Multi-Objective Optimization of Next-Generation Aircraft Collision Avoidance Software

by

John R. Lepird
B.S. Operations Research
B.S. Mathematical Sciences
United States Air Force Academy (2013)

Submitted to the Operations Research Center
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2015

Author .....
Operations Research Center
May 15, 2015

Certified by .....
Michael P. Owen
Technical Staff, Lincoln Laboratory
Thesis Supervisor

Certified by .....
Dimitri P. Bertsekas
McAfee Professor of Engineering
Thesis Supervisor

Accepted by .....
Dimitris Bertsimas
Boeing Professor of Operations Research
Co-Director, Operations Research Center

Disclaimer: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Multi-Objective Optimization of Next-Generation Aircraft Collision Avoidance Software

by

John R. Lepird

Submitted to the Operations Research Center
on May 15, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Developed in the 1970's and 1980's, the Traffic Alert and Collision Avoidance System (TCAS) is the last safety net to prevent an aircraft mid-air collision. Although TCAS has been historically very effective, TCAS logic must adapt to meet the new challenges of our increasingly busy modern airspace. Numerous studies have shown that formulating collision avoidance as a partially-observable Markov decision process (POMDP) can dramatically increase system performance. However, the POMDP formulation relies on a number of design parameters—modifying these parameters can dramatically alter system behavior. Prior work tunes these design parameters with respect to a single performance metric. This thesis extends existing work to handle more than one performance metric. We introduce an algorithm for preference elicitation that allows the designer to meaningfully define a utility function. We also discuss and implement a genetic algorithm that can perform multi-objective optimization directly. By appropriately applying these two methods, we show that we are able to tune the POMDP design parameters more effectively than existing work.

Thesis Supervisor: Michael P. Owen
Title: Technical Staff, Lincoln Laboratory

Thesis Supervisor: Dimitri P. Bertsekas
Title: McAfee Professor of Engineering

Acknowledgments

I would like to thank the MIT Lincoln Laboratory for not only supporting my education and research over the past two years, but also providing a fantastic environment for me to develop as a student, an Air Force officer, and a person. Col (ret.) John Kuconis was instrumental in bringing me to the lab and to MIT, and for that I am incredibly grateful.

Specifically, I am thankful for the opportunity to work with all my friends and colleagues in Group 42. I would like to thank Dr. Wesley Olson for his continual leadership and support of my endeavors. This work would not be the same were it not for the technical guidance and support of Robert Klaus, Ted Londer, Jessica Holland, and Barbara Chludzinski. I am also grateful for the support and friendship of Robert Moss, Brad Huddleston, Rachel Tompa, and Nick Monath.

I would also like to thank my adviser, Professor Dimitri Bertsekas, for keeping me on track and ensuring that I got the full "MIT experience." His superlative technical advice was also very much appreciated.

I am incredibly thankful for the help of Professor Mykel Kochenderfer. His continual support and technical guidance were invaluable to this effort. Similarly, I cannot thank Dr. Michael Owen enough for his guidance during my entire time at the Lincoln Lab: his guidance saved me from numerous pitfalls and was instrumental in making my research both fruitful and enjoyable.

Finally, I would like to thank all my friends and colleagues at the MIT Operations Research Center for their help and insights into this work, as well as their friendship. Although this list is far from incomplete, I would specifically like to thank Zeb Hanley, Kevin Rossillon, Dan Schonfeld, Mapi Testa, and Mallory Sheth — my experience at MIT would not have been the same without you.

This work was sponsored by the FAA TCAS Program Office AJM-233, and I gratefully acknowledge Mr. Neal Suchy for his leadership and support. Interpretations, opinions, and conclusions are those of the authors and do not reflect the official position of the United States Government. This thesis leverages airspace encounter models that were jointly sponsored by the U.S. Federal Aviation Administration, the U.S. Air Force, and the U.S. Department of Homeland Security.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

The Traffic Alert and Collision Avoidance System (TCAS) is the last safety net for avoiding an airborne collision. Mandated on all large aircraft flying in United States and European Union airspace, TCAS monitors the airspace surrounding the airplane and alerts the pilot of incoming traffic. If necessary, TCAS then instructs the pilots how to maneuver in order to avoid an impending collision.

Although very effective, TCAS logic is nearly thirty years old. Advances in computer hardware and artificial intelligence have led to development of a next-generation aircraft collision avoidance system, dubbed the Experimental Aircraft Collision Avoidance System (ACAS-X). Although ACAS-X has shown itself capable of across-the-board improvement over the existing TCAS system, development is still underway. One aspect under development is changing certain parameters of ACAS-X to generate ideal performance.

Optimizing ACAS-X performance with respect to these parameters is very difficult. First, evaluating its performance can only be accomplished through extensive simulation: a computational challenge in and of itself. Second, to exacerbate matters, there is no single metric that completely summarizes "good" performance. One could simply use the number of near mid-air collisions, but optimizing only this metric would result in a logic that alerts pilots too frequently to be useful.

The problem of performance evaluation was largely solved by Kyle Smith in his Master's thesis[71] by using a machine learning technique known to engineers as

"surrogate modelling"[33]. This thesis focuses on the second problem: designing a complex system subject to multiple conflicting optimization criteria.

1.1 Contributions and Outline

This thesis offers two important contributions to the development of a next-generation aircraft collision avoidance system:

- We develop a novel approach to preference elicitation for engineering design optimization, allowing experts to meaningfully define a utility function by using pairwise comparisons between designs. We empirically show that this algorithm converges faster than existing algorithms to an expert's true utility function. We then demonstrate the usefulness of this approach by using it to incorporate preferences from aviation experts around the world into an optimization of ACAS-X. Experimental results indicate that the algorithm effectively captured the experts intent and resulted in a solution better tailored to their needs.

- We exploit properties of a certain behavior in the collision avoidance system to allow us to use a genetic algorithm designed to identify the Pareto frontier of this behavior. By identifying the frontier directly, we identify a major opportunity for improvement in the collision avoidance system. We then show that exploiting this opportunity results in across-the-board improvement for this aspect of aircraft collision avoidance.

The organization of this thesis is as follows:

- Chapter 2 provides an overview of aircraft collision avoidance systems, as well as recent contributions. Particularly, we focus on the formulation of aircraft collision avoidance as a partially-observable Markov decision process. We conclude by examining a global optimization procedure known as surrogate modelling used to optimize the logic.

- Chapter 3 presents our novel approach to preference elicitation. We describe our method in detail and show that it empirically converges faster with respect

to a loss function meaningful in global optimization. We then use this method to direct a surrogate modelling optimization of next-generation aircraft collision avoidance software and show that this resulted in a solution well-tailored to our design goals.

- Chapter 4 discusses our use of the NSGA-II genetic algorithm to optimize the behavior of traffic alerts in the next-generation aircraft collision avoidance system. Using results from this optimization, we propose, implement, and evaluate a change in the traffic alert logic that results in across-the-board improvement.
- Chapter 5 concludes this thesis and discusses opportunities for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Background

2.1 Aircraft Collision Avoidance

Early aircraft collision avoidance relied on the "big sky" principle: there was a lot of airspace available and few aircraft. However, as air traffic increased following the invention of the jet engine, this approach proved no longer feasible. After a collision of two airliners over the Grand Canyon in 1956, federal authorities began development of a last-resort system that would provide guidance in the absence of Air Traffic Control (ATC) [1].

Following a series of partially-effective solutions, TCAS was finally developed in the 1980's and was later mandated to be placed on all large aircraft [1]. TCAS provides both Traffic Alerts (TAs) and Resolution Advisories (RAs). TAs serve to warn the pilot of nearby aircraft, increasing situational awareness. Should the need arise, TCAS then issues the pilot an RA: an instruction to climb or descend at a certain rate. Commands are announced aurally as well as visually as shown in Figures 2-1 and 2-2.

TCAS functions by using a large set of heuristic rules to determine when to issue a TA and, if necessary, which RA to issue to the pilot. Through decades of development, the TCAS logic has become quite extensive and performs remarkably well [1]. However, it is extremely difficult to envision all possible aircraft encounters. In 2002, contradictory advice from an air traffic controller and TCAS system resulted

Figure 2-1: Example TA display and annunciation.

Figure 2-2: Example RA display and annunciation.

in a mid-air collision over Überlingen, Germany [5]. Although TCAS has since been corrected to be able to handle contradictory guidance of the nature provided over Überlingen, no guarantee can be made that another such flaw in TCAS does not exist, as it is impossible to enumerate all possible encounters *a-priori*, let alone pre-

scribe the solution to each one. A more robust solution to aircraft collision avoidance was needed — a system that could dynamically adapt on its own.

Furthermore, the airspace in the 21st century is a much busier place than the airspace for which TCAS was designed. Not only has conventional traffic, such as airliners and general aviation aircraft, increased, but modern airspace must also deal with the presence of Unmanned Aerial Vehicles (UAVs). Collision avoidance for UAVs is a particularly challenging problem, as without a pilot on board, the UAV is entirely dependent on its collision avoidance software [43].

By formulating aircraft collision avoidance as a partially observable Markov decision process (POMDP), research has shown that these goals can be achieved [44]. The inherent robustness in a probabilistic approach such as POMDP has shown across-the-board improvement in simulation over the rule-based methodology of TCAS: the system issues fewer alerts to the pilots while improving overall flight safety [44].

2.2 Partially Observable Markov Decision Processes

POMDPs are a general framework for making decisions under uncertainty [71]. Although a wide range of problems can be modeled as a POMDP [9, 65], solving them is usually computationally infeasible unless the problem has some special structure [65]. Aircraft collision avoidance is one such problem [44].

A POMDP is a generalization of an Markov Decision Process (MDP). In an MDP, the world is modeled as a Markov process [11] that can be in one of several (possibly infinitely many) *states*. In a Markov process, the world transitions between states randomly, where the only restriction is that the transition probabilities hold the Markovian property [11]. But in an MDP, there also exists an *agent*. By performing one of the agent's possible *actions*, the agent can change the state transition probabilities of the Markov process [9, 65].

More formally, an MDP is described as a tuple $\{S, A, T, R\}$. We have that

- S is the set of states of the Markov process.

- A is the set of actions the agent may take.
- T is the transition function that returns transition probabilities based on the system's state and the agent's action.
- R is the *reward* the agent receives based on the system's state and the agent's action.

Furthermore, we define a *policy* to be a mapping from the set of states to the set of actions: more intuitively, a prescription for what action the agent should perform based on the state of the world. The goal of an MDP is to find a policy that maximizes expected future reward.

For example, in aircraft collision avoidance, the state of the world is summarized by five variables [44]:

1. The relative altitude of the aircraft.
2. The vertical rate of the equipped aircraft.
3. The vertical rate of the intruder aircraft.
4. The advisory currently issued to the pilot.
5. The time until horizontal loss of separation.

If the relative altitude is less than 100 feet when the aircraft lose horizontal separation, then a Near Mid-Air Collision (NMAC) is declared to have occurred and the agent receives a large penalty. In order to dissuade excessively alerting the pilot, the agent also receives a small penalty when it issues an advisory to the pilot [44]. Dozens of other rewards and penalties exist to encourage or discourage agent behavior in specific scenarios. For example, the agent receives an especially harsh penalty for issuing a reversal — an advisory that contradicts previous advice, such as telling the pilot to climb five seconds after telling the pilot to descend — as such behavior can undermine pilot faith in the collision avoidance system [44].

An MDP generalizes to a POMDP when the agent can no longer observe the state directly [9, 65]. For example, in aircraft collision avoidance, we cannot calculate

the relative altitude of the aircraft exactly. The aircraft can only estimate its own altitude by using potentially noisy altimeter readings, and must rely on the intruder aircraft accurately broadcasting its estimated altitude. Thus, in contrast to an MDP, where a policy maps states to actions, in a POMDP, a policy must map the set of all observations (usually denoted Ω) to the set of actions.

Solving a POMDP optimally can be very difficult [65, 9], and must often be done so approximately. In ACAS-X, the POMDP is solved approximately as a QMDP [25]. A QMDP can be thought of as a hybrid of an MDP and a POMDP. At each iteration, the expected future reward is calculated for each state the agent may be in, weighted by the probability that the agent is in that state [25]. In some cases, this simplification can result in suboptimal policies, but prior work generally shows this to be an effective technique in the realm of ACAS-X [44].

2.3 Surrogate Modelling

By modifying the magnitude of the rewards used in the POMDP formulation, ACAS-X can perform significantly differently [44, 72, 71]. For example, if we decrease the penalty for alerting relative to the penalty for an NMAC, then the system will likely result in fewer simulated NMACs, but might alert pilots too often to be useful.

The mapping between the POMDP rewards to actual system performance is highly nonconvex [71]. This fact precludes the use of traditional optimization routines, such as gradient descent [71, 79]. Although many nonconvex problems can be optimized satisfactorily using heuristic techniques such as simulated annealing [76], genetic algorithms [70], or particle swarm optimization [57], these methods presume that the objective function is computationally easy to evaluate. This is not the case in ACAS-X, where the only way to evaluate performance is through extensive simulation. Distributed across 512 high-performance compute nodes on a grid, evaluation of a single point takes approximately 20 minutes — this means that a genetic algorithm with a meager population size of 100 would take nearly three months to produce 50 generations.

The reason traditional heuristic optimization techniques fare so poorly with computationally expensive objective functions is that they retain no "memory" of previous iterations [76, 70, 57]. They uphold a Markov-like property where future steps in the optimization are entirely independent of past steps in the iteration, conditioned on the current iteration step. For example, in a genetic algorithm, if the population optimizes out of an area of low fitness, then the only thing preventing the future offspring from being in the area of low fitness is the location of the current generation. It is entirely possible, even likely, that some offspring will return to this unfit area, despite the fact that earlier generations had optimized out of it.

A good strategy for nonconvex optimization for computationally expensive functions would be to keep a history of previous objective function values and use them to somehow direct the optimization. The *surrogate modelling* technique does exactly that. First, it interpolates the existing data with some function. Then, this function is used to determine which point should be tested next. We now examine these steps in turn.

2.3.1 Constructing a Surrogate Model

Suppose X is our input space: the set of all possible rewards we could input into our POMDP formulation. Then, we can view generating our POMDP solution and evaluating it through simulation as a function f that maps X into some objective space Y. For now, we assume that $Y = \mathbb{R}$ (the relaxation of this assumption is the underlying theme of this thesis). Because f is complex and difficult to evaluate, we seek a function f_S that approximates f well, but has desirable properties [33]. This is a quintessential machine learning task.

In machine learning, we have some sort of fixed yet unknown distribution $\mathbb{P}(\mathbf{x}, y)$ that is of interest to us. For example, $\mathbb{P}(\mathbf{x}, y)$ might be the probability distribution that maps images of handwritten digits to the actual digits the author intended to write (for some of us, a more noisy distribution than others). Because calculating $\mathbb{P}(\mathbf{x}, y)$ itself would be a herculean task, we wish to approximate $\mathbb{P}(\mathbf{x}, y)$ in some way [31, 64]. More formally, we seek a f_S that minimizes the *expected risk* $I[f_S]$ of our

approximation:

$$I[f_S] \triangleq \int_{X,Y} V(y, f_S(\mathbf{x}))\mathbb{P}(\mathbf{x}, y)d\mathbf{x}dy \tag{2.1}$$

where $V(y_1, y_2)$ is a relevant loss function [50] and $\mathbb{P}(\mathbf{x}, y)$ represents the probability of the underlying system of interest generating a $(\mathbf{x}, y)$ pair [31, 63, 64]. If we have some space $\mathcal{H}$ of candidate functions for f_S, ideally, we would simply pick $f_S = \arg\min_{f_S \in \mathcal{H}} I[f_S]$ [31, 63, 64].

Although expected risk is the ideal measure of performance, it is impossible to compute directly without knowing $\mathbb{P}(\mathbf{x}, y)$ *a-priori*; if we already knew $\mathbb{P}(\mathbf{x}, y)$, we would not need to approximate it. Consequently, one must instead take n samples from $\mathbb{P}(\mathbf{x}, y)$ and instead use the *empirical risk* $I_S[f_S]$ to measure approximation performance [31, 63, 64]:

$$I_S[f_S] \triangleq \frac{1}{n}\sum_{i=1}^{n} V(y_i, f_S(\mathbf{x}_i)) \tag{2.2}$$

As we evaluate our POMDP formulations at different values for $\mathbf{x} \in X$ and acquire matching performance measures $y \in Y$, we can calculate the empirical risk of any f_S given a loss function using Equation 2.2.

Clearly, if we choose our function hypothesis space $\mathcal{H}$ to be arbitrary, then evaluating Equation 2.2 on every possible $f_S \in \mathcal{H}$ and selecting the arg min is an impossible task [63, 64]. However, by requiring that $\mathcal{H}$ be a Reproducing Kernel Hilbert Space (RKHS) [8, 64], $\arg\min_{f_S \in \mathcal{H}} I_S[f_S]$ will have a special structure [63, 64]. The Representer Theorem [67] states that for any $\mathcal{H}$ that is a RKHS, then the $\arg\min_{f_S \in \mathcal{H}}$ is of the form

$$f_S(\mathbf{x}) = \sum_{i=1}^{n} c_i k(\mathbf{x}, \mathbf{x}_i) \tag{2.3}$$

where $k(\cdot, \cdot)$ is the kernel of our RKHS. For any loss function V, empirical risk can be minimized by selecting the optimal values for the c_i. If V is the square loss function, i.e. $V(y_1, y_2) = (y_1 - y_2)^2$, then the optimal values for c are those that solve the equation [63, 64]

$$\mathbf{K}c = y \tag{2.4}$$

where $\mathbf{K}$ is the Gram matrix of our sample [33], defined as

$$\mathbf{K} \triangleq \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{2.5}$$

In order to leverage the power of Equation 2.4, we elect to use the square loss function to construct our approximation to f that estimates performance based on POMDP rewards. We need only now select the kernel k that uniquely defines our RKHS. Common choices for k include [33]:

- *Linear Kernel:* $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$. This kernel is highly-used for its simplicity. When Equations 2.3 and 2.4 are combined with the linear kernel, they simply become the dual representation of Ordinary Least Squares (OLS) [56, 75].

- *Polynomial Kernel:* $k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$. This kernel allows for more flexibility than the linear kernel, and is often used in natural language processing [37, 48].

- *Gaussian Kernel:* $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{-||\mathbf{x}_1 - \mathbf{x}_2||_2^2}{\sigma^2}\right)$. This flexible kernel is often used for its relation to the Gaussian distribution and its ability to be interpreted probabilistically [33, 75, 64].

- *Kriging Kernel:* $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\sum_{j=1}^{n} \theta_j \left|\mathbf{x}_1^{(j)} - \mathbf{x}_2^{(j)}\right|^2\right)$, where $\mathbf{x}_i^{(j)}$ represents the j^{th} element of the vector $\mathbf{x}_i$. Originally used in geostatistics, the Kriging kernel allows for even more flexibility than the Gaussian kernel while still maintaining its probabilistic interpretation [33].

In keeping with the work of Kyle Smith [71], we use the Kriging kernel to define our RKHS. The values for parameters θ_j could be chosen through internal cross-validation [33], but when there are few data points and an exponentially large number of potential assignments to all the θ_j, this would be computationally infeasible for all but the

smallest models [33]. Instead, we take advantage of the probabilistic interpretation of the Kriging kernel. If the process generating our data is viewed as a Gaussian process, then the likelihood of our model f_S given our kernel is [33]:

$$\hat{\mu} = \frac{\mathbf{1}^\intercal \mathbf{K}^{-1} y}{\mathbf{1}^\intercal \mathbf{K}^{-1} \mathbf{1}} \tag{2.6}$$

$$\hat{\sigma^2} = \frac{(y - \mathbf{1}\hat{\mu})^\intercal \mathbf{K}^{-1} (y - \mathbf{1}\hat{\mu})}{n} \tag{2.7}$$

$$\mathbb{P}(f_S) = -\frac{n}{2}\log(\hat{\sigma^2}) - \frac{1}{2}\log(\det(\mathbf{K})) \tag{2.8}$$

where $\mathbf{1}$ is a vector consisting of 1's. Although Equation 2.8 is nonconvex, it can be optimized through the use of a heuristic optimization technique. Most literature uses a genetic algorithm to perform this optimization [33, 71, 72], so we elected to do the same in order to fit our "surrogate model" f_S.

Figure 2-3 shows a 3D plot of the surrogate model constructed using the maximum liklihood estimates for the kernel parameters σ_i. The samples were collected by varying the POMDP reward for issuing an alert and the reward for issuing a reversal alert, versus a utility function composed of a mixture of NMAC rate, alert rate, and reversal rate. Even for this simple utility function and only two parameters, we notice that the surface is nonconvex.

2.3.2 Exploiting a Surrogate Model

With our surrogate model in hand, we now must decide how to proceed. We have two primary objectives:

- *Exploitation.* Our goal is to find a good solution for our POMDP. Thus, we should pick a point that the surrogate model expects will perform well.

- *Exploration.* Our surrogate model is only an approximation of the underlying behavior. Thus, our new point should be chosen to reduce model approximation and give us a better model for the next iteration.

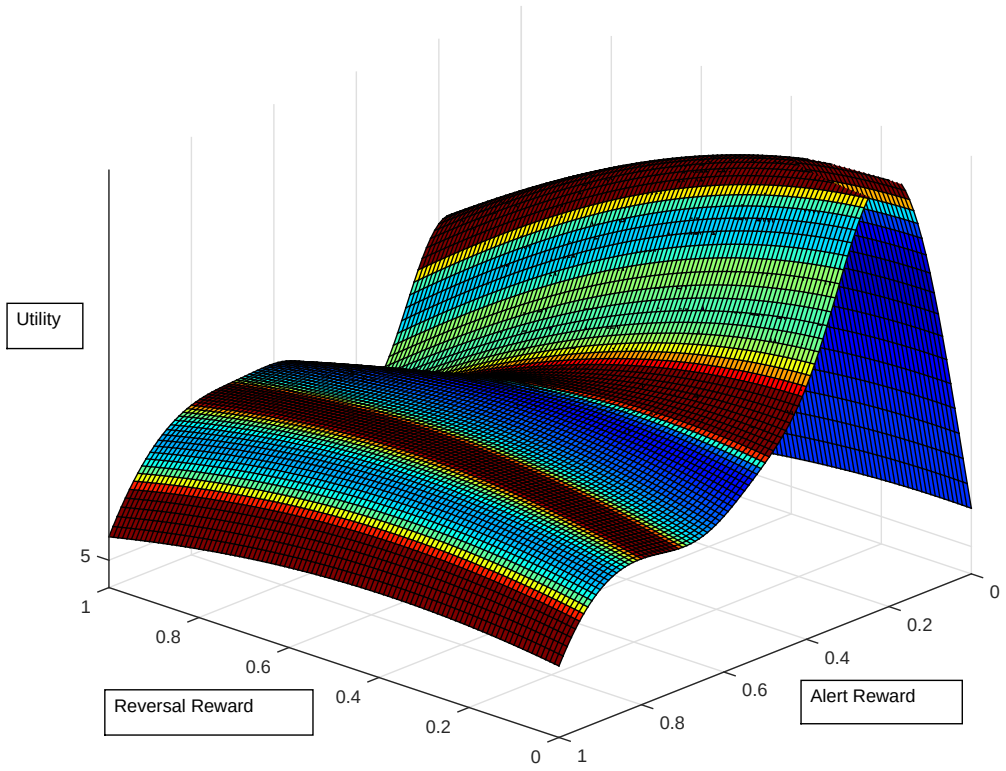

Figure 2-3: Fit of a surrogate model to ACAS data.

In general, these two objectives may not be mutually compatible. Suppose our surrogate model exploration is in a state like that depicted in Figure 2-4. Due to incomplete sampling, our model's optimum differs dramatically from the true optimum. At this state, the two objectives provide divergent goals. We could greedily exploit our existing model, or we could seek to improve the existing model by sampling in new areas.

Furthermore, we can see that performing purely exploration or purely exploitation leads to poor performance. With pure exploration, we may build an excellent model, but never use it. Pure exploitation yields decent results quickly, but is more likely to get stuck in local optima.

Again, we take advantage of our interpretation of the model and underlying truth as a Gaussian process: not only does our model include the estimate for each point,

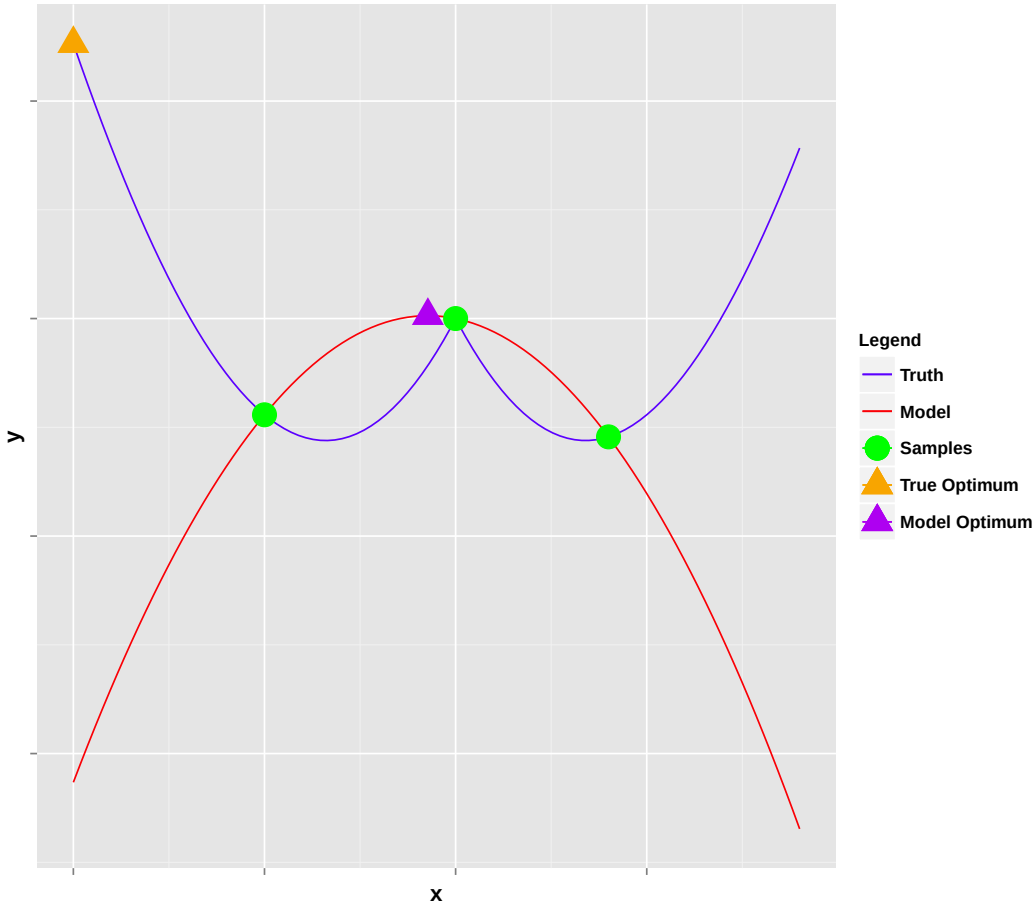

Figure 2-4: A simple surrogate model vs. reality

but we can also calculate the uncertainty of our estimate [33, 66]. If we have samples $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ and we are interested in calculating the uncertainty at a new point $\mathbf{x}$, then we define

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}, \mathbf{x_1}) \\ k(\mathbf{x}, \mathbf{x_2}) \\ \vdots \\ k(\mathbf{x}, \mathbf{x_n}) \end{bmatrix} \tag{2.9}$$

Then, our variance estimate at $\mathbf{x}$ is [66]:

$$\hat{s^2}(\mathbf{x}) = \sigma^2 \left(1 - \mathbf{k}^\intercal \mathbf{K}^{-1} \mathbf{k}\right) \tag{2.10}$$

We can now mathematically define the notions of exploitation and exploration. Exploitation seeks to pick a point that optimizes Equation 2.3, while exploration seeks a point that maximizes Equation 2.10.

Quantifying the uncertainty in our model also allows for sampling strategies that balance exploitation and exploration. One strategy could be to sample the point that has the highest probability of improving upon the existing best solution. However, this strategy can occasionally result in too much exploitation: for example, in Figure 2-4, the point with the highest probability is the model optimum. As we can see, even if the model is correct, this results in only slight improvement over the existing global optimum.

A strategy that takes the magnitude of expected improvement into account would mitigate this problem. If we are minimizing, the expected improvement of a point $\mathbf{x}$ is

$$E(I(\mathbf{x})) = (y_{min} - f_S(\mathbf{x})) \left[\frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{y_{min} - f_S(\mathbf{x})}{\hat{s}(\mathbf{x})}\right)\right] + \frac{\hat{s}(\mathbf{x})}{\sqrt{2\pi}} \exp\left[\frac{-\left(y_{min} - f_S(\mathbf{x})\right)^2}{2\hat{s}^2(\mathbf{x})}\right] \tag{2.11}$$

where erf indicates the error function. By maximizing expected improvement, we can strike a good balance between exploration and exploitation [33, 71]. Although we can easily evaluate Equation 2.11, optimizing it is nontrivial, as the function is nonconvex [33]. Again, we must resort to a genetic algorithm to find points with a high expected improvement.

In summary, one iteration of surrogate modelling proceeds as follows. First, we fit our surrogate model to existing samples by using a genetic algorithm to maximize posterior likelihood. Then, based on our new model, we use another genetic algorithm to select the next sample by maximizing expected improvement. After evaluating this next sample, we add it to our set of samples and repeat.

An astute reader will notice that we have ignored how one begins this loop; we have always assumed the existence of a sample. In theory, we could simply choose our first point at random. In practice, however, one often sees better performance using a space-filling design [33], such as Latin hypercubes [33, 74] or Sobol sequences

[19]. We adopted the former to remain consistent with existing literature [72, 71, 33].

Furthermore, we have thus far assumed that our output space is simply $\mathbb{R}$; that is, that we only have one objective we wish to optimize. The remainder of this thesis concerns itself with how to generate good solutions when this assumption is relaxed.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Preference Elicitation

3.1 Introduction

Making trade-offs between multiple objectives is fundamental to the engineering design process. A designer may want to optimize metrics such as cost, reliability, and performance of a system, but often an improvement in one objective must come at the expense of another. There are several different approaches one may take in multi-objective design optimization [7]. One approach is to generate what is known as the Pareto frontier, the collection of non-dominated points in the design space [27]. There are a wide variety of algorithms for finding points on the frontier [80, 28], such as the NSGA-II genetic algorithm [26]. However, although we can generate points on the Pareto frontier in polynomial time [26], the size of the Pareto frontier expands exponentially with the number of design objectives. This expansion means that we would need to generate exponentially many points to achieve the same level of coverage on the Pareto frontier [53]. To compound matters, these algorithms tend to presume that the objective function is computationally easy to evaluate [26, 7, 34]. In engineering design optimization, the objective function may be a computationally expensive simulation that cannot realistically be evaluated a large number of times. Recent work has mitigated some of the problems induced by an exponentially large Pareto frontier by using interactive preference elicitation to dynamically determine which area of the Pareto frontier is of most interest to the designer [23, 22]. Never-

theless, the computational burden of these methods—combined with the exponential growth of the number of non-dominated points—can make generating the Pareto frontier impractical [53].

A different approach is to adopt a heuristic quantitative criterion for defining a single best Pareto point. For example, a method called goal programming involves minimizing the distance between the objective measures of the design and the ideal objective measures [45]. One disadvantage of this approach is that it is often unclear what distance measure and choice of ideal point is appropriate. Another approach, called the ϵ-constraint method, constrains all but one of the objectives to be within some ϵ of their ideal value and then optimizes the remaining objective [34]. However, it is often far from clear what the ideal value and the corresponding ϵ for each objective should be [34].

If we are unable to apply the previous approaches, we can always define a utility function and then apply one of the many different single-objective optimization algorithms [7]. This approach is intuitive and easy to implement. However, it is difficult to define a meaningful utility function. The simplest approach is to define the utility function as a weighted sum of the individual metrics, but the designer must still specify the exact trade-off ratios between metrics. For example, when optimizing ACAS-X performance, we want to minimize both the number of collisions and the number of nuisance alerts [47]. Clearly, minimizing the number of collisions is more important than minimizing the number of nuisance alerts, but we must specify the exact trade-off ratio between the two. This is a difficult decision to make, as many values may appear appropriate—yet the optimization routine may return drastically different results depending on the choice.

One approach to creating a utility function is algorithmic preference elicitation [18]. The designer still must specify the functional form of the utility function, but instead of having to make the difficult decision of determining the parameters of the utility function directly, the designer answers a series of smaller, easier questions, such as "Do you prefer design X or design Y?" The algorithm then chooses the ideal choice of parameters for the designer. In practice, a user would start with a baseline

set of design parameters, run the optimization for a period, and then use preference elicitation on the existing solutions to determine the ideal parameters for use in the remainder of the optimization. This optimize-elicit-optimize loop has been previously used successfully in ant-colony optimization of assembly line balancing [23, 22].

In the following sections, we review existing algorithms for preference elicitation, and how they can be applied to engineering design optimization. We then introduce our own method for preference elicitation, specifically tailored for use in engineering design optimization. Finally, we apply our algorithm to optimization of ACAS-X.

3.2 Literature Review

Algorithms for preference elicitation have existed since the 1970's [69]. Generally, these early algorithms were viewed as an extension of learning theory [4], and were primarily of academic interest. In the 1990's, the increasing capability of computers resulted in a renewed interest for preference elicitation [61]. Querying consumers for preferences could result in better-tailored products and marketing, and preference elicitation could also be used as an aid to help decision makers quantify their notions of value.

Formulating an algorithm for preference elicitation generally requires two steps. First, a model for preference realization must be developed—the mechanism that the user applies to make decisions. Once this has been accomplished, this model can be inverted to infer the user's preferences from only observing the user's decisions.

Models for preference realization are usually derived from Multiattribute Utility Theory (MAUT) [42, 29]. MAUT is a relatively mature field that concerns itself with how people value trade-offs between multiple, competing objectives. Common concepts taken from MAUT are:

1. *Transitivity of preferences.* If someone prefers X to Y (denoted $X \succ Y$), and $Y \succ Z$, then $X \succ Z$ [29].

2. *Reflexive and transitive indifference.* If someone is indifferent between X and

Y (denoted $X \sim Y$) and $Y \sim Z$, then $X \sim Z$. Furthermore, $X \sim X$ [29].

3. *Trichotomy of preferences.* Either $X \succ Y$, $Y \succ X$, or $X \sim Y$ [29].

4. *Existence of Utility Function.* There exists a utility function $u(\cdot) : X \to \mathbb{R}$. Because of the properties of the real numbers, this presumes that the concepts 1-3 have been adopted [42].

5. *Additive utility.* Suppose $u(X)$ is the utility of X, where X has attributes $x_1, x_2, \ldots, x_n$. Then, the utility of X can be decomposed into $u(x) = \sum_{i=1}^{n} u_i(x_i)$, where $u_i(x_i)$ are marginal utility functions that depend only on individual attributes [29].

From these axioms, one can build a model for preference realization. For example, a common approach is to assume a simple linear utility function: if $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$, then $u(\mathbf{x}) = \mathbf{p}^\intercal \mathbf{x}$. Then, the task of preference elicitation is to simply learn the vector $\mathbf{p}$ that best matches the preferences of the user [69, 38]. Another common approach is to make the utility function noisy—this allows for the model to handle inconsistent or contradictory preferences, such as $\{X \succ Y, Y \succ Z, Z \succ X\}$ [69, 38].

Specifying the model is a nontrivial task, as there exists a natural tension between the modelling and inference steps. On one hand, the algorithm designer wants the preference realization model to be complex, in order to be able to explain as much of the user's decisions as possible. However, as the model becomes more complex, the inference step becomes more computationally difficult. Figure 3-1 shows how existing algorithms for preference elicitation fall on this spectrum.

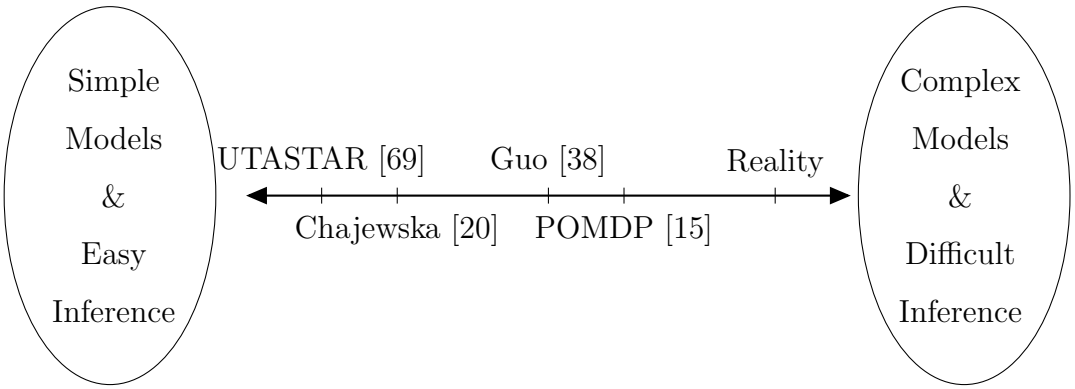

Figure 3-1: Spectrum of preference elicitation methods.

The strengths and drawbacks of the methods in Figure 3-1 will be discussed individually in the following sections.

3.2.1 Linear Programming Methods

An intuitive—and very widely-used—scheme for preference elicitation is the UTASTAR algorithm [69]. This algorithm relies on linear programming to determine the optimal utility function, with preferences modeled as soft constraints. The algorithm proceeds as follows.

First, we assume we have a set A consisting of m solutions, each having n observed metrics. For each $i \in \{1...n\}$, we rank the observations in A of the i^{th} metric from best to worst:

$$G_i \triangleq \left\{ x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)} \right\} \tag{3.1}$$

where $x_j^{(i)}$ is the j^{th} best observation of metric i found in the set A. We then define our decision variables $w_l^{(i)}$ to be

$$w_l^{(i)} \triangleq u_i \left(x_{l+1}^{(i)} \right) - u_i \left(x_l^{(i)} \right) \tag{3.2}$$

where $u_i(\cdot)$ is the marginal utility function for metric i. Then, by recursion, we have

that

$$u_i\left(x_j^{(i)}\right) = \sum_{l=1}^{j-1} w_l^{(i)} \tag{3.3}$$

and we set $u_i\left(x_1^{(i)}\right) = 0 \ \forall i \in \{1...n\}$. Finally, we let $u(\mathbf{x_j}) \triangleq \sum_i^n u_i\left(x_j^{(i)}\right)$. This means our marginal utility functions are piecewise linear with the absiccas of the corners at the observed values. Our overall utility function is the sum of all the marginal utility functions.

We then gather a set of pairwise comparison preferences from the user by posing queries like "Do you prefer X or Y?" Once this set has been obtained, we then simply formulate the preference elicitation problem as a linear program:

$$\begin{aligned}
&\min_{w_l^{(i)}} \sum_{j=1}^{m} \left(\sigma_j^+ + \sigma_j^-\right) \\
&\text{subject to:} \\
&u(\mathbf{x_i}) + \sigma_i^+ - \sigma_i^- - \left(u(\mathbf{x_j}) + \sigma_j^+ - \sigma_j^-\right) > 0 \\
&\left|u(\mathbf{x_i}) + \sigma_i^+ - \sigma_i^- - \left(u(\mathbf{x_j}) + \sigma_j^+ - \sigma_j^-\right)\right| < \epsilon \\
&\forall \mathbf{x_i} \succ \mathbf{x_j}, \mathbf{y_i} \sim \mathbf{y_j} \\
&w_l^{(i)} \geq 0 \ \ \forall i \in \{1 \ldots n\}, l \in \{1 \ldots j-1\} \\
&\sum_i \sum_l w_l^{(i)} = 1
\end{aligned} \tag{3.4}$$

where σ_i^+ and σ_i^- are the positive an negative slack variables for each observation.

The formulation in Equation 3.4 is simple, powerful, and very useful in many applications. It also doesn't require prior specification of the functional form of $u(\cdot)$.

However, the formulation in Equation 3.4 does suffer from serious drawbacks. First, the objective function created is difficult to conceptualize, meaning it will be heuristically difficult to detect when the algorithm has converged. Second, the objective function is not smooth, which is problematic for many numerical optimization techniques. Furthermore, the objective function is not defined for observations outside of the observed ranges of A. We want to feed this utility function into an optimization routine—if our optimization method is doing its job, we will likely encounter

such cases. Finally, the formulation in Equation 3.4 is unable to incorporate prior knowledge, meaning the algorithm will have to learn things that the designer may already know. For example, we may not know the exact tradeoff ratio between NMACs and RAs, but we do know that we want to minimize the number of NMACs.

As we can see, the majority of the issues with the UTASTAR algorithm stem from its attempt to proceed without first specifying the functional form of $u(\cdot)$. If we were to force $u(\cdot)$ to be strictly linear instead of using linear splines approach above, then the formulation in Equation 3.4 would simply reduce to

$$\begin{aligned} &\min_{\mathbf{p}} \sum_{j=1}^{m} \left(\sigma_j^+ + \sigma_j^-\right) \\ &\text{subject to:} \\ &||\mathbf{p}||_1 = 1 \\ &\mathbf{p}^\intercal \mathbf{x_i} + \sigma_i^+ - \sigma_i^- - \left(\mathbf{p}^\intercal \mathbf{x_j} + \sigma_j^+ - \sigma_j^-\right) < 0 \\ &\left|\mathbf{p}^\intercal \mathbf{y_i} + \sigma_i^+ - \sigma_i^- - \left(\mathbf{p}^\intercal \mathbf{y_j} + \sigma_j^+ - \sigma_j^-\right)\right| < \epsilon \\ &\forall \mathbf{x_i} \succ \mathbf{x_j}, \mathbf{y_i} \sim \mathbf{y_j} \end{aligned} \tag{3.5}$$

We can also incorporate priors into the formulation in Equation 3.5 by adding a term into the objective that penalizes the algorithm when it deviates from some prior estimate of $\mathbf{p}$. We can also reformulate Equation 3.5 as a quadratic program by changing our objective function to $\sum_{j=1}^{m} \left(\sigma_j^+ + \sigma_j^-\right)^2$ to ensure that our solution will be unique.

However, the formulation in Equation 3.5 is problematic even when reformulated as a quadratic program with the priors incorporated. The primary concern comes from the notion of optimization duality. In both of these formulations, a constraint will affect the solution if and only if it is binding at the optimum. In other words, a preference will only change $\mathbf{p}$ if it is strictly inconsistent with the currently optimal $\mathbf{p}$. This is concerning, as it means that the algorithm isn't using any information from some of our preferences. Consequently, linear and quadratic programming for-

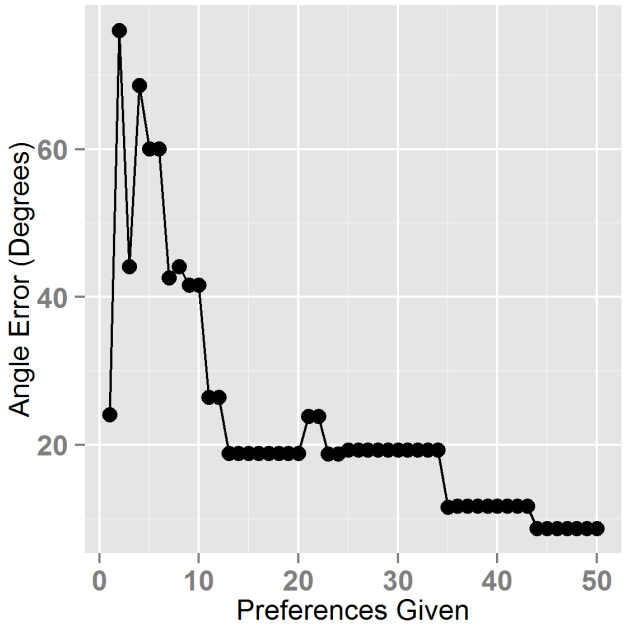

Figure 3-2: A typical convergence pattern for linear programming formulations.

mulations for preference elicitation cannot be optimal from an information-theoretic standpoint.

Figure 3-2 shows a typical convergence pattern of linear programming formulations. As we feed the algorithm more preferences, the angle between our estimated weight vector $\hat{\mathbf{p}}$ and the true angle vector eventually decreases to zero. However, we see that only a small percentage of the preferences decrease the error—the majority of the preferences do not change the estimate for $\mathbf{p}$. This phenomenon occurs because unless the new preference is strictly inconsistent with the current optimum, the optimal basis will remain the same.

Another problem lies in how we incorporated our prior knowledge. We are forced to artificially strike a balance between the feasibility portion of our objective and the prior portion of our objective, which can only be done by multiplying one of the two objectives by a coefficient and then manually tuning this tradeoff coefficient. But then we are back to the problem at which we started: we are trying to tune a sensitive value based on intuition—except now, we have far less of an intuition for what our coefficient should be.

3.2.2 Bayesian Methods

The UTASTAR algorithm views parameters of the utility function as fixed yet unknown constants. A natural step would be to relax this assumption and instead adopt a Bayesian mentality: parameters are viewed as being drawn from probability distributions.

Myopic Bayesian Strategies

Chajewska et al applied Bayesian methodology successfully for pre-natal consoling [20]. Their method assumes the existence of a set of outcomes $\mathbf{U} = \langle u_1, u_2, \ldots, u_n \rangle$, and the goal of the utility function is to infer the best outcome for the user. Each outcome is endowed with a univariate normal distribution, representing the algorithms belief of the user's utility for that outcome.

These distributions are updated by *standard gamble* queries to the user [78]. Standard gamble queries are a generalization of simple pairwise comparisons. Instead of simply being asked for preference between X and Y, an element of chance is incorporated. Queries are generally of the form, "Would you prefer X with 100% certainty, or Y with 50% certainty?" These queries are more flexible than simple pairwise comparisons, as they allow for queries that determine the exact ratio between the utilities of X and Y. However, this comes at a cost of a higher cognitive burden to the user [24], meaning responses will likely be noisier. Nevertheless, using Monte Carlo methods and properties of the normal distribution, these queries can be used to update the distribution on $\mathbf{U}$ efficiently [20].

Chajewska et al then propose a query strategy based on *expected posterior utility*. Expected posterior utility is "the average of the expected utilities arising from the two possible answers to the [query], weighted by how likely these two answers are" [20]. This also provides a natural way to define the value of the information encoded in the query, by simply taking the difference between the expected posterior utility and the current posterior utility.

Sanner and Guo [38] extend upon this work in several key ways. First, noting the

issue with the high cognitive burden of standard gable queries, they use only pairwise comparisons. Then, due to the nature of pairwise comparisons, they are able to use expectation propagation [55] instead of Monte Carlo methods to perform inference. Finally, the computational time saved by expectation propagation allows them to implement several different heuristics for query generation which can in some cases converge quickly to the user's true utility [38].

POMDP

The myopic approach for Bayesian preference elicitation is troubling, as greedy optimization can easily get stuck in local optima. Boutillier proposes formulating preference elicitation as a POMDP to generate an query strategy that is optimal in a long-term sense [15].

Formulation of the POMDP is simple [15]. The underlying state is the user's actual utility function. The actions available to the POMDP agent are the queries that it can postulate to the user, as well as the decision to stop eliciting preferences. The system has an infinite horizon with a discount factor. The POMDP agent receives a reward based on the utility of its state estimate when the decision to stop eliciting preferences is made.

Although the formulation is relatively simple, the implementation is very difficult. First, the action space is continuous, which requires optimization (such as gradient ascent) to even choose the optimal policy once the POMDP has been solved. Next, the POMDP has an infinite horizon, meaning solving the POMDP will have to address the asymptotic behavior of the system. However, the greatest issue is that the value function does not exist in closed form and is not in general piecewise linear convex [15]. This observation means that that standard methods for solving the POMDP cannot be used [15]. Approximating the value function via feedforward neural networks or grid-based models can allow the POMDP to be solved approximately [15], although this problem compounds with the complications of having a continuous action space and an infinite horizon.

Boutiller tested his framework on several problems of relatively small size. Gener-

ating an optimal preference elicitation policy for choosing between seven items took over seven hours of offline computation [15]. Furthermore, the optimized system generated would "often [get] stuck in loops or ask nonsensical questions" due to the inexactness of the value function approximation [15]. These problems, compounded with the fact that Boutiller's framework presupposes an existing set of solutions, make it impractical for use in our engineering design optimization.

Summary of Literature Review

As we have seen, although linear programming is a simple methodology that can converge to a user's utility function, its lack of robustness, difficulty to incorporate prior knowledge, and slow convergence are problematic. Existing Bayesian methods are useful in some domains, but they are impractical for our use in engineering design optimization, as they assume a set of solutions already exist. During our optimization, we only have a partial list of solutions, and simply assigning accurate utility to these existing solutions is not helpful. Instead, we seek to learn our utility function directly—a task that Chajewska et al claim to be "virtually impossible" [20].

3.3 Our Method

We assume that we know the functional form of our utility function $u(\cdot)$, but do not know some vector of parameters $\mathbf{p}$. We can let $u(\mathbf{x}) = \mathbf{p}^\intercal \mathbf{x}$, but we need not do so in the general case. Our preference elicitation scheme proceeds as follows:

1. From a family of examples, the expert is shown two examples, and states a preference between the two. For a query between items $\mathbf{x}$ and $\mathbf{y}$, the expert's response can be "I prefer $\mathbf{x}$ to $\mathbf{y}$" (denoted $\mathbf{x} \succ \mathbf{y}$), indifference between $\mathbf{x}$ and $\mathbf{y}$ (denoted $\mathbf{x} \sim \mathbf{y}$), or $\mathbf{y} \succ \mathbf{x}$. This response has been shown to pose a minimal cognitive burden on the user [24].

2. Based on this response, we update $\mathbf{p}$ appropriately.

3. Using the new estimate for $\mathbf{p}$, we select two new examples to pose to the expert and repeat this process until a stopping criterion is met.

This strategy is exactly the same as those found in the UTASTAR algorithm [69] and in the Bayesian model proposed by Guo and Sanner [38]. Our method differs from existing work in the way it updates its estimate for $\mathbf{p}$ and how it determines new queries to pose to the expert.

3.3.1 Model

We utilize a Bayesian methodology. We model our belief about $\mathbf{p}$ as a probability distribution and update this belief based on the preferences we observe. Figure 3-3 shows a factor graph [14] of our preference realization model.

We assume that $\mathbf{p}$ is distributed according to a multivariate normal with mean $\boldsymbol{\mu}$ and diagonal covariance matrix $\boldsymbol{\Sigma}$. When an expert realizes a preference, we assume they draw a value for $\mathbf{p}$ from its corresponding distribution, and then use this realization of $\mathbf{p}$ deterministically in our objective function to establish preference. Our goal in preference elicitation is to find an estimate for $\boldsymbol{\mu}$ that best fits our observations of preference. This model is similar in spirit to that presented by Sanner and Guo [38], but instead of learning the value of discrete attributes, we instead learn the parameters of our utility function.

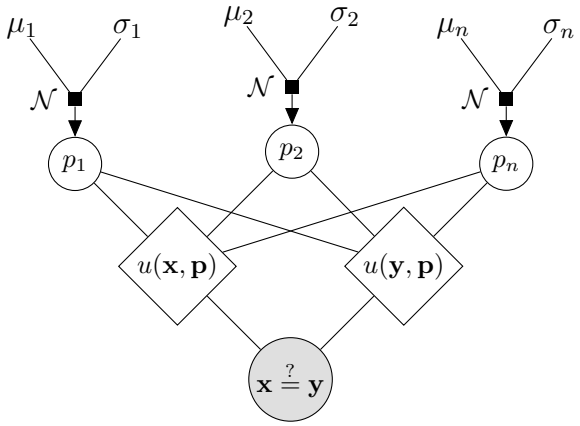

Figure 3-3: A factor graph for our model of preference realization.

3.3.2 Inference

Inference on Figure 3-3 is difficult, as we can only observe $\mathbf{p}$ indirectly through realizations of preference. Prior work uses expectation propagation [55] to exploit the graphical nature of the problem and perform efficient inference [38, 39]. However, this approach may not be desirable for engineering design optimization for several reasons. First, existing implementations of expectation propagation on this graph require the posterior of $\boldsymbol{\mu}$ to be a multivariate normal with a diagonal covariance matrix [38, 39]. Although this restriction can result in an approximate solution quickly, in the context of engineering design optimization, we are willing to wait a few more seconds for more accurate results. Furthermore, even if the posterior were to actually be a diagonal covariance multivariate normal distribution, expectation propagation would still only converge to an approximate solution for inference problems of this structure [38, 39, 55].

Instead of expectation propagation, we will analytically derive our posterior up to a normalizing constant. We let $\mathcal{D}$ denote our data—the set of preferences the expert has given us. From Bayes' Rule, we have that

$$\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathcal{D}) \propto \mathbb{P}(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.6}$$

The $\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ term is simply our prior, generally chosen to be a maximum entropy distribution given our knowledge of the parameter. For example, we often have a guess for the value of each element of $\boldsymbol{\mu}$ and know its sign, so an exponential prior on each element is a natural choice. The likelihood term differs from prior work in an important way. Previous approaches add preferences one at a time. The prior is updated using the preference as likelihood, and then the posterior is used as the prior for the next preference [20, 38]. This cycle forces the authors to use approximation algorithms, as the prior for the second preference is no longer Gaussian [38, 39]. Our approach avoids this problem by viewing the *set* of preferences as our data. Thus, as long as we are able to calculate our likelihood term, we can add a new preference by incorporating it into our data and computing our posterior from scratch using our

original prior.

We can evaluate the likelihood term in two different ways. First, we present an approach that exploits properties of linear functional forms of $u(\cdot)$ in order to perform efficient inference. We then discuss a second approach which uses numerical integration techniques to perform inference on nonlinear utility functions.

Linear Utility Functions

Suppose we have a linear utility function $u(\mathbf{x}) = \mathbf{p}^\intercal \mathbf{x}$. We know that if $\mathbf{x} \succ \mathbf{y}$, then

$$\mathbf{p}^\intercal \mathbf{x} > \mathbf{p}^\intercal \mathbf{y} \Rightarrow \mathbf{p}^\intercal (\mathbf{x} - \mathbf{y}) > 0 \tag{3.7}$$

In our model, $\mathbf{p} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. To make the following steps easier to follow, we let $\tilde{\mathbf{p}} \triangleq \mathbf{p} - \boldsymbol{\mu}$, which results in $\tilde{\mathbf{p}} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. Equation 3.7 then becomes

$$\tilde{\mathbf{p}}^\intercal (\mathbf{x} - \mathbf{y}) > -\boldsymbol{\mu}^\intercal (\mathbf{x} - \mathbf{y}) \tag{3.8}$$

In order to calculate $\mathbb{P}(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$, we must find the probability that Equation 3.8 is true. For example, if $\mathbf{x} - \mathbf{y} = \langle 1, 4 \rangle$, then this probability is the probability mass in the shaded region of Figure 3-4.

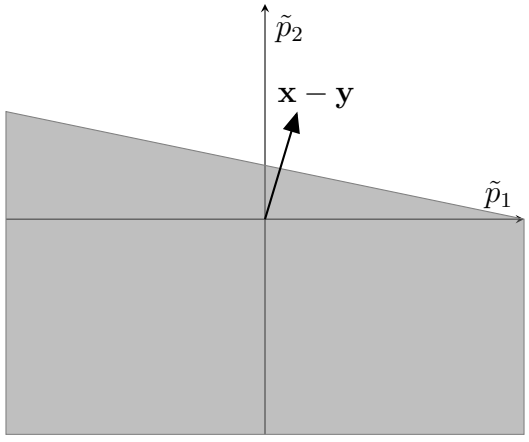

Figure 3-4: Region for which Equation 3.8 is true.

In order to integrate over the shaded region in Figure 3-4, we exploit the fact that the marginal distributions in any direction of multivariate normal distributions are

themselves univariate normal. The variance along the $\mathbf{x} - \mathbf{y}$ direction is

$$\sigma^2_{\mathbf{x}-\mathbf{y}} = (\mathbf{x} - \mathbf{y})^\intercal \boldsymbol{\Sigma} (\mathbf{x} - \mathbf{y}) \tag{3.9}$$

and the distribution has a mean of zero. Consequently,

$$\mathbb{P}\left(\tilde{\mathbf{p}}^\intercal(\mathbf{x} - \mathbf{y}) > -\boldsymbol{\mu}^\intercal(\mathbf{x} - \mathbf{y})\right) = \Phi\left(\frac{\boldsymbol{\mu}^\intercal(\mathbf{x} - \mathbf{y})}{\sigma^2_{\mathbf{x}-\mathbf{y}}}\right) \tag{3.10}$$

where $\Phi(\cdot)$ is the normal cumulative distribution function. Equation 3.10 is thus the probability that $\mathbf{x} \succ \mathbf{y}$. Our model specifies that realizations of $\mathbf{p}$ are independently drawn from their distributions. If we let $\mathcal{D}_\succ$ represent the subset of our preferences that are strict, then

$$\mathbb{P}(\mathcal{D}_\succ \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{(\mathbf{x},\mathbf{y})|\mathbf{x}\succ\mathbf{y}} \Phi\left(\frac{\boldsymbol{\mu}^\intercal(\mathbf{x} - \mathbf{y})}{\sigma^2_{\mathbf{x}-\mathbf{y}}}\right) \tag{3.11}$$

Equation 3.11 can be easily modified to account for indifference preferences. Instead of requiring that $\mathbf{p}^\intercal\mathbf{x} > \mathbf{p}^\intercal\mathbf{y}$, we simply require that $\mathbf{p}^\intercal\mathbf{x}$ and $\mathbf{p}^\intercal\mathbf{y}$ be within some ϵ of each other. After some experimentation, setting $\epsilon = \sigma^2_{\mathbf{x}-\mathbf{y}}/2$ has yielded good results, which are shown in the empirical experiments later in this thesis. Using this value for ϵ, the probability of our indifference preferences $\mathcal{D}_\sim$ is

$$\mathbb{P}(\mathcal{D}_\sim \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{(\mathbf{x},\mathbf{y})|\mathbf{x}\sim\mathbf{y}} \left(\Phi\left(\frac{\boldsymbol{\mu}\cdot(\mathbf{x} - \mathbf{y})}{\sigma^2_{\mathbf{x}-\mathbf{y}}} + 0.5\right) - \Phi\left(\frac{\boldsymbol{\mu}\cdot(\mathbf{x} - \mathbf{y})}{\sigma^2_{\mathbf{x}-\mathbf{y}}} - 0.5\right)\right) \tag{3.12}$$

Because $\mathcal{D}_\succ \bigcup \mathcal{D}_\sim = \mathcal{D}$, we can combine Equations 3.11 and 3.12 to form our likelihood function:

$$\mathbb{P}(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbb{P}(\mathcal{D}_\succ \mid \mu, \boldsymbol{\Sigma})\mathbb{P}(\mathcal{D}_\sim \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3.13}$$

We now know our posterior up to a normalizing constant, so we can use Markov Chain Monte Carlo (MCMC) to sample directly from the posterior [54]. If $\mathbf{p}$ has low dimensionality, the Metropolis algorithm will generally give the best samples. As the dimensionality increases, Gibb's sampling may be more effective. [35] The efficiency

of both methods is—as usual—dependent on the data. In our experience, we have seen good performance with the Metropolis algorithm for up to a twelve-dimensional $\mathbf{p}$, as demonstrated later in our application section. Alternatively, if we only want a point-estimate of $\boldsymbol{\mu}$, we can use an optimization algorithm to solve for the maximum a-posteriori (MAP) estimate. If we let $\mathbf{d} = \mathbf{x} - \mathbf{y}$, the μ_i partial derivatives of the natural logarithm of Equation 3.6 are

$$\frac{\partial \log(\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathcal{D}))}{\partial \mu_i} = \frac{\partial \log(\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma}))}{\partial \mu_i} + \left(\sum_{(\mathbf{x},\mathbf{y})|\mathbf{x}\succ\mathbf{y}} \frac{\mathbf{d}_i}{\sigma^2_{\mathbf{x}-\mathbf{y}}} \frac{\phi\left(\frac{\boldsymbol{\mu}\cdot\mathbf{d}}{\sigma^2_{\mathbf{x}-\mathbf{y}}}\right)}{\Phi\left(\frac{\boldsymbol{\mu}\cdot\mathbf{d}}{\sigma^2_{\mathbf{x}-\mathbf{y}}}\right)} \right) + \left(\sum_{(\mathbf{x},\mathbf{y})|\mathbf{x}\sim\mathbf{y}} \frac{\mathbf{d}_i}{\sigma^2_{\mathbf{x}-\mathbf{y}}} \frac{\phi\left(\frac{\boldsymbol{\mu}\cdot\mathbf{d}}{\sigma^2_{\mathbf{x}-\mathbf{y}}} + 0.5\right) - \phi\left(\frac{\boldsymbol{\mu}\cdot\mathbf{d}}{\sigma^2_{\mathbf{x}-\mathbf{y}}} - 0.5\right)}{\Phi\left(\frac{\boldsymbol{\mu}\cdot\mathbf{d}}{\sigma^2_{\mathbf{x}-\mathbf{y}}} + 0.5\right) - \Phi\left(\frac{\boldsymbol{\mu}\cdot\mathbf{d}}{\sigma^2_{\mathbf{x}-\mathbf{y}}} - 0.5\right)} \right) \tag{3.14}$$

where $\phi(\cdot)$ designates the probability density function of the normal distribution. As long as the log-priors on $\boldsymbol{\mu}$ are partially-differentiable, then we can use Equation 3.14 in a gradient ascent method, such as BFGS [79], to solve for the MAP estimate to arbitrary precision.

For computational purposes, it is often prudent to perform inference only on $\boldsymbol{\mu}$ by assuming $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix and σ^2 is a fixed constant. This assumption can result in better convergence, and empirically we have found that the model is not very sensitive to the choice of σ^2. If σ^2 is relatively small, the algorithm converges somewhat faster if there are no inconsistent preferences, but somewhat slower otherwise.

In fact, if we fix $\boldsymbol{\Sigma}$ as described above, then we can prove that the posterior is log-concave for log-concave priors. Our proof begins with several lemmas.

Lemma 1. *If $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix, and $B \in \mathbb{R}^{n\times n}$ is a matrix consisting of ones, then the product DBD is positive semidefinite.*

Proof. If we let $d_i \mid i \in \{1 \dots n\}$ represent the elements on the diagonal of D, then

the product DBD is of the form

$$DBD = \begin{pmatrix} d_1^2 & d_1 d_2 & \cdots & d_1 d_n \\ d_2 d_1 & d_2^2 & \cdots & d_2 d_n \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 & d_n d_2 & \cdots & d_n^2 \end{pmatrix} \tag{3.15}$$

By inspection, we see that vectors of the form $v_i = \langle -d_i, 0, \ldots, 0, d_1, 0, \ldots \rangle$, where the d_1 is the ith element, are linearly independent vectors, all with eigenvalues of zero. There are $n-1$ of these eigenvectors. The nth eigenvector is $\langle d_1, d_2, \ldots, d_n \rangle$, which by inspection has a corresponding eigenvalue of $\sum_i d_i^2 \geq 0$. Thus, because an $n \times n$ matrix has exactly n eigenvalues, we have found them all, and all are non-negative. Therefore, by definition DBD is positive semidefinite. □

Lemma 2. *The function* $\Phi(x+a) - \Phi(x-a)$ *is log-concave for all* $a \in \mathbb{R} \mid a \geq 0$.

Proof. We know that $\phi(x)$ is log-concave[6]. Furthermore, we know that integrals of log-concave functions are log-concave and products of log-concave functions are themselves log-concave[59]. We let

$$g(z) = \begin{cases} 1 & \text{if } x - a \leq z \leq x + a \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

We know that $g(z)$ is log-concave because it is an indicator function of a convex set. Thus, $\phi(z)g(z)$ is log-concave and

$$\int_{\mathbb{R}} \phi(z)g(z)dz = \int_{x-a}^{x+a} \phi(z)dz = \Phi(x+a) - \Phi(x-a) \tag{3.17}$$

is also log-concave. □

If we let $\mathbf{d} = \mathbf{x} - \mathbf{y}$, then the likelihood of our strict preferences is

$$\mathbb{P}(\mathcal{D}_{\succ} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{(\mathbf{x},\mathbf{y})|\mathbf{x}\succ\mathbf{y}} \Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2}\right) \tag{3.18}$$

Lemma 3. *The likelihood of our strict preferences is log-concave.*

Proof. We take take the logarithm of Equation 3.18, yielding

$$\log(\mathbb{P}(\mathcal{D}_{\succ} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})) = \sum_{(\mathbf{x},\mathbf{y})|\mathbf{x}\succ\mathbf{y}} \log\left(\Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2}\right)\right) \tag{3.19}$$

We let $H(\log(\Phi))$ be the Hessian matrix of one of the terms in the sum of Equation 3.19. We have that

$$H(\log(\Phi)) = \frac{f(\boldsymbol{\mu})}{\sigma_d^2} \begin{pmatrix} d_1^2 & d_1 d_2 & \cdots & d_1 d_n \\ d_2 d_1 & d_2^2 & \cdots & d_2 d_n \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 & d_n d_2 & \cdots & d_n^2 \end{pmatrix} \tag{3.20}$$

where

$$f(\boldsymbol{\mu}) = \frac{-\phi^2(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2})}{\Phi^2(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2})} + \frac{-\boldsymbol{\mu}\phi(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2})}{\Phi(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2})} \tag{3.21}$$

We know that Equation 3.21 must be non-positive because it is of the same form as the second derivative of the logarithm of $\Phi(x)$, which is known to be log-concave[6].

By Lemma 1, the matrix in equation 3.20 must be positive semidefinite. Because it is being multiplied by a nonpositive scalar, $H(\log(\Phi))$ must be negative semidefinite. Therefore, the Hessian of Equation 3.19 is a sum of negative semidefinite matrices, which must also be negative semidefinite. Thus, the likelihood of our strict preferences is log-concave. □

Lemma 4. *The likelihood of our indifference preferences is log-concave.*

Proof. We proceed in similar fashion as above. The logarithm of our indifference preferences is

$$\log(\mathbb{P}(\mathcal{D}_{\sim} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})) = \sum_{(\mathbf{x},\mathbf{y})|\mathbf{x}\sim\mathbf{y}} \log\left(\Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} + 0.5\right) - \Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} - 0.5\right)\right) \tag{3.22}$$

If we let $\log(\Delta\Phi)$ represent one of the terms in our sum, we have that

$$H(\log(\Delta\Phi)) = \frac{g(\boldsymbol{\mu})}{\sigma_d^2}\begin{pmatrix} d_1^2 & d_1 d_2 & \cdots & d_1 d_n \\ d_2 d_1 & d_2^2 & \cdots & d_2 d_n \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 & d_n d_2 & \cdots & d_n^2 \end{pmatrix} \tag{3.23}$$

where

$$g(\boldsymbol{\mu}) = \frac{-\left(\phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} + 0.5\right) - \phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} - 0.5\right)\right)^2}{\left(\Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} + 0.5\right) - \Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} - 0.5\right)\right)^2} + \frac{-\boldsymbol{\mu}\left(\phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} + 0.5\right) - \phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} - 0.5\right)\right)}{\Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} + 0.5\right) - \Phi\left(\frac{\boldsymbol{\mu}^{\intercal}\mathbf{d}}{\sigma_{\mathbf{d}}^2} - 0.5\right)} \tag{3.24}$$

Equation 3.24 must always be nonpositive, as it is simply the second derivative of $\Phi(\mu + 0.5) - \Phi(\mu - 0.5)$, which we proved to be log-concave in Lemma 2. Thus, we have that the Hessian of Equation 3.22 is negative semidefinite and the liklihood of our indifference preferences is log-concave. □

We now finally arrive at our result of interest.

Theorem. *If the prior on $\boldsymbol{\mu}$ is log-concave, then the posterior on $\boldsymbol{\mu}$ is log-concave. Furthermore, if the prior on $\boldsymbol{\mu}$ is strictly log-concave, then the posterior on $\boldsymbol{\mu}$ is strictly log-concave.*

Proof. By Lemmas 3 and 4, we have shown that all of the terms in the products of Equation 3.6 are log-concave, with the exception of the prior. Because the product of log-concave functions is log-concave, if the prior is log-concave, then the posterior will also be log-concave, and if the prior is strictly log-concave, then the posterior will also be strictly log-concave. □

This theorem is extremely useful, as most commonly-used priors—such as the exponential and normal distributions—are strictly log-concave. This results in several important qualities for our posterior. First, if strict log-concavity holds, then there exists exactly one local optimum, and that optimum is globally optimal [10]. Furthermore, if strict log-concavity holds, BFGS will approach this global optimum in superlinear time [58]. Finally, BFGS has been able to solve nonlinear optimization problems with millions of variables [21], meaning we need not worry about our inference technique's ability to scale to high dimensions. To our knowledge, this is the first approach proven to converge quickly and accurately to the global MAP estimate for any realistic number of dimensions.

Nonlinear Utility Functions

The likelihood function may not be able to be represented analytically for arbitrary functional forms of the utility function. If we fix our covariance matrix to a constant as described above, then our likelihood function is in general

$$\mathbb{P}(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \int_{\mathbb{R}^n} I(\mathbf{x}, \mathbf{y}, \mathbf{p}) \prod_{i=1}^{n} \phi\left(\frac{p_i - \mu_i}{\sigma}\right) d\mathbf{p} \tag{3.25}$$

where $I(\mathbf{x}, \mathbf{y}, \mathbf{p})$ is in indicator function which is one whenever our utility function ranks $\mathbf{x}$ and $\mathbf{y}$ in the same order as our expert's preference and zero otherwise.

We can use an integral approximation technique to evaluate the right side of Equation 3.25. For low dimensions, sparse grid quadrature rules can provide an efficient estimate [36]. For higher dimensions, the most efficient numerical integration technique is Monte-Carlo simulation.

With this approximation for for $\mathbb{P}(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$, we can either approximate the MAP, or draw samples directly from the posterior. However, we must modify our methodology to account for the fact that our likelihood function is now both noisy and expensive to compute.

BFGS requires a large number of function evaluations and can struggle with noisy functions, and consequently is no longer the preferred method for optimizing the likeli-

hood function to get the MAP. Instead, it is more efficient to use surrogate modelling to find the MAP. Because our samples are now noisy, we must use regularization in our surrogate model to avoid overfitting [33].

Alternatively, we can sample directly from the posterior using "pseudo-marginal" MCMC. Although using noisy estimates for the likelihood function does result in slower mixing, it still generates valid samples from the posterior [3]. In our experience, we achieved best mixing using the "Monte Carlo within Metropolis" variety of pseudo-marginal MCMC [3]. Although this variant of pseudo-marginal MCMC is the most costly per step, we have found that the superior mixing is worth the extra computation.

Pseudo-marginal MCMC does pose significant computational challenges, but modern tools are starting to make this feasible. We implemented our algorithm in Julia [13] and parallelized the inner integral approximation across four local cores. Using 1000 Monte Carlo samples to approximate $\mathbb{P}(\mathcal{D} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and 1000 total pseudo-marginal MCMC samples, we saw good convergence. On an Intel 3.5 GHz processor running Linux, this process takes approximately 20 seconds for a three-dimensional $\mathbf{p}$ and 20 preferences. For consumer-end preference elicitation, such as an internet radio station automatically selecting the next song to play, this runtime would be prohibitively long. However, for engineering design applications, this delay may be tolerable if linear functional forms of $u(\cdot)$ are unacceptable.

3.3.3 Query Generation

Some preference queries are more useful than others. For example, suppose our utility function is $u(x_1, x_2) = p_1 x_1 + p_2 x_2$ and we know p_1 exactly, but have no information about p_2. Then, learning the expert's preference between two designs with different values for x_1 and identical values for x_2 yields no new information. In contrast, knowledge about the user's preference between two designs with identical values for x_1 and different values for x_2 allows us to decrease our uncertainty about p_2.

We utilize the concept of the *entropy* of a distribution [68] in order to analyze the

effectiveness of a query. The entropy of a distribution is defined as

$$H(X) \triangleq -\int f_X(x) \log_2 \left(f_X(x)\right) dx \tag{3.26}$$

where $f_X(x)$ is the probability density function of the random variable X.

Because we can sample from the posterior of the distribution, we can estimate its entropy by fitting a kernel density approximation to the samples and calculating the entropy of this approximation [46]. In order to determine the information encoded by the query, we calculate the expected entropy for each possible comparison using the following formula:

$$\begin{aligned}\mathbb{E}\left(H\left(\mathbf{x} \stackrel{?}{=} \mathbf{y}\right)\right) \approx \hat{H}(\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathcal{D}, \mathbf{x} \succ \mathbf{y})\mathbb{P}(\mathbf{x} \succ \mathbf{y}))+ \\ \hat{H}(\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathcal{D}, \mathbf{x} \prec \mathbf{y})\mathbb{P}(\mathbf{x} \prec \mathbf{y})) + \hat{H}(\mathbb{P}(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathcal{D}, \mathbf{x} \sim \mathbf{y})\mathbb{P}(\mathbf{x} \sim \mathbf{y}))\end{aligned} \tag{3.27}$$

where $\hat{H}(\cdot)$ denotes our entropy estimate via MCMC sampling and kernel density estimation.

In order to determine the most effective query, we simply look at every possible pairwise comparison between our family of candidate solutions and choose the one with the lowest expected entropy, i.e., the query that most decreases our posterior uncertainty. This entropy-minimization technique, pioneered in the 1950s [51], has been used successfully in prior preference elicitation work [41, 52, 49] as well as other machine learning work, such as pattern recognition [17].

If the number of possible comparisons is large, then this can be a computationally intensive task, as each comparison requires three MCMC simulations. Fortunately, it is trivially parallizable. Determining the best comparison for a linear utility function among 100 possible comparisons with 10 pre-existing preferences takes approximately 10 seconds on a four core 3.5 GHz processor and scales linearly with the number of possible comparisons.

Many existing preference elicitation algorithm use a related, but fundamentally different, technique to generate queries. Instead of choosing the query that minimizes

posterior entropy, they choose the query that has the highest expected value of information [20, 41, 38]. This reduces the likelihood of the algorithm posing queries between designs that are of no interest to the designer or designs that could not exist. Maximizing expected value of information empirically results in the algorithm choosing the best item from a set with the fewest number of queries [20, 38]. Unfortunately, maximizing expected value is intractable for our engineering design optimization application. We wish to maximize over the set of all possible designs, rather than maximizing utility over a set of already known designs—in order to calculate the expected value of information, we would have to be able to evaluate every possible design. If we could do this, we would have no need of optimization in the first place. However, if we use the the entropy minimization technique in the optimize-elicit-optimize loop, we know will not encounter infeasible designs: all the designs available for comparison have already been generated by the previous stage of optimization. Finally, although entropy-maximization may pose queries between designs that are intrinsically of no interest to the designer, the designer's preference between these designs may help the optimization make similar value choices between better performing designs.

3.4 Results

In order to test the performance of this algorithm, we first contrive a simple example to examine the algorithm's convergence properties. Then, we apply the algorithm to optimizing an aircraft collision avoidance system.

3.4.1 Proof of Concept

We first test the ability of our algorithm to converge to a known utility function. We let $u(\mathbf{x}) = \mathbf{p}^\intercal \mathbf{x}$ and arbitrarily set $\mathbf{p} = \langle 5, -1, 2 \rangle$. We generated 100 designs randomly, with each of the three metrics being drawn randomly from a uniform distribution over the interval $[0, 1]$. Our goal is to measure how effectively our algorithm can estimate $\mathbf{p}$ by receiving only pairwise preferences between the designs.

One way to measure preference elicitation algorithm performance is to calculate

the difference in utility between the estimated optimal solution and the true optimal solution [20]. Although this definition is useful in applications where the set of possible solutions is already known, in an optimization context we do not know the global optimal solution, and simply assigning utility to existing solutions does not help us generate new ones. Instead, we are far more concerned with matching the actual underlying utility function globally, as it will be the force that drives our optimization routine.

To measure how close we are to the global utility function, we compare our estimate for the parameter vector $\hat{\mathbf{p}}$ to the known $\mathbf{p}$. One might be tempted to use a norm of $\hat{\mathbf{p}} - \mathbf{p}$ to measure our accuracy. However, for linear utility functions, it is impossible to learn $\mathbf{p}$ to anything beyond a normalizing constant by only using pairwise comparisons, as if $\mathbf{p}^\intercal\mathbf{x} > \mathbf{p}^\intercal\mathbf{y}$, then $c\mathbf{p}^\intercal\mathbf{x} > c\mathbf{p}^\intercal\mathbf{y}$ for all $c \in \mathbb{R}^+$. Thus, we use the *angle* between our estimate $\hat{\mathbf{p}}$ and $\mathbf{p}$ as our loss function V:

$$V(\hat{\mathbf{p}}, \mathbf{p}) = \arccos\left(\frac{\hat{\mathbf{p}}^\intercal\mathbf{p}}{\|\hat{\mathbf{p}}\|_2\|\mathbf{p}\|_2}\right) \tag{3.28}$$

In fact, this notion of error angle will correlate highly with the loss in utility of the global optimum. If our utility function is linear, then we know that the global optimum will occur on the boundary of the design set [12]. Thus, the error in angle will correspond to how far away the estimated optimum is from the true optimum on the boundary. For example, if the set of all designs forms a unit hypersphere around the origin and the true $\mathbf{p}$ is normalized, then the loss in utility from an error angle of θ is simply $1 - \cos(\theta)$. This correspondence is shown in Figure 3-5.

We test our method using randomly-generated queries and queries generated using the entropy minimization technique. Our method was compared against a modification of the UTASTAR [69] algorithm.[1] We also test a heuristic for query generation common in the literature of selecting the current best solution and the one with the highest probability of improving upon the incumbent [38]. These four methods are

[1] As described in the source, the UTASTAR algorithm fits a piecewise linear approximation to the utility function. However, such approximations do not extrapolate well globally, so we added a constraint that forces the estimated utility function to be exactly linear.

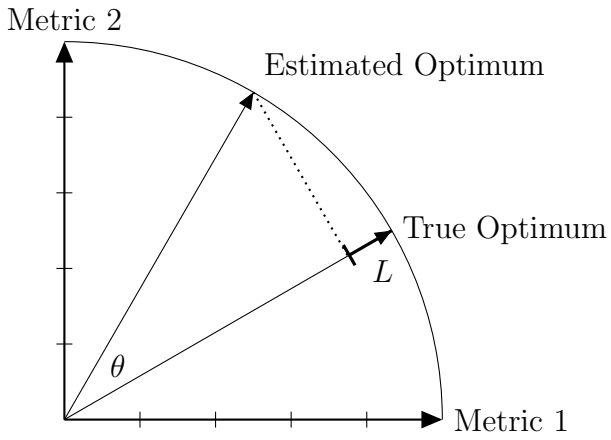

Figure 3-5: Correspondence between loss in utility L and error angle θ for a circular design set.

shown in Table 3.1.

Many existing preference elicitation methods are not tested because they are not applicable to our task of learning parameters of the utility function itself. Most existing methods simply try to find the optimal item from a given set without trying to find the functional form of the utility function directly [38, 41, 77, 20]. Furthermore, we know that methods based on the expected value of information [20, 38, 77] are not useful, as the utility over all possible designs is dependent on the exact structure of the set of all possible designs—although we might be able to perform this integration for a toy problem where we prescribe this set, it is extremely doubtful we would ever be able to calculate it for a real engineering design optimization problem.

All tests begin with the same first query, which was randomly generated. For the Bayesian tests, a flat prior was used. Each test was repeated 100 times.

With an Infallible Expert

We first run the above algorithms using an infallible expert—in our test, a subroutine that calculates the true utility of both of the query objects, and returns the true preference between the two. Figure 3-6 shows the mean loss for each method as a function of the number of preferences given to the algorithm.

The entropy-based learning method performs best, followed by the Bayesian ran-

Table 3.1: Description of the four algorithms tested.

Method	Description
UTASTAR with Random	The UTASTAR Algorithm with randomly chosen queries to the expert. UTASTAR has no method for posing queries to the expert.
Bayesian with Random	Our Bayesian approach, learning from queries chosen at random.
Bayesian with Best/Highest PI	Our Bayesian approach, learning from queries generated using the heuristic of querying the estimated best against the solution with the highest probability of improvement.
Bayesian with Min Entropy	Our Bayesian approach, learning from queries that most decrease expected posterior entropy.

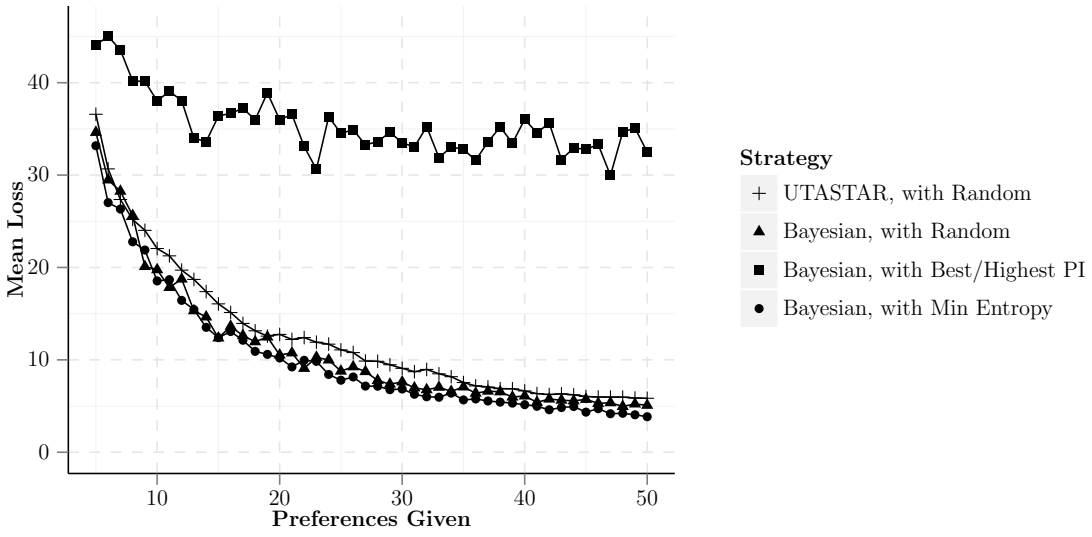

Figure 3-6: Mean loss as a function of preferences given from an infallible expert.

dom and linear programming methodologies. Performing significantly worse than the others is the best/highest probability of improvement heuristic. Although this heuristic fares well when a more traditional loss function is used [38], it performs poorly at learning the true underlying function. This is because this heuristic constrains the queries to always include the current optimum, even when a query between lower-ranked alternatives would have been more informative to global understanding.

Table 3.2 shows the statistical tests performed on the algorithm performance. We used the bootstrap [30] to generate 10^4 new samples from our performance data and calculated how often the mean performance of each algorithm did not match the ordering in Figure 3-6. This provides a result similar to a Student's t-test, but without requiring that our data be normally distributed. [30, 62] These results indicate that the trends exhibited in Figure 3-6 are statistically significant.

Table 3.2: Pairwise statistical comparison tests for best performing algorithms with an infallible expert.

Event	$\mathbb{P}$(Event)
Entropy $\geq$ UTASTAR	0.0000
Entropy $\geq$ Random	0.0140
Random $\geq$ UTASTAR	0.0046

With a Fallible Expert

We know that in reality no expert is infallible. Instead of a subfunction that observes the true utility to make comparisons, we incorporate the blunder model from Bradley and Terry [16], modified to account for indifference preferences in the manner described by Guo and Sanner [38]. Given true objective values $u(\mathbf{x})$ and $u(\mathbf{y})$, the probability of returning an indifference preference is

$$\mathbb{P}(\mathbf{x} \sim \mathbf{y} \mid u(\mathbf{x}), u(\mathbf{y})) = \exp\left(-\beta |u(\mathbf{x}) - u(\mathbf{y})|\right) \tag{3.29}$$

Given that an indifference preference is not returned, the probability of returning a strict preference is

$$\mathbb{P}(\mathbf{x} \succ \mathbf{y} \mid u(\mathbf{x}), u(\mathbf{y}), \neg(\mathbf{x} \sim \mathbf{y})) = \frac{\exp\left(\alpha(u(\mathbf{x}) - u(\mathbf{y}))\right)}{1 + \exp\left(\alpha(u(\mathbf{x}) - u(\mathbf{y}))\right)} \tag{3.30}$$

The parameters α and β allow this model to represent experts who have varying degrees of confusion. For this test, we let $\alpha = 1$ and $\beta = 0.1$, which led to a reasonable number of indifference and inconsistent preferences in our contrived scenario.

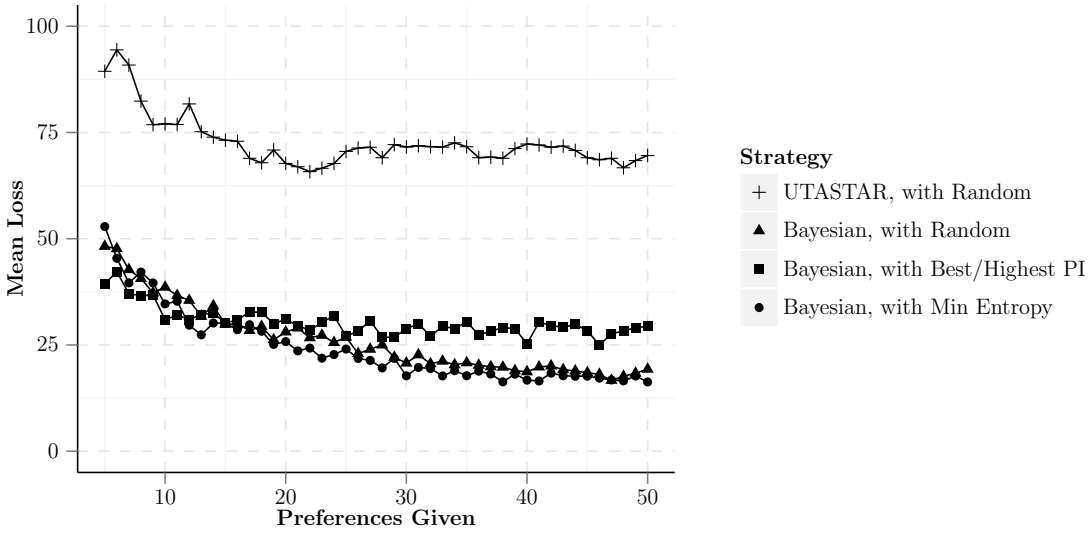

Figure 3-7: Mean loss as a function of preferences given from a fallible expert.

After programming this model of fallibility into our test, we again ran 100 samples of our four methodologies. Figure 3-7 shows the decay of error means as more preferences are added, and Table 3.3 shows the bootstrap statistical tests comparing the difference between the best performing algorithms.

Table 3.3: Pairwise statistical comparison tests for best performing algorithms with a fallible expert.

Event	$\mathbb{P}$(Event)
Entropy $\geq$ Best/Highest PI	0.0000
Entropy $\geq$ Random	0.0000
Random $\geq$ Best/Highest PI	0.0000

Clearly, our Bayesian methodology is far more robust to noise than the linear programming formulation, which only performs marginally better than randomly guessing weight vectors. Even the "Best with Highest Probability of Improvement" heuristic, when used with our framework, performs dramatically better. Furthermore, the entropy-based query selection method continues to outperform selecting random queries and the best/highest expected improvement heuristic.

It is also interesting to note that UTASTAR generally seems to be unable to

converge beyond a certain point. This highlights one of the fundamental problems in robust linear optimization: the linear program has likely determined that the actual solution lies within some polyhedral set, but is forced to always pick an extreme point of the polyhedron [12]. If the most likely solution is in the interior, linear programming will never be able to select it. When there are no contradictory preferences, the polyhedron of possible solutions grows smaller with the addition of the cutting plane defined by each preference, allowing the algorithm to converge. The presence of contradictory preferences prevents the polyhedron from reducing its volume beyond a certain point, as inconsistent preferences will re-expand it.

We have shown that our Bayesian approach performs very well. It converges quickly and accurately to the true utility function, even in the presence of noisy preferences. Encouraged by these results, we then used our approach to optimize ACAS-X.

3.4.2 Application to Aircraft Collision Avoidance

Background

Optimizing the POMDP penalties in ACAS-X is incredibly difficult. The mapping between the penalties and system performance is known to be non-convex [71] — although many real-world non-convex optimization problems can be optimized satisfactorily by using heuristics such as simulated annealing [76], genetic algorithms [70], or particle swarm optimization [57], all these methods presume that the objective function is computationally easy to evaluate [76, 70, 57]. Evaluating our POMDP solution is not so: at 25 minutes per evaluation, a genetic algorithm with a meager population size of 100 would take over nearly three months to produce 50 generations of solutions. The slow evaluation time also precludes the use of multi-objective optimiztion procedures based on these heuristic methods, such as the NSGA-II [26] or ant-colony optimization [23, 22]. We therefore use the surrogate modelling optimization method [71].

This technique requires a single objective, but we are concerned with several

objectives, such as the safety of the system and the number of nuisance alerts. We could use goal programming to define a single metric, but it is unclear what the ideal point should be, and even less clear what the distance metric should be. The ϵ-constraint method is just as troublesome, as we often do not know what constraints would be appropriate at the onset of optimization, and even if we did, the mapping between the POMDP penalties and the system performance is too complex for us to even be able to define the constraints. Consequently, we resort to the simplest solution: defining a utility function. For each of these metrics, we define the marginal utility function to be

$$u_{\text{metric}} = \exp\left(\frac{(\text{metric})^2}{(\text{metric target})^2}\right) \tag{3.31}$$

and we let the overall utility be

$$u = \sum_{i \in \text{metrics}} p_i u_i \tag{3.32}$$

where p_i indicates the relative importance of achieving the target rate for that metric. We casually refer to these p_i as the metric's "weight." By adjusting these weights, the behavior of our surrogate modelling optimization can drastically change. To learn these weights, we will use our preference elicitation algorithm.

Small-Scale Testing

In order to test the effect of our preference elicitation on our optimization routine, we perform a simple test. We vary only two of the most important parameters in our POMDP formulation and measure only the safety and nuisance alert metrics. Initial weights were chosen naïvely and are shown in Table 3.4. After 50 surrogate modeling iterations, we branch our optimization as shown in Figure 3-8. One branch continues with the naïve weights, while the other uses new weights derived from expert preference elicitation on the first 50 samples. The prior distribution used on each weight was an exponential distribution with a mean of the naïve weight.

As a basis for the preferences, the expert choose policies that struck a suitable balance between operational suitability and safety. Table 3.4 compares the weights

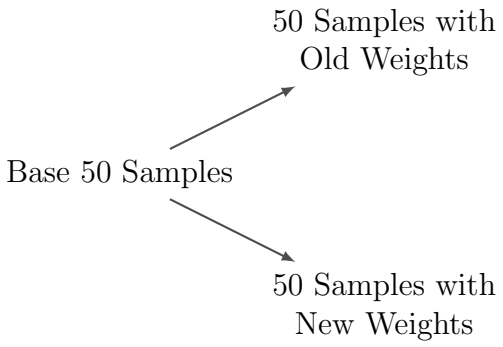

Figure 3-8: Our Small-Scale Optimization Test.

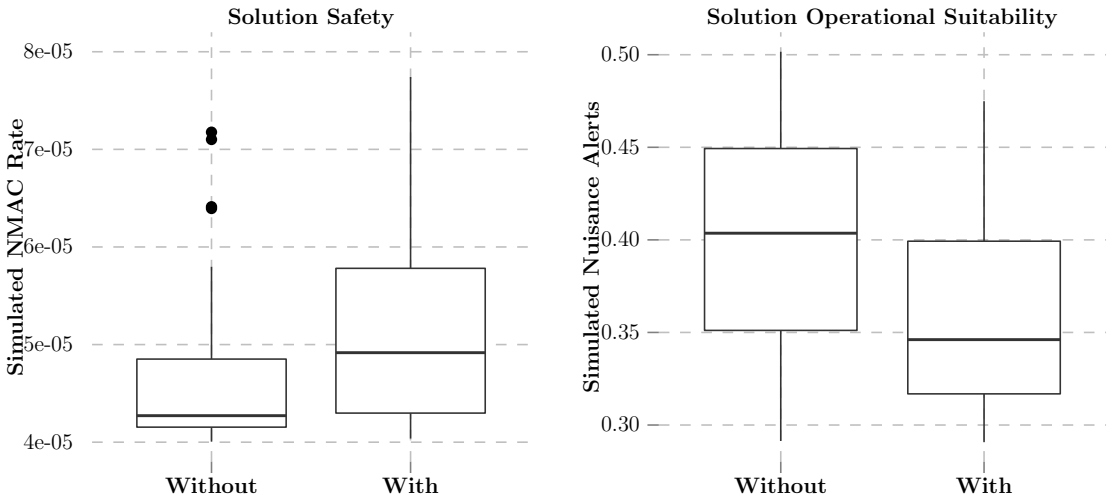

Figure 3-9: Distribution of solutions found with and without preference elicitation. The base 50 samples are omitted.

used before and after preference elicitation.

Table 3.4: Weights before and after preference elicitation.

Metric	Original Weight	Inferred Weight
NMAC Rate	0.990	0.508
RA Rate	0.010	0.492

Figure 3-9 shows the distribution of solutions returned by the optimization in a box-and-whisker plot with and without the use of preference elicitation. Without the preference elicitation, the optimization routine searches far too heavily in the safety

domain without sufficient regard to the operational suitability. With the aid of several pairwise comparisons, the optimization routine returns results far more tailored to the expert's goals.

Application

Surrogate modelling optimization of the aircraft collision avoidance system POMDP took several months. The eight most important penalties in the POMDP were tuned using the surrogate modelling framework, and twelve safety and operational suitability metrics were measured for each design. Due to the potentially sensitive nature of the trade-offs involved in designing an aircraft collision avoidance system, we have included the metric values but omitted the names of the metrics. In doing so, we hope to demonstrate the usefulness of our preference elicitation algorithm without putting our sponsor's priorities on public trial.

We began our optimization by setting each of the p_i in Equation 3.32 by intuition for each of our twelve metrics. These values are shown in the "Weights, Before" column of Table 3.5. We then ran the surrogate modelling optimization for several weeks with these p_i, generated and evaluating a large number of ACAS-X policies. The average performance of these policies is shown in the "Mean Metric Values, Before" column of Table 3.5. We took five of the top performing policies and presented them to the international aviation safety community, consisting of the Federal Aviation Administration (FAA), the Single European Sky Air traffic management Research (SESAR) project, potential commercial vendors, and pilot associations. We asked them to provide preferences between these designs based on their stake in the project in addition to heuristic feedback about the quality of the designs. The heuristic feedback amounted to the following:

- Performance in metrics 1 and 2 are performing above expectations.
- Metrics 3 and 4 should be improved. If need be, this may be at the expense of metrics 1 and 2.
- Metric 12 needs improvement.

Table 3.5: Effect of preference elicitation on collision avoidance optimization.

	Weights		Mean Metric Values	
	Before	After	Before	After
Metric 1	0.0750	0.0540	$2.567 \cdot 10^{-5}$	$2.566 \cdot 10^{-5}$
Metric 2	0.2250	0.0932	$2.227 \cdot 10^{-6}$	$2.228 \cdot 10^{-6}$
Metric 3	0.1225	0.1472	0.4977	0.4657
Metric 4	0.0350	0.0589	0.4643	0.4274
Metric 5	0.1050	0.0771	0.1097	0.1265
Metric 6	0.0175	0.1190	0.0033	0.0034
Metric 7	0.0350	0.0385	0.0097	0.0129
Metric 8	0.0350	0.0448	0.0426	0.0295
Metric 9	0.1700	0.0371	0.1356	0.1479
Metric 10	0.0300	0.0470	0.6461	0.6298
Metric 11	0.1000	0.0500	0.1264	0.0123
Metric 12	0.0500	0.2320	0.0147	0.0143

Instead of using the heuristic feedback to manually adjust the weights, we simply took the preferences elicited from the international community and fed them into our preference elicitation algorithm. Priors on each mean were selected to be exponential with a mean of the previously used, intuition-based value for each p_i. We fixed the σ of our algorithm to be 0.1. After running our algorithm, we used the posterior mean of each μ_i as a point estimate for each p_i. These estimates are shown in in the "Weights, After" column of Table 3.5. As we can see, the new weight structure matched the heuristic feedback provided by the international community. The weights on metrics 1 and 2 decreased relative to metrics 3 and 4, and the weight on metric 12 increased.

The "Mean Metric Values, After" column of Table 3.5 shows the metric performance using the new weights derived from preference elicitation. As we hoped, the performance in metrics 3, 4, and 12 improved. Interestingly, the performance in metrics 1 and 2 did not degrade substantially after the weight change. Feedback from the program sponsors of the top designs returned from the second stage of optimization were very positive — they were satisfied with the balance of metrics these designs exhibited.

As promising as this may be, we note that one should be cautious in interpreting Table 3.5, as the metrics values before and after preference elicitation are not independent. Because the optimization after preference elicitation started from where the optimization before preference elicitation ended, we would naturally expect the metric values to improve. In order to perform a fully rigorous test, we would have to re-run the optimization without the preference elicitation and compare the two optimizations. Unfortunately, because of the large amount of resources and manpower necessary for these optimizations, we were unable to perform this experiment. That said, we do not think it is too much of a leap of faith to believe that an optimization with different metric weights will return different results.

We have since used the optimize-elicit-optimize loop above several times to incorporate preferences into our optimization. Each time, our preference elicitation algorithm performed as above: it satisfied as many preferences as possible, didn't stray too far from our prior estimates, and matched the heuristic commentary provided alongside the preferences. Although we recognize that the plural of "anecdote" is not "data," we believe that our successful applications of our algorithm demonstrate its usefulness to real-world engineering design optimization problems.

3.5 Discussion

We have shown that our method for preference elicitation is well-suited for use in engineering design optimization. Its inference method is less restrictive and more general than existing work, and its ability to use entropy-minimization to generate queries results in it converging faster to the true utility function than other preference elicitation algorithms. We then successfully used our framework to incorporate preferences from dozens of experts around the world into a multiple month-long optimization routine of ACAS-X.

Chapter 4

Multi-Objective Optimization

Most large-scale optimizations of engineering designs rely on the use of utility functions in some form, whether this utility function is specified manually, learned through some preference elicitation algorithm, or defined via some goal programming metric. Although this does not pose any issues if the utility function is defined perfectly, in reality we know that this will not be the case.

In this chapter, we review methods one can use for a multiobjective optimization problem without specifying a utility function. Although these methods are often limiting, we identify an aspect of ACAS-X conducive to these methods. In Section 4.2, we use a multiobjective genetic algorithm to identify shortcomings of the ACAS-X TA system, and then use the same genetic algorithm to tune a new TA policy to optimal performance.

4.1 Background

Figure 4-1 illustrates what occurs with a faulty utility function. Suppose the dots lie on the actual Pareto front, and the arrow indicates the direction of search specified by our (linear) utility function. We can visualize how the points on the Pareto front would be ranked by our utility function by projecting them onto the arrow.

As we can see, if our actual optimum is near the head of the arrow, the optimum will be ranked highly and likely be found during optimization of our utility function.

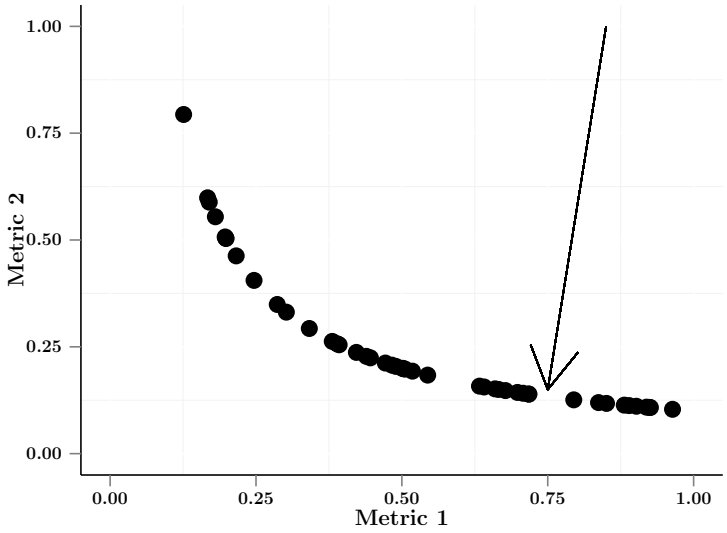

Figure 4-1: Utility function optimization on the Pareto front.

However, if our actual ideal point is further to the left, then the solution will have a low utility and thus be unlikely to be discovered during the optimization.

The ideal way to perform multi-objective optimization is to identify the Pareto front directly. One way to do this could be to modify the utility function slightly, re-optimize, and repeating this procedure until a section of the Pareto front has been identified. However, this approach suffers from two fundamental flaws. First, it is not immediately clear how much spread on the Pareto front will be achieved by modifying the utility function—if we modify our utility function too little, the solutions will be too similar to our previous optimization; if we modify it too much, our Pareto front might have large gaps in it. Second, the optimizations will likely have to perform many redundant optimization steps, i.e. optimizing out of regions in which all objectives are bad.

Genetic algorithms provide a more natural way to deal with multi-objective optimization [28]. Each population member's fitness is determined not only by its objective function values, but also its distance to other members in the population. Numerous different genetic algorithms have existed and been used successfully in practice [32, 40, 73, 81].

Among these, the most effective [81] is the second implementation of the Non-

dominated Sorting Genetic Algorithm (NSGA-II) [26]. The fitness function of the NSGA-II is straightforward. First, each individual is assigned a "Dominance Count." Individuals on the current Pareto front of the population are assigned a domination count of 0. These individuals are then removed, and the Pareto front is again recalculated. Members on this new Pareto front are given a domination count of 1, and this procedure continues until enough individuals are identified to form the parents of the next generation. If the last dominance front provides too many individuals for the population size, then the individuals whose absence makes the largest gap in the Pareto front are chosen [26]. A single generation of the NSGA-II runs in $\mathcal{O}(mn^2)$ time, where m is the number of objectives, and n is the number of population members [26].

Although the $\mathcal{O}(mn^2)$ computational overhead of the NSGA-II is rarely a problem, its large number of calls to the function to be optimized can be problematic if the function is computationally expensive. If we run 100 generations of a population size of 100, then the function must be called a total of 5000 times—if we were to apply this to ACAS-X as a whole, this would take over four months.

This chapter applies the NSGA-II algorithm to the ACAS-X project. A fundamental insight into how TAs are generated allows for our function of interest to be optimized directly using the NSGA-II algorithm.

4.2 Traffic Alert Optimization

Traffic Alerts (TAs) warn a pilot of nearby aircraft that could potentially become a threat, ensuring the pilot is alert and prepared for an RA to be issued. TCAS simply extends its RA heuristics to greater values of τ to determine if an TA should be issued [1]. However, this results in the TCAS TA logic falling prey to the same issues as the RA logic: the system is inherently unrobust, and therefore must be made very sensitive to ensure acceptable safety.

Puntin and Kochenderfer [60] previously developed a method for issuing TAs that appeared to be successful for ACAS-X. However, as the ACAS-X project evolved,

their method proved no longer viable. In this section, we review their approach and discuss its empirical shortcomings. Aided by the NSGA-II [26], we then modify their strategy, resulting in across-the-board improvement of the ACAS-X TA logic.

4.2.1 Traffic Alert Mechanism

After the implementation of ACAS-X RA logic as an POMDP, simply extending the τ value was no longer a sensible option [60]. One approach would be to develop another POMDP model specifically for TA logic. However, Puntin and Kochenderfer describe some of the problems with such an approach:

> In order to do this optimization, it is necessary to define a probabilistic dynamic model and a cost function that defines the objective of the system. The dynamic model would capture the response of the pilots to the generation of the RA. Although the function of the TA is not to instruct the pilots to maneuver, pilots often do, and so this should be accounted for in the model. The model can also capture the fact that a TA often improves the swiftness of the pilot response to an RA. The resulting TA cost can be a function of whether an NMAC occurs and the disruptiveness of the alerts. In order to implement such an optimization, the current model used for optimizing the RAs would have to be expanded to account for the additional TA state data, resulting in larger lookup tables [60].

This approach could be feasible, but would require extensive study and analysis. Furthermore, the increase in the state space would dramatically increase the size of the cost table, making it potentially too big to run on available aircraft hardware. Puntin and Kochenderfer instead propose an alternate procedure: when the system looks up the per action cost for RAs, it uses these costs to determine if an TA should be issued. Specifically, the logic follows Algorithm 1 to determine if an TA should be issued or turned off [60]. This algorithm works by using the clear-of-conflict reward as a proxy for safety. Generally, the lower the reward for issuing clear-of-conflict, the

```
1  // TA On Logic
2  if COC_COST < COC_ON AND
3  COC_COST − min{{COSTS} \ { COC_COST} } < COST_DIFFERENTIAL
4  then
5  |  TA ← ON;
6  end
7  // TA-Off Logic
8  if COC_COST > COC_OFF AND
9  TIME_SINCE_RA ≥ 6 seconds AND
10 TA_DURATION ≥ 6 seconds
11 then
12 |  TA ← OFF
13 end
```

Algorithm 1: ACAS-X TA logic.

less safe the aircraft is. The same logic is used in Line 8 to determine when a TA should be turned off, after the minimum TA time has been achieved.

However, the clear-of-conflict cost alone was determined to be insufficient for determining if a TA should be turned on. The logic encoded in Line 2 requires that not only the clear-of-conflict reward to be sufficiently low, but also that the reward of the next-best action is within some threshold of the clear-of-conflict reward. Puntin and Kochenderfer explain the logic behind this:

> The [COST_DIFFERENTIAL] threshold was added to reduce the rate of nuisance TAs. Without the [COST_DIFFERENTIAL] threshold, there were many TAs caused by the COC cost crossing the on threshold when all other actions had much higher costs. Implementing a [COST_DIFFERENTIAL] threshold requirement suppressed the TAs when an RA was not likely due to the large separation between the cost of COC and the other actions [60].

By looking only at the action costs already calculated for the RA logic, this approach results in no increase in offline optimization, no increased table storage requirements, and very little online computation. However, this algorithm requires specification of the COC_ON (Line 2), COC_OFF (Line 8), and COST_DIFFERENTIAL (Line

3) parameters. Optimal values for these parameters are far from clear, and can only be learned though optimization.

4.2.2 Optimization

Like the rest of ACAS-X, TA logic performance can only be measured through extensive simulation. For assessing TA performance, three metrics are deemed most important [60]:

- *Number of Traffic Alerts.*
- *Number of surprise RAs.* A surprise RA is an RA that did not have an TA at least six seconds prior.
- *Number of segmented TAs.* A segmented TA is a TA that goes off, but then back on again later in the encounter. This behavior is perceived to undermine pilot confidence in the system.
- *Average TA duration.* A TA that runs too long after the threat has been resolved could also undermine pilot confidence in the system.

In mathematical terminology, we thus have a function $f : \mathbb{R}^3 \to \mathbb{R}^4$ that we wish to optimize.

Puntin et. al. optimized these parameters by discretizing the parameter space and evaluating the solutions at all discretized points [60]. Although trivial to implement, this approach took enormous computing resources, taking over a week on a high performance compute cluster. Furthermore, if the discretization is too coarse, then good solutions could be missed.

We could optimize TA logic through our surrogate modelling optimization procedure. However, this procedure is slow and requires specification of an objective function — we have already shown this to be problematic. Instead, we exploit the fact that the metrics of interest in TA optimization are largely independent of pilot's response to TAs. This observation was confirmed by flight safety experts at the FAA.

By instructing our simulated pilots to ignore TAs and only respond to RAs, we can drastically speed up evaluation of TAs logic on simulations. The positions, belief states, RA costs and actions will always be the same, regardless of what values we assign to the TA parameters. Thus, if we collect the RA cost values at every point in time for every simulation, we can effectively simulate different TA logic by simply performing Algorithm 1 on the archived RA costs. Instead of actually simulating aircraft dynamics for hundreds of thousands of aircraft encounters, we need only parse a file.

A natural concern for this methodology is the memory requirement. At each time step, we need to collect the following values to run Algorithm 1:

- A time index of the simulation (8 bit integer).
- The clear-of-conflict cost (64 bit floating point).
- The difference between the clear-of-conflict cost and the next best alternative (64 bit floating point).
- Whether an RA was issued at this timestep (8 bit boolean).

This results in 144 bits of data per simulation timestep. The average simulation duration is approximately 100 seconds, and observations are recorded at one Hertz. Consequently, we can fit information from one million simulations into memory on a high performance computer: it only takes up 13.4 GB.

This methodology dramatically decreases logic evaluation performance. The time to evaluate a single aircraft collision encounter is cut from 0.25 seconds to $1.66 \cdot 10^{-5}$ seconds. Including overhead costs, simulating our TA encounter set of 100,000 encounters directly takes approximately three minutes on a 64 node high performance compute cluster; our parsing evaluation strategy evaluates the same encounter set locally in serial in 1.67 seconds.

This dramatic decrease in runtime allows us to directly optimize the TA logic without creating a series of surrogate models. Because we are dealing with a multi-

objective optimization problem of relatively low dimension, the NSGA-II is a natural choice.

We can further exploit the conditional independence inherent in our problem. The number of TAs and surprise TAs is dependent only on specification of the `COC_ON` and `COST_DIFFERENTIAL` parameters. Furthermore, given a set of values for the `COC_ON` and `COST_DIFFERENTIAL` parameters, the number of segmented TAs and the average TA time can be tuned by only modifying the `COC_OFF` parameter. Thus, instead of optimizing a function $f : \mathbb{R}^3 \to \mathbb{R}^4$, we can optimize $f : \mathbb{R}^2 \to \mathbb{R}^2$ and then simply tune a function $g : \mathbb{R} \to \mathbb{R}^2$. This independence drastically reduces the dimensionality of the optimization problem, reducing our runtime by orders of magnitude.

4.2.3 Results

Initial Results

After collecting the cost data for 100,000 simulations based on real-world traffic encounters, we ran the NSGA-II to optimize our traffic alert performance. We used a parent population size of 100, the simulated binary crossover [2] breeding technique, and ran the algorithm for 50 generations. Run in serial, this optimization takes approximately 40 minutes.

Figure 4-2 shows the results of this optimization alongside TCAS performance on the same dataset. This result is concerning for ACAS-X. In order to issue the same number of TAs, ACAS-X would have to risk tripling the number of surprise RAs; to keep the number of surprise RAs the same, the number of TAs would have to double.

The result is not a relic of the NSGA-II. In an effort to validate these results, a week-long brute force space search was performed. This search yielded results of a similar nature to Figure 4-2, indicating that the optimization method was performing as expected, and that the underlying issue lie in the POMDP TA logic.

This conflicts with the results presented by Puntin *et al* [60]. Since their publication, ACAS-X RA logic has undergone significant changes designed to increase safety in certain encounters with high vertical rates as well as to reduce the number of alters

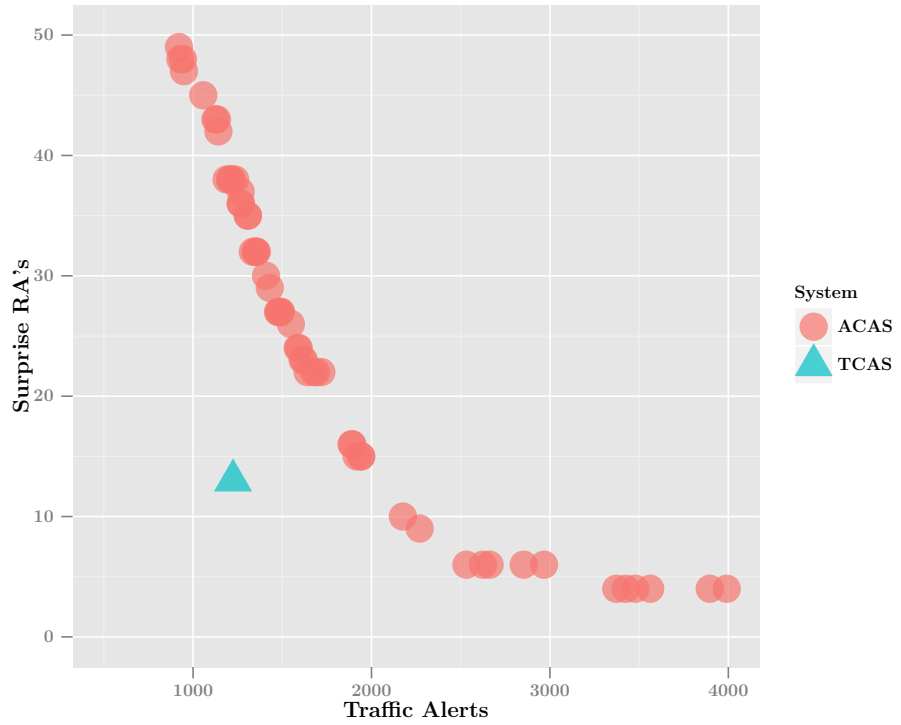

Figure 4-2: Traffic alert Pareto front.

in certain planned level-off scenarios. These changes fundamentally altered the cost behavior in many scenarios, resulting in different TA performance.

Traffic Alert Logic Change

The results of our NSGA-II optimization indicated that Algorithm 1 is not sufficient to outperform TCAS for our POMDP policy. In order to investigate the source of this performance problem, we created a plot of our POMDP policy. If aircraft velocities are constant, then the alert issued by ACAS-X is uniquely determined by the relative altitudes of the aircraft and the time until the aircraft's Closest Point of Approach (CPA). Figure 4-3 shows the ACAS-X policy for two level aircraft flying directly at each other at a speed of 250 knots. For example, if the intruder aircraft is 20 seconds from closest point of approach and is 200 feet above the ACAS-X-equipped aircraft, the ACAS-X aircraft will receive command to "climb at 1500 feet per minute." The fraction associated with each RA refers to the acceleration in G's at which pilots are

instructed to respond to this alert.

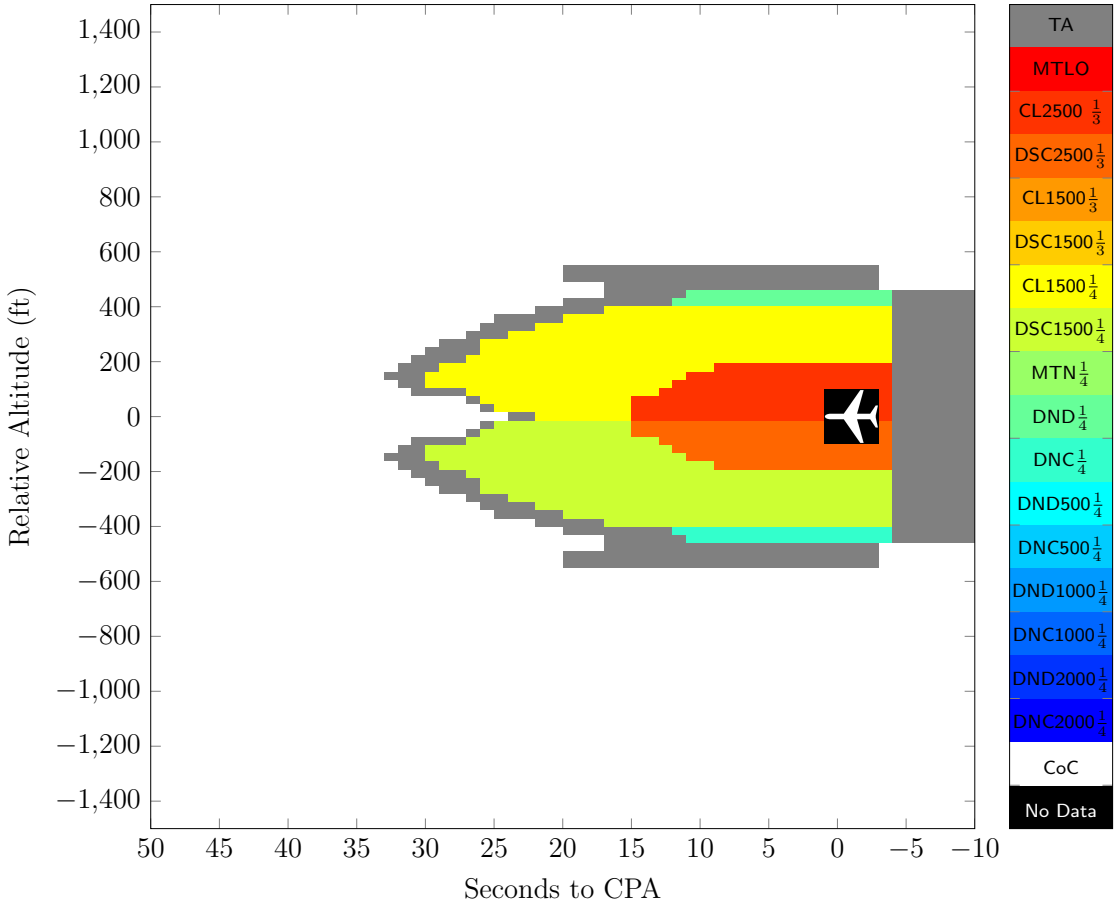

Figure 4-3: Traffic Alert policy plot for original logic.

Figure 4-3 reveals the failure modes of Figure 1. First, there exist several "teeth" in the policy at the edges of the alerting region (approximately at ± 600 ft, 20 seconds until CPA). These edges lead to unnecessary TAs: if the system was in clear-of-conflict when the aircraft were closer in altitude, then there is no reason why the aircraft should be in a traffic alert at this altitude difference. These edges are likely a result of the altitude discretization used in the POMDP interacting in a complex way with Algorithm 1.

The other failure mode in Figure 4-3 is the large gap at the center of the policy (at 0 altitude difference, 25 seconds to CPA). In the RA logic, such a gap can be explained due to the uncertainty in the POMDP: if the system is unsure of which aircraft has a higher altitude, than it is safer to wait a few seconds to decide which

aircraft should climb rather than running the risk of giving the wrong aircraft the climb order and driving the aircraft into one another. However, for TA logic, such a delay is senseless: the system *will* alert in a few seconds regardless of what happens; the system is only delaying to decide *which* alert is optimal. Thus, in the gap, the clear of conflict reward is very low, but all the alternatives are simply worse.

This observation gives insight on how to improve Algorithm 1: the AND condition at Line 2 is causing the policy to delay issuing a TA in the gap, as the clear-of-conflict reward is low, but no alternative is sufficiently close. A better policy might be to use and OR condition at Line 2: this would allow the `COST_DIFFERENTIAL` condition to activate most TAs, but allow the `COC_ON` condition to activate TAs in the gaps in Figure 4-3.

After implementing the OR variety of Algorithm 1, we created a policy plot for our new TA algorithm. This is shown in Figure 4-4.

Figure 4-4 shows that both fundamental problems with the policy depicted in Figure 4-3 are resolved. As expected, using the OR condition resulted in the gap being filled, reducing the number or surprise RAs in our policy. An unexpected benefit was the removal of the "teeth" from Figure 4-3. Thus, based on this policy plot, we would expect that the OR policy should in result both fewer alerts and fewer surprise RAs than the AND policy in Algorithm 1.

Post-Change Results

We then ran the NSGA-II optimization algorithm on the new OR-based policy. The results from this optimization are shown in Figure 4-5.

As we had hoped, switching the AND to OR significantly shifted the policy, resulting in far fewer TAs for every surprise RA. In fact, there are points on the OR Pareto curve that completely dominate TCAS performance, achieving both fewer TAs and surprise RAs.

The Pareto front from Figure 4-4 was given to FAA experts, who analyzed and selected the policy most suited to their use cases. After selecting the values for `COC_ON` and `COST_DIFFERENTIAL`, we simply manually tuned the `COC_OFF` threshold until the

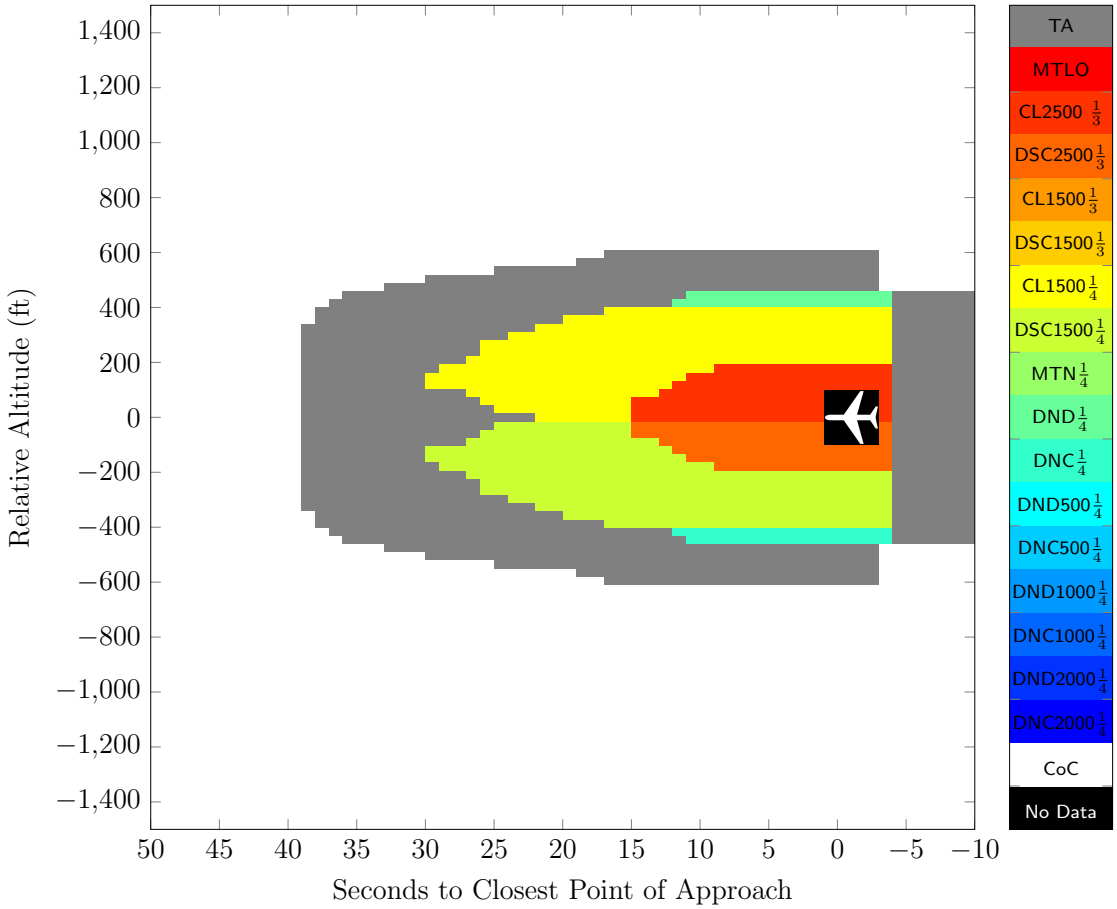

Figure 4-4: Traffic Alert policy plot for modified OR logic.

number of segmented TAs was reduced to an acceptable level. In fact, the optimal level for `COC_OFF` resulted in the OR-based policy having fewer segmented TAs and a lower average TA duration. Table 4.1 shows how our final policy compares to TCAS in the four metrics of interest.

Table 4.1: Comparison of TCAS and ACAS traffic alert performance.

	TAs	Surprise RAs	Segmented TAs	Mean TA Duration
TCAS	1224	13	31	24.33
ACAS-X	1056	13	11	22.86
% Reduction	13.7%	0%	64.5%	6.1%

Table 4.1 shows across-the-board improvement in the ACAS-X TA logic with respect to our objective metrics. However, not all performance can be captured in

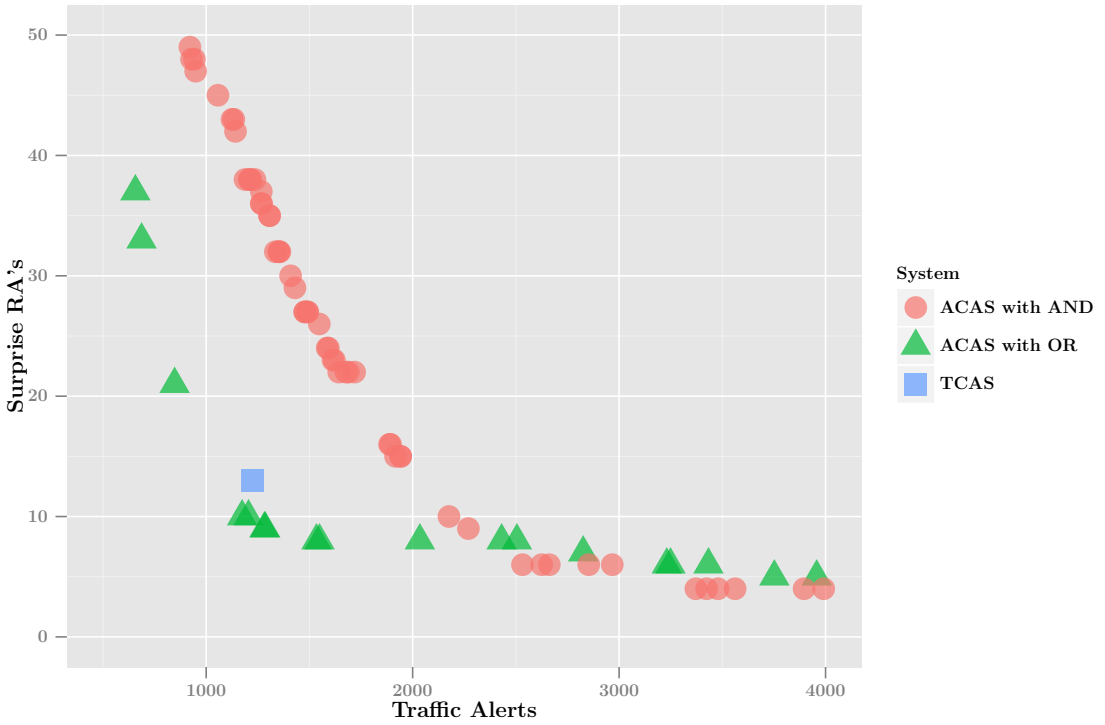

Figure 4-5: Pareto Front after Logic Change.

a single objective metric. One such abstract measure of performance is the shape of the distribution of the TA leadtime: the amount of time before an RA was issued that a TA was active. For example, if a TA was issued at seven seconds and an RA was issued at seventeen seconds, then this encounter would have a ten second TA leadtime. This is a generalization of the "surprise RA" metric.

Figure 4-6 shows the distribution of TA leadtimes for both the TCAS and ACAS-X systems. The general shape of the distribution is similar; in both systems, most RAs have a ten to twenty second leadtime, which is considered optimal by FAA experts. Furthermore, we also note that our ACAS-X methodology results in fewer TAs with a very long leadtime. This is also promising: long TAs can also be dangerous, as the pilot may have forgotten about the TA by the time the RA is issued.

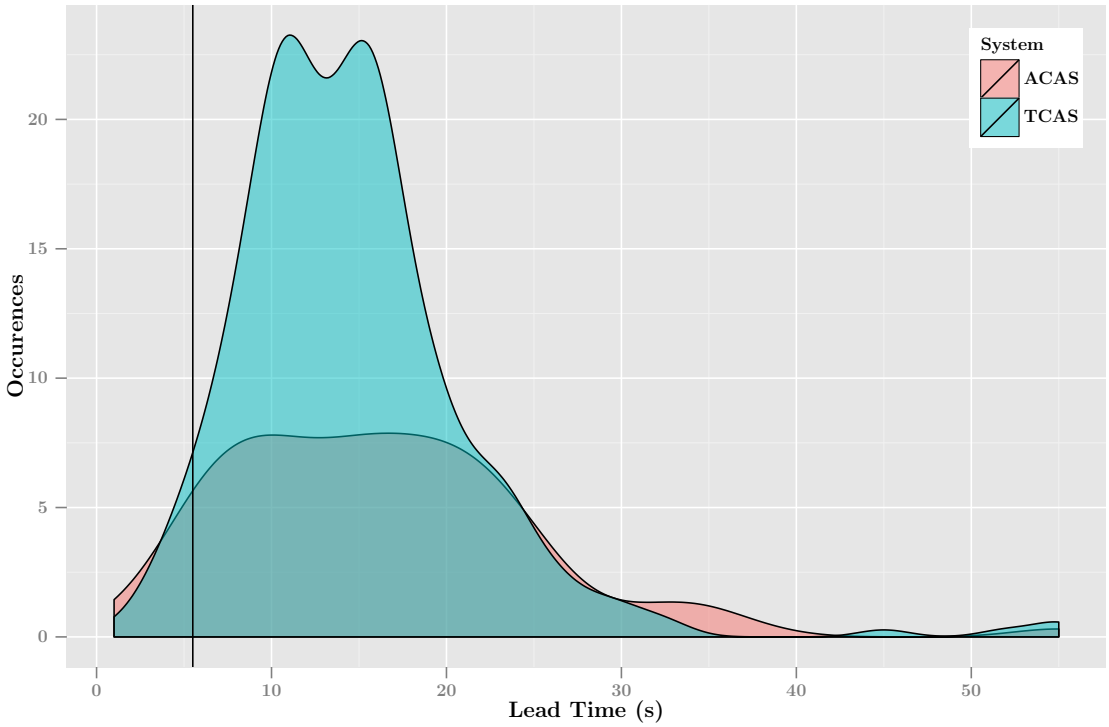

Figure 4-6: Distribution of time difference between TA and RA after logic change. The vertical bar at six seconds is the threshold for a surprise RA.

4.3 Discussion

Although not practical in all applications, identifying the Pareto front directly can be very useful in engineering design optimization. In the case of ACAS-X TA logic, its use demonstrated that a fundamental change in the TA logic was necessary. By examining the POMDP policy, we were able to identify and implement that change. This change resulted in a shift in the Pareto front, providing us a number of designs that dominated TCAS performance. The final point selected for use outperformed TCAS dramatically in all relevant performance metrics.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we have implemented two approaches to deal with multi-objective optimization in the realm of aircraft collision avoidance. First, we developed a novel algorithm for preference elicitation to allow the designers to more accurately create utility functions. Then, we applied a multi-objective genetic algorithm to optimize the behavior of traffic alerts in the ACAS system.

To develop our preference elicitation algorithm, we began by examining existing literature, and determined that although useful in some consumer-end applications, none existed that were amiable to eliciting a utility function for engineering design optimization. By exploiting properties of an existing model, we developed a faster, more accurate, and less-restrictive inference technique. We also developed a new approach to posing queries to the expert based on posterior entropy maximization. We then empirically showed that our method converged faster to a user's true preference model than existing algorithms. This result also held when we used a more complex response model. Finally, we applied this algorithm to the surrogate modeling optimization of ACAS-X.

When optimizing traffic performance in ACAS-X, we showed that we can quickly evaluate encounters when we only modify traffic alert logic parameters. This speedup enabled us to use the NSGA-II genetic algorithm to identify the Pareto front of our

solution space. The use of this algorithm allowed us to identify a fundamental flaw in the TA logic, which we corrected. Re-running the genetic algorithm then resulted in across-the-board improvement of the traffic alert behavior in the ACAS-X system.

5.2 Further Work

Within our preference elicitation algorithm, further work will be done on query selection. Sampling from the posterior of every possible preference realization is computationally expensive. It may be possible to find a heuristic to quickly determine which queries have a chance of being informative, and simply use the posterior sampling method to break the tie between these top queries.

It may also be possible to exploit the fact that our inference method does not require our covariance matrix to be diagonal—in other words, we could introduce correlation between the realizations of the elements of our parameter vector. Intuitively, it might make sense for there to be a small, negative correlation between the elements: if the expert overestimates the value of one metric, he or she is likely underestimating the value of others. Adding this negative correlation could make our method converge more quickly to a real expert's true utility function.

Work on the TA system also remains. First, the new system will be analyzed encounter-by-encounter by FAA experts to ensure that said TA are reasonable from a pilot acceptability standpoint — there may be cases in which pilots actually want an TA, which would have been missed by our TA minimization paradigm.

A larger concern lies in that the TA logic is build exclusively off the RA logic. Because the RA state space only extends 40 seconds prior to CPA, it is currently impossible for a TA to be issued before then. Thus, if a TA is issued at 35 seconds prior to CPA, then it will *always* result in a surprise RA. This problem cannot be mitigated without expanding the state space. However, extending the state space to include time steps up to 60 seconds would result in a 50% increase in both runtime for solving the POMDP as well as memory required for the optimal policy. The memory problem can be mitigated at the expense of code complexity by storing only

the clear of conflict cost and the next best option cost for time steps greater than 40, resulting in an increased memory requirement of only 6.25%. However, in either case, the increase in memory size could require the system to be implemented on a different hardware, potentially dramatically increasing program costs. A study will be undertaken to evaluate the benefit of expanding the state space beyond 40 seconds as well as if the system would require a hardware upgrade.

It may also be possible to generalize TA logic using techniques from machine learning. First, we would run a simulation and collect the costs for each action at each time step. Then, based on the simulation results, we can determine which time steps should have had a TA present; a reasonable approach would be to mark the six seconds prior to any RA as time steps that should have a TA. By doing this for a large number of simulations, we create a set of labeled data. We can thus use machine learning on this data to create a model that predicts whether or not a TA should be present based on the cost values. ACAS-X could then use this model in real time to determine when it should give a TA. Preliminary efforts have shown this to be a feasible methodology; unfortunately, these results are too experimental for this publication.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] Federal Aviation Administration. *Introduction to TCAS-II*, February 2011.

[2] Ram Bhusan Agrawal, Kalyanmoy Deb, Kalyanmoy Deb, and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. Technical report, 1994.

[3] Christophe Andrieu and Gareth Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009.

[4] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[5] Dylan Asmar. Airborne collision avoidance in mixed equipage environments. Master's thesis, Massachusetts Instituite of Technology, 77 Massachusetts Ave, Cambridge, MA, 2013.

[6] Mark Bagnoli and Ted Bergstrom. Log-concave probability and its applications. *Economic Theory*, 26(2):445–469, 2005.

[7] Ashok D. Belegundu and Tirupathi R. Chandrupatla. *Optimization Concepts and Applications in Engineering*. Cambridge University Press, second edition, 2011.

[8] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*, volume 3. Springer, 2004.

[9] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[10] Dimitri P Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[11] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 2002.

[12] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to Linear Optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[13] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing, 2012. arXiv cs-PL/1209.5145.

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *AAAI Innovative Applications of Artificial Intelligence Conference*, pages 239–246, 2002.

[16] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3-4):324–345, 1952.

[17] Matthew Brand. Pattern discovery via entropy minimization. Technical Report TR-98-21, MERL– A Mitshubishi Electric Research Laboratory, 1998.

[18] Darius Braziunas. Computational approaches to preference elicitation. *Department of Computer Science, University of Toronto, Tech. Rep*, 2006.

[19] Sebastian Burhenne, Dirk Jacob, and Gregor P Henze. Sampling based on sobol sequences for monte carlo techniques applied to building simulations. *Proceedings of Building Simulation*, 2011.

[20] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *AAAI Innovative Applications of Artificial Intelligence Conference*, pages 363–369, 2000.

[21] Weizhu Chen, Zhenghao Wang, and Jingren Zhou. Large-scale l-bfgs using mapreduce. In *Advances in Neural Information Processing Systems*, pages 1332–1340, 2014.

[22] Manuel Chica, Oscar Cordón, Sergio Damas, and Joaqun Bautista. Including different kinds of preferences in a multi-objective ant algorithm for time and space assembly line balancing on different nissan scenarios. *Expert Systems with Applications*, 38(1):709 – 720, 2011.

[23] Manuel Chica, Oscar Cordón, Sergio Damas, and Joaqun Bautista. Interactive preferences in multiobjective ant colony optimisation for assembly line balancing. *Soft Computing*, pages 1–13, 2014.

[24] Vincent Conitzer. Eliciting single-peaked preferences using comparison queries. *Journal of Artificial Intelligence Research*, 35:161–191, 2009.

[25] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(12):35 – 74, 1995. Planning and Scheduling.

[26] K. Deb, A Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr 2002.

[27] Kalyanmoy Deb. Multi-objective optimization. In *Search Methodologies*, pages 403–449. Springer, 2014.

[28] Kalyanmoy Deb and Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[29] James S Dyer. Mautmultiattribute utility theory. In *Multiple criteria decision analysis: state of the art surveys*, pages 265–292. Springer, 2005.

[30] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):pp. 1–26, 1979.

[31] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[32] Carlos M Fonseca, Peter J Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *ICGA*, volume 93, pages 416–423, 1993.

[33] Alexander I.J. Forrester, András Sóbester, and Andy J. Keane. *Engineering Design via Surrogate Modelling: A Practical Guide*. American Instituite of Aeronautics and Astronautics, 2008.

[34] Xavier Gandibleux. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, volume 52. Springer Science & Business Media, 2002.

[35] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, number 6, pages 721–741. IEEE, 1984.

[36] Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3-4):209–232, 1998.

[37] Yoav Goldberg and Michael Elhadad. splitsvm: Fast, space-efficient, non-heuristic, polynomial kernel computation for nlp applications. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, pages 237–240, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[38] Shengbo Guo and Scott Sanner. Real-time multiattribute Bayesian preference elicitation with pairwise comparison queries. In *International Conference on Artificial Intelligence and Statistics*, pages 289–296, 2010.

[39] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill: A bayesian skill rating system. In *Advances in Neural Information Processing Systems*, pages 569–576, 2006.

[40] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*, pages 82–87. Institute of Electrical and Electronics Engineers, 1994.

[41] Neil Houlsby, Ferenc Huszar, Zoubin Ghahramani, and Jose M. Hernández-lobato. Collaborative gaussian processes for preference learning. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2096–2104. Curran Associates, Inc., 2012.

[42] Ralph L Keeney and Howard Raiffa. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press, 1993.

[43] A. Khvilivitzky. Visual collision avoidance system for unmanned aerial vehicles, December 3 1996. US Patent 5,581,250.

[44] Mykel J. Kochenderfer, Jessica Holland, and James Chryssanthacopoulos. Next-generation airborne collision avoidance system. *Lincoln Laboratory Journal*, 19(1):17–33, 2012.

[45] JSH Kornbluth. A survey of goal programming. *Omega*, 1(2):193–205, 1973.

[46] A Kramer, J. Hasenauer, F. Allgower, and N. Radde. Computation of the posterior entropy in a bayesian framework for parameter estimation in biological networks. In *IEEE Conference on Control Applications*, pages 493–498, 2010.

[47] James K. Kuchar. Methodology for alerting-system performance evaluation. *Journal of Guidance, Control, and Dynamics*, 19(2):438–444, 1996.

[48] Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 24–31. Association for Computational Linguistics, 2003.

[49] Neil Lawrence, Matthias Seeger, and Ralf Herbrich. Fast Sparse Gaussian Process Methods: The Informative Vector Machine. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems*, pages 609–616, 2003.

[50] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting Structured Data*, 2006.

[51] D.V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4).

[52] D MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, July 1992.

[53] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[54] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[55] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Conference on Uncertainty in Artificial Intelligence*, pages 362–369, 2001.

[56] Quirino Paris. The dual of the least-squares method. Technical report, March 2012.

[57] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[58] Michael J.D. Powell. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. *Nonlinear programming*, 9:53–72, 1976.

[59] András Prékopa. On logarithmic concave measures and functions. *Acta Scientiarum Mathematicarum*, 34:335–343, 1973.

[60] Brendon Puntin and Mykel J. Kochenderfer. Traffic alert optimization for airborne collision avoidance systems. *Encounters*, 1(786,088):2–300, 2013.

[61] Rita Almeida Ribeiro. Fuzzy multiple attribute decision making: A review and new preference elicitation techniques. *Fuzzy Sets and Systems*, 78(2):155 – 181, 1996. Fuzzy Multiple Criteria Decision Making.

[62] John Rice. *Mathematical statistics and data analysis*. Cengage Learning, 2006.

[63] Ryan Rifkin, Gene Yeo, and Tomaso Poggio. Regularized least-squares classification. *Nato Science Series Sub Series III Computer and Systems Sciences*, 190:131–154, 2003.

[64] Lorenzo Rosasco and Tomaso Poggio. *A Regularization Tour of Machine Learning: MIT-9.520 Lecture Notes*, December 2014. Manuscript.

[65] Stuart Russelll and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, 3 edition, 2009.

[66] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

[67] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *Computational learning theory*, pages 416–426. Springer, 2001.

[68] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423,623–656, 1948.

[69] Yannis Siskos. *Encyclopedia of Optimization*, volume 1. Springer, 2008.

[70] SN Sivanandam and SN Deepa. *Genetic Algorithm Optimization Problems*. Springer, 2008.

[71] Kyle Smith. Collision avoidance system optimization for closely spaced parallel operations through surrogate modeling. Master's thesis, Massachusetts Instituite of Technology, 77 Massachusetts Ave, Cambridge, MA, 2013.

[72] Kyle Smith, Mykel J. Kochenderfer, Wesley Olson, and Adan Vela. Collision avoidance system optimization for closely spaced parallel operations through surrogate modeling. In *AIAA Guidance, Navigation, and Control Conference*, 2013.

[73] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[74] Michael Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

[75] Andrea Tacchetti, Pavan K. Mallapragada, Matteo Santoro, and Lorenzo Rosasco. GURLS: A least squares library for supervised learning. *Journal of Machine Learning Research*, 14:3201–3205, 2013. Source Code.

[76] Peter J.M. van Laarhoven and Emile H.L. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, volume 37 of *Mathematics and Its Applications*, pages 7–15. Springer Netherlands, 1987.

[77] Paolo Viappiani and Craig Boutilier. Optimal bayesian recommendation sets and myopically optimal choice query sets. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2352–2360. Curran Associates, Inc., 2010.

[78] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton university press, 60th anniversary commemorative edition, 2007.

[79] Ya-xiang Yuan. A modified BFGS algorithm for unconstrained optimization. *IMA Journal of Numerical Analysis*, 11(3):325–332, 1991.

[80] Ali MS Zalzala and Peter J Fleming. *Genetic Algorithms in Engineering Systems*, volume 55. Institution of Engineering and Technology, 1997.

[81] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms. In *Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–301. Springer Berlin Heidelberg, 1998.