $K$  is the kernel function used in the training. However, if we examine the constraints used in the SVM training, we actually have

$$h - y \sum_{i=1}^m \alpha_i K(\mathbf{v}_i, \mathbf{v}_1 - \mathbf{v}_2) \leq 0. \quad (5.15)$$

For the inequality constraint to reflect the observation, the first set of inequalities (Inequalities (5.14)) should be applied rather than the second set (Inequalities (5.15)). In fact, one can easily find violations to the observations from an estimator training by SVM. To circumvent this difficulty, we propose to use polynomial and Gaussian bases as features in active learning for preference estimation. For example, we can model the utility as a linear summation of polynomial bases of  $\mathbf{x}$ . Alternatively, we can also use Gaussian bases to have

$$f(\mathbf{x}) = \sum_{i=1}^n w_i \exp(-\lambda \|\mathbf{x}_i - \mathbf{x}\|_2^2). \quad (5.16)$$

where  $\mathbf{x}_i, i = 1, \dots, N$  are the  $N$  vertices (designs) on  $\mathcal{D}$  and the number of coefficients to estimate is equal to the number of vertices. Readers familiar with artificial neural networks can recognize that this is exactly the RBF neural net formulation. We shall emphasize that besides the capability of approximating nonlinear functions, the use

of polynomial and Gaussian bases has another important motivation in preference estimation: Consider the fact that the training data involves only the differences of features (denoted as  $\mathbf{Z}$ ) rather than the features  $\mathbf{V}$ , it is possible to have two different queries with the same expression. For example, given three different designs with features  $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ , the difference vectors  $\mathbf{v}_1 - \mathbf{v}_2$  and  $\mathbf{v}_2 - \mathbf{v}_3$  can be the same. In the case when both Designs 1 and 3 are more preferred than Design 2, the data will contain contradictory observations on the same query expression, which may deteriorate the performance of the training. High-order polynomial bases and Gaussian bases are used in order to reduce the chance that the entire queried sample set  $\mathbf{Z}$  contains replicates.

### 5.2.3 Estimation of $\mathbf{w}$

Let us go through the standard techniques in estimating  $\mathbf{w}$  before discussing how the estimation can be done actively. Following the assumptions to derive the logit model (see Section 2.1 for the derivation), the negative log-likelihood for observations  $\mathbf{y}$  and  $\mathbf{Z}$  can be written in the form of a loss function:

$$L_{ML} = \sum_{i=1}^n \log(1 + \exp(-\theta \mathbf{w}^T \mathbf{z}_i)), \quad (5.17)$$

where  $\theta$  is the parameter for the extreme value distribution. An estimation of  $\mathbf{w}$  can be derived by minimizing  $L_{ML}$ . A machine learning approach can also be used for the same purpose. A concise estimation proposed by [Abernethy et al. \(2008\)](#) minimises the following regularized penalty (this is a ridge regression formulation, or called a regularized network in [Abernethy et al. \(2008\)](#)):

$$L_{SL} = \sum_i (1 - y_i \mathbf{w}^T \mathbf{z}_i)^2 + C \mathbf{w}^T \mathbf{w}. \quad (5.18)$$



Here the index “SL” stands for statistical learning to differentiate this loss formulation from the maximum likelihood one, and  $C$  is a weight on the regularization term  $\mathbf{w}^T \mathbf{w}$  that adjusts the balance between training error and model complexity. To offer a more practical perspective of regularization,  $C$  controls the belief in observations against prior knowledge. This is a necessary treatment to achieve robust training since adaptive questionnaires can be subject to endogeneity: Questions are influenced by the response errors to earlier answers, as noted in [Abernethy et al. \(2008\)](#). The advantage of this estimation is that it has an analytical solution. However, it suffers from the drawback that situations where  $y_i \mathbf{w}^T \mathbf{z}_i > h$  should not be penalized. A remedy to this is the following:

$$L_{SL} = \sum_i \max\{0, h - y_i \mathbf{w}^T \mathbf{z}_i\}^2 + C \mathbf{w}^T \mathbf{w}, \quad (5.19)$$

where  $h$  is a parameter that controls the scale of  $\mathbf{w}$ . We call this formulation SVM training as it is similar to the soft-margin SVM formulation, and also note that this formulation is close to the ranking SVM formulation in [Joachims \(2002\)](#). In this study, we set  $C = 1/s$  where  $s$  is the number of queries (iterations) made so far. This setup ensures that the importance of complexity control goes down as more observations are available. The scaling parameter  $h$  is set at 1.

To justify the use of this formulation for conjoint analysis, we compare the penalties used in maximum likelihood estimation (Equation (5.20)) and in machine learning (Equation (5.21)) in the case of  $y = 1$  and with  $u = -\mathbf{w}^T \mathbf{z}$ . The penalties are inverted to be bounded within 0 and 1. We show in Figure 5.2 that these two penalties from  $L_{ML}$  and  $L_{SL}$  are similar.

$$\text{Penalty}_{ML}(u)^{-1} \triangleq \frac{1}{1 + \exp(\theta u)} \quad (5.20)$$

$$\text{Penalty}_{SL}(u)^{-1} \triangleq \exp(-\max\{0, h + u\}^2) \quad (5.21)$$

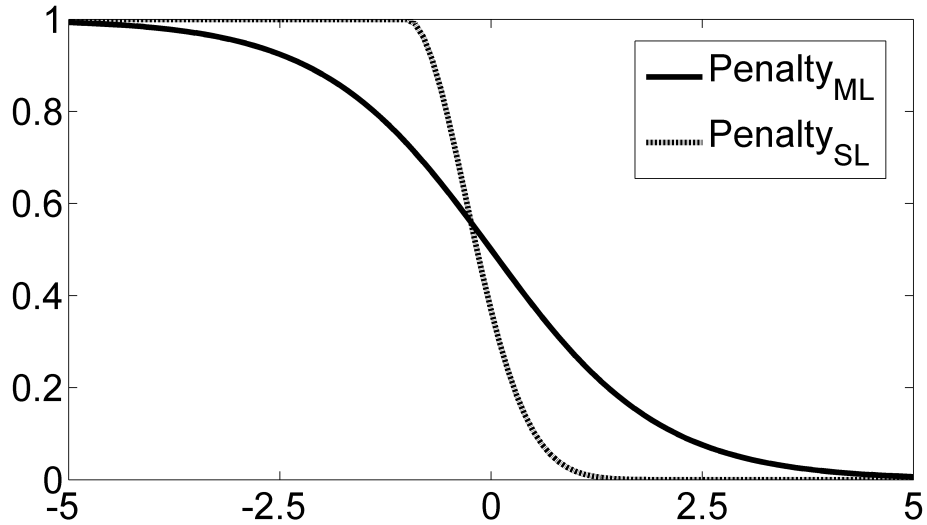


Figure 5.2:  $\text{Penalty}_{ML}$  and  $\text{Penalty}_{SL}$  in the range  $u \in [-5, 5]$  for preference realization;  $\theta = 1$ ,  $h = 1$ .

Above we discussed how  $\mathbf{w}$  can be estimated for individuals. Obviously, if all constraints (constraints in Equation (5.19) or penalties in Equations (5.18)) are observed, the resulting  $\hat{\mathbf{w}}$  will be able to estimate the preference function (the scale of the function value is not determined though). However, the number of candidate pairs to query, i.e., the number of rows in  $\mathbf{Z}_M$  is usually enormous and only a limited number of queries can be made in any questionnaire. The focus of this study is to understand which subset of the queries shall be used to make the estimation efficient. Below we apply active learning to this problem.

#### 5.2.4 Active learning on preference estimation

The drawback of non-adaptive methods is ignoring previous information about  $\mathbf{w}$  during the query process. As an improvement, adaptive query maximizes the determinant of  $\Omega$  sequentially with the updated knowledge about  $\mathbf{w}$  obtained at each iteration. The following approach is proposed in [Abernethy et al. \(2008\)](#).

To start with, we derive the closed-form solution for the regularization network

formulation (Equation (5.18)) at stage  $s$ :

$$\hat{\mathbf{w}}_s = (\mathbf{Z}_s^T \mathbf{Z}_s + C_s \mathbf{I}_s)^{-1} \mathbf{Z}_s^T \mathbf{1}_n. \quad (5.22)$$

We then choose the next query pair  $\mathbf{z}_{s+1}$  in a way that  $e_{s+1}$  in Equation (5.5) is maximized according to  $\hat{\mathbf{w}}_s$  which leads to  $\hat{\mathbf{w}}_s^T \mathbf{z}_{s+1} = 0$ . To elaborate, since  $e_i = P_{1i} P_{2i}$  where  $P_{ji}$  is the probability of design  $j$  in the  $i$ th query being chosen and  $P_{2i} = 1 - P_{1i}$ , it is clear that  $e_i \leq 1/4$  with the equality reached when  $P_{1i} = P_{2i} = 1/2$ . This leads to  $\hat{\mathbf{w}}_s^T \mathbf{z}_{s+1} = 0$ , which is exactly utility balance. In other words,  $\mathbf{z}_{s+1}$  must lie in the space span by the projection matrix  $\mathbf{P}_s \triangleq \mathbf{I} - \hat{\mathbf{w}}_s \hat{\mathbf{w}}_s^T / \hat{\mathbf{w}}_s^T \hat{\mathbf{w}}_s$ . A practical view of utility balance, considering the definition of  $\mathbf{z}$  being the difference between two designs, is that we query the two designs that are believed to be indifferent according to the current estimator  $\hat{\mathbf{w}}$ .

Recall that the covariance of the estimators can be indicated by the Hessian of the loss function (see subsection 2.1.3 or *McFadden and Train (2000)*). Considering the Hessian as a ellipsoid in the space of  $\mathbf{z}$ , a new query  $\mathbf{z}_{s+1}$  will be preferable if its existence morphs the Hessian towards a sphere, in which way the determinant of the Hessian is increased. Taking utility balance into consideration, it is therefore suggested that  $\mathbf{z}_{s+1}$  should be the eigenvector associated with the second minimum eigenvalue of the projected Hessian of the objective in Equation (5.18) onto  $\text{span}(\mathbf{P}_s)$ . The Hessian can be analytically derived as

$$\mathbf{H}_s = C_s \mathbf{I} + \mathbf{Z}_s^T \mathbf{Z}_s, \quad (5.23)$$

and the projected Hessian is

$$\mathbf{Q}_s = \mathbf{P}_s \mathbf{H}_s. \quad (5.24)$$

One thing we should clarify is that  $\mathbf{Q}_s$  has at least one zero eigenvalue because it is

projected to a subspace one dimension lower than the original space  $\mathcal{V}$ . And  $\hat{\mathbf{w}}_s / \|\hat{\mathbf{w}}_s\|_2$  is the eigenvector associated with that zero eigenvalue. Obviously  $\hat{\mathbf{w}}_s / \|\hat{\mathbf{w}}_s\|_2$  is inappropriate to be  $\mathbf{z}_{s+1}$  since we require utility balance. Therefore the eigenvector we are looking for should have the second minimum eigenvalue (can be zero) and is thus perpendicular to  $\hat{\mathbf{w}}_s$ .

We show below that when the projection matrix is not changed by the new query, i.e.,  $\hat{\mathbf{w}}$  stays the same so that  $\mathbf{P}_s = \mathbf{P}_{s+1}$ , this strategy of choosing  $\mathbf{z}_{s+1}$  increases  $\mathbf{Q}_s$  by 1: At the  $s + 1$ th iteration we have  $\mathbf{Q}_{s+1} = \mathbf{Q}_s + \mathbf{P}_s \mathbf{z}_{s+1} \mathbf{z}_{s+1}^T$ . Since  $\mathbf{z}_{s+1}$  is the eigenvector of  $\mathbf{Q}_s$  associated with the second minimum eigenvalue  $\lambda_{\mathbf{Q}}$  and  $\|\mathbf{z}_{s+1}\|_2 = 1$ , we have

$$\begin{aligned} \mathbf{Q}_{s+1} \mathbf{z}_{s+1} &= \lambda_{\mathbf{Q}} \mathbf{z}_{s+1} + \mathbf{P}_s \mathbf{z}_{s+1} \\ &= (\lambda_{\mathbf{z}} + 1) \mathbf{z}_{s+1}. \end{aligned} \tag{5.25}$$

Since the projection matrix will usually be changed with new observations, this strategy of choosing  $\mathbf{z}_{s+1}$  is a heuristic that tries to increase of the determinant of a matrix that is comparable to the information matrix. This step is called minimizing maximum uncertainty.

In summary, the active learning method has two steps: 1) Compute the estimator ( $\hat{\mathbf{w}}$ ); 2) Find  $\mathbf{z}$  as the eigenvector associated with the second minimum eigenvalue of the projected Hessian  $(\mathbf{I} - \hat{\mathbf{w}}_s \hat{\mathbf{w}}_s^T / \hat{\mathbf{w}}_s^T \hat{\mathbf{w}}_s) (C_s \mathbf{I} + \mathbf{Z}_s^T \mathbf{Z}_s)$ . In practice, since the candidates  $\mathbf{Z}_M$  are determined by the discretized design space and these may not be coincide with the eigenvectors of  $\mathbf{Q}$ , we choose  $\mathbf{z}$  that maximizes the following similarity measure:

$$\text{similarity} = \frac{|\mathbf{z}_0^T \mathbf{z}|}{\|\mathbf{z}\|_2} - \frac{|\mathbf{w}^T \mathbf{z}|}{\|\mathbf{z}\|_2}, \tag{5.26}$$

where  $\mathbf{z}_0$  is the eigenvector of  $\mathbf{Q}$  with the second minimum eigenvalue. Notice that this query strategy is similar to that of [Chang et al. \(2005\)](#) with the difference that

we explicitly minimize the estimation variance by increasing the determinant of the information matrix while [Chang et al. \(2005\)](#) justifies a similar approach empirically. Also, the two parts in this similarity measure can be weighted differently to achieve search with biased focus on either utility balance or minimization of maximum uncertainty. In the special case where the first term is neglected, we have a greedy algorithm with pure utility balance.

### 5.2.5 Test setup

For tests in this section, we use a two-dimensional design with 5 levels for each dimension which makes  $N = 25$  and  $M = 300$ . Each dimension is bounded by  $[-1, 1]$ . We compare the performance of the active learning algorithm against random and  $D$ -optimal designs. The set of functions we test includes a linear function in Equation (5.27) (Fn. 1), a polynomial function in Equation (5.28) (Fn. 2), the Branin function in Equation (3.17) (Fn. 3) and the Camelback function in Equation (3.15) (Fn. 4).

$$f(x_1, x_2) = x_1 + x_2. \tag{5.27}$$

$$f(x_1, x_2) = x_1^2 + x_1x_2 - x_2^3. \tag{5.28}$$

The performance of an algorithm is measured by the error rate of the estimated function:

$$\text{error rate} = \frac{\text{number of pairs with wrong outcomes}}{\text{total number of pairs}}. \tag{5.29}$$

The spread of Gaussian bases is set to  $\lambda = 1/N$ . With regard to polynomial bases, we use polynomials of order 3 and arrange terms in the following order:  $x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2$  and  $x_2^3$ . For the random algorithm, we report the mean and standard deviation of the error rate from 100 tests. The setup for  $D$ -optimal designs is elaborated in Subsection 5.2.6. In all tests, the candidate set  $\mathbf{Z}_M$  is normalized so that the candidates are centered around the origin and has compatible scales on each

dimension.

A set of tests for different purposes is provided: We start in Subsection 5.2.7 to compare generalization errors on the test functions using active learning,  $D$ -optimal and random query under 10, 20 and 50 queries; Subsection 5.2.8 illustrates how the linearity assumption may fail when nonlinear preference is present. In these two subsections we assume no faulty choice from the subject, i.e., the random error  $\varepsilon$  in Equation (5.1) is zero and thus Design 1 is preferred over Design 2 if and only if  $\mathbf{w}^T(\mathbf{v}_1 - \mathbf{v}_2) > 0$  and the two designs are labeled as indifferent if and only if  $\mathbf{w}^T(\mathbf{v}_1 - \mathbf{v}_2) = 0$ . Subsection 5.2.9 examines the number of queries required to statistically identify the correct features based on a set of questionnaire. Here we introduce a parameter  $\theta$  into the choice decision that represents the subject’s faulty choice rate. More specifically, following the choice model in Section 2.1, the probability of Design 1 being preferred over Design 2 is  $1 / (1 + \exp(-\theta \mathbf{w}^T(\mathbf{v}_1 - \mathbf{v}_2)))$ , i.e., a smaller  $\theta$  represents higher chances of faulty choices. In all tests, the test function values on the discretized space are normalized to have zero means and standard errors so that  $\theta$  will have equal effects on these test functions.

### 5.2.6 D-optimal design setup

Recall that D-optimal design requires to find such  $\mathbf{Z}$  that the determinant of the information matrix  $\mathbf{Z}^T \mathbf{Z}$  is maximized. The physical meaning of maximizing this determinant can be considered as to make the samples uncorrelated (orthogonal) and having large variances. However, finding such optimal designs is not a convex problem and heuristics have been proposed to achieve near-optimal designs. For example, see approximation theory (*Harman and Trnovska (2009)*). In this study, we produce near-optimal designs simply by enumerating over a large number of random  $\mathbf{Z}$  given the query sizes and pick the ones that have largest determinant. It shall be mentioned that near-optimal  $\mathbf{Z}$ s are produced separately for different query sizes,

i.e.,  $\mathbf{Z}$  with 10 queries is not a subset of that with 20 or 50 queries. One difficulty we encounter is that while the minimum query size is 10, the dimensionality of  $\mathbf{Z}$  when using Gaussian bases is 25, leading to undetermined information matrices and zero determinants under any design. However, we noticed that the rank of the entire candidate set is much lower than its dimensionality, i.e., all candidate  $\mathbf{z}$  lie in a small subspace of the feature space. Therefore the dimensionality of all candidates can be reduced by projecting them using the eigenvectors of the few largest eigenvalues. The largest 5 eigenvalues are used in the following test using D-optimal designs with Gaussian bases. In the case of polynomial bases, since the number of basis is 9 and is lower than the minimum query size, dimension reduction is not necessary.

### 5.2.7 Performance of active learning, $D$ -optimal and random designs on 2D problems

We test the algorithms in scenarios with known and unknown functional forms of the preference functions. Both the linear and the polynomial preferences (Fn. 1 and 2) are tested under the premise that the functional form is known, in which case we directly use the known features in the active learning process. For example,  $x^2$ ,  $xy$  and  $y^3$  are used as features rather than the design variables during the test on the polynomial preference. Under the realistic condition that functional forms are unknown, we test the algorithms with both full polynomial bases of order 3 and Gaussian bases. Training-wise, the analytic estimator proposed by [Abernethy et al. \(2008\)](#) is used during the querying. The more accurate SVM training is applied when all data are collected. The error data from all test settings (algorithm  $\times$  function  $\times$  scenario  $\times$  number of query) are reported in [Table 5.1](#).

The best performers among the three algorithms are highlighted under each test setting. Several important observations from the tests are as follows:

- Regardless of the scenario (known or unknown functional form of the prefer-

Table 5.1: Generalization errors of active learning,  $D$ -optimal and random  
(Lower values are better)

#query = 10	Active		$D$ -optimal		Active		$D$ -optimal			
	Random	Known	Random	Known	Random	Unknown, polynomial basis	Random	Unknown, Gaussian basis		
Fn. 1	<b>0</b>	0.120(0.041)	<b>0</b>	<b>0</b>	<b>0.087</b>	0.3657(0.124)	0.1	0.143	0.312(0.107)	<b>0.113</b>
2	<b>0.03</b>	0.128(0.067)	0.057		<b>0.130</b>	0.366(0.110)	0.140	<b>0.143</b>	0.322(0.097)	0.323
3	n/a	n/a	n/a		0.370	0.361(0.118)	<b>0.207</b>	0.333	0.333(0.102)	<b>0.233</b>
4	n/a	n/a	n/a		<b>0.220</b>	0.379(0.104)	0.240	<b>0.227</b>	0.329(0.089)	0.430
#query = 20										
1	<b>0</b>	0.001(0.013)	<b>0</b>		<b>0</b>	0.195(0.052)	0.160	<b>0.093</b>	0.272(0.071)	0.107
2	<b>0</b>	0.093(0.040)	0.050		<b>0.107</b>	0.188(0.057)	0.177	<b>0.057</b>	0.279(0.069)	0.183
3	n/a	n/a	n/a		<b>0.177</b>	0.200(0.060)	0.247	<b>0.21</b>	0.236(0.056)	0.267
4	n/a	n/a	n/a		<u>0.273</u>	0.249(0.046)	<b>0.173</b>	<b>0.143</b>	0.279(0.068)	0.203
#query = 50										
1	<b>0</b>	0.101(0.002)	<b>0</b>		<b>0</b>	0.124(0.017)	0.100	<b>0.033</b>	0.181(0.027)	0.117
2	<b>0</b>	0.069(0.021)	<b>0</b>		<b>0</b>	0.115(0.025)	0.147	<b>0.013</b>	0.157(0.026)	0.147
3	n/a	n/a	n/a		<b>0.050</b>	0.126(0.025)	0.157	<b>0.077</b>	0.138(0.026)	0.260
4	n/a	n/a	n/a		<b>0.127</b>	0.187(0.021)	0.200	<b>0.043</b>	0.159(0.027)	0.180



ence), the active learning and  $D$ -optimal methods are better than random in most cases especially with large number of iterations (queries).

- In most cases, the error rate for active learning constantly goes down with more queries. This is not achieved by  $D$ -optimal design since with a different iteration number we have different sets of pairs ( $\mathbf{z}$ ) that maximize the determinant of the information matrix. Such a set may not perform better than a set with fewer queries on a specific problem. In the active learning case on the Camelback function with 20 iterations (the underlined cell in the table), the performance is worse than that with 10 iterations. This happens due to the choice of  $C$ , which balances the complexity control and the training error. As a matter of fact, tuning  $C$  to 1 from the original  $10^6$  will in this case result in an improved error rate of 0.173. Further research on how the balancing parameter should be tuned in the training is needed.

In summary, both active learning and  $D$ -optimal design outperform random in most cases. The difference between the two will be investigated in Subsection 5.2.10 with the existence of subject faulty choices.

### 5.2.8 Limitation of the linearity assumption

The limitation of the linearity assumption is rarely brought up in non-adaptive questionnaires since it only provides estimates after the entire data collection is done. Such a limitation results in poor training performance during active learning on non-linear preferences, which is an indicator of incorrect features at the early stages of an interaction. Table 5.2 shows the generalization error under the linearity assumption for the various nonlinear functions can be even worse than flipping a coin.

Table 5.2: Generalization errors under the linearity assumption

#query = 10	Active	$D$ -optimal
2	0.337	0.293
3	0.457	0.367
4	0.530	0.940
#query = 50		
2	0.337	0.337
3	0.367	0.360
4	0.940	0.940

### 5.2.9 Identification of preference features

We acknowledge that keeping the entire basis set as features in learning is cumbersome, especially with the fact that when the right features are used, convergence of the estimation is much improved for the same query size, as indicated in Table 5.1. Adding to that, keeping the entire basis set causes higher computational cost and simply introduces irrelevant features for estimation. Here we propose to identify features after pretests, under the assumption of no subject heterogeneity, i.e.,  $\mathbf{w}$  is a deterministic vector. The idea is to understand the importance of the bases by analyzing the statistics of estimators derived from a set of questionnaires. In the following simulated test, we conduct 1000 questionnaires and test the following null hypothesis on each element of  $\hat{\mathbf{w}}$ ,

$$H_0 : \hat{w}_i = 0. \tag{5.30}$$

Since the mean of  $\hat{\mathbf{w}}$  asymptotically equals the true parameter  $\mathbf{w}$ , if the null hypothesis is accepted then it is equally safe to accept  $w_i = 0$ . Regarding the setup, we test the linear function from Equation (5.27) and the polynomial function from Equation (5.28). Active learning is applied to each test to get the estimators. Table 5.3 reports the  $p$ -values of the hypothesis tests for each parameter under different number of iterations (queries). For tests on the linear preference,  $p$ -values on the first two

Table 5.3:  $p$ -values of  $H_0 : \hat{w}_i = 0$  for estimators on linear and polynomial preferences (Highlighted are significantly lower  $p$ -values)

#query	$p_{w_1}$	$p_{w_2}$	$p_{w_3}$	$p_{w_4}$	$p_{w_5}$	$p_{w_6}$	$p_{w_7}$	$p_{w_8}$	$p_{w_9}$
Test on linear function, Low faulty choice rate ( $\theta = 10$ )									
50	<b>0.001</b>	<b>0.001</b>	0.907	0.875	0.921	0.197	0.134	0.121	0.208
20	<b>0.001</b>	<b>0.000</b>	0.800	0.767	0.863	0.716	0.620	0.321	0.508
10	<b>0.055</b>	<b>0.000</b>	0.832	<b>0.082</b>	0.786	0.749	0.715	<b>0.081</b>	<b>0.091</b>
Test on linear function, High faulty choice rate ( $\theta = 2$ )									
50	<b>0.282</b>	<b>0.339</b>	0.999	0.833	0.932	0.735	0.714	0.821	0.632
20	<b>0.294</b>	<b>0.253</b>	0.945	0.893	0.953	0.914	0.735	0.778	0.801
10	<b>0.368</b>	<b>0.282</b>	0.893	0.887	0.990	0.889	0.828	0.697	0.602
Test on polynomial function, Low faulty choice rate ( $\theta = 10$ )									
50	0.963	0.872	<b>0.125</b>	<b>0.087</b>	0.837	0.860	0.898	0.917	<b>0.084</b>
20	0.997	0.799	<b>0.049</b>	<b>0.105</b>	0.931	0.908	0.824	0.957	<b>0.084</b>
10	0.771	0.702	<b>0.035</b>	<b>0.123</b>	0.861	0.530	0.526	0.793	<b>0.090</b>
Test on polynomial function, High faulty choice rate ( $\theta = 2$ )									
50	0.963	0.946	<b>0.437</b>	<b>0.487</b>	0.820	0.929	0.888	0.960	<b>0.374</b>
20	0.874	0.749	<b>0.374</b>	<b>0.513</b>	0.818	0.825	0.862	0.942	0.685
10	0.960	0.946	0.570	0.665	0.849	0.891	0.972	0.753	<b>0.431</b>

elements of the estimator (representing bases  $x_1$  and  $x_2$ ) should have value close to zero while others away from zero. With regard to the nonlinear preference, close-to-zero  $p$ -values should occur on the 3rd, 4th and last element (representing bases  $x^2$ ,  $x_1x_2$  and  $x_2^3$ ). The results in Table 5.3 show that having more queries in each test is beneficial for feature identification, and feature identification becomes more difficult when subjects are prone to faulty choices.

### 5.2.10 Robustness of active learning

Remember that we implemented two attempts to improve the robustness of active learning under subject faulty choices. One is to regularize the Hessian of the loss function when only limited observations are available; The other is to minimize maximum uncertainty, i.e., to diversify the queries. The first attempt aims to adjust the Hessian so that it is close to the true unknown information matrix and the second attempt tries to increase the determinant of the adjusted Hessian. Here we compare

Table 5.4: Robustness of active learning  
(Lower values are better)

		#query = 10		#query = 20		#query = 50	
		Low fault	High fault	Low	High	Low	High
Fn. 1	active	<b>0.092</b>	0.293	<b>0.004</b>	<b>0.197</b>	<b>0.000</b>	<b>0.018</b>
	D-opt.	0.114	<b>0.200</b>	0.160	0.223	0.101	0.255
Fn. 2	active	0.169	0.317	<b>0.144</b>	0.293	<b>0.017</b>	<b>0.128</b>
	D-opt.	<b>0.143</b>	<b>0.234</b>	0.171	<b>0.242</b>	0.142	0.320
Fn. 3	active	0.348	0.348	<b>0.219</b>	0.310	<b>0.083</b>	0.189
	D-opt.	<b>0.207</b>	<b>0.207</b>	0.250	<b>0.253</b>	0.157	<b>0.157</b>
Fn. 4	active	<b>0.223</b>	0.322	0.265	0.303	<b>0.134</b>	<b>0.185</b>
	D-opt.	0.241	<b>0.295</b>	<b>0.190</b>	<b>0.295</b>	0.327	0.387

the performance of active learning algorithm and D-optimal design with the existence of subject faulty choices to evaluate the robustness of the two. We conduct 1000 tests on the four test functions we used before. Subject faulty choice is modeled in the same way as in the previous test, with  $\theta = 2$  for high and  $\theta = 10$  for low fault rate. The test function values on the discretized space are normalized to have zero mean and standard error. Table 5.4 presents the performance comparison.

We highlighted every best performer among the three algorithms under a certain test function and a certain level of faulty choice. The finding here is consistent with the deterministic results from Subsection 5.2.7: D-optimal design has better performance than the adaptive algorithms when the query size is small; while the adaptive ones have an edge when a larger number of queries are made. Examine more closely, one will find that while the performance of D-optimal design is not correlated with the query size, that of the adaptive methods is almost always improved when the query size is raised. The rationale behind these observations is that when only limited query is available, D-optimal design ensures the diversity of queries and also prevents mislead queries generated by faulty choices. On the other hand, when we can afford a larger number of queries, active learning on the existing knowledge refines the estimation more effectively, while it is not guaranteed for a D-optimal design with more queries to work better than the ones with less queries. This is an important

finding that not only justifies the usage of D-optimal design in short questionnaires but also indicates that when more queries are allowed, one should switch to an adaptive algorithm to take its advantage.

### 5.3 Preference Identification

In preference identification, the subject is asked to identify the preferred designs from a set of choices and thus the binary outcome is interpreted as the definite preference for these choices, e.g., with a pair of Designs 1 and 2, the subject may assign 1 to Design 1 and 0 to Design 2, indicating that Design 1 is preferred independently of Design 2 and Design 2 is not-preferred independently of design 1. Without loss of generality, we query one design at a time. Let the preference model be

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{v}(\mathbf{x}) + \varepsilon, \quad (5.31)$$

where  $\varepsilon$  is a random error term not explained by  $\mathbf{x}$ . It is safe to assume that the subject prefers a design  $\mathbf{x}$  when  $f(\mathbf{x}) > 0$  and vice versa, i.e.  $y = 1 \Leftrightarrow f(\mathbf{x}) > 0$  and  $y = 0 \Leftrightarrow f(\mathbf{x}) < 0$ . The queries  $\mathbf{V}$  and the observations  $\mathbf{y}$  are then processed in a learning algorithm to derive the estimators  $\hat{\mathbf{w}}$ . The decision boundary is then approximated as  $\{\mathbf{x} \in \mathcal{D} | \hat{\mathbf{w}}^T \mathbf{v}(\mathbf{x}) = 0\}$ . Labeling the complete set of  $\mathbf{v}$  will provide a close approximation of the true parameters and the true decision boundary, but is costly considering the enormous amount of candidates. Therefore this section investigates how active learning can be applied to preference identification.

#### 5.3.1 Justification of the loss function

We show in this subsection that the preference identification problem is mathematically equivalent to the estimation problem. To do this, we derive the loss function for training in preference identification. Let the data at hand be  $\mathbf{x}_i$  and  $y_i$  for  $i = 1$

to  $n$ . Assuming that  $\varepsilon_i$  for  $i = 1$  to  $n$  are i.i.d. normal with zero mean and standard deviation of  $\sigma_\varepsilon$ , the probability of  $y_i = 1$  conditioned on  $\mathbf{x}_i$  and  $\mathbf{w}$  is

$$\begin{aligned}
P(y_i = 1 | \mathbf{w}, \mathbf{x}_i) &= P(\mathbf{w}^T \mathbf{v}(\mathbf{x}_i) + \varepsilon_i > 0) \\
&= P(\varepsilon_i > -\mathbf{w}^T \mathbf{v}(\mathbf{x}_i)) \\
&= \int_{-\mathbf{w}^T \mathbf{v}(\mathbf{x}_i)}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_\varepsilon} \exp\left(-\left(\frac{\tau^2}{2\sigma_\varepsilon^2}\right)\right) d\tau \\
&= \frac{1}{2} \left(1 - \operatorname{erf}\left(-\frac{\mathbf{w}^T \mathbf{v}(\mathbf{x}_i)}{\sqrt{2}\sigma_\varepsilon}\right)\right), \tag{5.32}
\end{aligned}$$

where  $\operatorname{erf}(x)$  is the error function

$$\operatorname{erf}(x) = \frac{\pi}{\sqrt{2}} \int_0^x \exp(-\tau^2) d\tau. \tag{5.33}$$

Therefore the negative log-likelihood of the observations is

$$\begin{aligned}
L_{ML}(\mathbf{w}) &= - \sum_{i: y_i=1} \log(P(y_i = 1 | \mathbf{w}, \mathbf{x}_i)) - \sum_{j: y_j=-1} \log(P(y_j = -1 | \mathbf{w}, \mathbf{x}_j)) \\
&= \sum_{i=1}^n \log\left(\frac{1}{2} \left(1 - \operatorname{erf}\left(-\frac{y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i)}{\sqrt{2}\sigma_\varepsilon}\right)\right)\right)^{-1}. \tag{5.34}
\end{aligned}$$

The maximum likelihood estimator  $\hat{\mathbf{w}}$  minimizing the loss  $L_{ML}$ . We can also consider the problem from a machine learning perspective. For the same data set, we define a loss function as

$$L_{SL}(\mathbf{w}) = \sum_{i=1}^n \max\{0, h - y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i)\}^2. \tag{5.35}$$

This loss function essentially penalizes violations of the inequality constraints

$$y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i) \geq h, \tag{5.36}$$

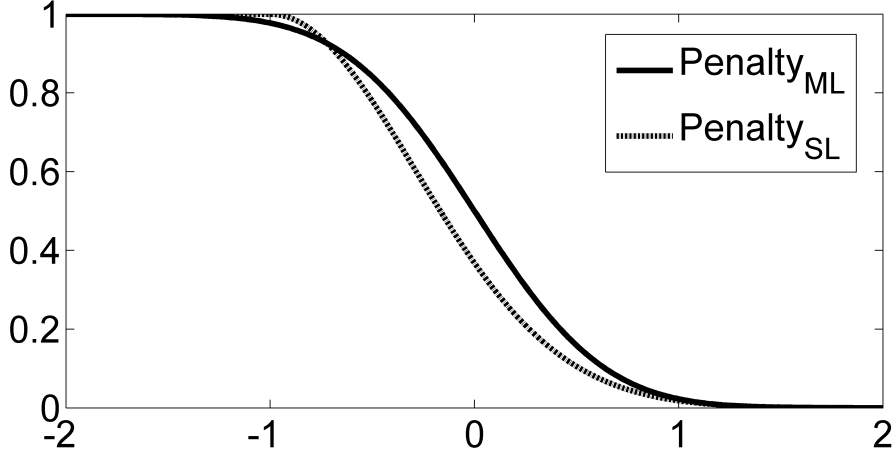


Figure 5.3:  $\text{Penalty}_{ML}$  and  $\text{Penalty}_{SL}$  in the range  $u \in [-2, 2]$  for preference identification;  $\sigma_\varepsilon = 1$ ,  $h = 1$ .

where  $h > 0$  controls the magnitude of the norm of  $\mathbf{w}$ . Using summation of logarithmic penalties, Equation (5.35) can be rewritten as

$$L_{SL}(\mathbf{w}) = \sum_{i=1}^n \log \exp(\max\{0, h - y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i)\}^2). \quad (5.37)$$

Following the previous section, we compare the penalties used in maximum likelihood estimation (Equation (5.38)) and in machine learning (Equation (5.39)) in the case of  $y = 1$  and with  $u = -\mathbf{w}^T \mathbf{z}$ . The penalties are inverted to be bounded within 0 and 1. Figure 5.2 shows the similarity between these two penalties.

$$\text{Penalty}_{ML}(u)^{-1} \triangleq \frac{1}{2} \left( 1 - \text{erf}\left(\frac{u}{\sqrt{2}\sigma_\varepsilon}\right) \right) \quad (5.38)$$

$$\text{Penalty}_{SL}(u)^{-1} \triangleq \exp(-\max\{0, h + u\}^2) \quad (5.39)$$

It should be noted that for  $\text{Penalty}_{ML}$ , the shape within the transitional region ( $[-1, 1]$  in Figure 5.3) is controlled by the assumptions on  $\varepsilon$ , i.e., the mean of  $\varepsilon$  controls the position of the transitional region on  $u$  while the standard deviation controls the curvature. Similarly for  $\text{Penalty}_{SL}$ , the position of the transitional region

is controlled by  $h$  and the curvature by the weight (or power) on the penalty term  $\max\{0, h + u\}$ .

Both  $L_{ML}$  and  $L_{SL}$  are convex in  $\mathbf{w}$  and therefore have their unique globally optimal solutions. In addition, we may also include complexity control in this problem, following [Toubia et al. \(2007a\)](#). For maximum likelihood estimation, let the prior knowledge be that  $\mathbf{w}$  follows a normal distribution with zero mean and covariance of  $\Sigma_{\mathbf{w}}$ . Thus the probability of observing  $y_i = 1$  will be

$$\begin{aligned}
P(y_i = 1 | \mathbf{w}, \mathbf{x}_i) &= \frac{P(\mathbf{w}^T \mathbf{v}(\mathbf{x}_i) + \varepsilon_i > 0)P(\mathbf{w})}{\int P(\mathbf{w}^T \mathbf{v}(\mathbf{x}_i) + \varepsilon_i > 0)P(\mathbf{w})d\mathbf{w}} \\
&\propto P(\mathbf{w}^T \mathbf{v}(\mathbf{x}_i) + \varepsilon_i > 0)P(\mathbf{w}) \\
&\propto \frac{1}{2} \left( 1 - \operatorname{erf}\left(-\frac{\mathbf{w}^T \mathbf{v}(\mathbf{x}_i)}{\sqrt{2}\sigma_\varepsilon}\right) \right) \exp\left(-\frac{1}{2}\mathbf{w}^T \Sigma_{\mathbf{w}}^{-1} \mathbf{w}\right). \quad (5.40)
\end{aligned}$$

When  $\mathbf{w}$  are i.i.d. with standard deviation  $\sigma_{\mathbf{w}}$ , we have

$$L_{ML}(\mathbf{w}) = - \sum_{i=1}^n \left( \frac{1}{2} \left( 1 - \operatorname{erf}\left(-\frac{y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i)}{\sqrt{2}\sigma_\varepsilon}\right) \right) \right) + \frac{1}{2\sigma_{\mathbf{w}}^2} \mathbf{w}^T \mathbf{w}. \quad (5.41)$$

Similarly for machine learning with regularization, we also have

$$L_{SL}(\mathbf{w}) = \sum_{i=1}^n \left( \max\{0, h - y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i)\}^2 \right) + C \mathbf{w}^T \mathbf{w}. \quad (5.42)$$

Notice that this is exactly the same loss function as in preference estimation, with the features  $\mathbf{v}$  replacing the feature difference  $\mathbf{z}$ . Therefore the identification and estimation problem are mathematically equivalent.

### 5.3.2 Application of active learning

In the previous section, we followed two rules to adaptively learn preference: 1) utility balance and 2) minimize maximum uncertainty. Let us now validate these two rules for preference identification. The practical meaning of utility balance can be



instantly seen in the current problem setting: A design  $\mathbf{x}$  has utility balance when it is on the decision boundary  $\{\mathbf{x} \in \mathcal{D} | \mathbf{w}^T \mathbf{v}(\mathbf{x}) = 0\}$ , which means that the probabilities of this query being labelled as 1 or  $-1$  are equally 0.5 (one can verify this by inserting  $\mathbf{w}^T \mathbf{v}(\mathbf{x}) = 0$  into Equation (5.32)).

Recall that minimizing maximum uncertainty is equivalent to minimizing the volume of the variance ellipsoid of  $\hat{\mathbf{w}}$ . Below we derive the form of  $\text{cov}(\hat{\mathbf{w}})$  and as we will see, maximizing the information matrix  $\mathbf{V}^T \mathbf{V}$  will be an approximation of minimizing the variance of  $\hat{\mathbf{w}}$  for both  $L_{ML}$  and  $L_{SL}$ . First, for  $L_{ML}$  we have

$$\frac{\partial L_{ML}}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\exp(-u_i^2/(2\sigma_\varepsilon^2)) y_i \mathbf{v}(\mathbf{x}_i)}{\sqrt{2}\sigma_\varepsilon \int_{u_i/\sigma_\varepsilon}^{\infty} \exp(-\tau^2) d\tau}, \quad (5.43)$$

and

$$\frac{\partial^2 L_{ML}}{\partial \mathbf{w} \partial \mathbf{w}^T} = \sum_{i=1}^n \frac{\exp(-u_i^2/(2\sigma_\varepsilon^2))}{2\sigma_\varepsilon^2 \int_{u_i/\sigma_\varepsilon}^{\infty} \exp(-\tau^2) d\tau} \left( \frac{\exp(-u_i^2/(2\sigma_\varepsilon^2))}{\int_{u_i/\sigma_\varepsilon}^{\infty} \exp(-\tau^2) d\tau} - \frac{\sqrt{2}u_i}{\sigma_\varepsilon} \right) \mathbf{v}(\mathbf{x}_i) \mathbf{v}(\mathbf{x}_i)^T, \quad (5.44)$$

where  $u_i = -y_i \mathbf{w}^T \mathbf{v}(\mathbf{x}_i)$ . If we approximate  $\hat{\mathbf{w}}$  with one Newton step with the initial guess being  $\hat{\mathbf{w}}_0 = \mathbf{0}$ , we will arrive at

$$\begin{aligned} \hat{\mathbf{w}} &= -\left(\frac{\partial^2 L_{ML}}{\partial \mathbf{w} \partial \mathbf{w}^T}\right)^{-1} \frac{\partial L_{ML}}{\partial \mathbf{w}} \\ &= \sqrt{\frac{\pi}{2}} \sigma_\varepsilon (\mathbf{V}^T \mathbf{V})^{-1} \mathbf{V}^T \mathbf{y}. \end{aligned} \quad (5.45)$$

To formulate the covariance of  $\hat{\mathbf{w}}$ , first notice that  $y_i$  is binary with probability  $P(y_i = 1) = p_i$  and  $P(y_i = -1) = 1 - p_i$ , where  $p_i$  is determined by  $\mathbf{w}$ ,  $\mathbf{x}_i$  and the distribution of  $\varepsilon_i$ . With the assumption that  $\varepsilon_i$  for  $i = 1$  to  $n$  are i.i.d., it is easy to show that  $y_i$ s are uncorrelated and thus  $\text{cov}(\mathbf{y})$  is diagonal. To be more specific, we have  $\text{cov}(\mathbf{y}) = \text{diag}(4p_1(1 - p_1), \dots, 4p_n(1 - p_n))$ . Therefore, the covariance of  $\hat{\mathbf{w}}$  is

$$\text{cov}(\hat{\mathbf{w}}) = \frac{\pi \sigma_\varepsilon^2}{2} (\mathbf{V}^T \mathbf{V})^{-1} \mathbf{V}^T \text{cov}(\mathbf{y}) \mathbf{V} (\mathbf{V}^T \mathbf{V})^{-1}. \quad (5.46)$$

Treating  $\text{cov}(\mathbf{y})$  with our null knowledge, i.e.,  $p_i = 0.5$  for all  $i$ , we come to

$$\text{cov}(\hat{\mathbf{w}}) = \frac{\pi\sigma_\varepsilon^2}{2} (\mathbf{V}^T\mathbf{V})^{-1}. \quad (5.47)$$

Therefore, at any level of noise  $\sigma_\varepsilon$  and before any observation is made, one shall design the queries  $\mathbf{V}$  to have large determinant so that the covariance of the resulting estimator will be small. Readers shall also find this conclusion similar to the discussion in Section 2.1.

We can now derive the same conclusion for  $L_{SL}$ . Since  $L_{SL}$  is not differentiable everywhere, the gradient and Hessian only exist when  $\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) \neq h$ :

$$\frac{\partial L_{SL}}{\partial \mathbf{w}} = \sum_{i=1}^n \mathbf{1}_{\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) < h} - (1 - y_i\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i))y_i\mathbf{v}(\mathbf{x}_i), \quad (5.48)$$

$$\frac{\partial^2 L_{SL}}{\partial \mathbf{w}\partial \mathbf{w}^T} = \sum_{i=1}^n \mathbf{1}_{\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) < h} \mathbf{v}(\mathbf{x}_i)\mathbf{v}(\mathbf{x}_i)^T, \quad (5.49)$$

where  $\mathbf{1}_{\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) < h}$  is an indicator function which has ones when  $\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) < h$  and zeros otherwise in  $\mathcal{D}$ . Taking one Newton step from  $\hat{\mathbf{w}}_0 = \mathbf{0}$ , we have

$$\hat{\mathbf{w}} = \left( \sum_{i=1}^n \mathbf{1}_{\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) < h} \mathbf{v}(\mathbf{x}_i)\mathbf{v}(\mathbf{x}_i)^T \right)^{-1} \sum_{i=1}^n \mathbf{1}_{\hat{\mathbf{w}}^T\mathbf{v}(\mathbf{x}_i) < h} y_i \mathbf{v}(\mathbf{x}_i). \quad (5.50)$$

Notice that Equation (5.50) is similar to Equation (5.45) with a few rows in  $\mathbf{V}$  missing. Therefore, maximizing the determinant of the information matrix can be used with both estimation formulations during active learning.

It is important to mention that the inverse of  $\mathbf{V}^T\mathbf{V}$  usually does not exist due to the high dimensionality of features and the limited number of queries, in which case Equation (5.41) and Equation (5.42) are used for training and the regularized information matrix  $\mathbf{H} = C\mathbf{I} + \mathbf{V}^T\mathbf{V}$  is used for choosing the next query, where  $\mathbf{I}$  is an identity matrix.

### 5.3.3 Active learning on preference identification

Following the above discussion, the strategy for identification is almost the same as that for estimation:

1. At iteration 0, an initial set  $\mathbf{V}_0$  is generated and labelled as  $\mathbf{y}_0$ ;  $\hat{\mathbf{w}}_0$  is trained using either maximum likelihood or machine learning.
2. At iteration  $s$ ,
  - (a) Regularize the Hessian matrix to be  $\mathbf{H} = C_s \mathbf{I} + \mathbf{V}_s^T \mathbf{V}_s$ , where  $C_s$  is set to be  $1/s$  so that the impact of regularization decreases when more observations are available.
  - (b) Denote the plane perpendicular to  $\hat{\mathbf{w}}_{s-1}$  as  $\mathbf{P}_s = \mathbf{I} - \hat{\mathbf{w}}_{s-1} \hat{\mathbf{w}}_{s-1}^T / \|\hat{\mathbf{w}}_{s-1}\|^2$ . Project the Hessian to  $\mathbf{P}_s$ :  $\mathbf{Q}_s = \mathbf{P}_s \mathbf{H}_s$ .
  - (c) Find the eigenvector of the second minimum eigenvalue of  $\mathbf{Q}_s$ , and denote it as  $\mathbf{v}_0$ .
  - (d) Find  $\mathbf{v}_{s+1}$  that maximizes the similarity:  $|\mathbf{v}_0^T \mathbf{v}_{s+1}| - |\hat{\mathbf{w}}_{s-1}^T \mathbf{v}_{s+1}|$ .
3. Make a query with  $\mathbf{v}_{s+1}$  to get  $y_{s+1}$ ; train with the new data and go back to 2 until maximum iteration number is reached.

### 5.3.4 Simulated tests and results

To start with, we show how the active learning algorithm works in a simple preference identification problem with two dimensions, five levels on each dimension, and with the polynomial preference function defined in Equation (5.28). Gaussian bases are used as features in this demonstration. Figure 5.4 illustrates the identification process of finding the decision boundary  $f(\mathbf{x}) = 0$  on this function. In this figure, each block on the 2D space represents a design. The black blocks represent designs such that  $f(\mathbf{x}) < 0$  and the white ones  $f(\mathbf{x}) > 0$ . The circled blocks are designs with

$y = -1$  and dotted ones  $y = 1$ . The true boundary under discretization is shown in the last subfigure. The corresponding testing errors at each iteration are listed under each sub-figure. These errors are defined as rates of misclassified designs over the total number of designs.

We now compare performance of the proposed algorithm against D-optimal design on 2D, 3D and 5D tests. All test functions are represented by weighted sums of Gaussian bases where one of the basis parameter is set to a higher value compared to the others to model a spherical preferred set in the corresponding space. We call the vertex associated with the high value parameter the preference center, e.g., with 2 dimensions and 10 levels on each dimension, a preference center at [5,4] represents the vertex at the 5th level on the first dimension and at the 4th on the second. We enable subject faulty choices using the choice probability model in Equation (5.32) with  $\sigma_\varepsilon = \sqrt{2}/4$  (low faulty choice rate) and  $\sigma_\varepsilon = 1$  (high faulty choice rate). With each test setup, we conduct 100 tests and  $\#query$  amount of queries. Table 5.5 summarizes test settings. The performances are reported in Table 5.6 as the means of generalization errors and their standard errors in the bracket. The better performance in each test setting is highlighted. This results show that the active learning algorithm provides better generalization errors than D-optimal design when the dimensionality of the design space is low or only a unique preference center exists. D-optimal, on the other hand, is shown to be more useful when multiple preference centers exist in high dimensional space. The similar result is also reported in *Osugi et al. (2005)* and *Baram et al. (2004)* as an inevitable tradeoff between exploitation and exploration. The strategy we take here is combining utility balance (exploitation) and minimization of maximum uncertainty (exploration) together as a unified criterion. However, how these two considerations should be weighted in the query process is yet to be explored.

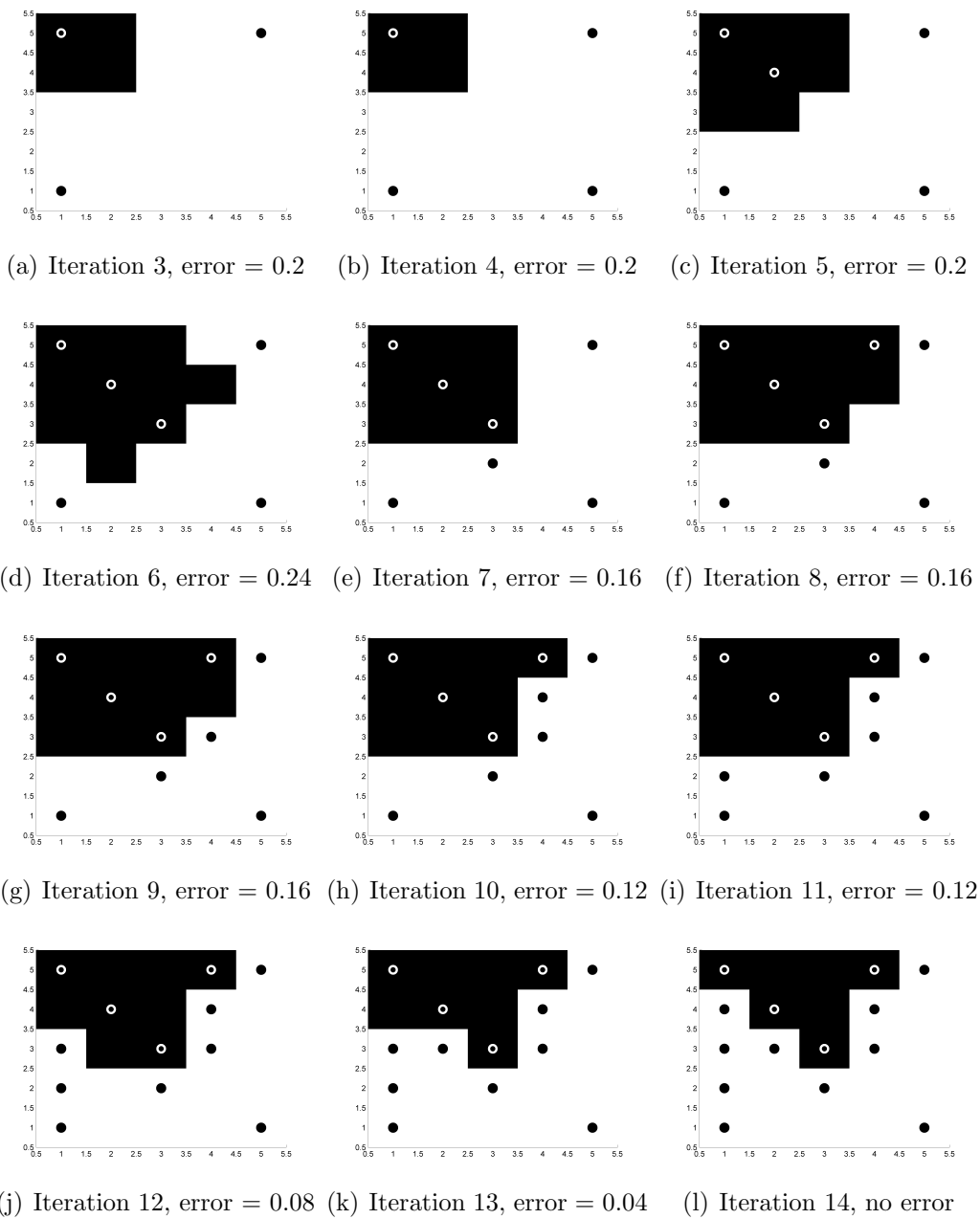


Figure 5.4: Demonstration of active learning for preference identification. Each block on the 2D space represents a design. The black blocks represent designs such that  $f(\mathbf{x}) < 0$  and the white ones represent  $f(\mathbf{x}) > 0$ . The circled blocks are designs with  $y = -1$  and dotted ones with  $y = 1$ .

Table 5.5: Preference identification test settings

test ID	#dim	#level	#parameter	preference center	#query
1	2	10	100	[5,5]	20
2	2	10	100	[1,1][5,5][10,10]	20
3	5	3	243	[2,2,2,2,2]	50
4	5	3	243	[1,1,1,1,1][2,2,2,2,2][3,3,3,3,3]	50

Table 5.6: Preference identification test results

test ID	active learning error	D-optimal error
Low faulty choice rate ( $\sigma = \sqrt{2}/4$ )		
1	<b>0.000</b> (0.000)	0.050(0.021)
2	<b>0.070</b> (0.000)	0.078(0.015)
3	<b>0.066</b> (0.003)	0.108(0.018)
4	0.091(0.000)	<b>0.059</b> (0.017)
High faulty choice rate ( $\sigma = 1$ )		
1	<b>0.001</b> (0.003)	0.044(0.018)
2	<b>0.070</b> (0.000)	0.094(0.016)
3	<b>0.075</b> (0.018)	0.133(0.026)
4	0.089(0.003)	<b>0.071</b> (0.018)

## 5.4 Concluding Remarks

In this chapter we provided a thorough technical review of the active learning literature from both the economics and machine learning communities. The core concept of active learning is to select queries that achieve both high information gain (using utility balance) and high diversity of the queried samples (to minimize maximum uncertainty). We argue that feature identification requires more attention in active learning research and the challenge still remains as how to find a balance between two conflicting interests: The need to increase the feature size to handle nonlinearity in the preference and the need to limit the feature size for manageable parameter estimation. We proposed to use polynomial and Gaussian bases as features rather than using kernel tricks in preference estimation since the former approach achieves fewer violations to observations in training. We showed that the same bases can also be applied to preference identification to handle nonlinear decision boundaries.

The simulated tests for preference estimation showed that active learning has more robust performance in most linear and nonlinear tests when a large number of queries is available while D-optimal design has an edge under a limited query size. The simulated tests for preference identification showed that the active learning algorithm outperforms D-optimal design when a single preference center exists or the dimensionality of the space is low. The results from both studies are consistent: Active learning is more favorable when we have some knowledge about the model or the model is easy to estimate; D-optimal design, as a conservative option, should be used when little knowledge has been obtained or when we cannot afford a large number of queries.

## CHAPTER VI

# Collaborative Filtering in Preference Elicitation

The motivation of this chapter naturally follows the previous discussion on preference elicitation. We have investigated querying strategies for individuals, and strategies will be repeated for a large number of tests. Considering that preference from a population many well be clustered, would it be possible for the query algorithm to learn from previous successes and failures and improve itself over time? In this chapter, we focus on the preference elicitation problem and discuss a variety of heuristics that may help to improve the search algorithm. The problem is formulated in Section 6.1, followed by the main discussion on heuristics in Section 6.2. Section 6.3 evaluates the effectiveness of these heuristics with empirical tests. Section 6.4 discusses the difference between this work and other collaborative filtering literature. We conclude with Section 6.5.

### 6.1 Problem Formulation

We consider a black-box optimization problem with binary outputs. Without loss of generality, we focus on pairwise comparisons for each query, i.e., a query contains two designs and the output will be two binary digits representing the preference order of the pair. Slightly different from the settings in Chapter III and IV, here we consider a discrete design space and thus optimization of a merit function can



be substituted by enumeration over all valid candidates and the search algorithm becomes deterministic. Notation-wise, let the design space be  $\mathcal{D}$ , with  $p$  dimensions and  $l$  levels for each dimension. All candidate designs form the set  $\mathbf{X} \triangleq \{\mathbf{x}_i\}_{i=1}^N$ , where  $N = l^p$ . A query is a set of two designs  $\{\mathbf{x}_i, \mathbf{x}_j\}$ , where  $i, j \in \{1, \dots, N\}$ . A path for test  $t$ , denoted as  $I_t$ , is defined as a sequence of indices of the queried designs, i.e.,  $I_t$  is a permutation of a subset of  $\{1, \dots, N\}$ . Notice that the last element in  $I_t$  has the best preference value among all queried designs. Let the  $t$ th test preference function be  $f_t \triangleq f(\mathbf{x}, \mathbf{w}_t)$ , parameterized by  $\mathbf{w}_t$  following some unknown distribution. We are interested in finding a search algorithm  $A$  which evolves its strategy depending on previous search history:  $I_{t+1} = A(f_{t+1}, \{I_\tau\}_{\tau=1}^t)$  so that the average length of paths, or equivalently, the query size, goes down along  $t$ . We elaborate the development of some heuristics below.

## 6.2 Heuristics

We first show that the path set  $\{I_\tau\}_{\tau=1}^t$  on  $t$  tests actually forms a tree  $T_t$  if in all tests the search algorithm starts with the same pair. Here the root vertex represents the initial query and each following vertex represents a query generated from the previous user responses. The leaf vertices represent optimal solutions of these tests. To be specific, in the context of pairwise comparison, the root vertex contains the first pair of designs and it leads to two edges, either Design 1 better than Design 2 or the other way, when designs are always differentiated. Each following vertex contains the information table of what designs have been queried and which ones are preferred. The estimator  $\hat{\mathbf{w}}$  can be directly derived from this table. A path to a vertex is a sequence of vertices and edges from the root vertex to that vertex. Associated with each vertex is a set  $P$  of path lengths from previous searches that reached this vertex during a search. For example, a vertex with  $P = \{6, 8, 9\}$  says that three searches have arrived at this vertex after 6, 8 and 9 queries.

### 6.2.1 Simplification of a search tree (JUMP0)

We propose a straightforward heuristic as follows:

1. During search  $t$  and  $s$ th query, we compare the current estimator  $\hat{\mathbf{w}}_s$  to all recorded estimators from previous searches to see if the current search status already exists in the record.
2. If  $\hat{\mathbf{w}}_s$  exists, let the current vertex be  $n_s$  and the next vertex to query be  $n_{(s+1)}$ . Let  $n_{(s+\delta)}$  be the vertex following  $n_s$  that first leads to more than one vertices, then directly query  $n_{s+\delta}$  and remove  $n_{s+1}, \dots, n_{s+\delta-1}$  if they exist. Figure 6.1 illustrates this strategy.
3. If  $\hat{\mathbf{w}}_s$  does not exist, then update the tree with this new vertex and proceed using the default EGO Search algorithm.

The logic of this heuristic approach is easy to understand: The search algorithm is designed to balance exploration and exploitation so that it does not jump to a false conclusion too soon. However, with the observation  $T_t$ , it would be effective if we simplify the search by eliminating the unnecessary exploratory queries. The limitation of this heuristic is obvious: On one hand, with limited tests, jumping to conclusion can derive inefficient heuristics; on the other hand, with abundant tests, vertices that have single input and output edge can be rare, rendering the heuristic less attractive.

### 6.2.2 Exploration of more efficient trees (JUMP1)

Still working with the search tree and denoting all possible leaf vertices (optimal solutions) be  $\{n_i^*\}_{i=1}^r$ , we now propose an algorithm that explores alternative paths from  $n_0$  to  $\{n_i^*\}_{i=1}^r$  that will not be generated by the initial search algorithm. The basic idea, as illustrated in Figure 6.2, is to jump to a leaf vertex after every  $d$

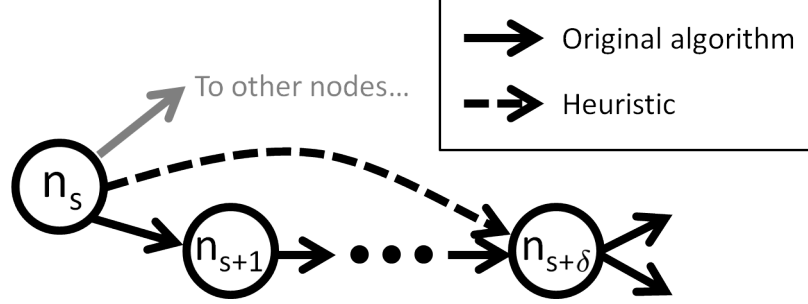


Figure 6.1: We jump from vertex  $n_s$  to vertex  $n_{s+\delta}$  if there is no alternative path from  $n_s$  and through  $n_{s+1}$ .

normal queries to establish new paths from that leaf vertex to other leaf vertices. For clustered optimal solutions, it could be more efficient for the search algorithm to query a near-optimal design and then search locally from there, reducing the average of search path lengths of that cluster. Let the current vertex be  $n_s$  which has a set of leaf vertices accessible from it. We propose to pick a leaf vertex to jump to that has the highest average path length  $\bar{P}$  associated with the vertex. The intuition here is that  $\bar{P}_i$  represents the penalty one had by not using this leaf vertex  $n_i^*$  as the jump vertex in the previous tests. Therefore jumping to this vertex could potentially lead to better overall search performance. Using this heuristic, the average path length  $\bar{P}$  on each leaf vertex changes dynamically. This creates more opportunities for exploration. One can see that this algorithm is similar in concept to JUMP0 but has a more advantageous jump. We summarize this algorithm as follows:

1. During search  $t$  and some  $s$ th query such that  $\text{mod}(s, d_t) = 0$  where  $d_t$  is an algorithm parameter, evaluate and rank the similarity between the current  $\hat{\mathbf{w}}_{st}$  and all estimators from previous searches at the  $s$ :

$$\text{similarity}_\tau = \frac{\hat{\mathbf{w}}_{st}^T \hat{\mathbf{w}}_{s\tau}}{\|\hat{\mathbf{w}}_{st}\|_2 \|\hat{\mathbf{w}}_{s\tau}\|_2}. \quad (6.1)$$

2. Form a candidate set  $\mathbf{N}_s$  of all vertices from previous tests at the  $s$ th query

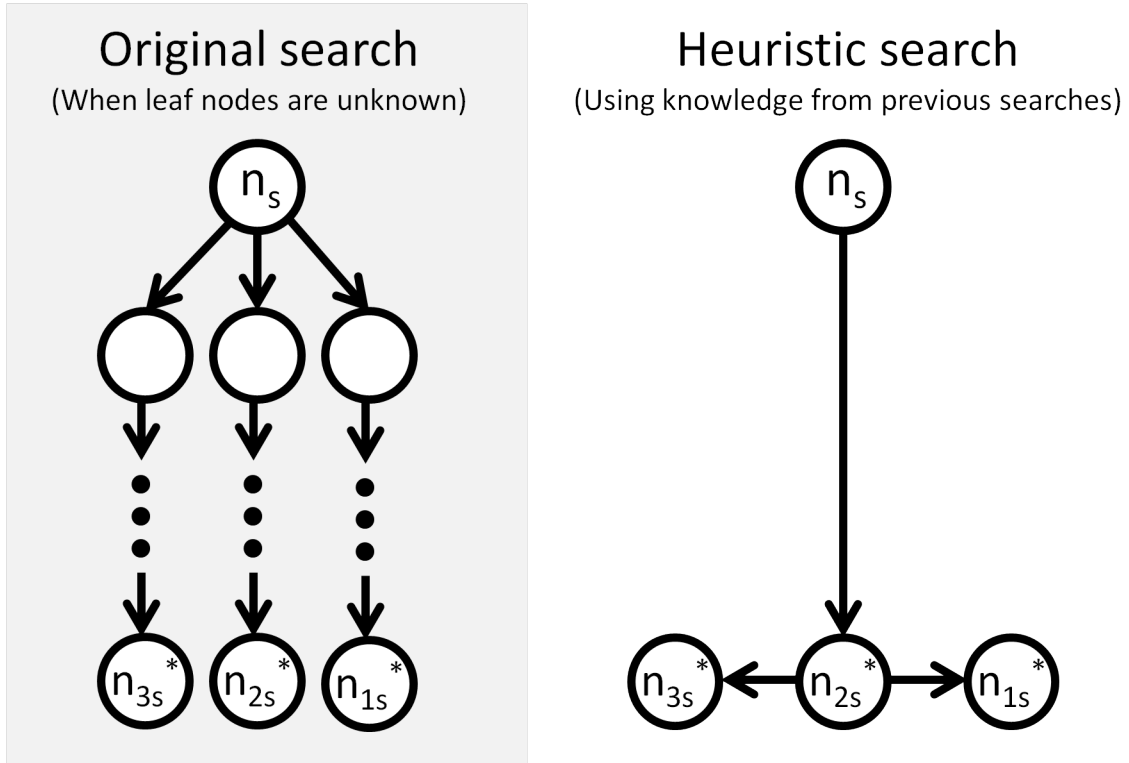


Figure 6.2: Concept of JUMP1: Jumping to a conclusion (leaf vertex) based on previous knowledge, and search from there.

that have the highest similarity value. Very likely, these vertices are identical to the current vertex if  $T_t$  is based on abundant tests.

3. Let  $\mathbf{N}_s^*$  be the set of unique leaf vertices that is accessible from  $\mathbf{N}_s$ . Let  $\mathbf{P}_s^*$  be the set of average path length from  $\mathbf{N}_s$  to  $\mathbf{N}_s^*$ . Choose the leaf vertex from  $\mathbf{N}_s^*$  that has the largest average path length from  $\mathbf{P}_s^*$  as the next query design (Recall that only one design is needed in a pairwise comparison scenario with the assumption that indifferent designs do not exist). See Figure 6.3 for an illustration.
4. Update  $T_t$  with the new paths.
5. If  $\text{mod}(s, d_t) \neq 0$ , proceed using the default EGO Search algorithm.

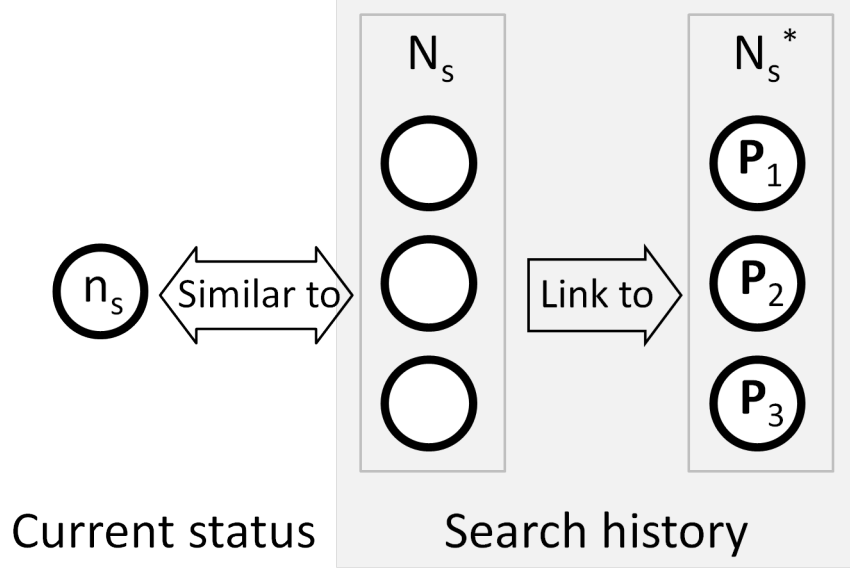


Figure 6.3: JUMP1 algorithm: From the current status  $n_s$ , find a set  $N_s$  from  $T_t$  that is most similar to  $n_s$ . Find the leaf vertex set  $N_s^*$  accessible from  $N_s$ . Pick the next query design as the one from  $N_s^*$  that has the highest average path length  $\bar{P}$ .

### 6.2.3 2D Demonstration of the exploration heuristic

For visualization purposes, we demonstrate the effect of JUMP1 on a 2D design space with 10 levels on each dimension. The designs associated with each vertex on  $\mathcal{D}$  are numbered following Figure 6.4. The test preference is parameterized by  $\mathbf{w}$  in the form:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{v}(\mathbf{x}), \quad (6.2)$$

where  $\mathbf{v}(\mathbf{x})$  are the Gaussian bases at  $\mathbf{x}$ :

$$v_i(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|_2^2). \quad (6.3)$$

Here  $\mathbf{w}$  has all zeros except one element, whose index is drawn uniformly from the set  $\{15, 24, 25, 33, 34, 35, 36, 44, 45, 55, 68, 77, 78, 86, 87, 88, 89, 97, 98\}$  and magnitude is 1. This setup creates preference functions that have unique optimal solutions distributed on the given set. Figure 6.4 shows the locations of the optimal solutions generated from this setup, mimicking the situation where two clusters of preferences exist.

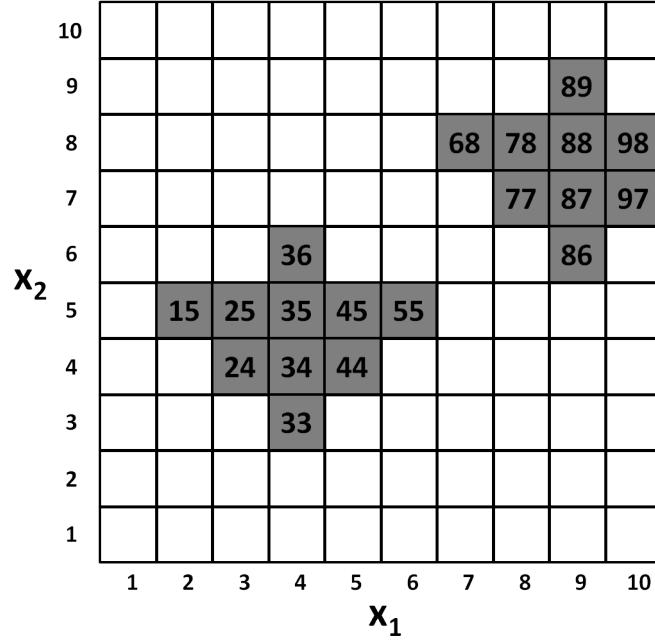


Figure 6.4: Setup of the 2D demonstration for JUMP1: The test optimal solutions are uniformly distributed on the shaded locations. The indices are of ascendant order starting from the left down corner as 1 and ending at the top right corner as 100.

We use an experiment to show that applying JUMP1 can successfully reduce the search cost over time. The experiment consists of a sequence of 1000 random tests generated from the above setup to mimic the preference elicitation tasks occurring along the time line. These tests are divided into 10 batches, where each contains 100 tests. The algorithm updates its strategy after every batch of tests is executed. After an initial batch using the original search algorithm, every following batch uses the heuristic search once every three queries, i.e.,  $d = 3$ . For each heuristic search step, it uses the accumulated knowledge from history, i.e., all history paths towards one leaf vertex are used for heuristic decision making. The initial query for each test is set as  $\{1, 100\}$ . Figure 6.5 compares the average search path lengths of each batch of tests with and without JUMP1. To see the effect of how search path lengths are shifted, we show the comparison in Figure 6.6 of the last 200 tests with and without using the knowledge generated from the previous experiment. It is clear that by using knowledge from previous searches, a new search can be done more efficiently.

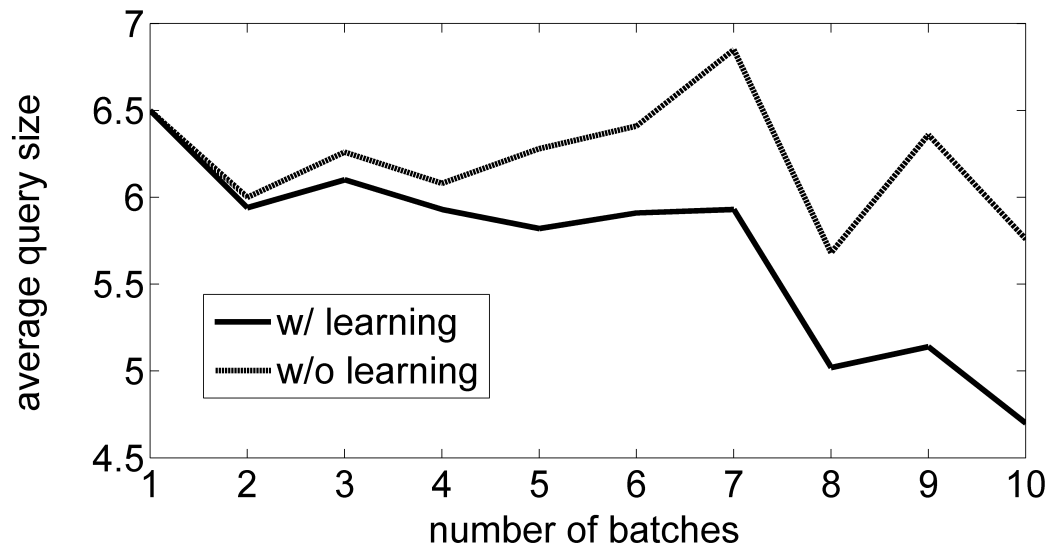


Figure 6.5: Average path lengths for 10 batches of random tests with and without JUMP1. This comparison shows that JUMP1 can effectively reduce the search cost with different  $d$ .

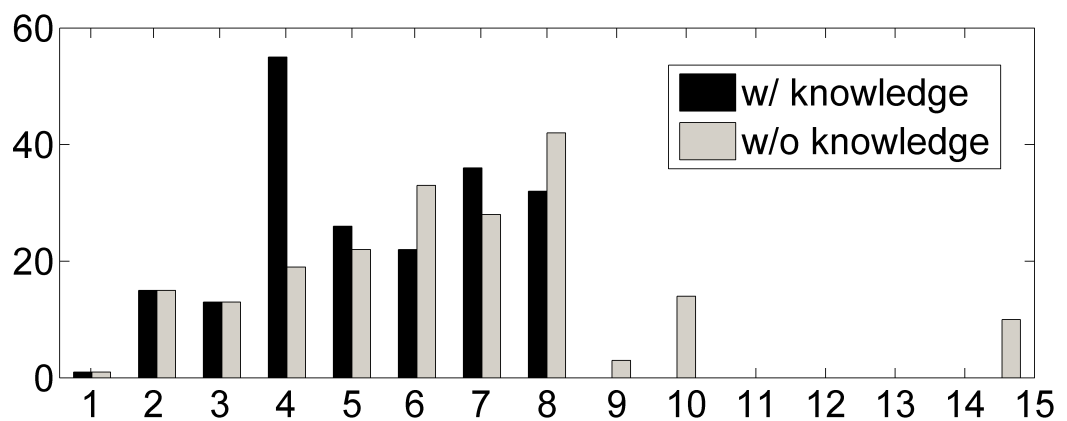


Figure 6.6: Histogram of path lengths from 200 random tests with and without the search tree generated from the experiment in Figure 6.5.

To better understand the behaviour of JUMP1, we show in Figure 6.7 the evolution of the search path for a specific preference optimized at  $x_{86}$  from the previous experiment result. This sequence of searches demonstrates how JUMP1 explores different possible leaf vertices and finds search paths more effective than the original one. The behaviour of JUMP1 is similar to JUMP0 when all vertices between the current vertex and the leaf vertex have one input and output.

One shall also notice that although a good search path is found during the experiment (at Steps 3 and 5), it is not guaranteed to be preserved in future tests due to the dynamic change of  $\mathbf{P}$  at each vertex, i.e., a search strategy maybe effective for one specific target. For example, in this case, going directly to  $\mathbf{x}_{86}$  is effective since the optimal solution is at  $\mathbf{x}_{86}$ , but the strategy may be not effective for other preferences. Therefore, the heuristic may change once  $\bar{\mathbf{P}}$ s on leaf vertices are updated upon the next batch of tests. We observe, however, that through batches of tests, the average query size (path length) over random tests from a given distribution of  $\mathbf{w}$  decreases by adopting JUMP1. For instance, Figure 6.8 shows the average query size of the two clusters (i.e., optimal solutions in  $\{\mathbf{x}_{15,24,25,33,34,35,36,44,45,55}\}$  and in  $\{\mathbf{x}_{68,77,78,86,87,88,89,97,98}\}$ ) from the experiment.

#### 6.2.4 Enhancement to JUMP1 (JUMP2)

Although JUMP1 achieves its purpose, we observed that it does not book-keep effective searches due to the dynamic change of  $\bar{P}$  at each leaf vertex. Addressing this problem leads to the following algorithm (JUMP2):

1. During search  $t$  and some  $s$ th query such that  $\text{mod}(s, d_t) = 0$  where  $d_t$  is an algorithm parameter, rank the similarity between the current  $\hat{\mathbf{w}}_{st}$  and all estimators from previous searches at the  $s$ . Same as JUMP1.
2. Form a candidate set  $\mathbf{N}_s$  of all vertices from previous tests at the  $s$ th query that have the highest similarity value. Same as JUMP1.



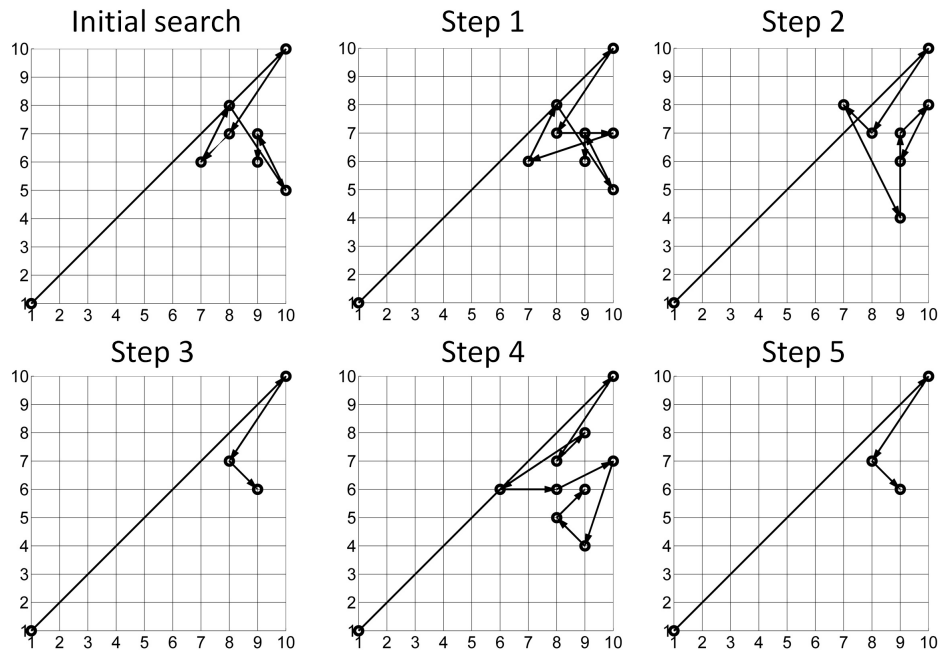


Figure 6.7: The evolution of search path for a preference function optimized at index 86. JUMP1 explores different paths and finds better ones than from the original search algorithm.

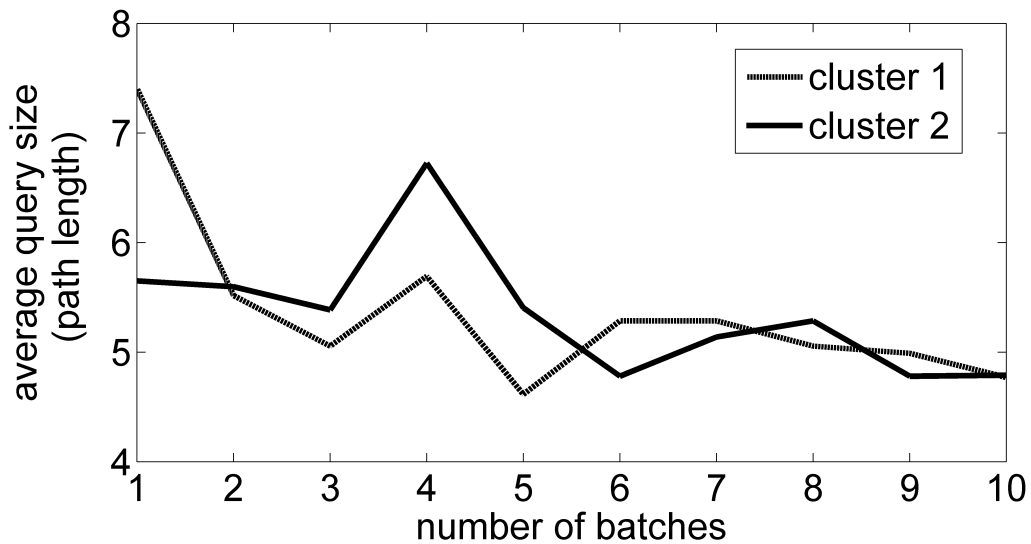


Figure 6.8: The average query size of the two clusters (optimal solutions at  $\{\mathbf{x}_{15,24,25,33,34,35,36,44,45,55}\}$  and in  $\{\mathbf{x}_{68,77,78,86,87,88,89,97,98}\}$ ) along number of batches of tests.

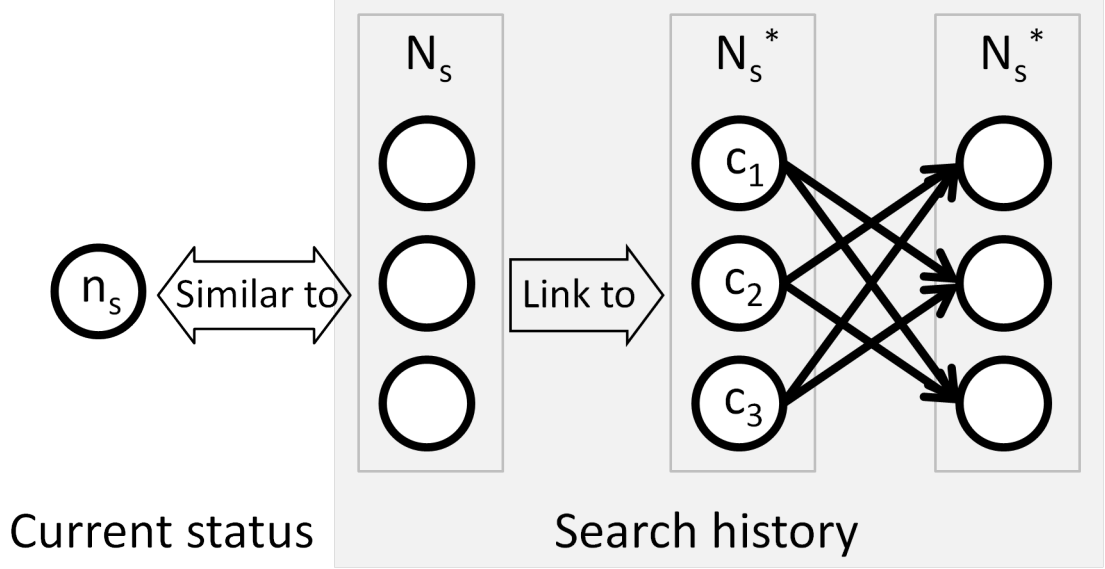


Figure 6.9: JUMP2 algorithm: From the current status  $n_s$ , find a set  $N_s$  from  $T_t$  that is most similar to  $n_s$ . Find the leaf vertex set  $N_s^*$  accessible from  $N_s$ . Pick the next query design as the one from  $N_s^*$  that has the lowest average path length to all other leaf vertices accessible from it.

3. Let  $N_s^*$  be the set of unique leaf vertices that is accessible from  $N_s$ . Denote the  $i$ th leaf vertex from  $N_s^*$  to be  $n_{is}^*$ . Let  $c_{jis}^*$  be the path length occurred when the optimal solution is some  $n_j$  and the search passes through  $n_{is}^*$ . Let  $\bar{c}_{is}^*$  be the average length of  $c_{jis}^*$  for all  $j$ . Choose the leaf vertices from  $N_s^*$  that have the smallest  $\bar{c}_{is}^*$  as the next query points. See Figure 6.9 for an illustration.
4. Update  $T_t$  with the new paths. Update  $\bar{c}_{is}^*$  if leaf vertex  $n_{is}^*$  is queried.
5. If  $\text{mod}(s, d_t) \neq 0$ , proceed using the default EGO Search algorithm.

The only difference from JUMP1 is the criterion of which vertex to jump to. Rather than querying the designs that have the largest average query size if they are not queried at iteration  $d$ , we instead query the ones that have the best overall performance explored so far. Notice that  $\bar{c}_{is}^*$  are zeros at initialization, therefore this algorithm will explore possible leaf vertices and gradually converge to the strategy with near-optimal performance.

### 6.2.5 2D Demonstration of the enhanced heuristic

We use the same experiment setup as before ( $d = 3$ ). To demonstrate the difference between JUMP1 and JUMP2 as well as to show how algorithm update frequency will affect both algorithms, we present in Figure 6.10 the performance of JUMP1 and JUMP2 under different update frequencies (1, 10 and 100 tests per update) over the same 1000 random tests. For visualization purposes, the results are shown in groups of 50 tests where each group represents a certain stage of the evolution of the heuristic strategy. The performance at each stage is presented by the average of the means of the query sizes of each test function. The performance of the default algorithm is also shown for comparison. Notice that the default performance is not constant throughout all the stages since not every test function is realized during a single stage. A few observations from this comparison are as follows:

- Both heuristics outperform the default search algorithm throughout the randomized test.
- While JUMP2 converges to a fixed strategy almost instantly after the initialization of the first 200 tests, performance of JUMP1 does not converge; JUMP2 also has overall better performance than JUMP1.
- Both JUMP1 and JUMP2 are affected by the strategy update frequency. Nonetheless, the performance of JUMP1 is affected more drastically than that of JUMP2. Also notice that JUMP2 converges to slightly different strategies under different update frequencies. This could happen when a large path length is first recorded on a leaf vertex that has optimal performance on average, leading to the false impression that this vertex is inferior to other alternatives and is not used in future tests.

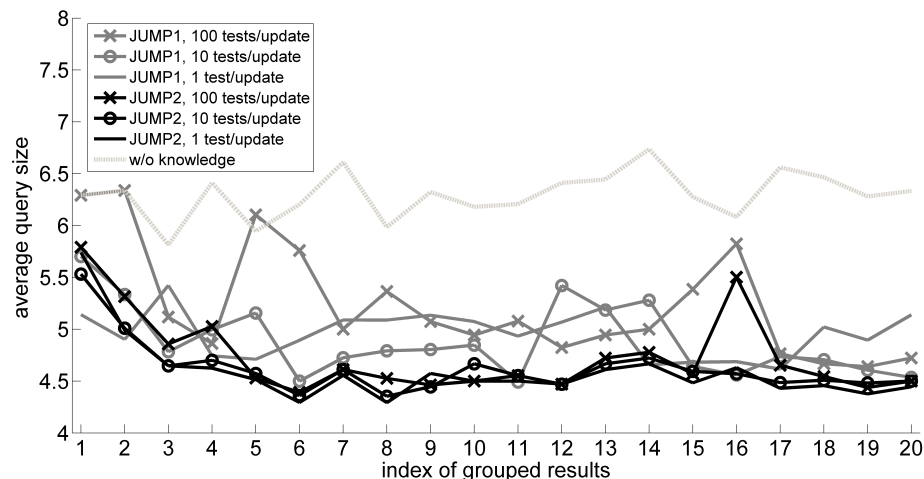


Figure 6.10: Performance of JUMP1 and JUMP2 on 1000 random tests with update frequency: 1, 10, 100 tests per update. Both algorithms switch to heuristic search after every 2 iterations. For visualization purpose, the 1000 test results are grouped and we show the average query size of each group.

### 6.2.6 Update of initial guess (JUMP3)

All of the experiments we conducted so far use the fixed initial pair  $\{x_1, x_{100}\}$ . It is quite intuitive that updating the initial guess during the experiment could help to improve further the overall performance since such a move potentially reduces the need for exploration. As a pilot experiment, we add to the previous algorithm a mechanism that changes the initial pair of designs to the most visited two after every 200 tests. Along with that, the default search algorithm is switched to pure exploitation mode after the first change of initial guess, i.e., the EGO merit function is the decision function without consideration of the variance of predictions. As the last modification to the previous algorithm, we blank the knowledge once the initial guess is changed. This is required since the knowledge from previous searches are irrelevant and will confuse upcoming searches. Figure 6.11 compares the result of this setup and that of JUMP2, both updating the algorithm after every test.

Examination of this performance figure reveals that the benefit of changing the initial guess is usually not instant. In fact, the performance almost always gets worse

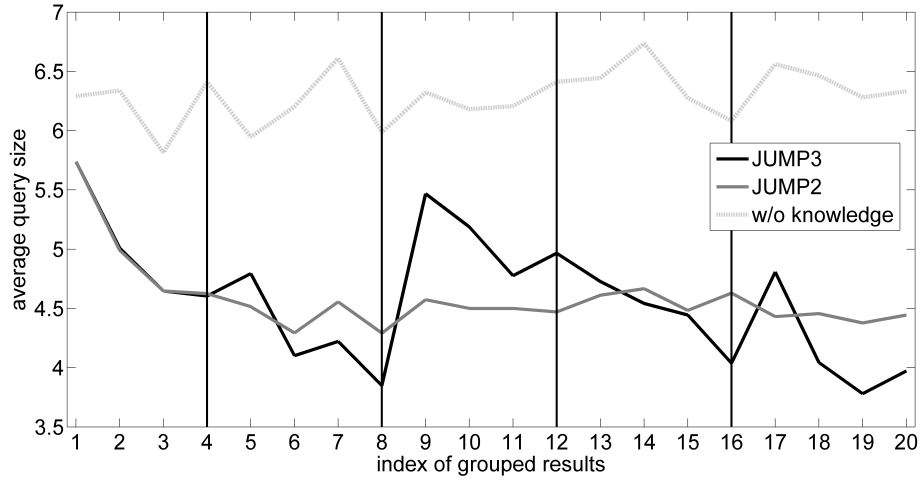


Figure 6.11: Performance of JUMP2 and JUMP3 on 1000 random tests. Both algorithms update their search strategy after every test and switch to heuristic search after every 2 iterations. The figure is segmented to show performance of JUMP3 with different initial guesses. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group.

right after the changes. However, its benefit eventually appears as the average query size gradually goes below the performance of JUMP2 without changing the initial guess. We also noticed from this pilot experiment that fixing the frequency of the change may not be the best strategy since the entire knowledge needs to be rebuilt every time the initial guess is changed. We propose a simple strategy to address this difficulty. The key idea is to keep the current initial guess if there is no significant change in the distribution of the leaf vertices. Formally, for a certain number of tests (200 tests in the following experiment), we group the leaf vertices into  $q$  clusters based on their design variables. Recall that  $q$  is the number of designs in a query. For the  $i$ th cluster, we assume that the observed leaf nodes are realized according to some normal distribution centred at  $\bar{\mathbf{x}}_i$ , which can be derived by maximizing the likelihood of the current realization, as shown in the following derivation. The variance of the normal distribution is neglected in the following derivation since it will not affect the

solution.

$$\begin{aligned}
\bar{\mathbf{x}}_i^* &\triangleq \arg \max_{\bar{\mathbf{x}}_i} \prod_{j \in \text{cluster}_i} \exp(-\|\mathbf{x}_j - \bar{\mathbf{x}}_i\|_2^2) \\
&= \arg \max_{\bar{\mathbf{x}}_i} \sum_{j \in \text{cluster}_i} -\|\mathbf{x}_j - \bar{\mathbf{x}}_i\|_2^2 \\
&= \frac{\sum_{j \in \text{cluster}_i} \mathbf{x}_j}{\text{number of designs in cluster } i}.
\end{aligned} \tag{6.4}$$

We then find a set of leaf nodes  $\{\mathbf{x}_i^*\}_{i=1}^q$  that are closest to their corresponding  $\bar{\mathbf{x}}_i^*$  for each cluster. Let the current initial guess be a set  $\{\mathbf{x}_i\}_{i=1}^q$ . We compute the change in the likelihood of the observation as follows.

$$\Delta L \triangleq \left| \frac{L(\{\mathbf{x}_i^*\}_{i=1}^q) - L(\{\mathbf{x}_i\}_{i=1}^q)}{L(\{\mathbf{x}_i\}_{i=1}^q)} \right|, \tag{6.5}$$

where

$$L(\{\mathbf{x}_i^*\}_{i=1}^q) = \sum_{i=1}^q \sum_{j \in \text{cluster}_i} -\|\mathbf{x}_j - \mathbf{x}_i^*\|_2^2. \tag{6.6}$$

We only change the initial guess when  $\Delta L$  is over a threshold (0.2 in the following experiment). Also remember that when the initial guess is changed, the previous search history will be discarded. Figure 6.12 compares this adaptive initial guess change strategy with the previous fixed one. The improvement is obvious and expected, since there is no need to frequently change the initial guess in this 2D experiment once we settle the initial pair close to the center of the two clusters. Indeed, the adaptive strategy only changed its initial guess once.

### 6.2.7 2D Demonstration of initial guess update

The major purpose of changing the initial guess is to start an interaction (questionnaire) with more relevant queries. We have shown above that this strategy can be useful when we have a fixed distribution of optimal designs. Its deployment, how-

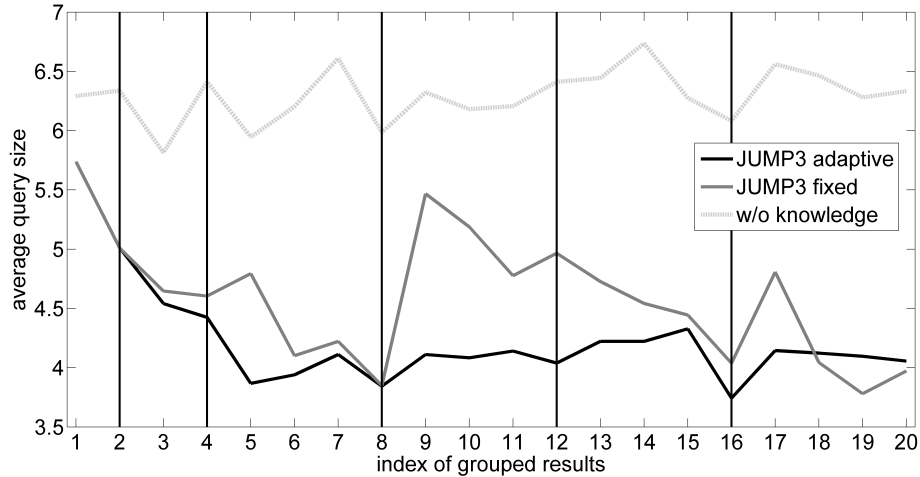


Figure 6.12: Performance of JUMP3 with fixed and adaptive initial guess change frequency on 1000 random tests. Both algorithms update their search strategy after every test and switch to heuristic search after every 2 iterations. The figure is segmented to show performance with different initial guesses. For visualization purpose, the 1000 test results are grouped and we show the average query size of each group.

ever, is more meaningful in a situation where the distribution changes over time, which is often the case with regard to human preference. Below we show the effect of JUMP3 with adaptive initial guess change in an experiment where the locations of optimal designs (leaf vertices) gradually move in the 2D design space as shown in Figure 6.13. Figure 6.14 compares the average query size of JUMP3, JUMP2 and the default search algorithm. The improvement using JUMP3 is significant.

### 6.3 Discussion

Throughout the development of the heuristics, we used a simple 2D problem with clustered optimal designs. It is natural to wonder what performance can the heuristic algorithm achieve in other situations. Below we present a situation, still in 2D, where the optimal designs from different subjects are scattered all around the design space, as shown in Figure 6.15. Figure 6.16 shows the performance of JUMP2, JUMP3 and the default search algorithm. From here we can see that improvements

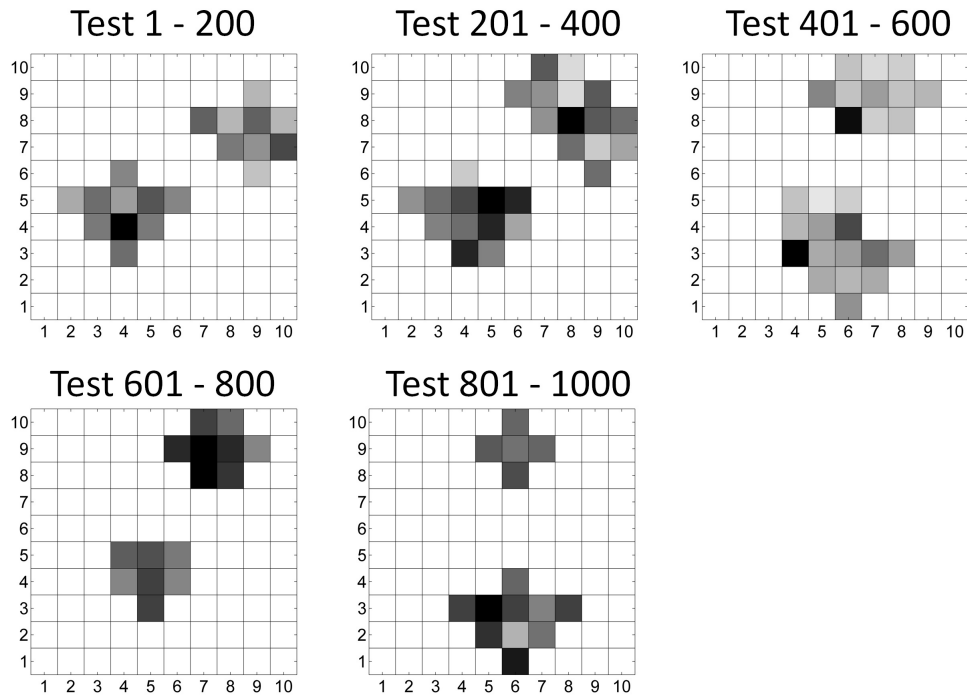


Figure 6.13: Densities of the optimal designs along time. The five figures show how the optimal designs are distributed at different stages of the experiment. The grey scale indicates the frequency of occurrence.

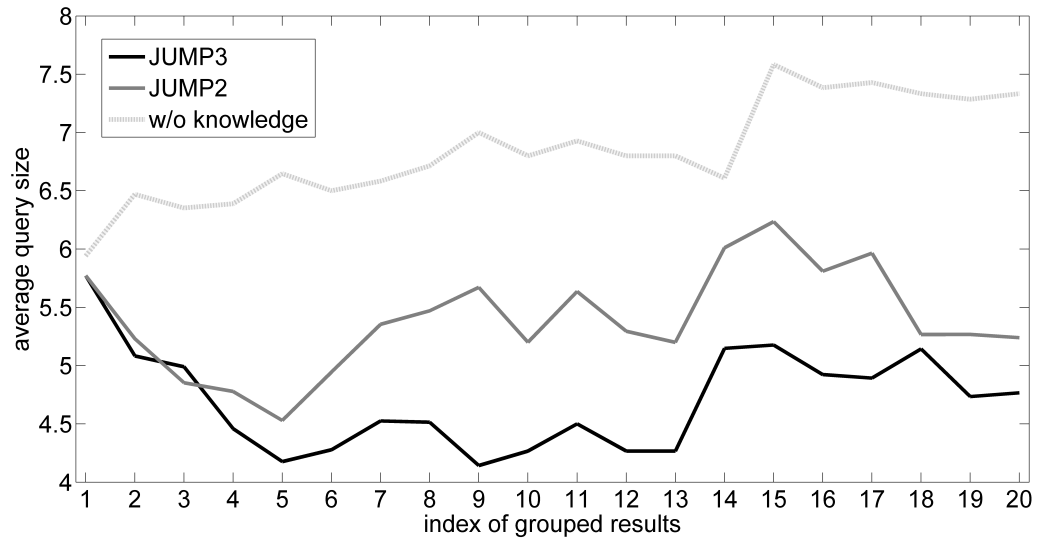


Figure 6.14: Performance of JUMP2, JUMP3 and the default algorithm on 1000 random tests. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 2 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group.



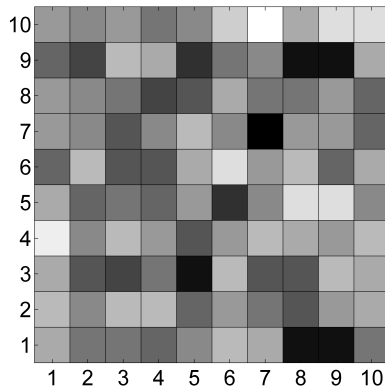


Figure 6.15: A situation where optimal designs are not clustered. The grey scale indicates the frequency of occurrence.

in the performance are still visible by using JUMP2 and JUMP3. However, the difference between JUMP2 and JUMP3 is negligible, which leads to the implication that changing the initial guess is only valuable when the optimal designs are clustered.

To further understand the usability of the proposed heuristics, we conducted two experiments on a design space with 5 dimensions and each with 3 levels. A total of 1000 random tests is conducted on the default algorithm, JUMP2 and JUMP3. In the first experiment, we randomize the optimal designs using a uniform distribution on the entire space, representing the worst scenario for the heuristic to perform. In the other experiment, the optimal solutions are generated uniformly from the small set of indices  $\{64, 65, 66, 222, 223, 224\}$  to represent the scenario where almost every subject falls into confined categories of preferred designs. Figure 6.17 and 6.18 compare the performances in the first and second experiment accordingly. The comparison here shows that the heuristics we proposed do not have universal application. They will be favorable only when the optimal designs from the population are clustered, and their value becomes marginal when possible optimal designs scatter in a high dimensional space. This observation, however, is not counter-intuitive, as the performance of any collaborative filter will rely on the commonality among subjects.

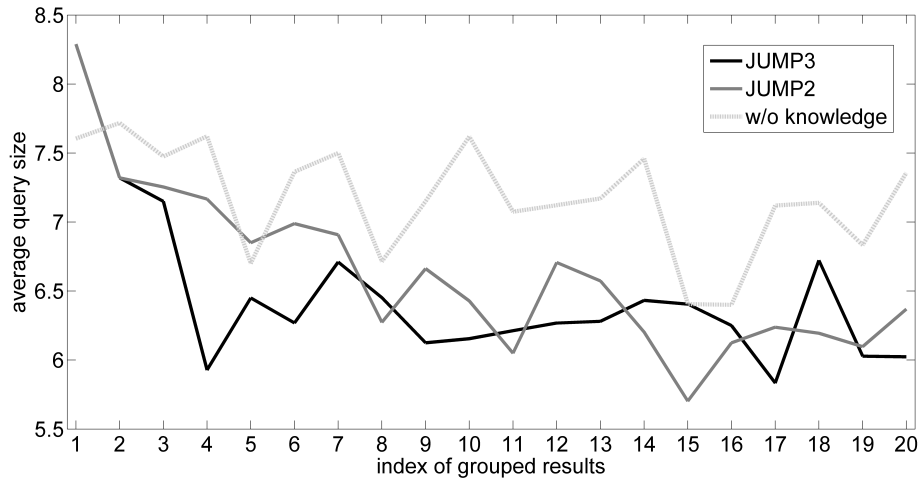


Figure 6.16: Performance of JUMP2, JUMP3 and the default algorithm on the 1000 random tests with optimal designs located in Figure 6.15. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 2 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group.

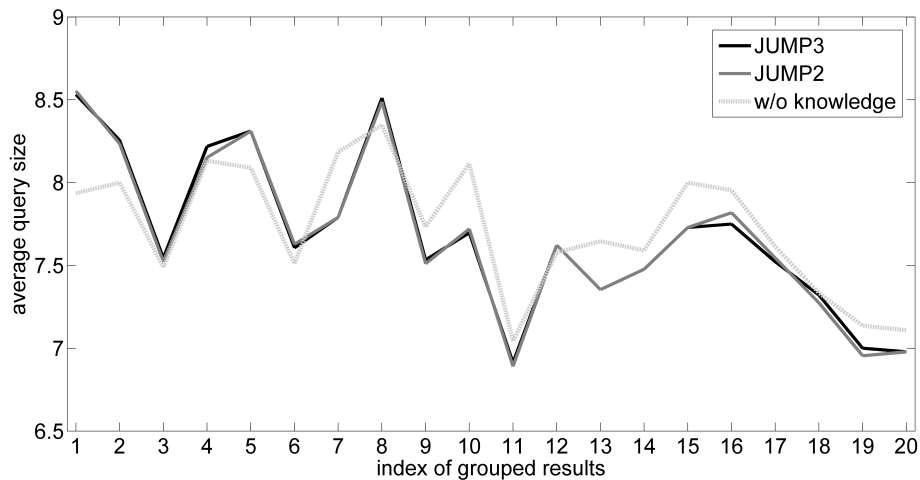


Figure 6.17: Performance of JUMP2, JUMP3 and the default algorithm on the 1000 random tests on a design space with 5 dimensions and 3 levels for each. The optimal designs of these random tests are uniformly generated on the entire space. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 7 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group.

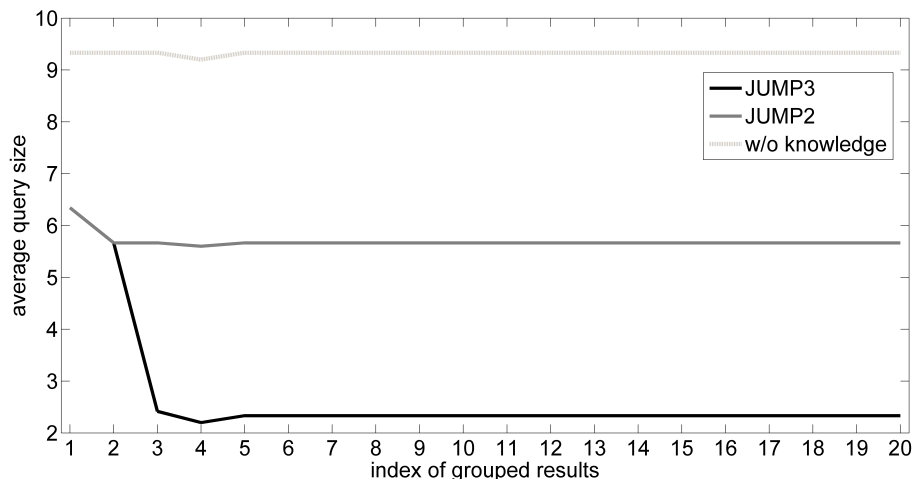


Figure 6.18: Performance of JUMP2, JUMP3 and the default algorithm on the 1000 random tests on a design space with 5 dimensions and 3 levels for each. The optimal designs of these random tests are uniformly generated on a small set of indices of the space. Both JUMP2 and JUMP3 update their search strategy after every test and switch to heuristic search after every 7 iterations. For visualization purposes, the 1000 test results are grouped and we show the average query size of each group.

## 6.4 Relationship With Collaborative Filtering

One may recall that in Chapter I we reviewed collaborative filtering as one popular form of recommender system. The work we presented in this chapter is very similar to a collaborative filter in that every heuristic search is based on the similarity between the current subject preference model and all previous subject models. A heuristically queried design is indeed a recommended item from other subjects with similar interests. There is, however, a crucial difference between the objectives of the two. The purpose of a recommender system is to provide new items to the subject, here in preference elicitation, the purpose of an interaction is to find out the most preferred design. Therefore, the heuristically queried design is not only likely to be preferred by the subject, but also likely to reduce the query size the most. On the other hand, while most collaborative filters operate on millions of items and users, we have not explored the scalability issue of the proposed algorithm.

## 6.5 Concluding Remarks

This chapter investigated how collaborative preference elicitation could work to reduce the query size by incorporating information from previous searches. Four algorithms were proposed. The key concept of these algorithms is to jump to a likely conclusion at the lowest average penalty. Using simulated tests, we showed that our most refined algorithm (JUMP3) can reduce the average query size of a population from the default EGO Search algorithm. This reduction is especially significant when the preferences of a population are clustered.

## CHAPTER VII

### Conclusions

#### 7.1 Summary

Understanding user preference has long been a challenging topic in the design research community. We have adopted econometric survey and data analysis methods to link engineering design and marketing, achieving design solutions that are sound from both engineering and business perspectives. This approach, however, only refines existing design from historical data while what we really need is to allow users to explore concepts and express their preferences through rich media, and create a channel for those user preferences to reach and inform the designers. With rapid developments in machine learning and virtual reality, this ambition is not only possible in research but also for employment in practice. The underlying research problem we are interested in is how to extract preferences from humans interactively and how to capture these preferences precisely within short time. This dissertation focused on addressing this problem essentially as a mathematical problem.

Throughout the dissertation, we worked in the context of a human-computer interaction where in each iteration, the subject is queried with a set of designs and required to choose a few preferred ones out of the set. The computer learns from the accumulated knowledge and creates the next set of designs to query, the responses upon which will be the most informative with respect to the subject's preferences.

We also use the assumption that preference can be expressed as a function of the designs, with or without random errors.

Three problems were discussed surrounding how effective queries can be made:

1. The objective of *preference elicitation* is to locate the most preferred design of a subject. Thus, making queries is equivalent to an algorithmic search to optimize a preference function.
2. In the *preference identification* problem, we consider designs as binary classified, i.e., a design is either preferred or not-preferred. Therefore capturing preferred designs is equivalent to finding a classifier for these two classes.
3. In *preference estimation*, we discussed how to estimate the entire preference function with adaptive queries.

The major effort was devoted to preference elicitation, which was shown as a unique “black-box” optimization problem with only binary choice data, instead of real-valued ones. We developed in Chapters III and IV a search algorithm that queries the next design that has high predicted preference and high uncertainty in its prediction. To predict user preference from binary choices, we first introduced a classification method in Chapter III by treating designs as binary classified. We then proposed a more accurate preference prediction method in Chapter IV by modeling accumulated binary choices as a comparison tree. The search algorithm developed in Chapter III can be computationally expensive when the dimensionality of the design space is high, making the algorithm less usable in real-time interactions. We identified that the cause to this is the high nonlinearity of the merit function on which new queries depend. This nonlinear function has multiple local optima in the high preference prediction region which causes the employed genetic algorithm to converge slowly (up to 1000 generations). An alternative merit function was thus developed that has one extra dimension but greater smoothness. We empirically showed that

this new form of merit function enhanced the computational efficiency of the search algorithm significantly.

The preference elicitation algorithm was then incorporated with an online 3D vehicle exterior modeling program. This combination allows free access to the developed interaction. Using this setup, we deployed a convergence test where the user is assigned a target design and is asked to find the target using the interaction. Our results showed that users can successfully find the target within 10 to 20 iterations, and that, although the underlying design features that users consider are unknown to the algorithm, it can still work well thanks to the kernel trick employed from the machine learning literature.

We then showed in Chapter V that preference estimation and preference identification problems are mathematically equivalent. While the former estimates the preference parameters in the feature space, the latter does so in the feature difference space, i.e., a space defined by the difference between features. The shared difficulty for both problems is to adaptively pick the most informative queries from an enormous set of candidates. We provided a thorough review of how preference estimation is done, starting in Chapter II and continued in Chapter V. The review showed that this difficulty can be addressed by the active learning technique separately developed in econometrics and computer science. The core concept of active learning is to select queries that achieve both high information gain (using utility balance) and high diversity of the queried samples (to minimize maximum uncertainty).

In Chapter VI we investigated how collaborative preference elicitation can work to reduce the query size by incorporating information from previous searches. Four algorithms were proposed. The key concept of these algorithms is to jump to such a conclusion at the lowest average penalty. Using simulated tests, we showed that our most refined algorithm (JUMP3) can reduce the average query size of a population from the default EGO Search algorithm. This reduction is especially significant when

the preferences of a population are clustered.

From a high-level viewpoint, the theme throughout this dissertation is the balance between exploration and exploitation during a querying process. Conceptually, this work is about how to ask relevant but not redundant questions during an interaction.

## 7.2 Contributions

1. The major contribution of this work comes along with the solution to the preference elicitation problem. As we discussed, preference elicitation is indeed a unique optimization problem with binary responses. The engineering community has researched how a real-valued “black-box” function shall be optimized using statistical methods, and the econometric community has studied how to build preference models based on binary choice data. Nonetheless, this combination of “black-box” optimization and binary choice response is put together here for the first time.
2. In addition to formulating and proposing search algorithms to elicit preference, the improvement in the computational cost of the search algorithm itself against existing approaches used for real-valued “black-box” optimization is an important contribution, also for solving traditional simulation-based optimization problems.
3. We incorporated the proposed search algorithm in an online human-computer interaction to extract subjects’ preferences on three-dimensional vehicle exterior designs in real time. This is the first such development for a highly complex design problem. The preliminary convergence test results from real subject tests show convergence of these interactions. The experiments conducted using the test website along with the implementation of the search algorithm open a promising avenue for connecting engineering design and human-computer inter-



action.

4. Collaborative preference elicitation was introduced to speed up a querying process by taking into account search histories from previous subjects. We proposed heuristic algorithms that gradually evolve and converge to search rules that minimize the average query size of a population. This is a novel way for capturing mathematically the popular “crowdsourcing” processes.
5. We showed that the preference identification problem can be mathematically formulated and solved in the same way as the preference estimation problem using active learning. We showed with simulated tests that active learning can enhance the performance in both estimation and identification problems within some level of subject faulty choice rate. More importantly, the introduction of the kernel trick from the machine learning research to the solution of these problems improves performance when the underlying models are nonlinear, therefore eliminating the controversial linearly additive utility assumption. Introducing this technique to the design community will help marketing researchers to improve validity of models from stated (survey) data.

### 7.3 Assumptions and Limitations

Assumptions used in this dissertation lead to limitations of the proposed methods. One of the major assumptions is the definition of preference in the form of a function defined on a design variable space. Although widely adopted, this assumption can be easily challenged at different levels: From a high level view point, it is still under question whether human preference can be measured. Taking this for granted, it is yet to be justified whether the preference can be defined on the given design space and whether the features introduced are appropriate. Leaving these aside, a great limitation still remains the fact that preference usually changes through time rather

than staying fixed as it was before the interaction happens. Therefore, it is important to review, establish and evaluate more reasonable and sophisticated human decision making models to improve our understanding of user preferences based on the interaction data. With these limitations of the current work, there is still plenty of room for justifying and improving the proposed query algorithms for real-world use. Some of the specific items are discussed in the following section.

## 7.4 Future Work

### 7.4.1 Preference modeling with enriched user interaction data

One important feature of the online interaction environment is to allow rotation, panning and zooming of individual 3D models. These user actions can be recorded and are helpful in understanding what design features the user is looking at. For example, by analyzing what perspective the user is using before she made a choice, we can not only know what designs are judged better than the others but also on what basis this judgement is made. The mathematical interpretation of user actions will help to refine the prediction of preference by refining the weights on features involved in the training. Formally, let  $\mathbf{S}$ ,  $n \times n$  be such that  $\mathbf{S}_{ij}$  is the index of the iteration when item  $i$  and  $j$  are compared by the subject. Let  $\mathbf{e}_{\mathbf{S}_{ij}}$ ,  $k \times 1$  be a binary vector of the length of design features;  $\mathbf{e}_\kappa$  takes 1 when the  $\kappa$ th feature is considered by the user in iteration  $\mathbf{S}_{ij}$ . Introducing an element-wise product operator  $\otimes$ , the training problem in Equation (4.3) can then be refined as:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{subject to} \quad & \mathbf{w}^T ((\mathbf{r}(\mathbf{X}, \mathbf{x}_j)) - \mathbf{r}(\mathbf{X}, \mathbf{x}_i)) \otimes \mathbf{e}_{\mathbf{S}_{ij}} \leq a, \quad \forall i, j \text{ such that } \mathbf{G}_{ij} = 1 \end{aligned} \quad (7.1)$$

Although mathematically simple, the practical difficulty in incorporating user actions in learning is visible: User action may involve a sequence of rotation, panning and zooming. The challenge is to justify by the recorded data what parts of the model the user is indeed looking at. Collaborative knowledge may also be helpful in this context. By analyzing user actions from a population, heuristics may be derived to interpret future user actions more efficiently.

Another important information we should utilize in the search is the demographic profile of users. Data including gender, age, and semantic description of design taste can be incorporated with search results, i.e., optimal designs, and together create a classifier for the population, so that future search will be able to start with designs potentially close to the subject’s preference by classifying the subject according to his demographic data.

#### **7.4.2 Calibration on weighted search**

As we show in Chapter III, failure to set the weights on exploration and exploitation properly can lead to much deteriorated search performance. However, throughout this dissertation, these weights are set manually based on repeated experiments and thus they are only in an empirical sense “optimized”. In real-world interactions, the weights may be different for different people and thus manually fixing the weights can lead to sub-optimal performance on average. One easy way to fix this is to provide the user a tuning slider in the interface. Collecting information on how users tune the weights may also help us to design the weights better in the first place.

#### **7.4.3 Feature addition and subtraction**

Throughout this dissertation we assumed that the preference is defined on the space of a fixed set of variables and can be represented as a linear summation of design features. While making such assumptions is questionable, as we mentioned,

the opportunity is that we may find ways to refine the set of variables we use by examining their influences on preference once some interaction data are collected. Variables will be subtracted if they have trivial impact and new variables related to those important existing ones shall be added to future interactions. For example, in the vehicle exterior design case, if we observe that users' preferences are affected mainly by the changes in the side silhouette design, more variables shall be introduced to enrich the variety of this aspect of the design so that more accurate preference can be captured in future tests.

#### **7.4.4 Linking user preferences and designer decisions**

Besides preference estimation, which has been widely applied, the solutions to the other two problems we discussed can help design decision making in their own ways:

1. We consider preference elicitation as both a data collection tool for designers to understand user preference trends and a convenient design tool for users to create and share their own creations. The data collected from user participation can be analyzed for specific interests. For example, one can analyze how user preferences are clustered and changing along time.
2. A preference identification implementation will help designers to create design constraints that cannot be explicitly expressed. For example, an experienced automobile exterior designer can train the algorithm so that it learns what car shapes are allowed and what are not. This expert knowledge will then be propagated as design constraints to other engineering departments.

The algorithms proposed in this dissertation must be calibrated and implemented in real-life design decision making scenario so that their usability can be evaluated better.

#### **7.4.5 Search towards a mutual target**

The last topic of great research interest but also practical usage is how to facilitate searches not just for individuals but for a group of designers to achieve a mutually preferable target design. In a real interdisciplinary design environment, the preference of one designer can be affected by inputs from other designers with different knowledge sets and focuses. Thus the optimal design of each individual changes throughout the design process. Adding complexity to it, the design team usually has a structure, meaning that the opinions of some members may have more priority than the others and shall be elicited and broadcast to other members. It would be interesting to investigate how a coordination as well as search mechanism can be developed to extract preferences of a group of designers in real time.

## BIBLIOGRAPHY

## BIBLIOGRAPHY

- Abernethy, J., Evgeniou, T., Toubia, O., and Vert, J. (2008), Eliciting consumer preferences using robust adaptive choice questionnaires, *IEEE Transactions on Knowledge and Data Engineering*, 20(2), 145–155.
- Adomavicius, G., and Tuzhilin, A. (2005), 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.
- Arora, N., and Huber, J. (2001), Improving parameter estimates and model prediction by aggregate customization in choice experiments, *Journal of Consumer Research*, 28(2), 273–283.
- Baeza-Yates, R., and Ribeiro-Neto, B. (1999), *Modern information retrieval*, Addison-Wesley, New York.
- Balabanović, M., and Shoham, Y. (1997), Fab: content-based, collaborative recommendation, *Communications of the ACM*, 40(3), 66–72.
- Baram, Y., El-Yaniv, R., and Luz, K. (2004), Online choice of active learning algorithms, *The Journal of Machine Learning Research*, 5, 255–291.
- Basudhar, A., and Missoum, S. (2010), An improved adaptive sampling scheme for the construction of explicit boundaries, *Structural and Multidisciplinary Optimization*, 42(4), 517–529.
- Billsus, D., and Pazzani, M. (1998), Learning collaborative information filters, in *Proceedings of the 15th International Conference on Machine Learning*, pp. 46–54, Morgan Kaufmann Publishers Inc.
- Billsus, D., and Pazzani, M. (2000), User modeling for adaptive news access, *User Modeling and User-adapted Interaction*, 10(2), 147–180.
- Breese, J., Heckerman, D., and Kadie, C. (1998), Empirical analysis of predictive algorithms for collaborative filtering, in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43–52.
- Bunch, D., Louviere, J., and Anderson, D. (1996), A comparison of experimental design strategies for multinomial logit models: The case of generic attributes, *University of California Davis Graduate School of Management Working Paper*, pp. 11–96.

- Chang, C., and Lin, C. (2011), Libsvm: a library for support vector machines, *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3), 27.
- Chang, E., Tong, S., Goh, K., and Chang, C. (2005), Support vector machine concept-dependent active learning for image retrieval, *IEEE Transactions on Multimedia*, 2.
- Chapelle, O., and Harchaoui, Z. (2005), A machine learning approach to conjoint analysis, *Advances in Neural Information Processing Systems*, 17, 257–264.
- Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. (2002), Choosing multiple parameters for support vector machines, *Machine Learning*, 46(1), 131–159.
- Chipperfield, A., and Fleming, P. (1995), The MATLAB genetic algorithm toolbox, in *Colloquium Digest-IEEE*, pp. 10–10, Citeseer.
- Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. (1999), Combining content-based and collaborative filters in an online newspaper, in *Proceedings of ACM SIGIR Workshop on Recommender Systems*, pp. 40–48, ACM.
- Cortes, C., and Vapnik, V. (1995), Support-vector networks, *Machine Learning*, 20(3), 273–297.
- Cox, D., and John, S. (1992), A statistical method for global optimization, in *IEEE International Conference on Systems, Man and Cybernetics*, pp. 1241–1246, IEEE.
- Cristianini, N., and Shawe-Taylor, J. (2000), *An introduction to support vector machines: and other kernel-based learning methods*, Cambridge University Press New York, NY, USA.
- Cui, D., and Curry, D. (2005), Prediction in marketing using the support vector machine, *Marketing Science*, pp. 595–615.
- Das, A., Datar, M., Garg, A., and Rajaram, S. (2007), Google news personalization: scalable online collaborative filtering, in *Proceedings of the 16th International Conference on World Wide Web*, pp. 271–280, ACM.
- Dawkins, R., and Pyle, L. (1991), *The blind watchmaker*, Penguin Harmondsworth.
- Dixon, L., and Szegö, G. (1978), The global optimization problem: an introduction, *Towards Global Optimization*, 2, 1–15.
- Drucker, H., Burges, C., Kaufman, L., Smola, A., and Vapnik, V. (1997), Support vector regression machines, *Advances in Neural Information Processing Systems*, 1(June), 155–161.
- Evgeniou, T., Boussios, C., and Zacharia, G. (2005), Generalized robust conjoint estimation, *Marketing Science*, 24(3), 415–429.



- Evgeniou, T., Pontil, M., and Toubia, O. (2007), A convex optimization approach to modeling consumer heterogeneity in conjoint estimation, *Marketing Science*, 26(6), 805–818.
- Frischknecht, B. (2009), Market Systems Modeling for Public versus Private Tradeoff Analysis in Optimal Vehicle Design, PhD Thesis, University of Michigan, Ann Arbor.
- Frischknecht, B., Whitefoot, K., and Papalambros, P. (2009), Methods for evaluating suitability of econometric demand models in design for market systems.
- GREEN, P., and RAO, V. (1971), Conjoint measurement for quantifying judgmental data, *Journal of Marketing Research*, 8, 355–63.
- Green, P., and Srinivasan, V. (1978), Conjoint analysis in consumer research: issues and outlook, *Journal of Consumer Research*, 5(2), 103.
- Green, P., Krieger, A., and Wind, Y. (2001), Thirty years of conjoint analysis: Reflections and prospects, *Interfaces*, 31(3), 56–73.
- Hardy, J. (1975), An implemented extension of branin’s method, *Towards Global Optimization*, pp. 117–142.
- Harman, R., and Trnovska, M. (2009), Approximate d-optimal designs of experiments on the convex hull of a finite set of information matrices, *Math. Slovaca*, 59, 693–704.
- Hauser, J., Toubia, O., Evgeniou, T., Befurt, R., and Dzyabura, D. (2010), Disjunctions of conjunctions, cognitive simplicity, and consideration sets, *Journal of Marketing Research*, 47(3), 485–496.
- Hermes, L., and Buhmann, J. (2000), Feature selection for support vector machines, in *15th International Conference on Pattern Recognition*, vol. 2, pp. 712–715, IEEE.
- Hoyle, C., Chen, W., Ankenman, B., and Nanxin, W. (2009), Optimal experimental design of human appraisals for modeling consumer preferences in engineering design, *ASME Journal of Mechanical Design*, 131(7), 071,008.
- Huber, J., and Zwerina, K. (1996), The importance of utility balance in efficient choice designs, *Journal of Marketing Research*, 33(3), 307–317.
- Joachims, T. (2002), Optimizing search engines using clickthrough data, in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 133–142, ACM.
- Johanson, B., and Poli, R. (1998), Gp-music: An interactive genetic programming system for music generation with automated fitness raters, *Genetic Programming*, pp. 181–186.

- Jones, D. (2001), The DIRECT global optimization algorithm, *Encyclopedia of Optimization*, 1, 431–440.
- Jones, D., Schonlau, M., and Welch, W. (1998), Efficient global optimization of expensive black-box functions, *Journal of Global Optimization*, 13(4), 455–492.
- Kanninen, B. (2002), Optimal design for multinomial choice experiments, *Journal of Marketing Research*, 39(2), 214–227.
- Kelly, J. (2008), Interactive genetic algorithms for shape preference assessment in engineering design, Ph.D. thesis, University of Michigan, Ann Arbor, Michigan.
- Kelly, J., and Papalambros, P. (2007), Use of shape preference information in product design, in *International Conference on Engineering Design, Paris, France*, Citeseer.
- Kim, H., and Cho, S. (2000), Application of interactive genetic algorithm to fashion design, *Engineering Applications of Artificial Intelligence*, 13(6), 635–644.
- Kosorukoff, A. (2001), Human based genetic algorithm, in *2001 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 3464–3469, IEEE.
- Kuhfeld, W. (2005), Marketing research methods in sas, *Experimental Design, Choice, Conjoint, and Graphical Techniques*. Cary, NC, SAS-Institute TS-722.
- Kuhfeld, W., Tobias, R., and Garratt, M. (1994), Efficient experimental design with marketing research applications, *Journal of Marketing Research*, 31(4), 545–557.
- Kumar, D., Hoyle, C., Chen, W., Wang, N., Gomez-Levi, G., and Koppelman, F. (2007), Incorporating customer preferences and market trends in vehicle package design, in *Proceedings of the ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, pp. 571–581.
- Kushner, H. (1964), A new method of locating the maximum of an arbitrary multippeak curve in the presence of noise, *Journal of Basic Engineering*, 86, 97–106.
- Lee, T., and Bradlow, E. (2007), Automatic construction of conjoint attributes and levels from online customer reviews, *University Of Pennsylvania, The Wharton School Working Paper*.
- Li, H., and Azarm, S. (2000), Product design selection under uncertainty and with competitive advantage, *ASME Journal of Mechanical Design*, 122(4), 411–418.
- Linden, G., Smith, B., and York, J. (2003), Amazon. com recommendations: Item-to-item collaborative filtering, *Internet Computing*, 7(1), 76–80.
- Liu, Y., and Zheng, Y. (2006), Fs\_sfs: A novel feature selection method for support vector machines, *Pattern Recognition*, 39(7), 1333–1345.

- Lloyd, A. (2003), Threats to the estimation of benefit: are preference elicitation methods accurate?, *Health Economics*, 12(5), 393–402.
- Locatelli, M. (1997), Bayesian algorithms for one-dimensional global optimization, *Journal of Global Optimization*, 10(1), 57–76.
- MacDonald, E., Gonzalez, R., and Papalambros, P. (2009), Preference inconsistency in multidisciplinary design decision making, *ASME Journal of Mechanical Design*, 131(3), 79–92.
- Mandel, M., Poliner, G., and Ellis, D. (2006), Support vector machine active learning for music retrieval, *Multimedia Systems*, 12(1), 3–13.
- McFadden, D. (1973), Conditional logit analysis of qualitative choice behavior, *Frontiers in Econometrics*, pp. 105–142.
- McFadden, D. (1980), Econometric models for probabilistic choice among products, *Journal of Business*, 53(3), S13–29.
- McFadden, D., and Train, K. (2000), Mixed mnl models for discrete response, *Journal of Applied Econometrics*, 15(5), 447–470.
- Michalek, J., Feinberg, F., and Papalambros, P. (2005), Linking marketing and engineering product design decisions via analytical target cascading, *Journal of Product Innovation Management*, 22(1), 42–62.
- Mitchell, T. (1974), An algorithm for the construction of “d-optimal” experimental designs, *Technometrics*, 16, 203–210.
- Mooney, R., Bennett, P., and Roy, L. (1998), Book recommending using text categorization with extracted information, in *Recommender Systems Papers from 1998 Workshop, Technical Report WS-98-08*.
- Netzer, O., et al. (2008), Beyond conjoint analysis: Advances in preference measurement, *Marketing Letters*, 19(3), 337–354.
- Neumann, J., Schnörr, C., and Steidl, G. (2005), Combined svm-based feature selection and classification, *Machine Learning*, 61(1), 129–150.
- Norman, D. (2002), Emotion & design: attractive things work better, *Interactions*, 9(4), 36–42.
- Osugi, T., Kun, D., and Scott, S. (2005), Balancing exploration and exploitation: A new algorithm for active machine learning, in *Proceedings of the 5th IEEE International Conference on Data Mining*, pp. 330–337, IEEE Computer Society.
- Pazzani, M. (1999), A framework for collaborative, content-based and demographic filtering, *Artificial Intelligence Review*, 13(5), 393–408.

- Pazzani, M., and Billsus, D. (1997), Learning and revising user profiles: The identification of interesting web sites, *Machine Learning*, 27(3), 313–331.
- Petiot, J., and Grognet, S. (2006), Product design: a vectors field-based approach for preference modelling, *Journal of Engineering Design*, 17(3), 217–233.
- Reid, T. (2010), Quantifying Perception-Based Attributes in Design: A Case Study on the Perceived Environmental Friendliness of Vehicle Silhouettes, PhD Thesis, University of Michigan, Ann Arbor, Michigan.
- Ren, Y. (2009), An interactive modeling environment for automotive exterior design, MS Thesis, University of Michigan, Ann Arbor.
- Rosenbrock, H. (1960), An automatic method for finding the greatest or least value of a function, *The Computer Journal*, 3(3), 175.
- Roy, N., and McCallum, A. (2001), Toward optimal active learning through sampling estimation of error reduction, in *Proceedings of the 18th International Conference on Machine Learning*, pp. 441–448, Citeseer.
- Salton, G. (1989), *Automatic Text Processing: The Transformation, Analysis, and Retrieval of*, Addison-Wesley.
- Sándor, Z., and Wedel, M. (2001), Designing conjoint choice experiments using managers prior beliefs, *Journal of Marketing Research*, 38(4), 430–444.
- Sanena, M. (2002), Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations, Ph.D. thesis, University of Michigan, Ann Arbor, Michigan.
- Sheth, B., and Maes, P. (1993), Evolving agents for personalized information filtering, in *Proceedings of the 9th Conference on Artificial Intelligence for Applications*, pp. 345–352, IEEE.
- Siah, E., Sasena, M., Volakis, J., Papalambros, P., and Wiese, R. (2004), Fast parameter optimization of large-scale electromagnetic objects using direct with kriging metamodeling, *IEEE Transactions on Microwave Theory and Techniques*, 52(1), 276–285.
- Sims, K. (1991), Artificial evolution for computer graphics, *Computer Graphics*, 25(4), 319–328.
- Sims, K. (1997), Galápagos, *Information online at: <http://www.genarts.com/galapagos>*.
- Soboroff, I., and Nicholas, C. (1999), Combining content and collaboration in text filtering, in *Proceedings of the International Joint Conferences on Artificial Intelligence*, vol. 99, pp. 86–91.

- Takagi, H. (2001), Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation, *Proceedings of the IEEE*, 89(9), 1275–1296.
- Todd, P., and Gigerenzer, G. (2000), Précis of simple heuristics that make us smart, *Behavioral and Brain Sciences*, 23(5), 727–741.
- Tokui, N., and Iba, H. (2000), Music composition with interactive evolutionary computation, in *Proceedings of the 3rd International Conference on Generative Art*.
- Tong, S., and Chang, E. (2001), Support vector machine active learning for image retrieval, in *Proceedings of the 9th ACM International Conference on Multimedia*, pp. 107–118, ACM.
- Tong, S., and Koller, D. (2002), Support vector machine active learning with applications to text classification, *The Journal of Machine Learning Research*, 2, 45–66.
- Toubia, O., Simester, D., Hauser, J., and Dahan, E. (2003), Fast polyhedral adaptive conjoint estimation, *Marketing Science*, 22(3), 273–303.
- Toubia, O., Hauser, J., and Simester, D. (2004), Polyhedral methods for adaptive choice-based conjoint analysis, *Journal of Marketing Research*, 41(1), 116–131.
- Toubia, O., Evgeniou, T., and Hauser, J. (2007a), Optimization-based and machine-learning methods for conjoint analysis: Estimation and question design, *Conjoint Measurement: Methods and Applications*, p. 231.
- Toubia, O., Evgeniou, T., and Hauser, J. (2007b), Optimization-based and machine-learning methods for conjoint analysis: Estimation and question design, *Conjoint Measurement*, pp. 231–258.
- Train, K. (2001), A comparison of hierarchical bayes and maximum simulated likelihood for mixed logit, *University of California, Berkeley*.
- Vapnik, V. (1982), Estimation of dependences based on empirical data, *NY: Springer-Verlag*.
- Vapnik, V. (1998), *Statistical learning theory*, vol. 2, Wiley New York.
- Wassenaar, H., Sudjianto, A., Cheng, J., and Chen, W. (2005), Enhancing discrete choice demand modeling for decision-based design, *ASME Journal of Mechanical Design*, 127(4), 514–523.
- Watson, A., and Barnes, R. (1995), Infill sampling criteria to locate extremes, *Mathematical Geology*, 27(5), 589–608.
- Wen, C., and Koppelman, F. (2001), The generalized nested logit model, *Transportation Research Part B: Methodological*, 35(7), 627–641.

- Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. (2001), Feature selection for svms, *Advances in Neural Information Processing Systems*, (13), 668–674.
- Whitefoot, K., Fowle, M., and Skerlos, S. (2011), Product design response to industrial policy: Evaluating fuel economy standards using an engineering model of endogenous product design, *Energy Institute at Haas Working Paper WP-214*. Berkeley, CA: University of California Energy Institute.
- Zhang, Y., Callan, J., and Minka, T. (2002), Novelty and redundancy detection in adaptive filtering, in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 81–88, ACM.
- Žilinskas, A. (1981), Two algorithms for one-dimensional multimodal minimization, *Optimization*, 12(1), 53–63.