EXPLORING AND EXPLOITING STRUCTURE IN LARGE SCALE SIMULATION OPTIMIZATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Nanjing Jian

August 2017

EXPLORING AND EXPLOITING STRUCTURE IN LARGE SCALE SIMULATION OPTIMIZATION

Nanjing Jian, Ph.D.

Cornell University 2017

The thesis explores how to solve simulation-based optimization problems more efficiently using information about the problem structure. Two primary ideas are explored. The first idea involves developing numerical methods to detect the structure of convexity in the output function of a simulation. The second idea exploits simulation traces to generate gradient-like information to guide optimization. The goal of this thesis is to provide insights on solving practical large-scale simulation optimization problems that defy solution by existing simulation-optimization algorithms. In doing so, we hope to inspire algorithmic developments on sub-classes of simulation optimization problems.

In the first part of this thesis, we give a asymptotically valid numerical procedure to detect the convexity in the output function of a simulation model. Since the simulated function could be expensive to evaluate, we adapt a Bayesian sequential sampling procedure, in which a set of new (noisy) samples is collected in each iteration to update a posterior model of the function evaluations. Based on that, the posterior probability that the underlying function is convex is estimated using Monte Carlo simulation, for which we develop variance reduction methods with significant improvement in the efficiency.

The second and third parts of this thesis are on exploiting gradient-like information from the simulation to solve large-scale simulation optimization problems. In the second part of the thesis, we give heuristics for optimizing the overnight rebalancing of the bike-sharing system in New York City with 466 stations and hence 932 decision variables. The heuristics use the number of failed trips incurred by each station to indicate approximate gradient directions, and show effective improvement given any starting solution. In the third part of the thesis we further propose the idea of pseudo-gradient defined as an approximate gradient estimated from the historical simulation traces without rerunning the simulation. We prove that when the objective is strongly convex and the pseudo-gradient is accurately approximated, a pseudo-gradient-search procedure will converge to the near-optimal solution. We give examples of the pseudo-gradient in the cases of rebalancing a bike-sharing system and scheduling and staffing agents in a multi-skill call center. The resulting search procedures perform better than the existing methods for the bike-sharing case and show comparable results to the state-of-art cutting-plane method for the call center example. The results indicate the effectiveness of pseudo-gradient search and its potential of replacing more sophisticated methods.

BIOGRAPHICAL SKETCH

Nanjing Jian grew up in Shanghai, China. When she was born, her mother had just completed her Ph.D. and began her tenure in Transportation Engineering, and her father was finishing up his Ph.D. in Thermal Engineering in the same university. After high school, Nanjing decided to leave her hometown and attend University of Wisconsin - Madison in the School of Engineering under her parents' influences. In her sophomore year, she took an introductory optimization course. The art of math modeling (and the professor's jokes) led her into the world of Operations Research. When she graduated with the highest distinction in Industrial Engineering, she decided to join the School of Operations Research and Information Engineering at Cornell University as a Ph.D. student.

Life as Ph.D. student was filled with hard-working days and rewarding moments, such as the times at the annual INFORMS and Winter Simulation Conference. Apart from doing her research in Simulation Optimization, Nanjing also enjoyed assisting in the teaching of many undergraduate and master level courses, including optimization, simulation, probability and statistics, spreadsheet modeling, and design of manufacturing systems. In her spare time, she tried various interesting courses offered by Cornell, including wine tasting, swing dancing, rock climbing, large boat sailing, archery, and handgun safety, and received her brown belt in Karate.

Upon graduating from Cornell, Nanjing will move to Seattle and join Amazon as a research scientist, using her simulation skills to improve their supply chain.

To my parents:

Xiaohong Chen and Ruimin Jian

ACKNOWLEDGEMENTS

First and foremost, I would like to sincerely thank my advisor Shane Henderson for his guidance and support over the years. I enjoyed our weekly meetings in which he patiently listens me, provides me with brilliant ideas and academic resources, and cheers me up with positive attitudes and kind offerings of chocolate in times of frustration. He has taught me how to think critically and guided my growth as a researcher, and his perspectives on combining theory and practice in the field of simulation optimization motivated me to tackle real-world problems. I could not imagine the completion of this thesis without his generous help.

Parts of this thesis are based on joint work with Susan Hunter of Purdue University, Daniel Freund, and Holly Wiberg during her time at Cornell University, to whom I owe my special appreciation. I also appreciate Damek Davis's help with the proof of Theorem 5.

I would like to thank Peter Frazier and Adrian Lewis for being members of my special committee, and James Renegar for serving as a proxy during my A Exam. They have provided many useful comments and the interesting courses they offered have inspired pieces of this thesis. I would also like to thank the entire faculty at the School of Operations Research and Information Engineering. Special thanks to Kathryn Caggiano, Peter Jackson, and John Callister (and of course, Shane Henderson) for enjoyable semesters of being a teaching assistant. Dating back to my undergraduate years, I give extra special thanks to Jeff Linderoth, whose enjoyable courses on optimization led me into the world of Operations Research.

Finally, I would like to thank my fellow graduate students and office mates Eric Ni, Angela Gao, Jerry Guo, and Yixuan Zhao, who accompanied me through the days and nights in Rhodes 288.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Consider a call center where calls of different types arrive according to a random process. The calls are handled by agents of different skill sets in random durations. How does one determine the most cost-efficient way of hiring agents without incurring excessive wait time? Now consider a bike-sharing system with stations all over a city, where customers arrive randomly at each station to check out a bike, and return it to another station after the ride. What is the optimal capacity of each station, so that there is enough space for the bike returns? And how to redistribute the bike supplies across the system, so there is enough bike where the demand is high? Many other problems can be found on `SimOpt.org` [57]. Due to the complexity and stochastic nature of such decision problems, it is difficult to establish reliable theory without oversimplifying the problem, but using simulation can often capture the system realistically. Simulation optimization is a useful tool for decision support under such conditions. It is the method of choosing decision variables to optimize some performance measures that can only be implicitly evaluated by a simulation model. For an introductory tutorial on the subject and references to the existing literature, see [44].

Simulation optimization problems are not easy to solve. The process usually involves many runs of the simulation model, especially when the problem is large-scale (has many decision variables), requiring considerable computational effort. When the problem lacks certain "nice" structure like convexity, monotonicity, or unimodality, or the decision variables are discrete with relatively large dimensions, there is usually limited (or no) choice from traditional simulation optimization methods. For example, for the (citi) bike-sharing system in New York City, there are over 450 stations, creating 450 integer-valued decision variables if we just want to optimize the bike level at each station, keeping the total number of bikes in the system constant. Exhausting all solutions is impossible, making ranking and selection infeasible for this problem. To use stochastic gradient search (see, e.g. [23]), each step would require at least 450 runs of simulation to evaluate a gradient using finite forward differences. A pure random search would randomly choose two stations among $450 * 449$ pairs to move a bike between them, and hope for an improvement. The same difficulty applies to the multi-skill multi-period call center example, where the decision variable is the number of agents to hire in each skill group and shift, and the dimension can easily scale to thousands of variables. A state-of-the-art heuristic uses a cutting plane approach combined with Sample Average Approximation, but each cut generation still requires gradient estimation by finite forward differences [5].

The more recent approaches for general simulation optimization problems with integer-valued decision variables include the Industrial Strength COMPASS (ISC) and Retrospective Search Using Piecewise Linear Interpolation and Neighborhood Search (R-SPLINE) (see [74] and [71]). The former is a three-stage random search method that is based on genetic algorithm and ranking and selection, and the latter linearly interpolates the objective between integer points and uses gradient search to optimize the interpolated continuous function. Since R-SPLINE relies on the gradient estimation similar to finite forward differences (on integer lattices), and ISC incurs large overhead when the problem dimension exceeds 10, they may not adapt efficiently to specific large problems like optimizing bike-sharing systems or multi-skill multi-period call centers.

The purpose of this thesis is to provide insights on solving practical large-scale simulation optimization problems (e.g. dimension of at least hundreds) that defy solution by existing simulation-optimization algorithms. In doing so, we hope to inspire algorithmic developments in simulation optimization on sub-classes of simulation-optimization problems. Two primary ideas are discussed in this thesis. The first idea (Chapter 2) involves exploring convexity in the output function of a simulation model. The second idea (Chapters 3 and 4) exploits information other than the final performance estimate from the simulation traces and use it to generate approximate gradient to guide optimization. Both ideas aim at gaining more understanding of the simulation model and treating it as a "white-box" instead of the more traditional "black-box" approaches.

1.2 Convexity Detection in Noisy Function Evaluations

Convexity can be exploited in many ways, including providing global convergence guarantee for gradient search procedures. Chapter 2 of this thesis provides a method to explore convexity in functions that can only be evaluated with noise (e.g. simulation output functions). More specifically, consider a real-valued function that can only be observed with stochastic simulation noise at a finite set of points in space. Given the observations, we wish to determine whether there exists a convex function that goes through the true function value at those design points. This discrete notion of convexity is useful when we can only afford to evaluate the true function on a constrained set of points, or when there are finite number of points in the feasible set (e.g. discrete decision variables constrained in a compact set). Our method can be used as a first-stage structural analysis of a noisy function before deploying any gradient-based or cutting plane methods that work most efficiently on convex functions. Apart from checking global convexity, one can use it to detect "basins of attraction" that contains local optimums. The method can also provide qualitative insights into simulation applications where theoretical models cannot be established without oversimplifying the reality.

Since each replication of the simulation is computationally costly, we use a Bayesian sequential sampling procedure. In each iteration, we obtain a new set of samples from the simulation and update a Bayesian conjugate prior model of the true function values. Based on that, the posterior probability that the sampled function is convex is estimated using Monte Carlo simulation. The asymptotic convergence of the procedure is given under both cases when the sampling variance is known and unknown. To improve the efficiency of the estimator, we provide three variance reduction methods - change of measure, acceptance-rejection, and conditional Monte Carlo. The former two methods depend on the likelihood ratios of Gaussian or Student-t posterior densities and exhibit substantial variability in performance. The recommended conditional Monte Carlo method achieves significant variance reduction, giving the highest efficiency among three methods.

The contents of Chapter 2 are contained in [45] that has been submitted for publication, where the initial ideas originate from [42]. The code is accessible from an open-access repository [40].

1.3 Using Gradient-like Information in Optimizing A Large-scale Bike-sharing System

Chapter 3 of this thesis takes an initial step in exploiting gradient-like information in large-scale simulation optimization, under the specific case of optimizing Citi Bike, the bike-sharing system in New York City. The Citi Bike system is a station-based bike-system where customers can check out bikes from any station and return to any other station. It has approximately 466 stations, 6074 bikes, and 15777 docks as of December 2015. The major use of the system is for daily commute, which creates unbalanced flows from and to the public transportation hubs and residential areas in the morning and afternoon rush-hour periods. To accommodate for this, Citi Bike rebalances the system by moving bikes between stations every night and relocate docks every month on a limited basis. We wish to find the optimal bike and dock allocation for each station at the beginning of the day (the decision variables), so that the expected number of customers having fail to find a bike ("failed-start") or a dock to return a bike to ("failed-end") over a day is minimized (the objective), keeping the total number of bikes and docks in the system constant (the constraint).

Given a starting configuration of the number of bikes and docks at each station, we use a discrete-event simulation to evaluate the objective. The model captures the random arrivals of customers at each station, their destinations, and the trip durations. The decision variables are integer-valued with simple constraints, so we could use a search procedure based on the phantom gradient (the gradient of the continuous function obtained by interpolating the objective, see [71]). However, since the decision variable has over 900 dimensions, it would be computationally infeasible to evaluate the phantom gradient directly via simulation using finite forward differences. This motivates us to attempt to identify gradient-like information from simulation traces, so that we can obtain useful search-direction information at very little additional computational cost. For this specific problem, we use the average number of failed-starts and failed-ends in the morning and afternoon rush-hour periods of a station obtained as the indication of the potential benefit from adding/removing a bike or a dock to/from this station. The resulting heuristics successfully make local improvements that better solutions found using any other techniques.

The idea of Chapter 3 originated from our tutorial [43], and some of the contents are contained in the conference paper [44] that is joint work with Daniel Freund and Holly M. Wiberg.

1.4 Using Pseudo-Gradients in Large-Scale Simulation Optimization

Building on Chapter 3, Chapter 4 of this thesis takes a further step in exploiting gradient-like information in large-scale simulation optimization and proposes the concept of pseudo-gradient. We propose the concept of pseudo-gradient as the approximate gradient estimated from the historical simulation traces without re-running the simulation. A pseudo-gradient is not uniquely defined, since it can be any quantitative or directional approximation to the true gradient. One example is to keep the sequence of events as is from the simulation traces, and calculate pseudo-gradient as the change in the objective after a small increase in the decision variables. This idea of "keeping the event list as is" is inspired by Perturbation Analysis that studies the change in the simulation trajectory with regard to a small change in the input parameters that may or may not affect the event list (see, e.g. [38], [26], [22], [24]). To help explain the potential contribution of pseudo-gradients, we prove the asymptotic convergence of gradient-search procedures to approximate optimal points when the objective is strongly convex and the pseudo-gradients are uniformly accurately approximated. Based on the pseudo-gradient, heuristics for optimizing the bike-sharing system and multi-skill multi-period call center are developed.

For the bike-sharing system described in the previous section, we record the sequence of attempted bike pick-ups and drop-offs from the simulation of the current solution. Using the sequence, we calculate the change in the objective value after adding one bike or dock to each station, giving the pseudo-gradient evaluated at the current solution. The availability of pseudo-gradient gives us the flexibility of making changes to many stations simultaneously, resulting in large steps towards the local minimum. When applied to the 466-station example from Chapter 3, the heuristic in Chapter 4 is shown to be more efficient, and converges to the same local minimum when started from different solutions.

In a multi-skill multi-period call center, calls of different types arrive randomly during the day, each requiring specific skills to be answered. Agents of different skill groups are assigned to different shifts. We wish to determine the optimal number of agents in each skill group and shift, so that the cost is minimized and the proportion of calls that are picked up late is under control. In this problem, we record the set of calls that are picked up late and the set of calls answered by the "last" agent in each group. The former is used to calculate the forward pseudo-gradient when one more agent is hired, and the latter is for the backward pseudo-gradient when an agent is removed from each group and shift. When applied to the largest instance we have seen in the literature, the pseudo-gradient search shows comparable result and affordable computational time.

1.5 Major Contributions

We view this thesis as offering three primary contributions. First, we provide an efficient method for numerically detecting convexity in a simulated function. This grants one a new opportunity to examine the structure of a model before developing a more insightful optimization method. Second, we propose new and practical gradient-search-like simulation-optimization methods to the problems of bike-sharing and multi-skill call center optimizations that can tackle the large-scale instances seen in practice. Finally, we (strive to) increase the attention of the simulation-optimization research community on large-scale practical problems that defy solution by existing simulation-optimization algorithms. In doing so, we hope to inspire algorithmic developments in simulation optimization on sub-classes of simulation-optimization problems.

CHAPTER 2

CONVEXITY DETECTION IN NOISY FUNCTION EVALUATIONS

2.1 Introduction

Our goal in this chapter is to develop a method to determine whether a real-valued function $g : S \subseteq \mathbb{R}^d \to \mathbb{R}$, observed at a finite set of points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$ in S with noise from a stochastic simulation model, is convex in the sense that a convex function f exists that coincides with g at $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$. That is, does there exist a convex function f that goes through the points $(\boldsymbol{x}_1, g(\boldsymbol{x}_1)), (\boldsymbol{x}_2, g(\boldsymbol{x}_2)), \ldots, (\boldsymbol{x}_r, g(\boldsymbol{x}_r))$? For example, g could be the expected profit in an inventory problem where the demand ξ is random, the starting inventory $\boldsymbol{x}$ of the day can take integer values $S = \{0, 1, \ldots, \infty\}$, and we only observe a simulation estimate of g at $\boldsymbol{x}$. We might choose a few integer values in S to test whether the expected profit is convex as a function of the starting inventory. Or g might represent the (true) expected waiting time for an ambulance as a function of base locations represented in latitude-longitude coordinates, where we observe g with noise through a stochastic simulation model of ambulance operations. We might then choose a finite set of base-location options in the city to test whether the waiting time is convex with regard to the location of the ambulance bases.

Convexity is a key structural property that can be exploited in many ways. One can use gradient-based methods (for smooth functions) or a cutting-plane based method (for nonsmooth functions) to quickly find the optimum, or bounds on the optimum, e.g., [54] and [28]. Even if a function is not globally convex, one might use our test to identify regions around local minima in which the restriction of the objective function is convex ("basins of attraction"). Such basins can provide information on the stability of a solution [70]. Beyond optimization, convexity can also provide insights into qualitative model behavior in simulation applications and elsewhere. This is especially useful when we have a sequence of similar simulation models with different inputs but the same structural property.

Most studies on convexity detection use a frequentist hypothesis-testing framework, and can be categorized with regard to how the null hypothesis is defined. The first category uses an infinite-dimensional functional approach. It defines the null hypothesis as $g \in \mathcal{C}$, where $\mathcal{C}$ is the cone of convex functions in an appropriate function space, whereas the alternative hypothesis is $g \notin \mathcal{C}$. The representative paper [47] models g as a Gaussian process assuming smoothness under the null hypothesis, and uses the L^r distance between g and $\mathcal{C}$ as the test statistic.

The second category, which is closer to our framework, works with finite-dimensional vectors. The null hypothesis is $\boldsymbol{g} \in \mathbb{C}$, where $\boldsymbol{g}$ is the restriction of g to a finite set of points, and $\mathbb{C}$ is the set of convex functions restricted to the same set of points. In this case, the noisy evaluations of $\boldsymbol{g}$ are modeled by a Gaussian random vector. [68] use the distance between this Gaussian vector and $\mathbb{C}$ as the test statistic, and show that this distance follows a chi-bar-square distribution, the parameters of which can be evaluated using simulation.

The third category fits a regression model with Gaussian noise on the observations of g and tests the hypothesis of convexity through the estimated model parameters. This approach is essentially testing whether there exists a convex function that could have generated the observed finite set of function values. In one dimension and under regularity conditions, [7] show that testing for the regression parameter is equivalent to testing $\boldsymbol{g} \in \mathbb{C}$, when $\mathbb{C}$ is defined with regard to nonnegative second order Vandermonde determinants. Their test is based on the idea that if a one-dimensional function is convex, then the sample mean of the function values in a partition should be lower than certain linear combinations of the function values in neighboring partitions. Others fit cubic splines on the observations and use the second order derivatives at the knots to test for convexity, e.g. [17], [72] and [52]. For higher dimensions, [1] work with small and localized sets of data points and count all the possible convex and concave simplices to construct a test statistic. [49] uses a second-order parametric model for the data points, and is a good survey of early literature.

A closely related field to convexity tests is convex regression, where one fits a regression model to observations under the constraint that the fitted model is convex. Work in this direction includes [46], [2], [66], [35], [50], among which [66] provides a review of past work.

In this chapter, we give a Bayesian sequential algorithm that iteratively collects noisy evaluations of an unknown function g on a fixed and finite set of design points $\boldsymbol{x}$, and uses them to estimate the posterior probability that the function, when restricted to the design points, is convex. Our approach differs from previous research in two main ways. First, since the function estimates are obtained via Monte Carlo simulation, we can only observe the function g on a finite set of points $\boldsymbol{x}$. Thus the best we can provide is a probabilistic guarantee that there exists a convex function that coincides with the unknown function g at those points. Even if there exists such a convex function, it does not imply that g itself is convex, although nonexistence does imply nonconvexity of g. Second, we use a Bayesian conjugate prior model that updates a posterior on the function values every time we collect a set of new samples. Then the posterior probability of convexity is estimated separately through Monte Carlo simulation. Instead of having a fixed running time, as is the case with hypothesis testing, our algorithm can be stopped at any stage to output an estimated probability of convexity. Indeed, the Bayesian approach avoids the difficulty in frequentist hypothesis testing of multiple “looks” at the data, for example to decide when to terminate the test.

Our overall approach is to successively update a posterior distribution on the vector of (true) function values $\boldsymbol{g}$. In doing so we assume that the noise in the estimated function values is normally distributed and assume a conjugate prior so that we can use standard posterior updates. In simulation the normality assumption is common and reasonable, since one can batch multiple replications to obtain approximate normality through the central limit theorem. For a given posterior distribution, computing the probability of convexity for a sample from the posterior appears to be difficult. We use Monte Carlo to estimate this probability, providing three methods for reducing the variance of the Monte Carlo estimator of the posterior probability of convexity. The change of measure and acceptance-rejection methods reuse samples obtained in an earlier iteration to construct an unbiased estimator for the current iteration. These two methods can be useful in any sequential algorithm in a Bayesian framework, but need to be used with caution due to heavy-tailed behavior of the likelihood ratio that is needed in these methods. The conditional Monte Carlo method takes advantage of the spherical property of Gaussian and t distributions. It can be applied to the more general problem of estimating the probability that a Gaussian or t-distributed random vector lies in a polyhedron, which might arise, e.g., in solving linear feasibility problems described in [69].

This chapter is built upon [42], with the addition of new results, complete proofs, new variance reduction methods, and more extensive numerical results.

Notation: We use upper case Latin letters for random variables or sets, and lower case Latin letters for deterministic variables. Vectors are in bold, and matrices are in upper case Greek letters. We use A^T to denote the transpose of the matrix A. For a set S, S° is its interior. We use $\Rightarrow$ for convergence in distribution, and $\mathbb{1}\{B\}$ is the indicator function that takes the value 1 on the event B and 0 otherwise.

2.2 Problem Statement and Assumptions

Suppose we can obtain noisy evaluations of a real-valued function $g : S \to \mathbb{R}$ over a fixed and finite set of design points $\boldsymbol{x} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r\}$, $\forall i \in \{1, 2, \ldots, r\}$ in $S \subseteq \mathbb{R}^d$. Ideally, we would like to know whether the function g is convex on S or not. In the absence of any regularity assumptions on g, such as are assumed in [47], it appears that this question cannot be addressed in finite time. The rest of this chapter is focused on giving a probabilistic guarantee on the convexity of the finite-dimensional vector obtained by restricting the function g to the r points in $\boldsymbol{x}$, i.e., we restrict attention to $\boldsymbol{g} = (g(\boldsymbol{x}_1), g(\boldsymbol{x}_2), \ldots, g(\boldsymbol{x}_r)) \in \mathbb{R}^r$.

Definition 1. *Given a finite set of points $\boldsymbol{x}$, we define a vector $\boldsymbol{g}$ to be convex if and only if there exists a convex function g whose values on $\boldsymbol{x}$ coincide with $\boldsymbol{g}$, i.e. $g(\boldsymbol{x}) = \boldsymbol{g}$.*

Definition 2. *Define $\mathbb{C} = \mathbb{C}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r) \subseteq \mathbb{R}^r$ to be the set of all convex vectors on $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r)$, so that $\boldsymbol{g}$ is convex if and only if $\boldsymbol{g} \in \mathbb{C}$.*

The notion of vector convexity is weaker than (the usual) functional convexity, since if a function g is convex then its restriction $\boldsymbol{g}$ on $\boldsymbol{x}$ is convex under our definition. The converse is not true since we can arbitrarily extend $\boldsymbol{g}$.

The set $\mathbb{C}$ is a convex cone. In Section 2.5, we will see that $\boldsymbol{g} \in \mathbb{C}$ if and only if a certain linear system is feasible, and the linear system is then a tool to verify vector convexity. We will also show that $\boldsymbol{g}$ is strictly convex (in the sense that a strictly convex function g exists that agrees with $\boldsymbol{g}$ on $\boldsymbol{x}$) iff $\boldsymbol{g} \in \mathbb{C}^\circ$ in Appendix A.1.

We use a Bayesian approach, regarding $\boldsymbol{g}$ as an unknown realization from the prior distribution of a random vector $\boldsymbol{f}$. Let $(\boldsymbol{\xi}_j : j = 1, 2, \ldots)$ be an i.i.d. sequence of r-dimensional Gaussian random vectors, each with mean vector $\mathbf{0}$ and covariance matrix Γ, that is independent of $\boldsymbol{f}$. Our jth observation is then $\boldsymbol{Y}_j$, where $\boldsymbol{Y}_j = \boldsymbol{f} + \boldsymbol{\xi}_j$, $j = 1, 2, \ldots$. We denote the ith component of the vector $\boldsymbol{Y}_j$ by Y_{ij}, and interpret it as the jth sampled function value at the point $\boldsymbol{x}_i$.

The covariance matrix Γ is not necessarily diagonal, i.e., we do not necessarily constrain the observations to be (conditionally) independent between sampled points, conditional on $\boldsymbol{f}$. Thus, Common Random Numbers (CRN) can be employed within our framework. CRN induces positive correlation on the r dimensions of the noise $\boldsymbol{\xi}$, so that the structure of the underlying "true" function $\boldsymbol{f}$ can be better preserved than would be possible with conditionally independent observations [14]. For simplicity, we assume throughout that the covariance matrix Γ is positive definite.

2.3 Sequential Algorithm

In our algorithm, the information gained after n iterations of sampling is represented by a σ-field $\mathscr{A}_n$. Initially, before any sampling, we fix the r points $\boldsymbol{x}_i, i = 1, \ldots, r$ and the prior mean μ_0 and covariance matrix Λ_0 of the assumed Gaussian prior distribution of $\boldsymbol{f}$. At the beginning of the nth iteration ($n = 1, 2, \ldots$), we obtain a new observation $\boldsymbol{Y}_n$, and use that to update the posterior distribution on the function values, as described in Section 2.4. The information collected thus far is denoted $\mathscr{A}_n$, which is the sigma field generated from $\mathscr{A}_0$ and $\{\boldsymbol{Y}_j, j = 1, \ldots, n\}$. Thus, $\{\mathscr{A}_n\}_{n=0,1,2,\ldots}$ is a filtration. Once the posterior distribution has been updated, we separately estimate the posterior probability of convexity, $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$ as discussed in Section 2.5. At the end of each iteration, we can choose either to stop, or to continue with the current posterior as the prior of the next iteration. More precisely, the algorithm is as follows.

Algorithm 1 A sequential method for testing for convexity of the function

```
Input: The Gaussian prior $P(\boldsymbol{f} \in \cdot|\mathscr{A}_0)$, with hyperparameters $\{\mu_0, \Lambda_0\}$ of the
    function values $\boldsymbol{f}$.
1: Initialize $n = 0$.
2: repeat
3:     Set $n = n + 1$.
4:     Obtain a new vector $\boldsymbol{y}_n$ of $r$ noisy function values at $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r$.
5:     Update the posterior distribution of $\boldsymbol{f}|\mathscr{A}_n$ from the new samples $\boldsymbol{y}_n$ using
       $\boldsymbol{f}|\mathscr{A}_{n-1}$ as the prior, as in Section 2.4.
6:     Estimate $p_n = P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$ from the distribution of $\boldsymbol{f}|\mathscr{A}_n$ using the
       Monte Carlo method described in Section 2.5, obtaining a confidence interval
       $[\hat{p}_n - h_n, \hat{p}_n + h_n]$.
7: until stopped return A confidence interval $[\hat{p}_n - h_n, \hat{p}_n + h_n]$ of $p$.
```

In Step 6, the posterior probability that $\boldsymbol{f}$ is convex is estimated using Monte Carlo simulation. The samples are obtained from the posterior distribution $\boldsymbol{f}|\mathscr{A}_n$, unlike the samples from $\boldsymbol{f}$ used to update the posterior in Step 5. This avoids excessive sampling from the original $\boldsymbol{f}$, which may be computationally expensive.

2.4 Posterior Updates

We use a conjugate prior to update our belief about $\boldsymbol{f}|\mathscr{A}_n$ in each iteration n. Since we assumed that $\boldsymbol{Y} - \boldsymbol{f} \sim N(\mathbf{0}, \Gamma)$, this conjugate prior is normal-normal when Γ is known, and normal-inverse-Wishart when Γ is unknown. In this section we give the updating formula of the posterior distribution of $\boldsymbol{f}|\mathscr{A}_n$ in Step 5 of Algorithm 1 under these two scenarios, given the prior $\boldsymbol{f}|\mathscr{A}_{n-1}$. The formulae given here are standard; see, e.g., [16], [25], or [9].

2.4.1 Conjugate Prior under Known Sampling Variance

First, before any sampling we select a non-informative Gaussian prior with zero mean and large variance, i.e. $\boldsymbol{f}|\mathscr{A}_0 \sim N(\mu_0, \Lambda_0)$ in which $\mu_0 = \mathbf{0} \in \mathbb{R}^r$ and $\Lambda_0 \in \mathbb{R}^{r\times r}$ is a diagonal matrix with diagonal values that are large relative to the sampling variance (the diagonal of Γ). Alternative parameters for the Gaussian prior could be used in the presence of more information, and would not change the algorithm.

At iteration $n > 1$, the posterior from the last iteration $\boldsymbol{f}|\mathscr{A}_{n-1} \sim N(\mu_{n-1}, \Lambda_{n-1})$ is used as the prior for the current iteration. We then obtain $s \geq 1$ new objective-function samples $(y_{ij}, j = 1, 2, \ldots, s)$ at each of the design points $\boldsymbol{x}_i, i = 1, 2, \ldots, r$. The mean μ_n and covariance Λ_n of the posterior $\boldsymbol{f}|\mathscr{A}_n \sim N(\mu_n, \Lambda_n)$ are updated by

$$\begin{aligned}\Lambda_n^{-1} &= \Lambda_{n-1}^{-1} + s\Gamma^{-1}\\ \mu_n &= \Lambda_n(\Lambda_{n-1}^{-1}\mu_{n-1} + s\Gamma^{-1}\bar{\boldsymbol{y}}),\end{aligned} \tag{2.1}$$

where the i-th component of the r-dimensional vector $\bar{\boldsymbol{y}}$ is $s^{-1}\sum_{j=1}^{s} \boldsymbol{y}_{ij}$. One can adaptively choose the sample size s in each iteration, but for simplicity we use $s = 1$, meaning that only one new sample is obtained in each iteration.

The updating equation (2.1) involves matrix inversion. To reduce the computational effort, we use Cholesky factorization and the Sherman-Morrison-Woodbury formula as detailed in Appendix A.2.

2.4.2 Conjugate Prior under Unknown Sampling Variance

When the sampling variance Γ is unknown, the inverse-Wishart distribution provides a conjugate prior. First, we use an uninformative Jeffrey's prior, where the prior joint distribution on $\boldsymbol{f}$ and Γ is proportional to $|\Gamma|^{-(r+1)/2}$ [25] and $|A|$ denotes the determinant of the matrix A. To construct Jeffrey's prior, an initial set of r-dimensional samples $\boldsymbol{y}_j = (y_{ij}, i = 1, 2, \ldots, s), j = 1, 2, \ldots, s_0$ are used to estimate the parameters of the normal distribution for the mean $\boldsymbol{f}$ and the Inverse-Wishart distribution (Inv-Wishart) for the variance Γ. The initial sample size s_0 can be any positive integer. For a prior that reflects the data without being too costly, we choose $s_0 = r + 1$, where r is the number of design points. This choice also ensures that the inverse-Wishart distribution is concentrated on covariance matrices that are positive definite. More specifically [25],

$$\begin{aligned} \Gamma|\mathscr{A}_0, \boldsymbol{y} &\sim \text{Inv-Wishart}_{\upsilon_0}(\Xi_0^{-1}) \\ \boldsymbol{f}|\Gamma, \mathscr{A}_0, \boldsymbol{y} &\sim N(\mu_0, \Gamma/\kappa_0), \end{aligned} \tag{2.2}$$

where

$$\mu_0 = \frac{1}{s_0}\sum_{j=1}^{s_0} \boldsymbol{y}_j = \bar{\boldsymbol{y}}; \quad \kappa_0 = s_0; \quad \upsilon_0 = s_0 - 1; \quad \Xi_0 = \left(\sum_{j=1}^{s_0} (\boldsymbol{y}_j - \bar{\boldsymbol{y}})(\boldsymbol{y}_j - \bar{\boldsymbol{y}})^T\right)^{-1}.$$

In iteration $n \geq 1$, we obtain s samples y_{ij} on each of the points $\boldsymbol{x}_i, i = 1, \ldots, r$

and update the posterior of

$$\begin{aligned} \Gamma|\mathscr{A}_n &\sim \text{Inv-Wishart}_{\upsilon_n}(\Xi_n^{-1}) \\ \boldsymbol{f}|\Gamma, \mathscr{A}_n &\sim N(\mu_n, \Gamma/\kappa_n) \end{aligned} \tag{2.3}$$

by [25]:

$$\begin{aligned} \mu_n &= \frac{\kappa_{n-1}}{\kappa_{n-1}+s}\mu_{n-1} + \frac{s}{\kappa_{n-1}+s}\bar{\boldsymbol{y}}; \quad \kappa_n = \kappa_{n-1}+s; \quad \upsilon_n = \upsilon_{n-1}+s; \\ \Xi_n &= \Xi_{n-1} + S + \frac{\kappa_{n-1}s}{\kappa_{n-1}+s}(\bar{\boldsymbol{y}}-\mu_{n-1})(\bar{\boldsymbol{y}}-\mu_{n-1})^T, \end{aligned} \tag{2.4}$$

where the i-th component of the r dimensional vector $\bar{\boldsymbol{y}}$ is defined as $\bar{\boldsymbol{y}}_i = \sum_{j=1}^s y_{ij}/s,\ i = 1, \dots, r$, and the $r \times r$ matrix S is the sum of squared errors $\sum_{j=1}^s (\boldsymbol{y}_j - \bar{\boldsymbol{y}})(\boldsymbol{y}_j - \bar{\boldsymbol{y}})^T$. For simplicity we again choose $s = 1$ when updating, so that $\boldsymbol{y}_j = \bar{\boldsymbol{y}}$ and $S = 0$.

If a random $r \times r$ matrix Γ has the Inverse-Wishart distribution with parameters υ and Ξ^{-1}, whose density is proportional to $|\Xi|^{\upsilon/2}|\Gamma|^{-(\upsilon-r-1)/2}\exp\{-\text{tr}(\Xi\Gamma^{-1})/2\}$, where $\text{tr}(\cdot)$ is the trace of a matrix, the inverse Γ^{-1} has the Wishart distribution with parameters υ and Ξ. The Wishart distribution is a higher-dimensional generalization of the χ^2 distribution, and thus can be expressed similarly as the sum of squares of Gaussian random vectors. To generate a $\text{Wishart}_\upsilon(\Xi)$ distributed random matrix Γ, we generate $\upsilon > r$ independent, r-dimensional random vectors W_i distributed as $N(0, \Xi)$, and return $\Gamma = \sum_{i=1}^{\upsilon} W_i W_i^T$.

With the posterior covariance distributed as Inverse-Wishart and the posterior mean distributed as Gaussian conditioning on the covariance, the marginal distribution of the posterior mean $\boldsymbol{f}|\mathscr{A}_n$ is

$$\boldsymbol{f}|\mathscr{A}_n \sim t_{(\upsilon_n-r+1)}(\mu_n, \Xi_n/(\kappa_n(\upsilon_n-r+1))), \tag{2.5}$$

where $t_{(\upsilon_n-r+1)}(\mu_n, \Xi_n/(\kappa_n(\upsilon_n-r+1)))$ is a multivariate Student-t distribution with (υ_n-r+1) degrees of freedom, location parameter μ_n, and scale matrix $\Xi_n/(\kappa_n(\upsilon_n-$

$r+1)$). The density function of $\boldsymbol{f}|\mathscr{A}_n$ is thus proportional to $|\Xi_n/(\kappa_n(\upsilon_n - r + 1))|^{-1/2}\{1 + (\boldsymbol{f} - \mu_n)^T[\Xi_n/(\kappa_n(\upsilon_n - r + 1))]^{-1}(\boldsymbol{f} - \mu_n)\}^{-(\upsilon_n+1)/2}$ [25].

2.5 Convexity

Recall that $\boldsymbol{g} = (g(\boldsymbol{x}_1), g(\boldsymbol{x}_2), \ldots, g(\boldsymbol{x}_r))$, and the vector $\boldsymbol{g}$ is defined to be convex if and only if there exists a convex function that coincides with $\boldsymbol{g}$ on the set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r$. Equivalently, for each $i = 1, \ldots, r$ there exists a hyperplane $\{\boldsymbol{a}_i^T\boldsymbol{x} + b_i : \boldsymbol{x} \in \mathbb{R}^d\}$ that goes through $(\boldsymbol{x}_i, g(\boldsymbol{x}_i))$ and lies at or below all the other points $(\boldsymbol{x}_j, g(\boldsymbol{x}_j)), j \neq i$ ([53, p.539]; [4]). That is, $\boldsymbol{g}$ is convex if and only if there exists feasible solutions $\boldsymbol{a}_i \in \mathbb{R}^d, i = 1, \ldots, r$ and $\boldsymbol{b} \in \mathbb{R}^r$ to the linear system

$$\begin{aligned} &\boldsymbol{a}_i^T\boldsymbol{x}_i + \boldsymbol{b}_i = g(\boldsymbol{x}_i), \quad \text{for all } i \in \{1, \ldots, r\} \\ &\boldsymbol{a}_i^T\boldsymbol{x}_j + \boldsymbol{b}_i \leq g(\boldsymbol{x}_j), \quad \text{for all } i \in \{1, \ldots, r\} \text{ and } j \neq i,\ j \in \{1, \ldots, r\}, \end{aligned} \tag{LS}$$

with $\boldsymbol{b}_i$ being the i-th component of $\boldsymbol{b}$. Let the set of all $\boldsymbol{g}$ such that the corresponding LS is feasible be $\mathbb{C}$, which denotes the set of all convex vectors $\boldsymbol{g}$ with regard to the r design points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r$.

This large linear system can also be decomposed into r sub-systems, indexed by $i = 1, \ldots, r$:

$$\begin{aligned} &\boldsymbol{a}_i^T\boldsymbol{x}_i + b_i = g(\boldsymbol{x}_i) \\ &\boldsymbol{a}_i^T\boldsymbol{x}_j + b_i \leq g(\boldsymbol{x}_j), \quad \text{for all } j \neq i \text{ and } j \in \{1, \ldots, r\}, \end{aligned} \tag{LS(i)}$$

each with the variables $\boldsymbol{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$.

Transforming the question of whether a vector is convex to the feasibility of r linear systems allows us to use Monte Carlo simulation to estimate the posterior probability of convexity at the end of each iteration n. We first simulate m random samples from the posterior distribution of $\boldsymbol{f}|\mathscr{A}_n$. Then, for each generated sample, we determine feasibility (or lack thereof) for the linear systems (LS(i), $i = 1, \ldots, r$) in sequence. If *any* linear system is infeasible then we stop (skip the rest of the systems) and conclude that this generated sample is not convex, since one cannot define an appropriate hyperplane. The probability $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$ is then estimated by the sample average of the indicators of convexity for each sample as described more formally in Algorithm 2.

Algorithm 2 Subroutine used in Step 6 of Algorithm 1 to estimate $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$.

```
Input: The posterior marginal density of $\boldsymbol{f}|\mathscr{A}_n$ from (2.1) or (2.5).
1: Generate independent samples $\{\boldsymbol{y}_n^1, \boldsymbol{y}_n^2, \ldots, \boldsymbol{y}_n^m\}$ from the posterior marginal
   density of $\boldsymbol{f}|\mathscr{A}_n$.
2: for $k$ from 1 to $m$ do
3:     Set $\mathbb{1}\left\{\boldsymbol{y}_n^k \in \mathbb{C}\right\} = 1$.
4:     for $i$ from 1 to $r$ do
5:         Set $g(\boldsymbol{x}_i)$ as the $i$-th component of $\boldsymbol{y}_n^k$, $i = 1, \ldots, r$.
6:         Solve for the feasibility of LS($i$).
7:         if LS($i$) is infeasible then
8:             Set $\mathbb{1}\left\{\boldsymbol{y}_n^k \in \mathbb{C}\right\} = 0$.
9:             BREAK the inner loop and go to next $k$.
   return The center $\hat{p}_n = \sum_{k=1}^{m} \mathbb{1}\left\{\boldsymbol{y}_n^k \in \mathbb{C}\right\}/m$ and half-width $h_n = 1.96 s_n/\sqrt{m}$
   of a 95% confidence interval for $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$, where $s_n$ is the sample standard
   deviation of $\mathbb{1}\left\{\boldsymbol{y}_n^k \in \mathbb{C}\right\}, k = 1, \ldots, m$.
```

2.6 Asymptotic Validity of the Main Algorithm

We now establish that the posterior probability of convexity converges to 1 or 0, depending on whether $\boldsymbol{g}$ is convex or not, with one qualification. If $\boldsymbol{g}$ is convex but not strictly convex then it lies on the boundary of $\mathbb{C}$, and then certain arbitrarily small perturbations of the function values $\boldsymbol{g}$ will yield points outside $\mathbb{C}$. Since we estimate the function values $\boldsymbol{g}$ using simulation, we cannot rule out such perturbations, and so we should not expect the posterior probability of convexity to converge to 1 or 0. Accordingly, we only show that when $\boldsymbol{f}$ is strictly convex then the posterior probability of convexity converges to 1, and when $\boldsymbol{f}$ is not convex the posterior probability of convexity converges to 0. The remaining case where $\boldsymbol{f}$ lies on the boundary of $\mathbb{C}$ has probability 0 under our prior, which has a density with respect to Lebesgue measure.

Theorem 1. *Let $p_n = P(\boldsymbol{f} \in \mathbb{C} | \mathscr{A}_n)$ be the n-iteration posterior probability that $\boldsymbol{f}$ is convex as in Algorithm 1. As the number of iterations $n \to \infty$, $p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\} \to 0$ a.s.*

Proof. [42] has a sketch of the proof in the known variance case. We provide a complete proof that covers both the known and unknown variance cases here. First suppose that Γ is known. Proposition 5.16 of [9] then establishes that

$$\Lambda_n^{-1/2}(\mu_n - \boldsymbol{f}) \Rightarrow N(0, I), \text{ as } n \to \infty. \tag{2.6}$$

Using the posterior updating equations (2.1) when Γ is known, we have $\Lambda_n^{-1} = \Lambda_0^{-1} + n\Gamma^{-1}$. The square root operator is continuous over the set of positive definite matrices, so it follows that $\Lambda_n^{-1/2} = (\Lambda_0^{-1} + n\Gamma^{-1})^{1/2} \sim \sqrt{n}\,\Gamma^{-1/2}$. Slutsky's theorem applied to (2.6) then yields $\mu_n - \boldsymbol{f} \to 0$ in probability as $n \to \infty$. (All norms are Euclidean in this proof.)

Now, $P(\boldsymbol{f} \in \partial\mathbb{C}) = 0$, where $\partial\mathbb{C}$ denotes the boundary of $\mathbb{C}$, since $\boldsymbol{f}$ has a density and $\partial\mathbb{C}$ is a union of a finite number of lower-dimensional faces. Thus, it is sufficient to consider the cases (i) $\boldsymbol{f} \in \mathbb{C}^\circ$, the interior of $\mathbb{C}$, which corresponds to the set of strictly convex vectors (see Appendix A.1), and (ii) $\boldsymbol{f} \notin \mathbb{C}$.

On the event $\boldsymbol{f} \notin \mathbb{C}$, we can strictly separate the point $\boldsymbol{f}$ from $\mathbb{C}$ by a hyperplane. Hence we can define a random variable $D_{\boldsymbol{f}} > 0$ such that on the event

$\boldsymbol{f} \notin \mathbb{C}$, all points in $\mathbb{C}$ are at least a distance $D_{\boldsymbol{f}}$ from $\boldsymbol{f}$. (Arbitrarily define $D_{\boldsymbol{f}} = 1$ on the event $\boldsymbol{f} \in \mathbb{C}$.) Let $Z \sim N(0, I)$ be a standard normal random vector, defined on the same probability space as all other random variables, and independent of all else. On the event $\boldsymbol{f} \notin \mathbb{C}$, and using the fact that $\boldsymbol{f}$ has a normal posterior distribution with parameters μ_n and Λ_n,

$$\begin{aligned}
p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\} &= P(\boldsymbol{f} \in \mathbb{C} | \mathscr{A}_n) - 0 \\
&= P(\mu_n + \Lambda_n^{1/2} Z \in \mathbb{C} | \mathscr{A}_n) \\
&\leq P(\|\mu_n + \Lambda_n^{1/2} Z - \boldsymbol{f}\| \geq D_{\boldsymbol{f}} | \mathscr{A}_n) \\
&\leq P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2 | \mathscr{A}_n) + P(\|\Lambda_n^{1/2} Z\| \geq D_{\boldsymbol{f}}/2 | \mathscr{A}_n). \qquad (2.7)
\end{aligned}$$

Consider the first random variable $R_n = P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2 | \mathscr{A}_n)$ in (2.7). Then $R_n \geq 0$ and $ER_n = P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2)$. We will show that this expectation converges to 0 as $n \to \infty$, and hence by Markov's inequality, it immediately follows that $R_n \to 0$ in probability as $n \to \infty$.

To this end, since $D_{\boldsymbol{f}} > 0$, for any $\epsilon > 0$ we can find $\delta > 0$ such that $P(D_{\boldsymbol{f}} \leq \delta) \leq \epsilon$. Hence,

$$\begin{aligned}
P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2) &= P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2, D_{\boldsymbol{f}} \leq \delta) + P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2, D_{\boldsymbol{f}} > \delta) \\
&\leq P(D_{\boldsymbol{f}} \leq \delta) + P(\|\mu_n - \boldsymbol{f}\| \geq \delta/2) \\
&\leq \epsilon + P(\|\mu_n - \boldsymbol{f}\| \geq \delta/2). \qquad (2.8)
\end{aligned}$$

But $\|\mu_n - \boldsymbol{f}\| \to 0$ as $n \to \infty$ in probability, so the second term in (2.8) converges to 0 as $n \to \infty$, and since $\epsilon > 0$ was arbitrary, we see that $ER_n \to 0$ as $n \to \infty$.

A similar approach works for the second term, $R_n' = P(\|\Lambda_n^{1/2} Z\| \geq D_{\boldsymbol{f}}/2 | \mathscr{A}_n)$. Again, R_n' is non-negative, so that if its expectation $P(\|\Lambda_n^{1/2} Z\| \geq D_{\boldsymbol{f}}/2) \to 0$ as $n \to \infty$, then R_n' also converges to 0 in probability. As before, for any $\epsilon > 0$ we can find $\delta > 0$ such that $P(D_{\boldsymbol{f}} \leq \delta) \leq \epsilon$. Hence, as in (2.8)

$$P(\|\Lambda_n^{1/2} Z\| \geq D_{\boldsymbol{f}}/2) \leq \epsilon + P(\|\Lambda_n^{1/2} Z\| \geq \delta/2).$$

By the Cauchy-Schwarz inequality,

$$\begin{aligned} P(\|\Lambda_n^{1/2} Z\| \geq \delta/2) &\leq P(\|\Lambda_n^{1/2}\| \, \|Z\| \geq \delta/2) \\ &= P(\|Z\|^2 \geq t_n), \end{aligned}$$

where

$$t_n = \frac{\delta^2}{4} \frac{1}{\|\Lambda_n^{1/2}\|^2} \geq \frac{\delta^2}{4\|\Lambda_n\|}.$$

Now, $\Lambda_n \sim \Gamma/n$, hence $\Lambda_n^{1/2} \sim \Gamma^{1/2}/\sqrt{n}$, and so Markov's inequality gives

$$P(\|Z\|^2 \geq t_n) \leq \frac{r}{t_n} = \frac{4r\|\Lambda_n^{1/2}\|^2}{\delta^2} \to 0$$

as $n \to \infty$.

We thus conclude that (2.7) converges to 0 in probability as $n \to \infty$. Since $(p_n : n \geq 0)$ is a uniformly integrable martingale, it converges almost surely [73], and hence the almost sure limit is $\mathbb{1}\{\boldsymbol{f} \in C\}$.

On the other hand, on the event $\boldsymbol{f} \in \mathbb{C}^\circ$ we redefine $D_{\boldsymbol{f}}$ to be the radius of a ball, centered at $\boldsymbol{f}$, that is wholly contained in $\mathbb{C}$, and off this event we redefine $D_{\boldsymbol{f}} = 1$. We then find that, on the event $\boldsymbol{f} \in \mathbb{C}^\circ$,

$$\begin{aligned} p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\} &= P(\boldsymbol{f} \in \mathbb{C} | \mathscr{A}_n) - 1 \\ &= P(\mu_n + \Lambda_n^{1/2} Z \in \mathbb{C} | \mathscr{A}_n) - 1 \\ &\geq P(\|\mu_n + \Lambda_n^{1/2} Z - \boldsymbol{f}\| \leq D_{\boldsymbol{f}} | \mathscr{A}_n) - 1 \\ &\geq P(\|\mu_n - \boldsymbol{f}\| \leq D_{\boldsymbol{f}}/2, \|\Lambda_n^{1/2} Z\| \leq D_{\boldsymbol{f}}/2 | \mathscr{A}_n) - 1 \\ &\geq -P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2 | \mathscr{A}_n) - P(\|\Lambda_n^{1/2} Z\| \geq D_{\boldsymbol{f}}/2 | \mathscr{A}_n), \end{aligned}$$

and the proof follows as in the case where $\boldsymbol{f} \notin \mathbb{C}$. This concludes the proof when Γ is known.

Now consider the unknown variance case. From (2.5), $\boldsymbol{f}|\mathscr{A}_n$ follows a multivariate t distribution with $\upsilon_n - r + 1$ degrees of freedom, mean μ_n, and variance $\Lambda_n = \frac{\upsilon_n - r+1}{\upsilon_n - r-1}\Xi_n/(\kappa_n(\upsilon_n - r + 1))$. From (2.4), Ξ_n is of order n, and both υ_n and κ_n are of order n. Thus, $\Lambda_n \to \mathbf{0}$ as $n \to \infty$ a.s., where $\mathbf{0}$ is a matrix of zero components.

By Proposition 5.14 of [9],

$$\Lambda_n^{-1/2}(\mu_n - \boldsymbol{f}) \Rightarrow N(0, I), \text{ as } n \to \infty. \tag{2.9}$$

Then this together with $\Lambda_n \to \mathbf{0}$ a.s. yields $\mu_n - \boldsymbol{f} \to 0$ in probability as $n \to \infty$ by Slutsky's Theorem.

Again, it is sufficient to consider the cases $\boldsymbol{f} \in \mathbb{C}^{\circ}$, the interior of $\mathbb{C}$, or $\boldsymbol{f} \notin \mathbb{C}$.

As before, on the event $\boldsymbol{f} \notin \mathbb{C}$, we can define a random variable $D_{\boldsymbol{f}} > 0$ such that $D_{\boldsymbol{f}} = 1$ on the event $f \in \mathbb{C}$, and such that on the event $f \notin \mathbb{C}$, all points in $\mathbb{C}$ are at least a distance $D_{\boldsymbol{f}}$ from $\boldsymbol{f}$. Let $Z \sim N(0, I)$ be a standard normal random vector, defined on the same probability space as all other random variables, and independent of all else. On the event $\boldsymbol{f} \notin \mathbb{C}$, we use the fact that $\boldsymbol{f}|\Gamma$ has a normal posterior distribution with parameters μ_n and Γ/κ_n and take $\Gamma^{1/2}$ to be the symmetric square root of Γ. Then

$$\begin{aligned}
p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\} &= P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n) - 0 \\
&= E(P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n, \Gamma)|\mathscr{A}_n) \\
&= E(P(\mu_n + \Gamma^{1/2}Z/\sqrt{\kappa_n} \in \mathbb{C}|\mathscr{A}_n, \Gamma)|\mathscr{A}_n) \\
&\leq E(P(\|\mu_n + \Gamma^{1/2}Z/\sqrt{\kappa_n} - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}|\mathscr{A}_n, \Gamma)|\mathscr{A}_n) \\
&\leq E(P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2|\mathscr{A}_n, \Gamma)|\mathscr{A}_n) \\
&\qquad + E(P(\|\Gamma^{1/2}Z/\sqrt{\kappa_n}\| \geq D_{\boldsymbol{f}}/2|\mathscr{A}_n, \Gamma)|\mathscr{A}_n) \\
&= P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2|\mathscr{A}_n) + E(P(\|\Gamma^{1/2}Z/\sqrt{\kappa_n}\| \geq D_{\boldsymbol{f}}/2|\mathscr{A}_n, \Gamma)|\mathscr{A}_n).
\end{aligned} \tag{2.10}$$

The first term in (2.10) converges to 0 in probability exactly as before. The second term, R'_n say, is non-negative, so that if its expectation $P(\|\Gamma^{1/2}Z/\sqrt{\kappa_n}\| \geq D_{\boldsymbol{f}}/2) \to 0$ as $n \to \infty$, then R'_n also converges to 0 in probability. As before, for any $\epsilon > 0$ we can find $\delta > 0$ such that $P(D_{\boldsymbol{f}} \leq \delta) \leq \epsilon$ and hence

$$E(R'_n) \leq \epsilon + P(\|\Gamma^{1/2}Z/\sqrt{\kappa_n}\| \geq \delta/2). \tag{2.11}$$

Denoting the second term in (2.11) by Q_n, we see that

$$\begin{aligned}
Q_n &\leq P(\|Z\|\|\Gamma^{1/2}\| \geq \delta\sqrt{\kappa_n}/2) && (2.12) \\
&= P(\|Z\|^2 \geq \frac{\delta^2\kappa_n}{4\|\Gamma^{1/2}\|^2}) \\
&\leq E(\frac{4r\|\Gamma^{1/2}\|^2}{\delta^2\kappa_n}) && (2.13) \\
&= \frac{4r}{\delta^2\kappa_n}E(\lambda_{\max}(\Gamma)) && (2.14) \\
&\leq \frac{4r}{\delta^2\kappa_n}E(\mathrm{tr}(\Gamma)) \\
&= \frac{4r}{\delta^2(n+\kappa_0)}\sum_{i=1}^{r}E(\Gamma_{ii}),, && (2.15)
\end{aligned}$$

where $\lambda_{\max}(\Gamma)$ denotes the maximum eigenvalue of Γ, and $\mathrm{tr}(\Gamma)$ denotes the trace of Γ. The step (2.14) is by the definition of Euclidean norm of a real and symmetric matrix with non-negative eigenvalues. In (2.12) we use the sub-multiplicative property of the Euclidean norm [30], and (2.13) is by Markov's inequality. By the Jeffery's prior of $\Gamma|\mathscr{A}_n$ in (2.2), conditioning on an initial sample set $\boldsymbol{y}$, $\Gamma|\mathscr{A}_0, \boldsymbol{y} \sim \text{Inv-Wishart}_{\upsilon_0}(\Xi_0^{-1})$, with expectation $E(\Gamma) = (\upsilon_0 - \kappa_0 - 1)^{-1}\Xi_0^{-1}$. Thus the summation in (2.15) is finite, we conclude that $ER_n^{'} \to 0$ in probability, and further that $p_n - \mathbb{1}\{\boldsymbol{f} \in C\} \to 0$ in probability by (2.10). Again, since $(p_n : n \geq 0)$ is a uniformly integrable martingale, it converges almost surely [73], and hence the almost-sure limit is $\mathbb{1}\{\boldsymbol{f} \in C\}$.

On the other hand, in the case $\boldsymbol{f} \in \mathbb{C}^{\circ}$, $p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\} \geq -P(\|\mu_n - \boldsymbol{f}\| \geq D_{\boldsymbol{f}}/2|\mathscr{A}_n) - E(P(\|\Lambda_n^{1/2}Z\| \geq D_{\boldsymbol{f}}/2|\mathscr{A}_n, \Gamma)|\mathscr{A}_n)$ and the proof follows similarly as above. □

The result of Theorem 1 relates to the exact posterior probability of convexity, which we estimate using Monte Carlo. We next show that the Monte Carlo estimator from Section 2.5 of the exact probability converges to the same indicator provided that the Monte Carlo sample sizes increase without bound, through a uniform law of large numbers.

Corollary 2. *Let p_n^m be the m-sample estimator of $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$ from Algorithm 0. As $n \to \infty$ and $m = m(n) \to \infty$, $p_n^m - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\} \to 0$ in probability.*

Proof. We have $|p_n^m - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\}| \leq |p_n^m - p_n| + |p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\}|$, where $p_n = P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$ and $p_n^m = \frac{1}{m}\sum_{k=1}^{m} \mathbb{1}\{\boldsymbol{y}_n^k \in \mathbb{C}\}$. Let $\epsilon > 0$ be arbitrary. For the first term, Chebyshev's inequality gives

$$\begin{aligned}P(|p_n^m - p_n| > \epsilon) &= EP\left(\left|\frac{1}{m}\sum_{k=1}^{m}\mathbb{1}\{\boldsymbol{y}_n^k \in \mathbb{C}\} - P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)\right| > \epsilon \middle| \mathscr{A}_n\right) \\ &\leq E\left(\frac{Var(\mathbb{1}\{\boldsymbol{y}_n^k \in \mathbb{C}\}|\mathscr{A}_n)}{m\epsilon^2}\right) \leq \frac{1}{4m\epsilon^2} \to 0\end{aligned}$$

as $n \to \infty$ since $m = m(n) \to \infty$ as $n \to \infty$. This shows that $p_n^m - p_n \to 0$ in probability as $n \to \infty$. Also Theorem 1 shows that $|p_n - \mathbb{1}\{\boldsymbol{f} \in \mathbb{C}\}| \to 0$ in probability as $n \to \infty$. □

2.7 Variance Reduction Methods

In this section, we improve the vanilla Monte Carlo method through three variance-reduction methods. The change of measure and acceptance-rejection methods are likelihood-ratio-based methods that reuse samples generated in an earlier iteration, and the conditional Monte Carlo method reduces the variance through smoothing.

2.7.1 Change of Measure

Algorithm 0 can be computationally costly due to the need to solve up to mr linear feasibility problems LS(i), where m is the number of Monte Carlo samples and r is the number of design points. We can reduce the computational effort by reusing samples generated in a previous iteration through a change-of-measure method. The resulting estimator is based on the same principle used in the score-function method for simulation optimization [64], and that used in "green simulation" [18]. We will see that the resulting estimator is unbiased and has finite variance, but does not perform as well as we might hope.

Recall that in iteration n, Algorithm 0 generates m i.i.d. samples $\{\boldsymbol{y}_n^k : k = 1, 2, \ldots, m\}$ from the posterior marginal distribution of $\boldsymbol{f}|\mathscr{A}_n$ and produces m indicators $\{\mathbb{1}\left\{\boldsymbol{y}_n^k \in \mathbb{C}\right\} : k = 1, 2, \ldots, m\}$ of convexity. To reuse these samples, in iteration $n + \ell$, we instead output

$$\hat{p}_{n+\ell} = \frac{1}{m} \sum_{k=1}^{m} \mathbb{1}\left\{\boldsymbol{y}_n^k \in \mathbb{C}\right\} L_{n+\ell,n}(\boldsymbol{y}_n^k) \tag{2.16}$$

as an estimate of $p_{n+\ell} = P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_{n+\ell})$, where $L_{n+\ell,n}(\cdot) = \phi_{n+\ell}(\cdot)/\phi_n(\cdot)$ is the likelihood ratio of the densities of $\boldsymbol{f}|\mathscr{A}_{n+\ell}$ and $\boldsymbol{f}|\mathscr{A}_n$.

Theorem 3. *The change of measure estimator $\hat{p}_{n+\ell} = \mathbb{1}\{\boldsymbol{Y}_n \in \mathbb{C}\}L_{n+\ell,n}(\boldsymbol{Y}_n)$ is (conditionally) unbiased and has finite conditional variance, conditional on $\mathscr{A}_{n+\ell}$ for any n and $\ell \geq 1$.*

Proof. The proof for the known Γ case can be found in [42], and we provide a proof for the unknown Γ case here.

First, similar to the known variance case, the distribution of $\boldsymbol{f}|\mathscr{A}_n$, being multivariate t, is absolutely continuous with respect to the distribution of $\boldsymbol{f}|\mathscr{A}_{n+\ell}$. Thus $E(\hat{p}_{n+\ell}|\mathscr{A}_{n+\ell}) = p_{n+\ell}$, and $\hat{p}_{n+\ell}$ is unbiased. It remains to show that $E(\hat{p}_{n+\ell}^2|\mathscr{A}_{n+\ell}) < \infty$, so that conditional on $\mathscr{A}_{n+\ell}$ the estimator has finite second moment and hence variance.

When Γ is unknown, $\boldsymbol{f}|\mathscr{A}_n$ is distributed as multivariate t centered at μ_n with scale matrix $\Xi_n/[\kappa_n(\nu_n - r + 1)]$ and $\nu_n - r + 1$ degrees of freedom. Since $E(\hat{p}_{n+\ell}^2|\mathscr{A}_{n+\ell}) = E(\mathbb{1}\{\boldsymbol{Y} \in \mathbb{C}\}L_{n+\ell,n}^2(\boldsymbol{Y})|\mathscr{A}_{n+\ell})$, it suffices to show that $L_{n+\ell,n}^2(\boldsymbol{y})$ is bounded in $\boldsymbol{y}$ for any n. We will do this by showing that the ratio of an upper bound on the numerator to a lower bound of the denominator is bounded. Moreover, it suffices to obtain such a bound outside a compact set, since t densities are bounded above, and are bounded below on any compact set. Hence, we need only show a bound outside a compact set C that will be successively defined as we proceed.

Let $\lambda_{\max}$ be the largest eigenvalue of $\Lambda_{n+\ell} = \Xi_{n+\ell}/\kappa_{n+\ell}$. Then the squared density of $\boldsymbol{f}|\mathscr{A}_{n+\ell}$ evaluated at $\boldsymbol{y}$ is proportional to

$$\begin{aligned}
&\{1 + (\boldsymbol{y} - \mu_{n+\ell})\Lambda_{n+\ell}^{-1}(\boldsymbol{y} - \mu_{n+\ell})/(\nu_{n+\ell} - r + 1)\}^{-(\nu_{n+\ell}+1)} &(2.17)\\
&\leq \{1 + (\lambda_{\max})^{-1}||\boldsymbol{y} - \mu_{n+\ell}||^2/(\nu_{n+\ell} - r + 1)\}^{-(\nu_{n+\ell}+1)}\\
&\leq \left\{\frac{||\boldsymbol{y} - \mu_{n+\ell}||^2}{\lambda_{\max}(\nu_{n+\ell} - r + 1)}\right\}^{-(\nu_{n+\ell}+1)}\\
&= c_1||\boldsymbol{y} - \mu_{n+\ell}||^{-2(\nu_{n+\ell}+1)}, &(2.18)
\end{aligned}$$

where the constant $c_1 = \{\lambda_{\max}^{+}(\nu_{n+\ell} - r + 1)\}^{(\nu_{n+\ell}+1)}$. This constant depends on n but does not depend on $\boldsymbol{y}$.

Likewise, let $\lambda_{\min}$ be the smallest eigenvalue of $\Lambda_n = \Xi_n/\kappa_n$. Then the squared density of $\boldsymbol{f}|\mathscr{A}_n$ evaluated at $\boldsymbol{y}$ is proportional to

$$\begin{aligned}
&\{1 + (\boldsymbol{y} - \mu_n)\Lambda_n^{-1}(\boldsymbol{y} - \mu_n)/(\nu_n - r + 1)\}^{-(\nu_n+1)}\\
&\geq \{1 + \lambda_{\min}^{-1}||\boldsymbol{y} - \mu_n||^2/(\nu_n - r + 1)\}^{-(\nu_n+1)}\\
&\geq \left\{\frac{2||\boldsymbol{y} - \mu_n||^2}{\lambda_{\min}(\nu_n - r + 1)}\right\}^{-(\nu_n+1)} &(2.19)\\
&= ||\boldsymbol{y} - \mu_n||^{-2(\nu_n+1)}, &(2.20)
\end{aligned}$$

where the constant $c_2 = \{\lambda_{\min}(\nu_n - r + 1)/2\}^{(\nu_n+1)}$. The bound (2.19) applies for y outside the compact set $C = \{\boldsymbol{y} : \|\boldsymbol{y} - \mu_n\|^2 \leq \lambda_{\min}(\nu_n - r + 1)\}$.

Taking the ratio of (2.18) and (2.20), we have that for $\boldsymbol{y} \notin \mathbb{C}$,

$$\begin{aligned}
L^2_{n+\ell,n}(\boldsymbol{y}) &\leq \frac{c_1||\boldsymbol{y} - \mu_{n+\ell}||^{-2(\nu_{n+\ell}+1)}}{c_2||\boldsymbol{y} - \mu_n||^{-2(\nu_n+1)}} \\
&= \frac{c_1}{c_2} \cdot \frac{||\boldsymbol{y} - \mu_n||^{2(\nu_n+1)}}{||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_{n+\ell}+1)}} \\
&= \frac{c_1}{c_2} \cdot \frac{||\boldsymbol{y} - \mu_{n+\ell} + \mu_{n+\ell} - \mu_n||^{2(\nu_n+1)}}{||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_{n+\ell}+1)}} \\
&\leq \frac{c_1}{c_2} \cdot \frac{(||\boldsymbol{y} - \mu_{n+\ell}|| + ||\mu_{n+\ell} - \mu_n||)^{2(\nu_n+1)}}{||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_{n+\ell}+1)}} \\
&\leq \frac{c_1}{c_2} \cdot 2^{2\nu_n+1} \frac{||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_n+1)} + ||\mu_{n+\ell} - \mu_n||^{2(\nu_n+1)}}{||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_{n+\ell}+1)}} \\
&\leq \frac{c_1}{c_2} \cdot 2^{2\nu_n+1} \left\{ ||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_n - \nu_{n+\ell})} + \frac{||\mu_{n+\ell} - \mu_n||^{2(\nu_n+1)}}{||\boldsymbol{y} - \mu_{n+\ell}||^{2(\nu_{n+\ell}+1)}} \right\}. \qquad (2.21)
\end{aligned}$$

The second to last step uses Jensen's inequality $f((a+b)/2) \leq (f(a)+f(b))/2$ on the convex function $f(x) = x^{2(\nu_n+1)}$ where $x > 0$ and $\nu_n > 0$.

Finally, for the bracketed term in (2.21), we enlarge the "exclusion compact set" C if necessary so that $||\boldsymbol{y} - \mu_{n+\ell}||^2 \geq 1$ for $y \notin C$ and then the first term is at most 1 since $\nu_n < \nu_{n+\ell}$, and the second term is at most $||\mu_{n+\ell} - \mu_n||^{2(\nu_n+1)}$. Thus the term is bounded above for any fixed n. $\square$

Given that the change of measure estimator is unbiased and has finite variance, it is tempting to generate a single sample and re-use it for many iterations to save computational effort. Unfortunately, such an estimator has poor empirical performance. Figure 2.3 gives an example where the estimated probability of convexity is greater than 1. This happens especially later in the run when all of the linear systems are feasible, and the likelihood ratios $L_{n+\ell,n}(\boldsymbol{y}^n_k)$ occasionally take very large values.

Occasional large values of the likelihood ratio $L_{n+\ell,n}(\boldsymbol{y})$ might arise when the sample $\boldsymbol{y}$ is generated within the tail of ϕ_n. Indeed, Proposition 1 below shows that in at least one special case, $L_{n+\ell,n}$ has a heavy tail given any sampling trajectory $\mathscr{A}_n$. At first sight, this may appear to contradict Theorem 3, which states that given the posterior information $\mathscr{A}_{n+\ell}$ in iteration $n+\ell$, the change of measure estimator is bounded. But notice that in Theorem 3 we are conditioning on more information than in Proposition 1. In effect, Proposition 1 shows that given the posterior information $\mathscr{A}_n$ in iteration n, the change of measure procedure in iteration $n+\ell$ could have poor behavior, depending on the (random) samples that are used to update $\boldsymbol{f}|\mathscr{A}_{n+\ell}$ from $\boldsymbol{f}|\mathscr{A}_n$. Thus there is no contradiction between these two results.

Proposition 1. *When Γ is known and $r = 1$, given $\mathscr{A}_n$, $C_n = \sup_{\boldsymbol{y}\in\mathbb{R}^r} L_{n+1,n}(\boldsymbol{y})$ asymptotically (as $n \to \infty$) has the same distribution as $e^{\chi_1^2}$, where χ_1^2 is a non-central chi-square random variable with 1 degree of freedom.*

Proof. When $r = 1$, the density of $f|\mathscr{A}_n$ is $\phi_n \sim N(\mu_n, \sigma_n^2)$. Denote the true sampling variance as γ^2, and the precision $\gamma^{-2} = \lambda$. According to equation (2.1), we have

$$\begin{aligned}
\sigma_{n+1}^{-2} &= \sigma_n^{-2} + \lambda = \sigma_0^{-2} + (n+1)\lambda, \\
\sigma_n^{-2} &= \sigma_0^{-2} + n\lambda, \\
\mu_{n+1} &= \sigma_{n+1}^{2}(\sigma_n^{-2}\mu_n + \lambda\bar{z}),
\end{aligned}$$

where $\bar{z} \sim N(\mu, \gamma^2)$ for some constant μ. Then for any $y \in \mathbb{R}$, $\ln L_{n+1,n}(y)$ is

$$\begin{aligned}
&\ln\left(\frac{\phi_{n+1}(y)}{\phi_n(y)}\right)\\
&= \ln\left(\frac{\sigma_n}{\sigma_{n+1}}\right) - \frac{(y-\mu_{n+1})^2}{2\sigma_{n+1}^2} + \frac{(y-\mu_n)^2}{2\sigma_n^2}\\
&= \frac{1}{2}\ln\left(\frac{\sigma_0^{-2}+(n+1)\lambda}{\sigma_0^{-2}+n\lambda}\right) - \frac{1}{2}(y-\mu_{n+1})^2(\sigma_0^{-2}+(n+1)\lambda) + \frac{1}{2}(y-\mu_n)^2(\sigma_0^{-2}+n\lambda)\\
&= q_n + \frac{1}{2}\left[(y-\mu_n)^2(\sigma_0^{-2}+n\lambda) - (y-\mu_n+\mu_n-\mu_{n+1})^2(\sigma_0^{-2}+(n+1)\lambda)\right]\\
&= q_n + \frac{1}{2}\left[-\lambda(y-\mu_n)^2 - 2(\mu_n-\mu_{n+1})(\sigma_0^{-2}+(n+1)\lambda)(y-\mu_n)\right.\\
&\left.+ (\mu_n-\mu_{n+1})^2(\sigma_0^{-2}+(n+1)\lambda)\right]\\
&= q_n - \frac{\lambda}{2}\left\{y-\mu_n+\frac{1}{\lambda}\left[(\mu_n-\mu_{n+1})(\sigma_0^{-2}+(n+1)\lambda)\right]\right\}^2\\
&\qquad - \frac{1}{2}(\mu_n-\mu_{n+1})^2\left[(\sigma_0^{-2}+(n+1)\lambda) - \frac{1}{\lambda}(\sigma_0^{-2}+(n+1)\lambda)^2\right],
\end{aligned}$$

where $q_n = \frac{1}{2}\ln\left(\frac{\sigma_0^{-2}+(n+1)\lambda}{\sigma_0^{-2}+n\lambda}\right)$ is a constant. Maximizing over y gives

$$\begin{aligned}
\ln C_n &= \sup_y \ln\left(\frac{\phi_{n+1}(y)}{\phi_n(y)}\right)\\
&= q_n - \frac{1}{2}(\mu_n-\mu_{n+1})^2\left[(\sigma_0^{-2}+(n+1)\lambda) - \frac{1}{\lambda}(\sigma_0^{-2}+(n+1)\lambda)^2\right]\\
&= q_n - \frac{1}{2}(\mu_n-\mu_{n+1})^2\left[\sigma_{n+1}^{-2} - \frac{1}{\lambda}(\sigma_{n+1}^{-2})^2\right].
\end{aligned}$$

Here,

$$\begin{aligned}
\mu_n - \mu_{n+1} &= \mu_n - \sigma_{n+1}^2(\sigma_n^{-2}\mu_n + \lambda\bar{z})\\
&= (1 - \frac{\sigma_{n+1}^2}{\sigma_n^2})\mu_n - \sigma_{n+1}^2\lambda\bar{z}\\
&= \mu_n\left(1 - \frac{\sigma_0^{-2}+n\lambda}{\sigma_0^{-2}+(n+1)\lambda}\right) - \sigma_{n+1}^2\lambda\bar{z}\\
&= \mu_n\frac{\lambda}{\sigma_0^{-2}+(n+1)\lambda} - \sigma_{n+1}^2\lambda\bar{z}\\
&= \mu_n\frac{\lambda}{\sigma_{n+1}^{-2}} - \sigma_{n+1}^2\lambda\bar{z}\\
&= \lambda\sigma_{n+1}^2(\mu_n - \bar{z}),
\end{aligned}$$

where $\bar{z}$ is the average over the samples used to update the posterior. We have been using just one sample for the update, so $\bar{z} \sim N(\mu, \sigma^2)$, where σ^2 is the sampling variance.

Substituting, we obtain

$$\begin{aligned}
\ln C_n &= q_n - \frac{1}{2}(\mu_n - \mu_{n+1})^2 \left[\sigma_{n+1}^{-2} - \frac{1}{\lambda}(\sigma_{n+1}^{-2})^2\right] \\
&= q_n - \frac{1}{2}\lambda^2\sigma_{n+1}^4(\mu_n - \bar{z})^2 \left[\sigma_{n+1}^{-2} - \frac{1}{\lambda}(\sigma_{n+1}^{-2})^2\right] \\
&= q_n - \frac{1}{2}\lambda^2(\mu_n - \bar{z})^2 \left[\sigma_{n+1}^{2} - \frac{1}{\lambda}\right] \\
&= q_n - \frac{1}{2}\lambda^2(\mu_n - \bar{z})^2 \left[\frac{1}{\sigma_n^{-2} + \lambda} - \frac{1}{\lambda}\right].
\end{aligned}$$

Here $\mu_n - \bar{z} = \mu_n - (\mu - \sigma N) = \sigma N + \mu_n - \mu$, where N is a standard normal random variable, so $(\mu_n - \bar{z})^2 = \sigma^2(N + (\mu_n - \mu)/\sigma)^2$. Thus conditional on $\mathscr{A}_n$,

$$\begin{aligned}
\ln C_n &= q_n - \frac{1}{2}\lambda^2\sigma^2(N + (\mu_n - \mu)/\sigma)^2 \left[\frac{1}{\sigma_n^{-2} + \lambda} - \frac{1}{\lambda}\right] \\
&\sim \frac{1}{2}\left(N + \frac{\mu_n - \mu}{\sigma}\right)^2,
\end{aligned}$$

since $\lambda = \sigma^{-2}$ and $q_n \sim 0$ as $n \to \infty$. Here N is a standard normal random variable, so $\ln C_n$ is distributed as a non-central chi-square with 1 degree of freedom and non-central parameter $(\mu_n - \mu)^2/\sigma^2$; see, e.g., [27, p. 123]. $\square$

In fact, the proof for Proposition 1 applies to when n is finite too. In that case $\ln C_n$, conditional on $\mathscr{A}_n$, is a non-central chi-square random variable scaled by a constant of order $O(1/n)$ and shifted by another constant of order $O(1/n)$. The conclusion of Proposition 1 can be generalized to $r > 1$ when Γ is known and diagonal. Indeed, when Γ is diagonal the likelihood ratio decomposes into a product, so that $\ln C_n = \ln(\sup_{\boldsymbol{y}\in\mathbb{R}^r} L_{n+1,n}(\boldsymbol{y})) = \sup_{\boldsymbol{y}\in\mathbb{R}^r} \ln(\prod_{i=1}^r L_{n+1,n}(\boldsymbol{y}_i)) = \sum_{i=1}^r \sup_{\boldsymbol{y}_i\in\mathbb{R}} \ln L_{n+1,n}(\boldsymbol{y}_i)$. Proposition 1 then allows us to conclude that, conditional on $\mathscr{A}_n$, this is asymptotically conditionally distributed as $\chi_r^2/2$, where χ_r^2

is a non-central chi-square random variable with r degress of freedom. Since the tail probability of $\chi_r^2/2$ at a given point increases in r, we expect this heavy tail behavior to be more significant as r increases, i.e., as the number of design points increases. We conjecture that the likelihood ratio is similarly heavy-tailed in the cases where Γ is known but not necessarily diagonal and when Γ is unknown.

In summary, conditional on $\mathscr{A}_{n+\ell}$, the estimator $\hat{p}_{n+\ell}$ is unbiased and has finite variance, but its distribution may be heavy tailed given $\mathscr{A}_n$ only, depending on the samples obtained to update to $\boldsymbol{f}|\mathscr{A}_{n+\ell}$. Thus this estimator needs to be used with caution. We suggest that if the method is to be used, then one should do so with small ℓ, e.g., $\ell < 5$, based on simulation experiments described later.

2.7.2 Acceptance/Rejection

The change of measure estimator reuses all the samples obtained in an earlier iteration by outputting a Monte Carlo estimator that scales each indicator $\{I_n^k = \mathbb{1}\{\boldsymbol{Y}_n^k \in \mathbb{C}\} : k = 1, 2, \ldots, m\}$ by a likelihood ratio $L_{n+\ell,n}(\boldsymbol{Y}_n^k) = \phi_{n+1}(\boldsymbol{Y}_n^k)/\phi_n(\boldsymbol{Y}_n^k)$, where ϕ_n is the posterior density. An alternative is to reuse a subset of the samples from the previous iteration through acceptance-rejection.

Suppose that in iteration n, we have m i.i.d. Monte Carlo samples $\{\boldsymbol{y}_n^k : k = 1, 2, \ldots, m\}$ from $\boldsymbol{f}|\mathscr{A}_n$, together with the indicators $\{I_n^k = \mathbb{1}\{\boldsymbol{y}_n^k \in \mathbb{C}\} : k = 1, 2, \ldots, m\}$. Then, at iteration $n+1$, the k-th sample $\boldsymbol{y}_k^n$ will be accepted (reused) with probability $L_{n+1,n}(\boldsymbol{y}_k^n)/c$, where $c \geq \sup\{L_{n+1,n}(\boldsymbol{y}) : \boldsymbol{y} \in \mathbb{R}^r\}$. If the accepted indices are $A \subseteq \{1, 2, \ldots, m\}$, then $m - |A|$ additional samples can be generated from $\boldsymbol{f}|\mathscr{A}_{n+1}$ to ensure a total of m samples. The estimator is then just the usual Monte Carlo estimator based on all m samples, i.e.,

$\hat{p}_{n+1} = \left(\sum_{k\in A} \mathbb{1}\left\{\boldsymbol{y}_k^n \in \mathbb{C}\right\} + \sum_{k=1}^{m-|A|} \mathbb{1}\left\{\boldsymbol{y}_k^{n+1} \in \mathbb{C}\right\}\right)/m.$

When the sampling variance Γ is known, optimization shows that c is given by

$$\left\{\frac{|\Lambda_n|}{|\Lambda_{n+1}|} \exp\left[(\Lambda_{n+1}^{-1}\mu_{n+1} - \Lambda_n^{-1}\mu_n)^T\Gamma(\Lambda_{n+1}^{-1}\mu_{n+1} - \Lambda_n^{-1}\mu_n) + \mu_n^T\Lambda_n^{-1}\mu_n - \mu_{n+1}^T\Lambda_{n+1}^{-1}\mu_{n+1}\right]\right\}^{1/2},$$

where the parameters $\mu_n, \Lambda_n, \mu_{n+1}, \Lambda_{n+1}$ are defined as in Section 2.3. When Γ is unknown, c is the maximum of a ratio of polynomials and does not have a closed form, so we calculate it numerically.

The acceptance-rejection estimator is simply an average of i.i.d. samples, like the pure Monte Carlo estimator. The difference lies in how the samples are obtained. The probability of accepting a sample generated in iteration n is $1/c$, so the efficiency of this method is related to the constant c. According to Proposition 1, the likelihood $L_{n+1,n}(y)$ can take very large values, meaning that c can often be large. When c is large, very few of the earlier samples might be reused, so the majority of the m samples needed in the $(n+1)$th iteration are new. This may lower the efficiency of the acceptance-rejection method.

2.7.3 Conditional Monte Carlo

We can view a sample from the posterior distribution as consisting of both a direction Z on the $(r-1)$-sphere S^{r-1} and a step size T along that direction. We condition on the direction Z, and integrate the posterior over the interval of step sizes $[t_{\min}, t_{\max}]$ that yield points inside the convexity cone $\mathbb{C}$. Averaging the results over a number of uniformly generated directions gives an estimate of our desired probability.

For convenience, let $E_n(\cdot) = E(\cdot|\mathscr{A}_n)$ and $P_n(\cdot) = P(\cdot|\mathscr{A}_n)$. If $X \sim N(0, I)$

(known variance) or $X \sim t_{\nu_n}(0, I)$ (unknown variance), then

$$\begin{aligned} P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n) &= E_n\left(\mathbb{1}\left\{\Lambda_n^{1/2}X + \mu_n \in \mathbb{C}\right\}\right) \\ &= E_n\left(\mathbb{1}\left\{T\Lambda_n^{1/2}Z + \mu_n \in \mathbb{C}\right\}\right), \text{ for } Z \text{ uniform on } S^{r-1} \\ &= E_n\left(E_n\left(\mathbb{1}\left\{T\Lambda_n^{1/2}Z + \mu_n \in \mathbb{C}\right\}|Z\right)\right) \\ &= E_n(P_n(T \in [t_{\min}(Z), t_{\max}(Z)]\,|Z)) \\ &= E_n(F_{T|Z}(t_{\max}(Z)) - F_{T|Z}(t_{\min}(Z))). \end{aligned}$$

Here $F_{T|Z}$ is the conditional distribution function of T given $\mathscr{A}_n$ and Z. Thus, the posterior probability $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$ can be estimated using $F_{T|Z}$ and a way to calculate $t_{\max}(Z)$ and $t_{\min}(Z)$. Theorem 4 gives the former, and linear programs LS(i) (below) give the latter.

Theorem 4 (Distribution of $T|Z$)**.** *When the sampling variance* Γ *is known,* $F_{T|Z}(t) = (1 + \mathit{sign}(t)F_{\chi^2_r}(t^2))/2$*, where* $F_{\chi^2_r}(\cdot)$ *is the (cumulative) distribution function of a* χ^2 *r.v. with* r *degrees of freedom. When* Γ *is unknown,* $F_{T|Z}(t) = (1 + \mathit{sign}(t)F_{F(r,\nu_n)}(t^2/r))/2$*, where* $F_{F(r,\nu_n)}$ *is the distribution function of the F distribution with* r *and* ν_n *degrees of freedom.*

Proof.

$$\begin{aligned} &F_{T|Z}(t) = P_n(T \le t|Z) \\ &= \begin{cases} P_n(T \le 0|Z) + P_n(0 \le T \le t|Z), & \text{when } t \ge 0 \\ P_n(T \le 0|Z) - P_n(0 \le T \le -t|Z), & \text{when } t < 0 \end{cases} \\ &=1/2 + \text{sign}(t)P(||X||^2 \le t^2|Z)/2, \text{ for } X = TZ \end{aligned}$$

When Γ is known, $X \sim N(0, I)$, so $||X||^2 \sim \chi^2_r$. When Γ is unknown, $X \sim t_{\nu_n}(0, I) = N/\sqrt{Y/\nu_n}$ for independent $N \sim N(0, I)$ and $Y \sim \chi^2_{\nu_n}$. Therefore

$$||X||^2 = \frac{N^T N}{Y/\nu_n}$$

where $N^T N \sim \chi^2_r$, so $||X||^2/r \sim F(r, \nu_n)$. $\square$

To find $t_{\min}(Z)$ and $t_{\max}(Z)$, we can solve linear programs with objectives minimizing or maximizing t, with decision variables $t \in \mathbb{R}, \boldsymbol{a} \in \mathbb{R}^{r\times d}, \boldsymbol{b} \in \mathbb{R}^d$, and the constraints (LS), replacing $g(\boldsymbol{x})$ by $\mu + (\Lambda^{1/2}Z)t$:

$$\begin{aligned} & t_{\min} = \min\ t \quad (t_{\max} = \max\ t) \\ \text{s.t.}\quad & \boldsymbol{a}^T\boldsymbol{x} + \boldsymbol{b} = \mu + (\Lambda^{1/2}Z)t \\ & \boldsymbol{a}_i^T\boldsymbol{x}_j + \boldsymbol{b}_i \le \mu_j + (\Lambda^{1/2}Z)_j t, \ \text{ for all } i \in \{1,\dots,r\} \text{ and } j \ne i,\, j \in \{1,\dots,r\} \end{aligned} \tag{LP}$$

The linear program LP can be decomposed into r smaller LP's, with constraints LS(i) and variables $t \in \mathbb{R}, \boldsymbol{a}_i \in \mathbb{R}^r, b_i \in \mathbb{R}$:

$$\begin{aligned} & t_{\min}(i) = \min\ t \quad (t_{\max}(i) = \max\ t) \\ \text{s.t.}\quad & \boldsymbol{a}_i^T\boldsymbol{x}_i + b_i = \mu + (\Lambda^{1/2}Z)t \\ & \boldsymbol{a}_i^T\boldsymbol{x}_j + b_i \le \mu_j + (\Lambda^{1/2}Z)_j t, \ \text{ for all } j \ne i,\, j \in \{1,\dots,r\}, \end{aligned} \tag{LP(i)}$$

and then

$$t_{\min} = \max_{i=1,2,\dots,r} t_{\min}(i), \quad \text{and} \quad t_{\max} = \min_{i=1,2,\dots,r} t_{\max}(i).$$

This decomposition does not bring as much speed improvement as LS(i) does, because all the decomposed linear programs must be solved.

Now we have all the pieces needed for the conditional Monte Carlo method.

Algorithm 3 A conditional Monte Carlo estimator $\widetilde{p}_n$ for $p_n = P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$.

```
Input: Posterior distribution of $\boldsymbol{f}|\mathscr{A}_n$ obtained from Algorithm 1 with mean $\mu_n$
    and covariance $\Lambda_n$; Number of Monte Carlo samples $m$ needed
1: for $k = 1, \dots, m$ do
2:     Uniformly generate a vector $z_k$ on the surface of a unit sphere (by generating
       a standard Gaussian and normalizing it to a unit vector).
3:     Determine integration boundaries $t_{\min}(z_k)$ and $t_{\max}(z_k)$.
4:     Set $\widetilde{P}_n(k) = F_{T|Z}(t_{\max}(z_k)) - F_{T|Z}(t_{\min}(z_k))$.
5: Calculate the mean $p_n^m$ and standard deviation $s_n^m$ of $(\widetilde{P}_n(k) : k = 1, 2, \dots, m)$.
   return $\widetilde{p}_n = p_n^m$ as an estimator of $P(\boldsymbol{f} \in \mathbb{C}|\mathscr{A}_n)$, along with the half-width
   $\tilde{h}_n = 1.96 s_n^m / \sqrt{m}$ of a 95% confidence interval.
```

Relative to the other variance-reduction methods, conditional Monte Carlo takes much longer to produce an estimate in each iteration because it needs to solve two linear programs LP and cannot "skip" any of them as can be done when solving the decomposed feasibility problems LS(i).

2.8 Numerical Results

In this section we show numerical results on some test functions, assuming the more realistic case that the sampling variance is unknown. Since we can freely choose the r points to sample from (and evaluate them by running the simulation with the regarding inputs), we wish to locate the r points such that the method is able to detect non-convexity efficiently. Therefore, for a test function in d dimensions, we first sample $d + 1$ points uniformly at random within the (assumed compact) sample space S, and for each such random point, we generate a uniform random direction on the surface of the unit sphere. Each point-direction pair defines a line segment within S. Then we sample 3 points uniformly at random on each line segment. In such way, each group of 3 points on the same line are able to determine non-convexity independent of all the other samples. This method generates $r = 3(d+1)$ sample points. We select the points to lie on $d+1$ line segments because doing so seems to improve the performance of the convexity test relative to just sampling points uniformly within S. In each iteration of the sequential algorithm, we use $m = 100$ Monte Carlo samples from the posterior predictive distribution to estimate a 95% confidence interval for p_n.

Our procedure is implemented in Matlab and freely available in an online repository [40]. The repository contains two versions. The first version uses only a standard Matlab installation, solving linear programs using the built-in `linprog` function [51]. The second version requires the installation of the packages `CVX` [32, 31], which is a package for specifying and solving convex programs, and `Gurobi` [33], a commercial optimization solver. We suggest the second version if a user has the requisite licenses, since `Gurobi` seems more robust than `linprog`. For example, we have found cases where `linprog` was not able to find a feasible solution, whereas `Gurobi` did. However, because of the overhead of `CVX` in setting up the linear program in a format that `Gurobi` is able to read, `linprog` is usually faster when the problem dimension is low. When the problem dimension is high, the inefficiency of the interior-point-method used by `linprog` outweighs the overhead of `CVX`. Figure 2.1 compares solving times by these two solvers for the linear programs (LP), tested with the sample function $f(\boldsymbol{x}) = ||\boldsymbol{x}||^2, \boldsymbol{x} \in [-1,1]^d$ for different values of d.

All test cases are run on a desktop with a 4-core Intel Core i7-3770 3.40 GHz processor with 16G memory, running Matlab R2013a on 64-bit Windows 7.

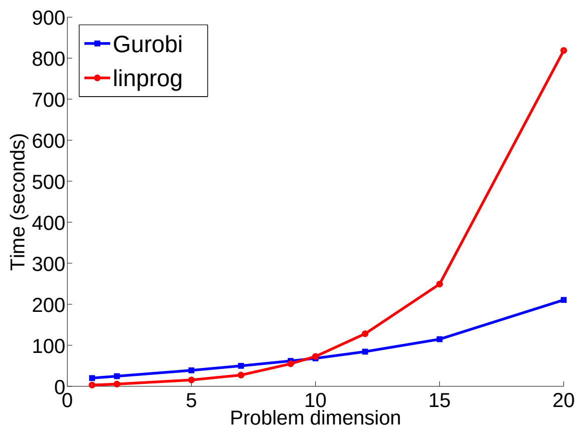


Figure 2.1: The solving time vs. testing function dimension for the two linear programs in the conditional Monte Carlo method using `Gurobi` and `linprog`. `Gurobi` is faster when the dimension exceeds 10, when 33 sample points are used.

2.8.1 A Strictly Convex Function

We use $f(\boldsymbol{x}) = ||\boldsymbol{x}||^2$ in this section as the test function.

First, we compare vanilla Monte Carlo with the variance reduction methods in Section 2.7, showing 95% confidence intervals for the estimated probability of convexity, the time per iteration, and the efficiency per iteration. Here the efficiency of the Monte Carlo estimator $\tilde{p}_n$ is defined as the inverse of the product of the computational time per replication and the variance of one replication; see, e.g., [29]. We first take the dimension $d = 1$, on the sample space $[-1, 1]$. The sampling covariance matrix has equal constant variances of 0.01 on the diagonal, and we use a Gaussian kernel of $10^{-4} \exp\{-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/2\}$ for the off-diagonal components. Hence the noise at different points is positively correlated, and the correlation is stronger between closer points [61].

We use `linprog` instead of `Gurobi` to avoid the time overhead incurred by `CVX`, since the dimension $d = 1$. For the change of measure method, a new set of samples is obtained every iteration for the first 30 iterations, and every 5 iterations thereafter. For the acceptance-rejection method we start to reuse samples only after the first 30 iterations. Thus the first 30 iterations of these two methods are exactly the same as vanilla Monte Carlo.

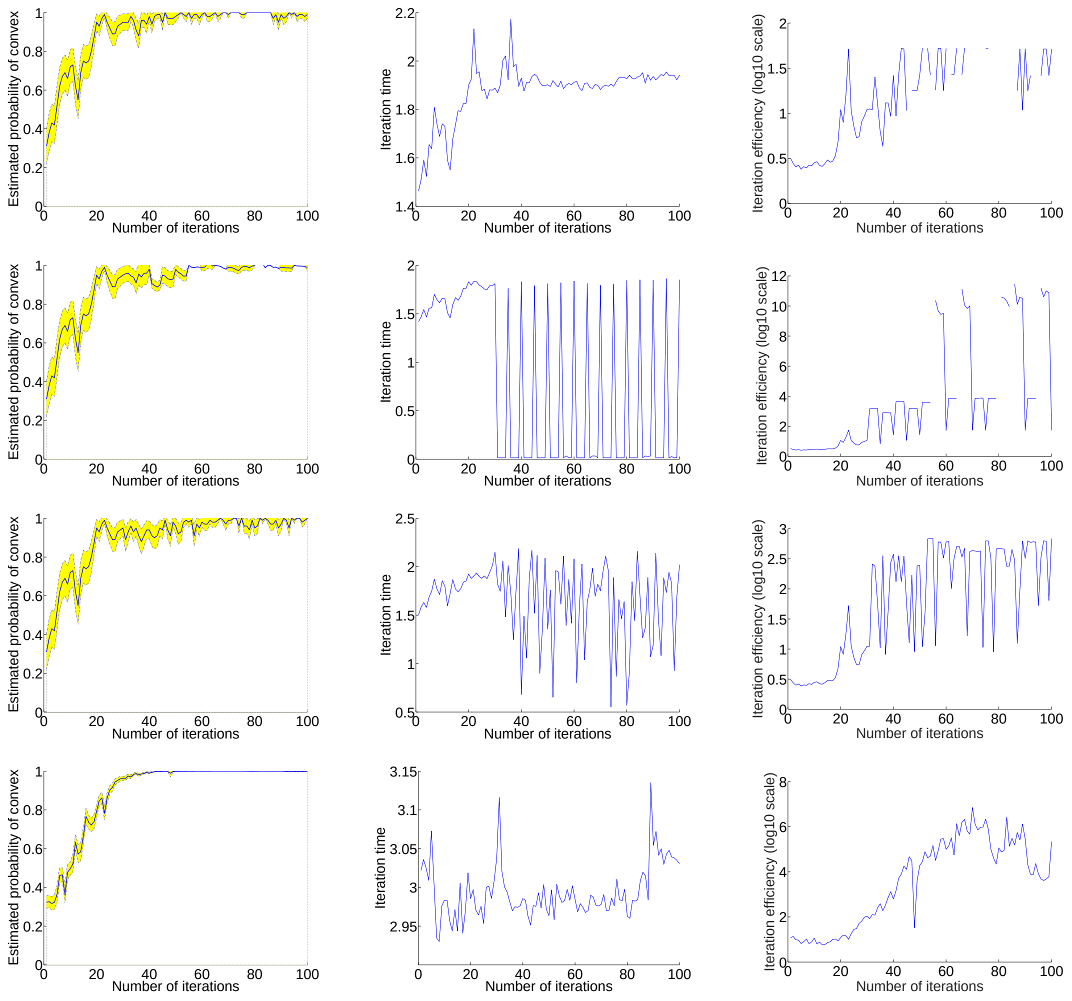


Figure 2.2: Comparisons of the estimated probability of convexity, the iteration time (in seconds), and the log (base 10) efficiency (left to right) of vanilla Monte Carlo, change of measure, acceptance-rejection, and conditional-Monte-Carlo (top to bottom) methods applied to a one-dimensional strictly convex function.

Figure 2.2 shows that the estimated probability of convexity increases to 1 for all methods. Later in the iterations, the change of measure method can return greater-than-one estimates due to the poor behavior of the likelihood ratio as discussed in Section 2.7.1. Among all methods, the conditional Monte Carlo method has the smallest variance but takes the longest time to compute. (This difference in the computational time becomes more significant for higher-dimensional test functions when we later experiment on a 30-dimensional function.) For the 1-dimensional convex function here, taking both computational time and variance into consideration, we observe that conditional Monte Carlo has the highest overall efficiency. The efficiency of vanilla Monte Carlo is the lowest. The efficiency plots occasionally break when the sample variance of the estimator is 0, where all the linear systems (LS) are feasible. This also happens for the change of measure method because that method corresponds with vanilla Monte Carlo every 5 iterations after the first 30. The efficiency of the change of measure method and the acceptance-rejection method both increase whenever they reuse the samples from a previous iteration because the linear feasibility problems need not be solved. The change of measure method occasionally has a very large efficiency because of the small sample variance of the estimate. This happens in later iterations when the posterior density is very concentrated. In this case all the Monte Carlo samples are close to the mean, giving almost identical posterior densities and similar likelihood ratios. When all reused samples are convex (corresponding to the iterations where the vanilla Monte Carlo method has infinite estimated efficiency), the change of measure estimator has almost 0 variance. However, due to the heavy tail behavior of the likelihood ratio, it is risky to trust the change of measure estimator values, as we see when the change of measure method estimates a probability greater than 1. The acceptance-rejection estimator has slightly lower estimated efficiency, but the estimator is more trustworthy in that it is statistically identical to vanilla Monte Carlo.

Consider now the 30-dimensional test function $f(x) = ||x||^2, x \in [-10, 10]^{30}$ with $r = 3(d+1) = 93$ sample points. The covariance matrix Γ has diagonal entries $\Gamma_{ii} = 0.04 f^2(\boldsymbol{x}_i)$, and off-diagonal entries $\Gamma_{ij} = 10^{-2} \exp\{-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/2\} 0.04 f(\boldsymbol{x}_i) f(\boldsymbol{x}_j)$. Hence the variance depends on the function value, and there is also modest positive correlation between any two design points depending on the distance between them. As before, for the change of measure method, a new set of samples is obtained every iteration for the first 30 iterations, and every 5 iterations thereafter, and for the acceptance-rejection method we start to reuse samples only after the first 30 iterations. We find that the change of measure and acceptance-rejection methods do not work very well on this example. Indeed, according to Proposition 1, the heavy-tail behavior of the likelihood ratio becomes more severe with more design points. With the likelihood ratio often taking very large values, the change of measure estimates evaluate to large values with wide confidence intervals, as shown in Figure 2.3 (notice the y-axis scale). Due to the same reason, the acceptance-rejection method reduces to vanilla Monte Carlo by rejecting almost all previous samples, so we omit that method from the results in Figure 2.3. In early iterations, conditional Monte Carlo takes more than 6 minutes to generate an estimate using `CVX` with `Gurobi` (`linprog` takes over 1 hour), and the iteration efficiency is around 0.20. In comparison, the vanilla Monte Carlo method only takes 80 seconds per iteration at the beginning of the iteration by solving the decomposed LS(i), giving around the same level of iteration efficiency. However, towards the end of the 100 iterations, conditional Monte Carlo is able to reduce the variance of the estimated probability so well that the efficiency improves beyond that of vanilla Monte Carlo. Therefore we recommend using conditional Monte Carlo (with `CVX` + `Gurobi`) if one can afford the running time, and vanilla Monte Carlo otherwise or when `CVX` is not installed.

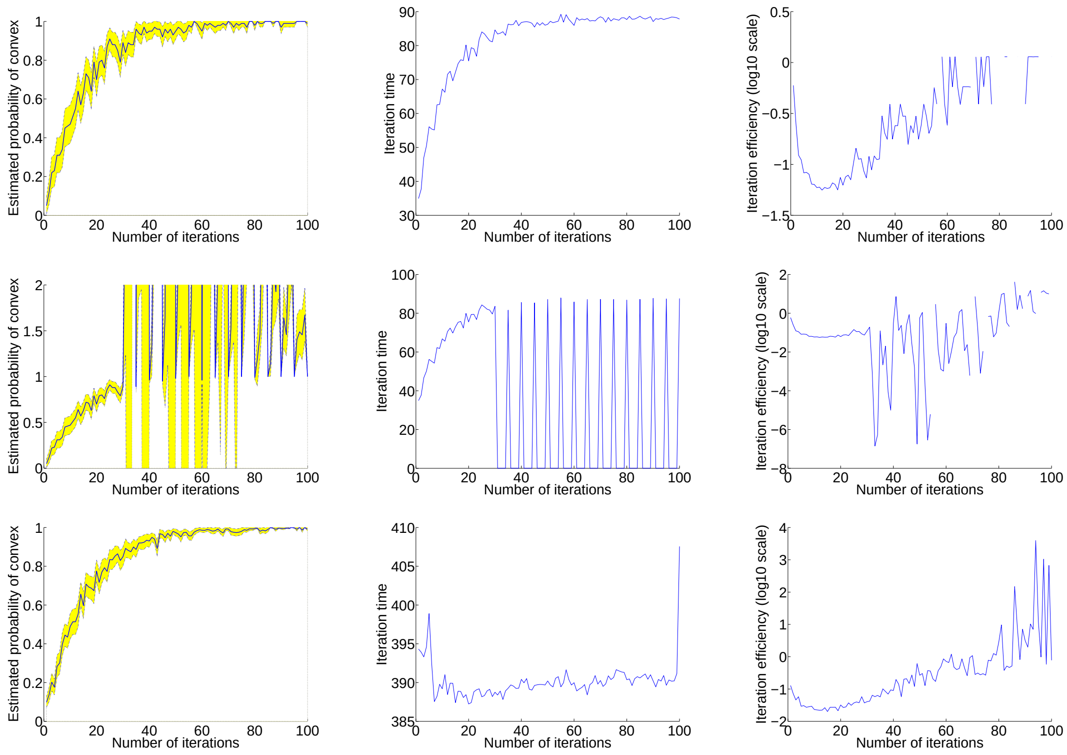


Figure 2.3: The estimated probability of convexity, the iteration time (in seconds), and the log (base 10) efficiency (left to right) of the vanilla Monte Carlo, change of measure, and conditional Monte Carlo methods (top to bottom) applied to a 30-dimensional strictly convex function.

2.8.2 A Non-Convex Function

Consider the function $f(\boldsymbol{x}) = -||\boldsymbol{x}||^2, \boldsymbol{x} \in [-1, 1]^d$. In order to make the problem "harder," we choose the covariance matrix Γ to be $d^2/4$ on the diagonal, so that the sampling standard deviation is bigger than half of the function value, and 0 on the off-diagonal. Figure 2.4 gives the results from the vanilla Monte Carlo estimator, with acceptance-rejection applied after the initial 30 iterations, for varying dimensions. The estimated probabilities of convexity hover near zero over all iterations, especially in lower dimensions. This is perhaps intuitive: with few iterations the noise in the estimated function values dominates, and in the presence of large noise *any* function will appear to be nonconvex, while after many iterations, the nonconvexities of the (true) function dominate and are detected.

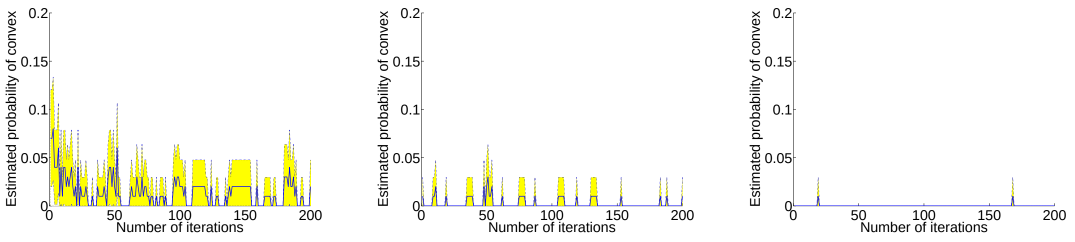


Figure 2.4: The estimated probability of convexity for the simple strictly non-convex function in dimensions 3 and 5 and 10 (from left to right). The estimates when the function is one-dimensional are not shown because they were effectively identically 0.

2.8.3 Linear Function

Linear functions are convex but lie on the boundary of the cone $\mathbb{C}$. Theorem 1 does not inform us of the likely behaviour of our algorithm in this case, because the event that the function lies on the boundary of $\mathbb{C}$ has measure 0 in the context of that result. Thus the posterior probability of convexity could converge to any number between 0 and 1 or not converge at all. Here we use a one-dimensional linear function $f(\boldsymbol{x}) = 0, \boldsymbol{x} \in [-1, 1]$, with a sampling covariance matrix that equals 10^{-4} on the diagonal and 0 on the off-diagonal.

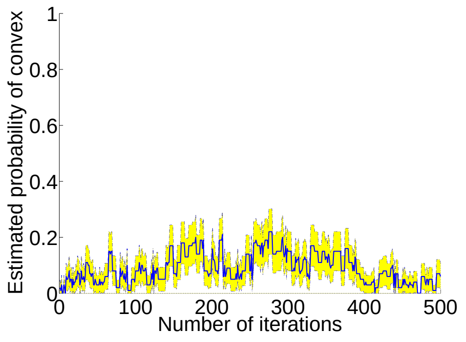


Figure 2.5: The estimated probability of convexity for a 1-dimensional linear function. The mean does not appear to converge.

As shown in Figure 2.5, the estimated probability does not converge to 0 or 1, but stays close to 0. When we increase the dimension, e.g., to 5, and keep the sampling variance the same, the estimated probabilities stay at 0 throughout the first 100 iterations. Changing the sampling variance does not change the qualitative nature of results because when the function is zero-valued the sampling variance only changes the "scale" of the observations. These results are perhaps to be expected because a linear function would only appear convex when the function noise at all design points "happens to" form a strictly convex function.

2.8.4 Output of a Simulation

Finally, we have also tested our algorithm on a more realistic example similar to the "Ambulances in a Square" problem from SimOpt [58]. In this problem, patient calls arrive in a one kilometer unit square $[0, 1]^2$ according to a Poisson process at a constant rate of 1 call every 2 hours. The (x, y) locations of the calls are i.i.d. and distributed with a density proportional to $1.6 - (|x - 0.8| + |y - 0.8|)$. Upon receiving a call, a nearest ambulance is dispatched, traveling to the scene at a constant speed of 60 km/h. Once arriving at the scene, the ambulance spends a Gamma-distributed scene time with mean 45 minutes and standard deviation 15 minutes, then returns to the base at a speed of 40 km/h if no other call is received. We are interested in the mean response time (time from when the call is received until the ambulance arrives at the call location) as a function of the location(s) of the ambulance base(s).

We sampled the base locations of the ambulance along (4 $\times$ the number of bases + 1) random lines in the unit square, with 3 points sampled on each line. Each base has two coordinates, so this is equivalent to $3(2d + 1)$ design points, where d

is the dimension of the sample space. We are using more design points than in our previous test cases because we wanted to try more points (and consequently more computation) on a real case. Similar to our other experiments, we obtain a sample of the mean response time on each set of sampled base locations from running the simulation until 360 calls receive a response (approximately 30 days). The mean response times of the sampled base locations are evaluated using common random numbers, which compares the locations using the exact same random call arrivals and scene times. The convexity of the mean response time as a function of the ambulance base locations is tested with one, two, and three ambulance bases, using the conditional Monte Carlo method. The estimated probabilities vs. iteration are plotted in Figure 2.6.

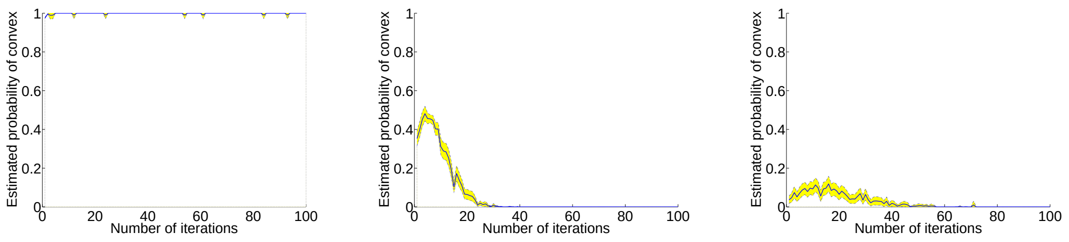


Figure 2.6: The estimated probability of convexity for the mean response time as a function of the base locations, when the number of bases is one, two, and three.

It seems that the mean response time is convex as a function of the base location when there is only one base, while it is not convex for more than one ambulance base. This agrees with our intuition that the location of a single ambulance base should have one global minimizer in the unit square. By plotting the posterior mean function, we found that the minimizer is located near the point $[0.46, 0.54]$, near, but slightly offset from, the mode $[0.8, 0.8]$, to balance the travel time to the farther corner $[0, 0]$. However, when there is more than one ambulance base, the objective does not have a single minimizer due to symmetry and the interactions between bases.

2.9 Conclusion

Given a function that can be observed on a finite number of points in the presence of noise, we have suggested a sequential algorithm to estimate the posterior probability that the function is convex. The method models the function values on a fixed set of design points using a Bayesian conjugate model, and estimates the probability of convexity by Monte Carlo simulation, using samples of the function values from the posterior distribution. This Bayesian procedure gives sequential estimates for function convexity. It is useful when a function is expensive to evaluate, e.g., the output of a large simulation, or when its values can only be obtained on a constrained set of points, e.g., a function defined on a discrete domain.

To improve the efficiency of our algorithm we have introduced three variance reduction methods - change of measure, acceptance-rejection, and conditional Monte Carlo. The first two methods reuse samples obtained in an earlier iteration to calculate an estimator in the current iteration. However, they both rely on the likelihood ratio of normal or Student-t posterior densities, which we prove could take extreme values due to its heavy-tail behavior. In our computational results, we observe that the change of measure method may give poor (e.g., greater than 1) estimates of the probability, and the acceptance-rejection method rejects most of the earlier samples and reduces to vanilla Monte Carlo when the number of design points is large. Finally, the conditional Monte Carlo method takes the longest to compute but is the most effective in variance reduction, giving the highest efficiency among all methods. We recommend using it with `CVX` and `Gurobi`, especially for high-dimensional functions, to ensure reasonable computational time.

A package containing the main algorithm and all variance reduction alternatives is available on Github [40].

CHAPTER 3

USING GRADIENT-LIKE INFORMATION IN OPTIMIZING A LARGE-SCALE BIKE-SHARING SYSTEM

3.1 Introduction

Citi Bike in New York City (NYC) is a large-scale bike-sharing system where a customer can check out a bike from any station and return the bike to any other station. Due to time-varying demand, this freedom of movement of bikes creates unbalanced flows in the system. For example, many customers ride bikes from residential areas to the financial district during the morning rush (6-10am) and reverse their origin and destination in the afternoon. Citi Bike relocates bikes in the system, both overnight (12-6am) and throughout the day. Overnight rebalancing operations can move many more bikes than during the day due to traffic congestion. We are interested in how many docks should be assigned to each station and, based on that dock assignment, how many bikes should be allocated to each station at the beginning of the day.

This optimization problem is an extension of the rebalancing problem in the bike-sharing literature, in that we attempt to rebalance docks in addition to bikes. Insight into the rebalancing problem has been obtained by [21] and [20] using mean-field analysis, showing, e.g., that in a system with perfectly balanced inflows and outflows at each station, that the optimal number of bikes is equal to half the combined station capacities plus the number on trips. Work that is more operational includes [37, 56] and [55] where the CTMC model we refer to later is developed. [62] and [63] develop a solution to bike (only) repositioning. Based on this work, [19] develop a practical recipe for bike repositioning. [65] develop related integer programming models for repositioning bikes (only). [67] use a time-space formulation that is more detailed than our models. [13], and [60] are examples of other repositioning algorithms.

We use discrete-event simulation to model the bike-sharing system, and thus we tackle the rebalancing problem over bikes and docks as a simulation-optimization problem. Ideally, we would apply standard simulation-optimization methods to solve the problem, but as discussed in [44] this seems computationally infeasible. In particular, the optimization problem (3.1) over 466 stations is an integer-ordered optimization problem with 932 decision variables. The scale of this problem renders methods based on random search principles unlikely to be effective. Indeed, since the numbers of bikes and docks are fixed, any local moves likely involve moving bikes and/or docks between randomly selected pairs of stations, and there are over 100,000 such pairs of stations. Moreover, if one attempts to perform a variant of gradient search where the gradients are estimated by finite (integer) differences, then each step of such an algorithm would require simulating 932 "neighboring" configurations (if forward differences are used).

These computational demands seem, to us, to be impractical. We instead develop heuristic search procedures that use statistics from a single simulation run in order to update the allocation of bikes and docks between stations. Our methods build on the heuristic introduced in [44] for allocating bikes to satisfy only the morning rush demand, to allocate both bikes and docks for a full day's operations. We exploit the simulation and use station-specific information from it to group the stations according to their estimated contributions to the objective. These "contributions" can reasonably be viewed as *approximate* gradients, but we make no claim that our approximate gradients are accurate or unbiased, nor do we claim that we find a locally optimal solution. We instead see the value of these algorithms in the improvements they can make in performance relative to that of starting solutions. Since the ideas effectively better any feasible configuration of bikes and docks, they potentially reduce the need for more sophisticated models like the CTMC model.

In obtaining the station-specific information and generating a search direction, we are using the simulation as a "white box" that provides significantly more information than just the final performance estimate that is the starting point for more traditional "black box" approaches. By grouping stations using that information we have also reduced the dimensionality of the problem. These can be invaluable in search algorithms for large-scale simulation optimization problems.

3.2 Preliminaries

This section introduces the problem and its input data, the discrete-event simulation model we use, and some alternatives for obtaining starting solutions for the simulation-optimization search. Parts of Sections 3.2.3, 3.2.4, and a time-homogeneous version of 3.2.4 are recaps of a discussion in [44].

3.2.1 Problem Statement

Our goal is to minimize the expected number of "unhappy" customers who want to check out a bike when a station is empty or return a bike when a station is full, by giving an initial bike allocation x_i and dock (capacity) allocation r_i at each station i. We assume that the total number of bikes b and the total number of docks c is fixed in the system. The level of bikes x_i assigned is constrained by the capacity r_i, and the capacity r_i is constrained between 16 and 60 by physical space limits at the station. (This latter restriction is a simplification, since the actual space limits vary from station to station.) In practice in NYC, docks mostly come in sets of 3, but we ignore that complication in the work that follows. We use ξ to denote the random objects in the system, including the random arrivals and departures of customers at each station as well as the trip durations, so the notation accommodates for the use of Sample Average Approximation [48]. Thus the problem can be formulated as

$$\begin{aligned} \underset{x,r}{\text{minimize}} \quad & f(x,r) = Ef(x,r;\xi) \\ \text{subject to} \quad & \sum_i x_i = b \\ & \sum_i r_i = c \\ & 0 \le x_i \le r_i, \ \forall i \\ & 16 \le r_i \le 60, \ \forall i \\ & x_i, r_i \text{ integer}, \ \forall i. \end{aligned} \tag{3.1}$$

The function $f(\cdot,\cdot)$ yields the number of unhappy customers in one day, and we estimate its expectation using simulation.

3.2.2 Input Data

We use real trip data from the 14 weekdays in the period December 1 to 20, 2015. We selected this time period because Citi Bike completed its most recent expansion up to 86th Street by November 2015 [15]. This period excludes the lower demands seen on weekends and during the holiday season starting from December 21. During December 1 to 20 there were 466 active stations in Manhattan and

Brooklyn, with a total capacity of $\sum_i r_i = 15777$ docks. The number of bikes used is approximately $b = 6074$. The daily number of trips in this period was 31,400, which is lower than the August average of 45,000, presumably because of cold weather. We subsequently adjusted the rates by a multiplier of 1.5 to align our data to an average non-winter month after expansion.

The input data used in the simulation consists of the flow rates between pairs of stations and the trip durations. The flow rates are taken to be piecewise constant over each of the 48 30-minute time intervals throughout the day, and are estimated from the data. The flow rate $\mu_{t,i,j}$ in time interval t, $t = 1, \dots, 48$ from station i to j is calculated from the total number of observed trips from station i to j in that interval, divided by the time that the station is not empty. (This accounts for the censoring that happens when no bikes are available, but does not account for the censoring that happens when a biker tries to return a bike to a full rack and must go to an adjacent station.) The trip durations are obtained using linear regression, fitting the log of the trip durations seen in data to the log of the predicted cycling durations from Google Maps. Figure 3.1 shows a scatter plot in log scale with the fitted regression line (right plot) on 85% of the data inside the central ellipse (left plot).

3.2.3 Discrete-Event Simulation Model

We adapt an existing simulation model [55] written in Python that operates in discrete time (minute by minute). The arrival process of potential bikers at stations are independent across stations, and at each station i is a time-varying Poisson process with rate $\mu_{t,i} = \sum_j \mu_{t,i,j}$ in time interval t, with the arrival times rounded to the nearest minute. The destination of a biker leaving station i in time interval t

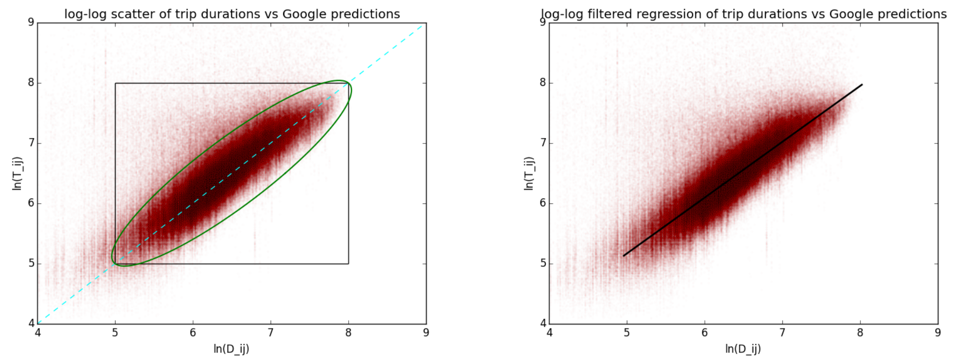


Figure 3.1: The regression line is $\ln(\text{observed}) = 0.93 \ln(\text{google}) + 0.53 + \epsilon)$, where ϵ is normally distributed with mean 0 and variance 0.066. The R^2 value of the fit is 0.806. Durations are measured in seconds.

has a multinomial distribution with the probability of going to station j estimated by $P_{t,i,j} = \mu_{t,i,j}/\mu_{t,i}$. The associated trip duration T_{ij} is lognormally distributed with parameters obtained from the regression model above, and also rounded to the nearest minute.

The system evolves as follows. In each new minute we generate and schedule the trips starting in that minute from each station i and assign a destination and duration to each trip. Next, all the trips scheduled to arrive or depart a station in this minute are executed as follows. If a departing trip cannot start now because the origin station i is empty, the customer leaves the system and the trip is recorded as a "failed-start". If an arriving trip cannot end because the destination station j is full, the state "failed-end" is triggered, and a new trip heading from j to the nearest station is scheduled. Customers make at most 3 attempts to return a bike, at which point we label the final trip as a "bad-end," which happens rarely (less than 1% of the trips). With this simulation model, the objective in (3.1) is

$$\min_x f(x) = E_x[\#\text{failed-starts}] + E_x[\#\text{failed-ends}] + E_x[\#\text{bad-ends}], \qquad (3.2)$$

where each expectation is estimated using sample-average approximation over a fixed set of replications of the simulation model.

We have sped up the basic model by a factor of approximately 40% by generating trips in 30-minute batches and using conditional uniform occurrences to generate the Poisson arrival processes. The time required to simulate one replication of an 18-hour day (assuming nothing happens from 12-6AM in which only 2.4% of the daily trips occur) is around 1.4 seconds on a desktop with 4-core Intel Core i7-3770 CPU 3.40 GHz processor, 16G memory, and Ubuntu 14.04 OS.

3.2.4 Starting Solutions for the Simulation Optimization

We introduce three alternative methods to generate starting solutions.

An Equal Allocation Solution

With the current capacity at each station fixed at its true value as of December, 2015, a naïve solution is to allocate some bikes to every station in proportion to its capacity. This solution does not take the flows in and out of stations into consideration, and so suffers from flow imbalances.

A Fluid Model Solution

Now consider flow rates but ignore randomness, so that in period t customers drop off bikes at station i with constant rate $\lambda_{t,i} = \sum_j \mu_{t,j,i}$ and pick up bikes at the same station with constant rate $\mu_{t,i}$. (For simplicity we assume that the trip durations are 0.)

The key idea is to calculate the minimum level of bikes and docks to start the day with so that the objective is 0. To achieve that, suppose the level of bikes in station i at minute q in the day $Y_i(q)$ is not constrained by 0 or the capacity. Then given any starting bike allocation x_i, $Y_i(q)$ is equal to the initial level x_i, plus the net flow of bikes in all the complete 30-minute intervals before q, and the net flow in the last less-than-30-minute interval that contains q. That is,

$$Y_i(q) = x_i + \left(\sum_{t=1}^{\lfloor q/30 \rfloor} 30(\lambda_{t,i} - \mu_{t,i}) \right) + (q - 30\lfloor q/30 \rfloor)(\lambda_{q,i} - \mu_{q,i}).$$

To avoid cost when the station is empty or full, in a perfect world we would start with $\hat{x}_i = x_i - \min_q Y_i(q)$ bikes and capacity $\hat{r}_i = \max_q Y_i(q) - \min_q Y_i(q)$ docks so there are sufficient bikes and docks throughout the day. Figure 3.2 gives an example of this "ideal" solution.

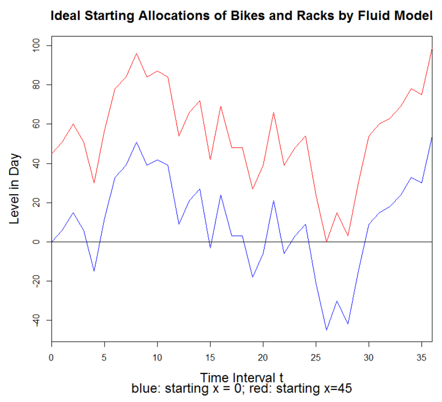


Figure 3.2: The bike level in an 18-hour day for the fluid model. The blue curve starts with 0 bikes, with lowest and highest levels of -45 and 54. The red curve starts with the "ideal" of 45 bikes and 99 docks.

Starting with the ideal $\hat{x}_i$ and $\hat{r}_i$ for all stations i would require many more bikes and docks than we have. Therefore, after obtaining these ideal levels, we scale the dock allocations to ensure that we don't exceed our available capacity. In doing so, some care is required to account for the integer nature of the dock allocations; we omit the details. A similar scaling is used for the bike allocations. An additional complexity results when these unequal scalings result in a station receiving more bikes than its allocated docks, but again we omit the details of our ad-hoc solution.

A shortcoming of the fluid model, noted in [44], is that it allocates almost no capacity to stations with nearly balanced inward and outward flow rates, irrespective of their magnitude.

A Continuous-Time Markov Chain Solution

[37] use a very similar model to ours, and show that under the admittedly very strong assumption that the objective function in our optimization problem is separable by stations. That is, the problem decomposes into a sum of functions, where the function for each station depends only on the inflow and outflow rates and the number of bikes and docks allocated to that station. Also, the change in the allocations at one station does not affect the inflow and outflow rates at its downstream and upstream stations. This dramatic simplification, together with a result establishing that the objective function can be extended to a piecewise-linear convex function, allows them to obtain bike and dock allocations by solving a linear integer program. Their result only applies to the case where the flow rates are constant in time. An extension of their result due to Freund, Henderson, and Shmoys allows us to solve the time-inhomogeneous problem, albeit still under the "objective separable by stations" assumption. The solution thus obtained is our third starting solution for simulation optimization.

3.3 Simulation Optimization

In this section we suggest four simulation-optimization heuristics that can be used to solve (3.1) over different time horizons and problem features. Each alternative is tested using the starting solutions in Section 3.2.4. The structures of the heuristics are similar. In each iteration we generate a trial solution and evaluate it with the discrete event simulation model in Section 3.2.3. If the trial solution improves the objective, then we move to the solution; otherwise we stay at the last solution. We use common random numbers to evaluate each trial solution using 30 replications of the simulation. In generating the trial solutions, we treat the simulation as a "white box," exploiting gradient-like information that can be gathered inside the model.

Recall that in the objective function (3.2), "# failed-starts" is the number of trips that cannot start because the origin station has no bikes, "# failed-ends" is the number of trips that cannot end because the destination station has no empty docks, and "# bad-ends" is the number of trips in which the customer finally abandons the bike after 3 failed attempts. Although "bad-ends" have serious consequences, they happen very rarely ($< 1\%$ of the trips) and thus are ignored in our methods when generating trial solutions. We denote the allocation of bikes (docks) at station i at the beginning of the day by $x(i)$ ($r(i)$), where i is the station id of one of the 466 stations. As a reminder, the total number of bikes in the system is $b = 6074$, and the capacity at each station is the status-quo as of December 2015.

3.3.1 Simulation Optimization Heuristics to Optimize Bike Allocations

First, consider a simpler problem that only optimizes the bike allocation x for a *fixed* capacity r equal to the status-quo of December 2015 by treating r in (3.1) as an input parameter. We start with a simple heuristic that optimizes the bike allocation only considering the morning rush (6-10am), similar to the one introduced in [44], then turn to optimizing over an 18-hour day (6-12am). We disregard 12-6am because only 2.4% of the daily trips occur in that interval.

Optimizing Bike Allocations for the Morning Rush-hour

Suppose for now we are only optimizing the objective (3.2) over the morning rush (6-10am).

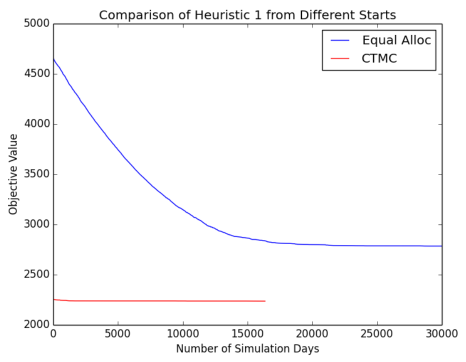
From m replications (days) of the 6-10am simulation with x as the initial allocation, we can estimate the objective evaluated at x from the counts of the failed-starts and failed-ends over all replications. Suppose we also obtain the list of *stations* where the failed-starts and first attempts of failed-ends (not including later attempts when the destination station is full) arose. Define the ordered list $statE(x, m, \ell)$ (we will suppress the arguments) as origin stations with the top ℓ # failed-starts, and $statF(x, m, \ell)$ as the destinations with the top ℓ # failed-ends. We expect that the statE stations have many failed-starts because they are too empty at 6am, whereas the statF stations have many failed-ends because they are too full at 6am. Thus, increasing the initial bike level x at a station in statE and decreasing it at a station in statF should decrease the # failed-starts and the # failed-ends in (3.2). This is mostly true for the morning rush-hour period, but is flawed if we are optimizing over the entire day, as discussed in Section 3.3.1. The method swaps w bikes between two randomly selected stations in these lists as described in Heuristic 4.

We use $m = 30$ replications. The list size for statE and statF is $\ell = 20$. The number of bikes allowed to move from/to each station in each iteration w is changed adaptively along the iterations. Initially $w = 3$, and whenever a consecutive of 100 iterations (100 trial solutions) cannot improve the objective, w is reduced by 1. When starting with a solution closer to optimality (like CTMC), w can be adjusted to start from 1, but here we keep it consistent for all starting solutions just for the fairness of comparison. The initial solution x_0 and the terminating one are evaluated with 50 and 100 independent replications, respectively, to obtain an independent estimate of the objective-function value, independent of the replications used in the search. We choose to stop the heuristic when it fails to improve the objective for 200 consecutive iterations (trial solutions generated).

The heuristic is tested from the proportional-allocation solution and the CTMC solution, giving Figure 3.3, in which the x-axis is the number of simulated days (replications), and the y-axis is the objective. The equal allocation solution starts with a 95% confidence interval for the objective of 4673 ± 35 and ends at 2775 ± 22. The CTMC solution starts with an objective of 2276 ± 30 and ends at 2236 ± 22. The heuristic makes great improvements to the equal allocation solution (42%), but not much to the better CTMC solution (2%).

Heuristic 4 Optimizing x for the morning rush.

```
Input: A starting solution $x_0$. The list size $\ell$ for statE and statF. The random seed for the simulation (daySeed). Number of replications $m$ for the simulation. Number of bikes $w$ allowed to move to/from each station in each iteration.
 1: initialize Set $k = 1$. Run simulation to evaluate $x_0$ and obtain the initial statE and statF.
 2: repeat
 3:     procedure GENERATE TRIAL SOLUTION
 4:         Randomly choose $s_{\mathrm{E}}$ from statE and $s_{\mathrm{F}}$ from statF such that $x_{k-1}(s_{\mathrm{E}}) + w \leq r(s_{\mathrm{E}})$ and $x_{k-1}(s_{\mathrm{F}}) - w \geq 0$.
 5:         Generate trial solution $x'$ by add $w$ bikes to $s_{\mathrm{E}}$ and removing $w$ bikes from $s_{\mathrm{F}}$ based on $x_{k-1}$.
 6:     procedure SIMULATE AND EVALUATE
 7:         Set random seed = seed.
 8:         Evaluate $x'$ using the sample average of the objective in (3.2) over $m$ replications of simulation.
 9:         if $x'$ shows improvement in the average objective then
10:             Set $x_k = x'$ and let $k = k + 1$. Record statE and statF from the simulation.
11:         else
12:             Go back to generate another trial solution.
13: until Stopped.
```



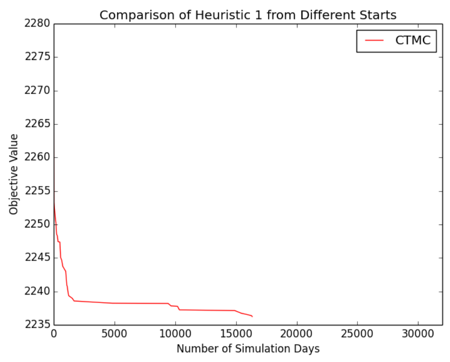


Figure 3.3: The objective of running Heuristic 4 (optimize bikes only for morning rush) starting from the equal allocation and CTMC solutions. The left one is the comparison of the two, and the right one is CTMC only on a magnified scale.

Optimizing Bike Allocations for the Entire Day

Consider the same bike-allocation-only problem as above, but instead evaluating the objective over an 18-hour day. When we use Heuristic 4, changing only the simulation period to 18 hours, the method was not able to find improvement for a long time. One of the problems is that most stations behave very differently in the morning and in the afternoon, but statE and statF are still chosen based on the counts of failed starts and failed ends over the *entire* day. If a station in statE has many failed starts in the morning, then adding bikes to the station at the beginning of the day helps in reducing the objective. However, consider a busy station in statE that fills in the morning and then empties in the afternoon. Adding bikes to this station at the beginning of the day makes # failed ends in the morning worse. Meanwhile, because the station starts to empty at the same time irrespective of adding bikes or not, it does not help to reduce the count of failed starts in the afternoon. Indeed, the sample path for the increased allocation of bikes couples with that of the previous allocation once the bike level hits 0 or the capacity. A symmetric problem arises with stations in statF that empty out in the morning and fill up in the afternoon; reducing their bike levels at the beginning of the day would increase the objective instead. Moreover, these stations with both failed starts and failed ends are typically those that have large traffic, e.g., near Penn Station, and thus contribute heavily to the objective.

To adjust Heuristic 4 to address this issue, we define the following 7 types of stations, with illustrations in Figure 3.4.

1. statEA: the stations that are empty in the morning and not full in the afternoon.

2. statEP: the stations that are empty in the afternoon and not full in the morning.

3. statFA: the stations that are full in the morning and not empty in the afternoon.

4. statFP: the stations that are full in the afternoon and not empty in the morning.

5. statBI: the stations that are full in the morning and empty in the afternoon.

6. statBD: the stations that are empty in the morning and full in the afternoon.

7. statC: the stations that contribute the least to the objective by rarely being full or empty.

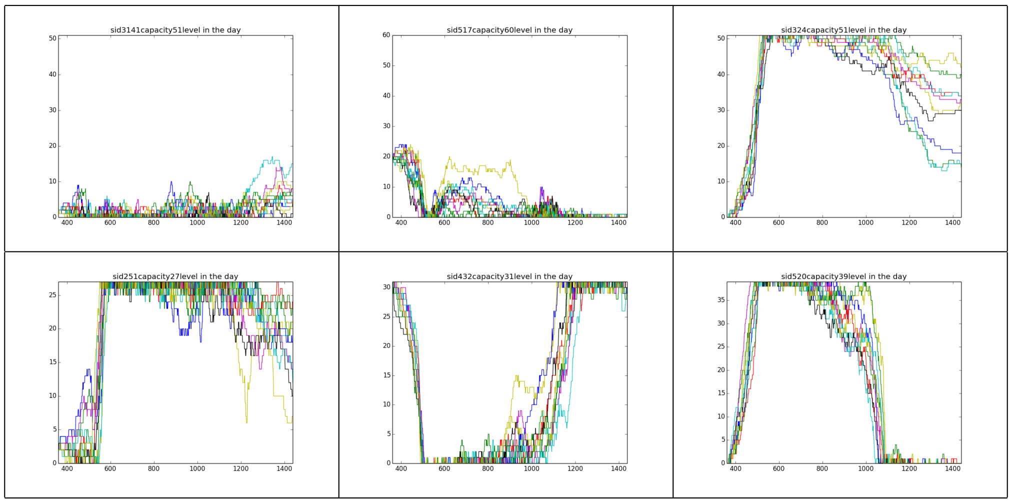


Figure 3.4: The bike levels for the example stations from the lists statEA, statEP, statFA, statFP, statBI, and statBD (ordered from left to right in rows) over 10 replications. The x-axis is in time from 6-12am.

It appears that statEA, statEP, and statBI need more bikes in the morning, whereas statFA, statFP, and statBD need fewer bikes in the morning. The stations in statC contribute the least to the objective, and thus are used as “back-up.” This inspires Heuristic 5 that changes the method for generating trial solutions, while keeping the rest of Heuristic 4 the same.

Heuristic 5 Optimizing x for the entire day.

```
1: procedure GENERATE TRIAL SOLUTION
2:     Randomly choose a station $s_{\text{type}}$ from each of the list types {EA, EP, FA,
       FP, BI, BD}.
3:     To generate trial solution $x'$ from $x_{k-1}$
4:     take $w$ bikes from each of $s_{FA}$, $s_{FP}$ and $s_{BD}$,
5:     give $w$ bikes to each of $s_{EA}$, $s_{EP}$ and $s_{BI}$.
6:     If any of the movements is not possible because of capacity restrictions, we
       substitute the station by a random station in statC that allows the movement.
```

With the same configurations of m, ℓ, w, and the stopping criteria as in Section 3.3.1, Figure 3.5 depicts the progress of this heuristic starting from the equal allocation and CTMC solutions when optimizing the bike allocation over the 18-hour period. The equal allocation starts with a 95% confidence interval of the objective of 12249 ± 89 and ends at 10428 ± 47 (-15%). The CTMC solution starts at 9239 ± 73 and ends at 9168 ± 46 (-1%). The percentage difference between the starting and ending objectives has decreased compared to when optimizing only over the morning rush, suggesting that the 18-hour day problem with its time-flow complexities is more difficult than the rush-hour problem. The fact that the heuristics started from different solutions are unable to close the gap in the ending solutions may also suggest that there is room for improvement for this 18-hour problem.

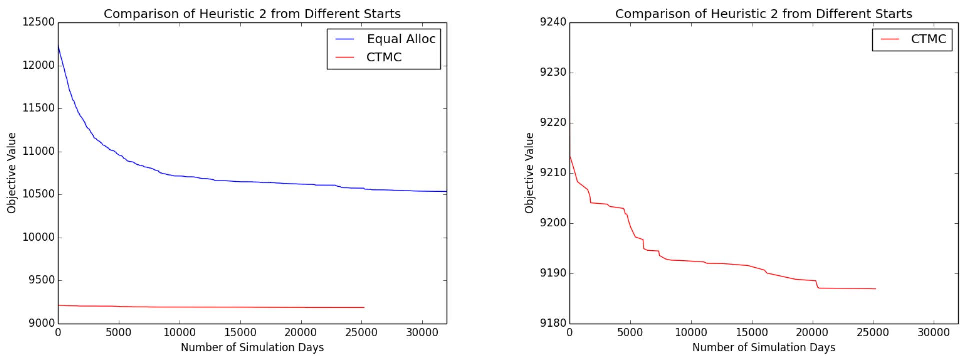


Figure 3.5: The objective of running Heuristic 5 (optimize bikes only for the entire day) starting from the equal allocation and CTMC solutions. The left plot is the comparison, and the right one is CTMC only on a magnified scale. The scale is different from Figure 3.3 since more failed trips are incurred over a 18-hour day.

3.3.2 Simulation Optimization Heuristics to Optimize Both Bike and Dock Allocations

Now we return to the original formulation (3.1), solving for the allocations of both bikes and docks, initially just over the morning rush and then for an 18-hour day.

Optimizing Bike and Dock Allocations for the Morning Rush-hour

To incorporate the movements of docks into Heuristic 4, notice that some stations in statE cannot receive more bikes because they are already full. Increasing the capacity at such stations allows us to allocate more bikes and thus reduces the # failed-starts. Similarly, some stations in statF start empty, so increasing the capacity at such stations allows them to receive more bikes and thus reduces the # failed-ends. The docks added to these two types of stations come from statC that is subject to the least amount of change in the objective due to the loss of a dock or a bike. Thus we change the procedure for generating the trial solution in Heuristic 4, giving Heuristic 6.

Heuristic 6 Optimizing x and r for the morning rush.

```
1: procedure GENERATE TRIAL SOLUTION
2:     Randomly choose stations $s_{\mathrm{E}}$ from statE and $s_{\mathrm{F}}$ from statF.
3:     To generate trial solutions $(x', r')$ from $(x_{k-1}, r_{k-1})$,
4:     if $x_{k-1}(s_{\mathrm{E}}) + w \leq r_{s_{\mathrm{E}}}$ and $x_{k-1}(s_{\mathrm{F}}) - w \geq 0$ then
5:         Take $w$ bikes from $s_{\mathrm{F}}$ and give $w$ bikes to $s_{\mathrm{E}}$.
6:     else if $x_{k-1}(s_{\mathrm{E}}) + w > r_{s_{\mathrm{E}}}$ and $r_{k-1}(s_{\mathrm{E}}) + w \leq 60$ then
7:         Take $w$ docks and $w$ bikes from a random station in statC and give them to $s_{\mathrm{E}}$.
8:     else if $x_{k-1}(s_{\mathrm{F}}) - w < 0$ and $r_{k-1}(s_{\mathrm{F}}) + w \leq 60$ then
9:         Take $w$ docks from a random station in statC and give them to $s_{\mathrm{F}}$. Excess bikes will go to $s_{\mathrm{E}}$.
```

This heuristic prioritizes moving bikes first, and if that fails, it moves docks to the stations that cause the failure. With the same configurations of m, ℓ, and w as in Section 3.3.1, Figure 3.6 gives the progress of this heuristic starting from the equal allocation, the fluid model, and the CTMC solutions, when optimizing both bike and dock allocations over the 18-hour day. The equal allocation starts with a 95% confidence interval of the objective of 4690 ± 45 and terminates at 1936 ± 18 (-59%), the CTMC solution starts with objective of 1379 ± 28 and ends at 1333 ± 13 (-3%), and the fluid model starts with objective of 1472 ± 32 and ends at 1378 ± 13 (-6%).

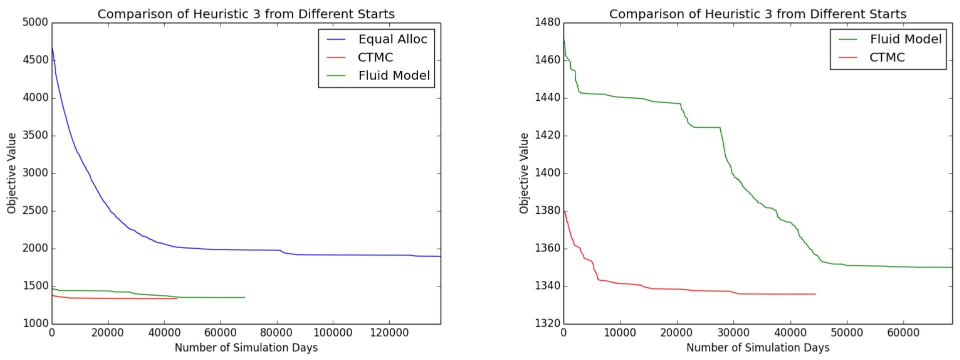


Figure 3.6: The objective of running Heuristic 6 (optimize bikes and docks for the morning rush) starting from the equal allocation, the fluid model, and CTMC solutions. The left plot is the comparison of all three, and the right one is the comparison between the CTMC and the fluid model solutions on a magnified scale.

Optimizing Bike and Dock Allocations for the Entire Day

Now we make changes to Heuristic 5, which optimizes the bike allocation over the entire day, to allow for dock movements. Similar to Heuristic 5, the following heuristic requires lists statEA, statEP, statFA, statFP, statBI, statBD, and statC from the last solution (x_{k-1}, r_{k-1}).

With the same configurations of m, ℓ, w, and the stopping criteria as in Section 3.3.1, Figure 3.7 gives the objective change of this heuristic starting from the equal allocation, the fluid model, and the CTMC solutions, when optimizing both bike and dock allocations for the 18-hour day. The equal allocation starts with a 95% confidence interval for the objective of 12249 ± 89 and terminates at 8915 ± 47 (-27%), the CTMC solution starts with objective 6937 ± 122 and terminates at 6681 ± 43 (-3%), and the fluid model solution starts with objective 7063 ± 89 and terminates at 6865 ± 44 (-3%). Note that here the starting objectives of CTMC and Fluid solutions are both smaller than those in Figure 3.5. This is because the solutions here optimize both bike and dock allocations, whereas the dock levels in

Heuristic 5 are taken as is from the reality.

Heuristic 7 Optimizing x and r for the entire day.

```
1: procedure GENERATE TRIAL SOLUTION
2:     Randomly choose a station $s_{\text{type}}$ from each of the list types in {EA, EP, FA, FP, BI, BD}.
3:     To generate trial solutions $(x', r')$ from $(x_{k-1}, r_{k-1})$,
4:     Move $w$ docks from a random station in statC to $s_{\text{BI}}$ if $r_{k-1}(s_{\text{BI}}) + w \leq 60$.
5:     Move $w$ docks from a random station in statC to $s_{\text{BD}}$ if $r_{k-1}(s_{\text{BD}}) + w \leq 60$.
6:     Try taking $w$ bikes from each of $s_{FA}$, $s_{FP}$ and $s_{BD}$ in $x_{k-1}$. If any of the movements is not allowed because the station is already empty, give the station $w$ docks from a random station in statC.
7:     Try giving $w$ bikes to each of $s_{EA}$, $s_{EP}$ and $s_{BI}$ in $x_{k-1}$. If any of the movements is not allowed because the station is already full, give the station $w$ docks and $w$ bikes from a random station in statC.
```

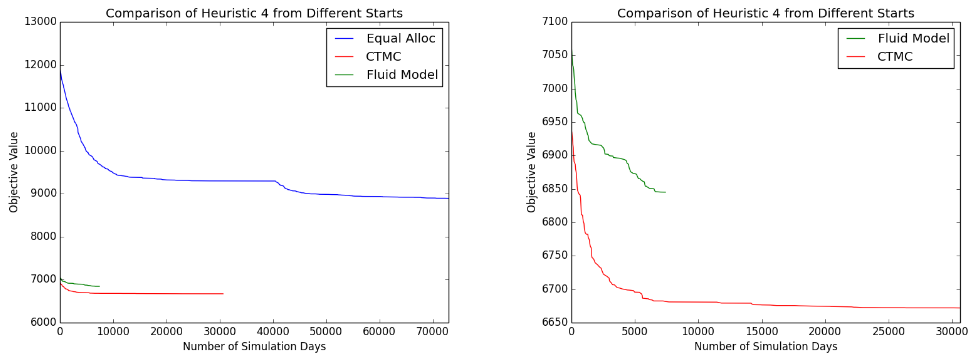


Figure 3.7: The objective of running Heuristic 7 (optimize bikes and docks for the entire day) starting from the equal allocation, the CTMC, and the fluid model solutions. The left plot is the comparison of all three, and the right one is the comparison between the CTMC and the fluid model solutions on a magnified scale.

3.4 Remarks

Our heuristics can make local improvements to the objective regardless of the starting solution, yielding practically relevant improvements over all solutions, although the improvements relative to the (already apparently well-performing) CTMC solutions are modest especially in the 18-hour problem. The starting solution plays an important role, because the heuristics cannot close the gap to the best solution we have seen. The heuristics are perhaps best viewed as using approximations for "gradients" (for this integer-variables problem) to guide reallocations of bikes and docks.

CHAPTER 4
USING PSEUDO-GRADIENT IN LARGE-SCALE SIMULATION OPTIMIZATION

4.1 Introduction

In last chapter, we have explored the idea of smartly making local improvements using simulation traces and applied it to the bike-sharing example. The resulting heuristics move only a few bikes and docks in each iteration according to a few priority rules, making modest improvements with every 30 replications of the simulation. The limitation of such heuristics seems to lie in the complicated structure and the small magnitude of the local changes. In each local step, by using the output of the latest simulation runs, we carefully choose a few stations that generated the most number of failed starts or failed ends in the morning or afternoon, and move a bike and/or a dock between them. This is based on the intuition that the chosen stations contribute most to the objective value. Such intuition is very similar to gradient search, except that we are not estimating the gradient itself, but instead using gradient-like information from the simulation traces. This has inspired us to further exploit the simulation traces to come up with direct approximations to the gradient, which we call pseudo-gradient, and use it to generate bigger and more efficient local steps. Similar to the heuristic from the last chapter, we expect this to be useful in solving large-scale simulation optimization problems where the solution space is so large that the direct gradient estimation using finite differences is too costly and a pure random search is unlikely to be effective.

The general idea of obtaining pseudo-gradient is as follows. We use simulation traces to approximate the system states after a small change in a single coordinate of the input parameter (i.e. decision variable). Then from the changed states we can estimate the partial derivative of the objective function in that coordinate. The information needed from simulation traces could be the sequence of events, which we assume does not alter with a small change in the decision variable. When the input parameter is in continuous space and the change is infinitesimal, this idea of "keeping the event list as is" is similar to Infinitesimal Perturbation Analysis (see, e.g. [38], [26], [22], [24]), for which asymptotic results have been established. [38] provides methods (e.g. event and state matching) for Finite Perturbation Analysis that can be applied to discrete changes, but it is nontrivial how to use those methods on a finite-horizon simulation with huge state space and complicated event list.

In this chapter, we illustrate how to numerically solve such simulation optimization problems using pseudo-gradient. We provide quality and affordable new methods to solving the large-scale problems of bike-sharing and multi-skill call centers with nearly 1000 decision variables. However, our contribution is not limited to these two specific problems. We give a general framework of local search using pseudo-gradient that can be applied to simulation optimization problems with both deterministic and stochastic constraints. We show that when the pseudo-gradient approximation is close enough and the optimized function has "nice" structural property, the search will lead to a near-optimal solution. We give ideas on how to generate approximate gradients from the simulation traces without re-running the simulation, as would be required with finite differences, for example. The savings in computational time makes it feasible to obtain quality solution to large-scale simulation optimization problems that were not tractable using traditional methods like gradient search or random search.

The chapter is organized as follows. Section 4.2 gives the outline of a search algorithm using pseudo-gradient. In Section 4.3 we show that when the objective function is continuous and strongly convex and the pseudo-gradient is a close approximation to the actual gradient, the pseudo-gradient search algorithm would terminate at a near-optimum solution. This proves what a gradient search using gradient information with errors (pseudo-gradients) can achieve in the best-case scenario of strongly convex functions and uniformly bounded errors. Sections 4.4 and 4.5 use two case studies to illustrate how pseudo-gradient search can be used in large-scale simulation optimization. Each section contains problem description, literature reviews, summary of algorithm, and numerical results for the regarding problem. The first problem is the (re)allocation of bikes and docks in a large bike-sharing system from last chapter, and we have shown that using pseudo-gradients gives a more efficient search procedure. The second case is the scheduling of agents of different skill groups and shifts in a multi-period multi-skill call center, and we compare the heuristic using pseudo-gradients with the state-of-art cutting plane method designated for call center scheduling problems.

4.2 Pseudo-gradient Search

Suppose we are solving the simulation optimization problem

$$\min_{x\in\Theta} f(x), \tag{4.1}$$

consisting of an objective $f : S \to [-\infty, +\infty]$ where $S \subseteq \mathbb{R}^d, d < \infty$, decision variables x, and constraints $x \in \Theta \subseteq S$. Here $f = Ef(x,\xi)$, where $f(x,\xi)$ represents the output of a simulation logic for the objective for one replication. The constraint set Θ could be deterministic or stochastic as $\Theta = \{x : Eh(x,\xi) \geq 0\}$, where $h(x, \xi)$ is the output of a simulation logic for the constraint function for one replication.

A pseudo-gradient provides gradient-like information calculated from the simulation traces. This is rather vaguely specified, as a pseudo-gradient is not unique given the simulation traces and could be anything that provides directional information for local improvement. For example, suppose f is continuous and proper, and $\nabla f(x) \in \partial f(x)$ where $\partial f(x)$ is the sub-differential set of f at point x. A pseudo-(sub)gradient $\widehat{\nabla} f(x)$ could be an approximation of the subgradient $\nabla f(x)$. When f is discrete, the i-th coordinate of $\widehat{\nabla} f(x)$ could be an estimation to the forward difference $f(x + e_i) - f(x)$. In the above-mentioned simple cases, the pseudo-gradient $\widehat{\nabla} f(x)$ is taken as an approximation of the true "gradient" and thus has the same dimension as x, but it does not have to, as shown in the bike-sharing example in Section 4.4.

For simplicity, suppose for now we have the unconstrained version of (4.1),

$$\min f(x), \tag{4.2}$$

We will discuss how to convert box-constrained and stochastic-constrained problems to this unconstrained version in Sections 4.4 and 4.5, respectively.

For the unconstrained problem (4.2), we use Algorithm 8 that is similar to a random search but uses pseudo-gradient $\widehat{\nabla} f$ to generate trial solutions. Algorithm 8 lacks precise details on how some steps are to be completed. In later sections we will be more concrete on these steps, especially the procedure to generate trial solution from pseudo-gradient in step 3. Note that each trial solution is evaluated using the same random seed over m replications of the simulation. This use of Common Random Numbers improves the signal to noise ratio in comparing different solutions. We usually use a shrinking step size in the procedure of generating trial solution, so that the algorithm is able to make faster progress in the initial iterations and does not "miss" the optimum in the later ones. The next trial solution is generated randomly in a small neighborhood, which allows for exploration while exploiting the "best" direction indicated by the pseudo-gradient. Finally, the stopping criteria is a combination of budget limitation and local optimality (defined by pseudo-gradient).

Algorithm 8 Pseudo-gradient Local Search

```
Input: A starting solution $x$. A random seed *seed*. Number of replications $m$
    and $m'$ for the simulation. Maximum number of consecutive failures before
    stopping $n_{\text{maxFails}}$.
 1: initialize Run $m$ replications of the simulation to evaluate the starting solu-
    tion. Calculate pseudo-gradient $\widehat{\nabla} f(x)$ using the traces. Set $n_{\text{fails}} = 0$.
 2: repeat
 3:     procedure GENERATE TRIAL SOLUTION
 4:         Generate a trial solution $x'$ using pseudo-gradient $\widehat{\nabla} f(x)$.
 5:     procedure SIMULATE AND EVALUATE
 6:         Set random seed = *seed*.
 7:         Evaluate $x'$ using the sample average of the objective over $m$ replications
    of simulation.
 8:         Record pseudo-gradient $\widehat{\nabla} f(x')$ calculated from the simulation traces.
 9:         if $x'$ shows improvement in the average objective then
10:             Set $x = x'$ and $\widehat{\nabla} f(x) = \widehat{\nabla} f(x')$.
11:             Set $n_{\text{fails}} = 0$.
12:         else
13:             Set $n_{\text{fails}} = n_{\text{fails}} + 1$ and go back to generate another trial solution.
14: until Either $n_{\text{fails}} >= n_{\text{maxFails}}$ or $\widehat{\nabla} f(x)$ shows no room for improvement.
Output: Set random seed = *seed* + 1 and evaluate $x$ using $m' > m$ replications of
    simulation. Output $x$ and the sample average of the objective (with confidence
    interval).
```

As any other simulation optimization method, this algorithm does not guarantee to output the global optimal or near-optimal solution, unless the objective function has special structure (see Section 4.3). We use the standard approach of restarting when the search is suspected to be trapped in the "basin of attraction" of a local minimum. This is done by resetting the step size to a large value, as applied in the bike-sharing example in 4.4.

4.3 Convergence Results

This section provides theoretical support to the near-optimality of a pseudo-gradient search algorithm under certain conditions. Throughout the section, we use the following common definition of proper function and strong convexity (see, e.g., [8]). Recall that in Algorithm 8, the optimized function f is unconstrained, so we assume $S = \mathbb{R}^d$ throughout the section.

Definition 3 (Proper function). *Suppose $\mathcal{X}$ is nonempty. A convex $f : \mathcal{X} \to [-\infty, +\infty]$ is proper if $-\infty \notin f(S)$ and $\{x \in \mathcal{X} : f(x) < +\infty\}$ is nonempty.*

Definition 4 (Strongly convexity). *Let S be a convex set and $f : S \to [-\infty, +\infty]$ be continuous and proper. f is σ-strongly convex if and only if $f(\cdot) - \sigma||\cdot||^2/2$ is convex, or equivalently, $f(y) \geq f(x) + \langle y - x, \nabla f(x)\rangle + \sigma||y - x||^2/2$, $\forall x, y \in S, \nabla f(x) \in \partial f(x)$, where $\partial f(x)$ is the subdifferential set of f at x.*

For simplicity, assume Algorithm 8 always stops only when

$$||\widehat{\nabla} f(x)|| \leq \epsilon_s \tag{4.3}$$

in line 14 for some small and positive ϵ_s ($||\cdot||$ is the Euclidean norm). Suppose the true optimal solution is x^* and the outputted solution of Algorithm 8 is $\widehat{x^*}$ when stopped. Then the following results are true.

Theorem 5. *Assume $f(x)$ is σ-strongly convex by Definition 4, and when Algorithm 8 stops with condition (4.3), the pseudo-gradient $\widehat{\nabla} f(x)$ satisfies $||\nabla f(\widehat{x^*}) - \widehat{\nabla} f(\widehat{x^*})|| \leq \epsilon$ for some with $\epsilon > 0$. Then $f(\widehat{x^*}) - f(x^*) \leq (\epsilon + \epsilon_s)^2/(2\sigma)$.*

Proof. Since f is closed (by continuity) and convex, we have $f^{**} = f$. By Theorem 18.15 in [8], f is σ-strongly convex if and only if the (convex) conjugate function $f^*(y) := \sup_x \langle y, x\rangle - f(x)$ has a $(1/\sigma)$-Lipschitz derivative. Thus by descent lemma,

$$f^*(\nabla f(x)) \le f^*(\nabla f(y)) + \langle \nabla f^*(\nabla f(y)), \nabla f(x) - \nabla f(y)\rangle + \frac{1}{2\sigma}||\nabla f(x) - \nabla f(y)||^2 \tag{4.4}$$

Since f is closed (by continuity) and convex and we have $f^{**} = f$. Thus $\nabla f^* = \nabla f^{-1}$ and $\nabla f^*(\nabla f(y)) = y$. Following from (4.4) we have

$$\begin{aligned} f^*(\nabla f(x)) &\le f^*(\nabla f(y)) + \langle y, \nabla f(x) - \nabla f(y)\rangle + \frac{1}{2\sigma}||\nabla f(x) - \nabla f(y)||^2 \\ &= f^*(\nabla f(y)) + \langle y, \nabla f(x)\rangle - \langle y, \nabla f(y)\rangle + \frac{1}{2\sigma}||\nabla f(x) - \nabla f(y)||^2. \end{aligned}$$

Therefore,

$$\begin{aligned} \langle y, \nabla f(y)\rangle - &f^*(\nabla f(y)) \\ &\le \langle x, \nabla f(x)\rangle - f^*(\nabla f(x)) + \langle y - x, \nabla f(x)\rangle + \frac{1}{2\sigma}||\nabla f(x) - \nabla f(y)||^2. \end{aligned} \tag{4.5}$$

By Fenchel-Young Inequality (see, e.g. [8]), we have

$$f(y) + f^*(\nabla f(y)) \ge \langle \nabla f(y), y\rangle.$$

Thus following from (4.5),

$$f(y) \le f(x) + \langle y - x, \nabla f(x)\rangle + \frac{1}{2\sigma}||\nabla f(x) - \nabla f(y)||^2.$$

Now suppose the optimal solution to (4.2) is x^*. Since f is strongly convex on the entire domain, $\nabla f(x^*)$ can be taken as a d-dimensional zero vector. Then take $x = x^*$ and $y = \widehat{x^*}$ in (4.3),

$$f(\widehat{x^*}) \le f(x^*) + \frac{1}{2\sigma}||\nabla f(\widehat{x^*})||^2. \tag{4.6}$$

Since $||\widehat{\nabla} f(\widehat{x^*})|| \leq \epsilon_s$,

$$||\nabla f(\widehat{x^*})|| \leq ||\nabla f(\widehat{x^*}) - \widehat{\nabla} f(\widehat{x^*})|| + ||\widehat{\nabla} f(\widehat{x^*})|| \leq \epsilon + \epsilon_s.$$

Thus $f(\widehat{x^*}) - f(x^*) \leq (\epsilon + \epsilon_s)^2/(2\sigma)$. □

Theorem 5 gives the bound on the optimality gap $||f(x) - f(x^*)||$ of a pseudo-gradient search when the pseudo-gradient is approximated uniformly within ϵ of the true gradient. Sometimes the bound on the approximation error is proportional to the size of the true gradient, so that the pseudo-gradient becomes closer to the true gradient as the search approaches the optimal solution. Suppose the ratio of the approximation error to the true gradient is small enough ($< 1/2$) and we stop at a small $\epsilon_s \leq \epsilon$, then the optimality gap of the pseudo-gradient algorithm would be tighter than that in Theorem 5.

Corollary 6. *Assume $f(x)$ is σ-strongly convex, and when Algorithm 8 stops with condition (4.3), the pseudo-gradient $\widehat{\nabla} f(x)$ satisfies $||\nabla f(\widehat{x^*}) - \widehat{\nabla} f(\widehat{x^*})|| \leq (1 + c||\nabla f(\widehat{x^*})||)\epsilon$, with $\epsilon > 0$ and $0 \leq c < 1/\epsilon$. Then $f(\widehat{x^*}) - f(x^*) \leq (\epsilon + \epsilon_s)^2/(2\sigma(1 - c\epsilon)^2)$.*

Proof. When stopped,

$$||\nabla f(\widehat{x^*})|| \leq ||\nabla f(\widehat{x^*}) - \widehat{\nabla} f(\widehat{x^*})|| + ||\widehat{\nabla} f(\widehat{x^*})|| \leq \epsilon + \epsilon_s + c||\nabla f(\widehat{x^*})||,$$

so with $0 \leq c < 1/\epsilon$,

$$||\nabla f(\widehat{x^*})|| \leq \frac{\epsilon + \epsilon_s}{1 - c\epsilon}.$$

Then by (4.6), $f(\widehat{x^*}) - f(x^*) \leq (\epsilon + \epsilon_s)^2/(2\sigma(1 - c\epsilon)^2)$. □

When $c = 0$, Corollary 6 reduces to Theorem 5. Otherwise, the bound in Collary 6 is looser than that in Theorem 5. Note that in both Theorem 5 and

Corollary 6, we assume a bound on the pseudo-gradient at the stopping solution of Algorithm 8. This is probably difficult to verify in general, but is implied if the bound apply to all points x in the entire domain.

Theorem 5 states that in the best-case scenario of strongly convex objective function, if we can derive a pseudo-gradient that uniformly approximates the true gradient within an error of ϵ, then the pseudo-gradient search can output a near-optimal solution when stopping at condition (4.3), with an optimality gap of order $(\epsilon + \epsilon_s)^2$. This describes the best we can expect from any gradient-search method that use gradient information with error, as in the case of pseudo-gradient.

Corollary 6 describes what happens when the approximation error is proportional to the size of the true gradient when x is far away from x^*, and bounded by ϵ when x is within a small neighborhood around x^*. It would achieve an optimality bound equal to or better than that in Theorem 5 when the approximation error is not too large with respect to the true gradient ($c\epsilon < 1/2$). This corollary motivates the use of shrinking step sizes. At the earlier iterations of Algorithm 8, we make more steps into the negative pseudo-gradient direction approximated from the last evaluated trial solution before re-running the simulation again to update the pseudo-gradient (more details in later sections). The intuition is that when the search is at a trial solution that is far away from the optimal solution, the pseudo-gradient only needs to be directionally consistent with the actual gradient in order to yield improvements. However, in later iterations of the search, when the trial solution is within a small neighborhood of the optimal solution, the pseudo-gradient needs to be more precise and updated more frequently so that we do not step away from the optimum. Using the result of Corollary 6, we show that this shrinking step size regime makes faster progress towards the optimal solution without sacrificing the near-optimality.

The conditions in Theorem 5 and its corollary are rarely met in real-world problems (for example, decision variables could be discrete so there is no notion of strong convexity). Yet we will show that the use of pseudo-gradients can still provide quick and significant progress in the search for a good solution later in Sections 4.4 and 4.5.

4.4 Citibike

4.4.1 Introduction

In this section, we revisit the simulation optimization of Citi Bike system from Chapter 3 and provide a brand new and more efficient search heuristic using pseudo-gradient. For an overview of the problem and details on the simulation model, see Chapter 3.

4.4.2 Problem Statement

As a reminder, we are trying to find the optimal bike level x and dock level r to start each station with, so that the expected daily number of failed trips $f(x, r)$ is minimized. The problem can be formulated as

$$
\begin{aligned}
\underset{x,r}{\text{minimize}} \quad & f(x,r) = Ef(x,r;\xi) \\
\text{subject to} \quad & \sum_i x_i = b \\
& \sum_i r_i = c \\
& 0 \le x_i \le r_i,\ \forall i \\
& 16 \le r_i \le 60,\ \forall i \\
& x_i, r_i \text{ integer},\ \forall i,
\end{aligned}
\tag{4.7}
$$

where the objective is evaluated using simulation via

$$\min_x f(x,r) = \ E_{x,r}[\#\text{failed-starts}] + E_{x,r}[\#\text{failed-ends}].$$

Similar to Chapter 3, the "bad-end" event where a customer abandons the bike is rare and thus omitted from the objective.

4.4.3 Generating Trial Solutions Using Pseudo-gradient

In this section, we will fill in the details in step 3 of Algorithm 8 to use it for the bike-sharing example. A pseudo-gradient in this example would approximate the change in the number of failed trips with regard to a small (integer-valued) change in (x, r). Since Algorithm 8 is designated for solving the unconstrained prolem (4.2), to use it on the bike-sharing problem (4.7), we need to make sure that the pseudo-gradient does not guide the trial solution out of the feasible set. For example, consider a station that has 20 docks and 20 bikes at the current solution. We cannot remove a dock without removing a bike from it. Neither can we add a bike to the station without adding a dock. However, we can add or remove a bike and a dock simultaneously from such station and the resulting solution would be feasible. Therefore, when considering possible changes to the current solution, we incorporate such "diagonal" changes in (x, r) to accommodate for the constraints.

More specifically, for each station i, we consider the following six changes, keeping all other stations $j \neq i$ the same.

C_1: $x_i' = x_i + 1$

C_2: $x_i' = x_i - 1$

C_3: $r_i' = r_i + 1$

C_4: $r_i' = r_i - 1$

C_5: C_1 and C_3

C_6: C_2 and C_4

For each change $C_s, s = 1, \ldots, 6$, we use the information from the last simulation run of (x, r) to calculate the estimated change in the total number of failed-starts and failed-ends. This can be achieved by recording the sequence I_i of (attempted) bike pick-ups (-1) and drop-offs $(+1)$ at each station i when started with x_i bikes and r_i docks. Assuming that the sequence does not change when the station is started with x_i' bikes and r_i' docks instead, we can trace through the bike level during the day and compute how many failed-starts and failed-ends this station would have incurred. This gives the estimated change in the objective $\Delta_{si} = f(x', r') - f(x, r)$ with regard to the change C_s in the starting configuration of station i, assuming it does not affect any other stations $j \neq i$. We define $\{\Delta_{si}, s = 1, \ldots, 6, i = 1, \ldots\}$ as the pseudo-gradient $\widehat{\nabla} f(x, r)$. Note that it has a larger dimension than (x, r) by taking diagonal changes into consideration, to support better trial solution generation within the feasible set, as mentioned in the beginning of this section.

If we calculate Δ_{si} for all stations i in the system, we can obtain an ordered list $\boldsymbol{\Delta_s}$ of the change in objective with regard to change type C_s. Since our problem (4.7) is constrained, we use the following "matches" to generate a trial solution that satisfies $\sum_i x'_i = b$ and $\sum_i r'_i = c$. Furthermore, each match is executed only if the resulting (x', r') is feasible with $0 \leq x'_i \leq r'_i$ and $16 \leq r'_i \leq 60$ for any i. This enables us to use Algorithm 8 as if the problem is unconstrained.

M_1: Move a bike from station i to j.

M_2: Move a dock from station i to j.

M_3: Move a bike and a dock from station i to j.

M_4: Remove a bike and a dock from station i and give them to station j and k, respectively.

M_5: Remove a bike from station j and a dock from station k and give them to station i.

The effect of each movement on the objective can be estimated by the sum of Δ_{si} of the relevant changes to the stations. For example, the estimated change in the objective from movement M_4 is approximately $\Delta_{6i}+\Delta_{1j}+\Delta_{3k}$. We wish to generate the trial solution such that the sum of such estimated changes is minimized.

In summary, the procedure of generating a trial solution in Algorithm 8 is presented in Heuristic 9. When embedded in Algorithm 8, we set an initial n_{match} in the beginning of the search, and reduce it by half every time the stopping criteria in line 14 is met. Then we restore n_{fails} to 0 to continue the search with a smaller step size until eventually $n_{\text{match}} = 1$ and the algorithm terminates. The required pseudo-gradient $\widehat{\nabla} f(x)$ in Algorithm 8 is here in the form of $\Delta_{si}, s = 1, \ldots, 6$ and i indexing each of the stations in the system, so taking values from 1 to 466.

Heuristic 9 Generation trial solution (x', r') for the bike-sharing example.

```
1: procedure GENERATE TRIAL SOLUTION
Input: $\{\Delta_{si}, s = 1, \ldots, 6, i = \text{all stations}\}$. Local randomization parameter $d_R$.
    Number of matches $n_{\text{match}}$ to generate.
2:     For each change $C_s$, sort $\{\Delta_{si}\}$ from smallest to largest.
3:     repeat
4:         for Each match $M_1$ to $M_5$ do
5:             for Each change $C_s$ corresponding to the match do
6:                 Choose station $i$ randomly with top $d_R$ smallest $\Delta_{si}$. Make sure
                   the $C_s$ is feasible for this station.
7:                 Recalculate $\Delta_{si}$ and ascend/descend it to maintain the ordered
                   list.
8:     until Total number of matches generated exceeds $n_{\text{match}}$.
```

4.4.4 Numerical Results

Based on the simulation model in Chapter 3, we implement Algorithm 8 with pseudo-gradient generated from Heuristic 9 on the bike-sharing example. In Algorithm 8, each trial solution is evaluated with $m = 30$ replications (days) of the simulation, and the final solution is evaluated independently with $m' = 50$ replications. The maximum number of consecutive failures before stopping is set to be $n_{\text{maxFails}} = 30$. In Heuristic 9, the local randomization radius d_R is set to be 30 stations. Each trial solution is initially generated with a "step size" of $n_{\text{match}} = 32$, and is reduced by half (until reaching 1) every time the stopping criteria in line 14 is met, as mentioned in the previous section.

We compare the pseudo-gradient search in this chapter with Heuristic 7 from Chapter 3 in the problem of optimizing the allocation of both bikes and docks over the entire day. Both search methods are started from the same solutions of equal allocation, the fluid model, and CTMC, and the trial solutions are evaluated using the same random number stream. Figure 4.2 compares the two methods by each starting solution and Figure 4.1 combines all starting solutions into one plot. The corresponding starting and ending objective values are given in Table 4.1 below.

	Starting Objective	Heuristic 7 Ending Objective	Pseudo-gradient Search Ending Objective
Equal Allocation	12249 ± 89	8915 ± 47 (-27%)	7071 ± 43 (-42%), 6636 ± 43 (-46%) after restart
CTMC	6937 ± 122	6681 ± 43 (-3%)	6679 ± 43 (-4%)
Fluid	7063 ± 89	6865 ± 44 (-3%)	6715 ± 43 (-5%)

Table 4.1: Comparison of the starting and ending objective values in 95% confidence intervals for Heuristic 7 and pseudo-gradient search started from the solutions of equal allocation, CTMC, and fluid model.

When the search started from the equal allocation solution terminates for the first time, we observe a gap between the ending objective value and that of the other two starting solutions. To check if the search is trapped in a local minimum, we restart the pseudo-gradient search by resetting n_{match} to the initial value of 32, as mentioned in Section 4.2. After the restart, the search is able to make further progress and close the gap between equal allocation and other starting solutions. We have also tried this trick on the search with the other two starting solutions, but it cannot further improve the objective value by much (results negligible and hence omitted here). In fact, Table 4.1 shows that the 95% confidence intervals for the ending objective values of three solutions overlap each other, indicating the potential value of the global optimum.

Compared to Heuristic 7 from the last chapter, the pseudo-gradient search is able to make much faster progress. When started from the "bad" solution of equal-allocation, the search takes around 3000 simulated days (around 1.5 hours) to reach a good solution (the first "basin" in Figure 4.2), and only around 1000 simulated days (30 minutes) when started from the better solutions of CTMC and fluid model. Thus the method is well within the budget for planning the daily re-balancing of a large-scale bike sharing system like Citibike. Further, it has the potential to solve problems even larger than the current system with 466 stations (932 decision variables), and may reduce the need for more sophisticated methods like CTMC.

The code and input data is available in [41], and all the experiments are run with Python 2.7 on a desktop with 4-core Intel Core i7-3770 CPU 3.40 GHz processor, 16G memory, and Ubuntu 14.04 OS.

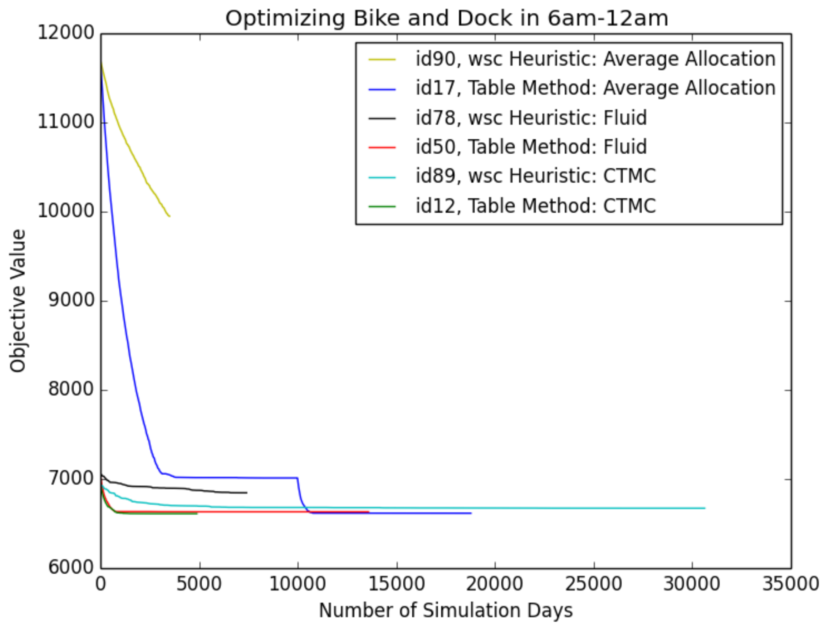


Figure 4.1: Combination of all starting solutions into one plot.

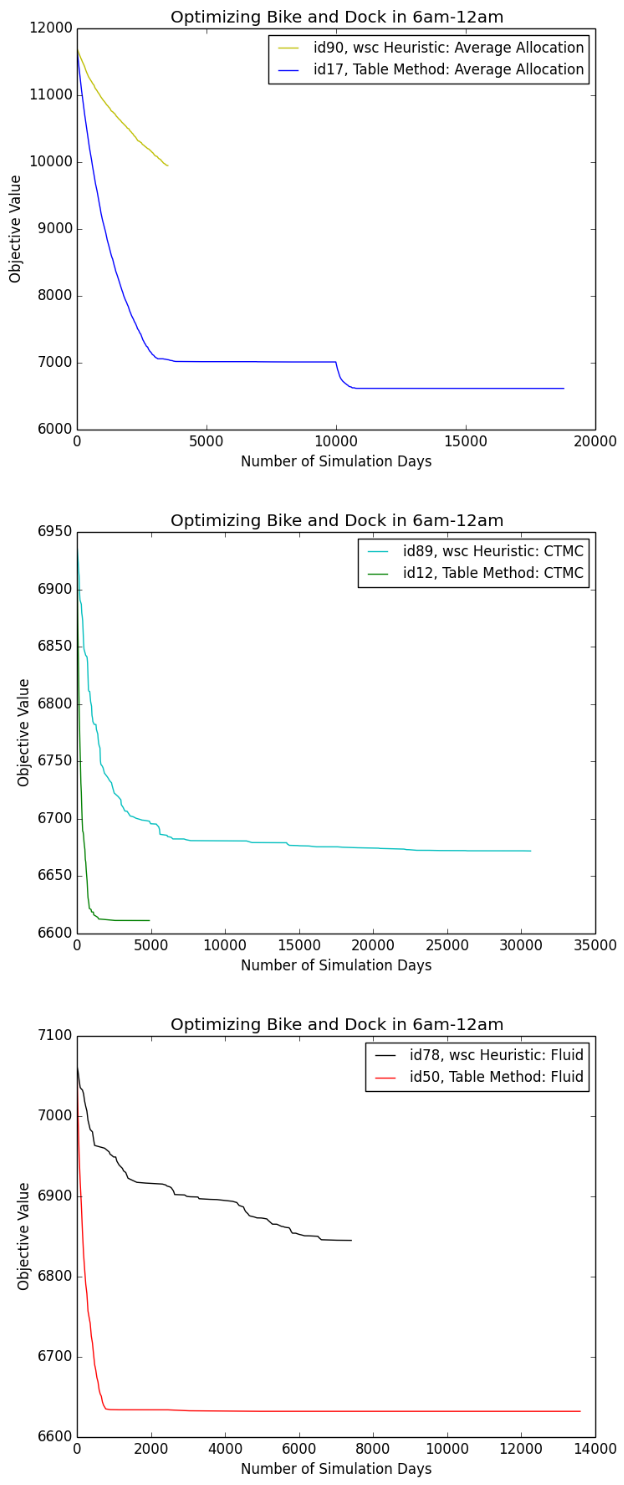


Figure 4.2: Comparison of objective value vs. number of simulated days between Heuristic 7 ("wsc Heuristic") and pseudo-gradient search ("Table Method"). The three plots are with starting solutions of equal allocation, CTMC, and fluid model, respectively.

4.5 Multi-period Multi-skill Call Center

4.5.1 Introduction

In this section, we consider a multi-skill call center, where calls of different types arrive randomly into the system during the day. Each type of calls require certain skill to be handled, and the agents working in the call center are grouped by their skill sets for training and hiring purposes. When a call arrives, it is assigned to an agent with the corresponding skill, and the agent is then occupied for the random duration of the call. If there are multiple feasible agents available, the call is assigned according to a routing rule determined by the preferences of the agents. If there is no agent with matching skills available, the caller will wait for a random patience time before abandoning the system. The quality of service at a call center is measured by the service level (SL), defined as the (expected) fraction of calls that are picked up within a certain time constraint. The SL constraint can be overall, by call type, and by period in the day. Agents work on a predetermined set of shift schedules with different starting times, durations, and breaks. The cost associated with a particular agent is given based on his/her skill set and the length of the shift. The goal is to find the optimal number of agents to schedule in each skill group and shift, so that the total cost is minimized without violating the service level requirement.

Despite that the cost function in the objective is deterministic and usually piecewise linear, a multi-skill call center problem is hard for many reasons. First, there is no closed-form formula or quick numerical algorithm for calculating the service level in the multi-skill setting. The only way to accurately estimate SL is by stochastic simulation ([5]). This makes the scheduling of a multi-skill call center a simulation optimization problem with integer variables and stochastic constraints. Furthermore, SL as a function of the number of agents is not concave (see, e.g. [3]), making the simulation optimization problem non-convex, so that no algorithm can guarantee a global optimum ([44]). Finally, each period during the day can be covered by many shifts, and each call type can be handled by many agent groups that share the same required skill set. This has created heavy dependency between different coordinates of the decision variable, resulting in many near-optimal solutions.

For multi-skill call centers, two types of problems have been defined in the literature. First, a *staffing* problem divides the day into small periods, and the goal is to determine the number of agents of each skill group to hire for each period. Since the problem is solved independently in each period, the staffing problem can also be regarded as a single-period problem. For staffing problem with a single overall SL constraint, [59] uses a local search with approximate SL to solve a Lagrangian formulation of the problem. For SL constraints by call type and period, [6] uses a local search with SL approximated from a loss-delay function. [12] applies a cutting-plane method with SL estimated by simulation. Since the problem is non-convex, the cutting-plane method may generate cuts that eliminate feasible solutions, for which [12] provides heuristics to deal with the difficulty. The second type is the multi-period *scheduling* problem, where agents working on an admissible set of shifts, and the goal is to find the optimal number of agents assigned to each skill group and shift. A scheduling problem is harder to solve than a staffing problem because of the larger problem dimension and the dependency between overlapping shifts. Thus [10] decomposes the scheduling problem into two steps by first solving for the optimal staffing level for each period in the day. Then the staffing solution is converted to a feasible schedule using the predetermined shifts.

It is found that such two-step approach usually overestimates the number of agents needed, so [5] chooses to optimize the staffing and the scheduling simultaneously by extending the cutting-plane method of [12] to the multi-period setting.

Different from all the approaches mentioned above, the simulation-based optimization method provided in the section uses pseudo-gradient to drive local search steps. For simplicity and ease of computation, we only include the overall SL constraint, and provide a Lagrangian formulation with a three-step optimization framework in Section 4.5.2. However, the essence of our method is not constrained by the Lagrangian formulation - we can foresee the pseudo-gradient introduced in Section 4.5.4 to be used for generating cuts in a cutting plane method like [12]. The numerical experiments in Section 4.5.5 shows that our method is able to output feasible and reasonably optimal solutions. For the staffing problem, the outputted solutions of the pseudo-gradient search are close to the experiment results in [59] for their comparison with [12]. For the scheduling problem, we are not able to compare the method directly with anything in the literature due to the difference in problem settings and input data.

4.5.2 Problem Statement and Lagrangian Formulation

For the scheduling problem, assume there are K types of calls, I groups of agents, and a total of Q different shifts. The cost matrix matrix $C = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})$ contains the cost $c_{i,q}$ of hiring an agent of group i and shift q. The goal is to find the optimal number of agents $x_{i,q}$ to hire in each group i and shift q, so that the total cost $\sum_{i=1,\ldots I} \sum_{q=1,\ldots,Q} c_{i,q} x_{i,q}$ is minimized, and the overall service level, defined as the expected fraction of calls picked up within 20 seconds, is bounded below by α. Define the service level $g : \mathbb{R}^{I \times Q} \to [0,1]$

as a function of the schedule $X = (x_{1,1}, \ldots, x_{1,Q}, \ldots, x_{I,1}, \ldots, x_{I,Q})$. Then the scheduling problem can be formulated as (4.8).

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & p(X) = \sum_{q=1,\ldots,Q} c_{i,q} x_{i,q} \\ \text{subject to} \quad & g(X) \geq \alpha \\ & X \geq 0 \text{ and integer.} \end{aligned} \tag{4.8}$$

For the staffing problem, We simply remove the subscript q from C and X, so that they become vectors of length I.

The service level function $g(\cdot)$ is estimated using a discrete-event simulation model with the following parameters. The day is divided into J periods, and in each period $j \in \{1, \ldots, J\}$, calls of type k arrive into the system according to a stationary Poisson process with rate λ_{jk}. The service time of the call handled by agents of group i is exponentially distributed with mean μ_{ik}. The routing logic R defines the priority R_{ik} of agents of group i for calls of type k. Lower values of R_{ik} represent higher preferences, and ∞ means agents in group i cannot handle calls of type k. If there is no available agent who can pick up the call, the caller will wait in the system for an exponentially distributed patience time with mean ρ.

To use Algorithm 8, we convert problem (4.8) into an unconstrained one using Lagrangian transformation. We assign the Lagrangian multiplier $\beta \geq 0$ to the constraint $g(X) \geq \alpha$. The Lagrangian form of the problem (4.8) is

$$\max_{\beta \geq 0} \min_{X \in \mathbb{N}^{I \times Q}} p(X) + \beta(\alpha - g(X)), \tag{4.9}$$

where $\mathbb{N} = \{0, 1, 2, \ldots\}$. By weak duality (see, e.g. [11]), the optimal value of this Lagrangian problem serves as a lower bound to that of the original problem

(4.8). If we ignore the integrality of X and assume that $g(\cdot)$ is concave, the original problem becomes convex, and the bound is tight as a result of strong duality.

However, the service level function $g(\cdot)$ is typically not concave even if X is continuous. In fact, it is convex in each coordinate of X when the coordinates are small, and concave for larger coordinates ([12]). The intuition is that when the number of agents in a certain group is zero, adding just one or two may not improve the service level by much. The service level starts to increase at a faster rate when the number of agents is large enough. Eventually, when there are many agents so that the service level is close to one, adding more agents would be unnecessary. In order to solve this difficulty, we define $\sum X = \sum_{i=1,\dots I} \sum_{q=1,\dots,Q} x_{i,q}$ and decomposes the inner layer of (4.9) into two steps:

$$\max_{\beta \geq 0} \min_{n \in \mathbb{N}} \min_{\substack{\sum X = n, \\ X \in \mathbb{N}^{I \times Q}}} p(X) + \beta(\alpha - g(X)). \tag{4.10}$$

The idea is that by constraining $\sum X = n$ for large enough n in the inner problem, we try to limit the local search of X inside the concave region of $g(\cdot)$, so that we have a "convex" problem with "strong duality" (not exactly because X is integer-valued). In result, the optimal value of the Lagrangian problem would be closer to that of the original problem, and we can switch the order of optimizations in (4.10).

For ease of notation, we define

$$f_n(\beta, X) = p(X) + \beta(\alpha - g(X)).$$

Then problem (4.10) can be decomposed to the following three subproblems.

$$f^* = \min_{n \in \mathbb{N}} f_n^*, \tag{I}$$

$$f_n^* = \max_{\beta \geq 0} f_n(\beta), \tag{II}$$

$$f_n(\beta) = \min_{\substack{\sum X = n, \\ X \in \mathbb{N}^{I \times Q}}} f_n(\beta, X). \tag{III}$$

4.5.3 Search Framework for the Lagrangian Problem

In this section, we provide a 3-step search framework used for solving the decomposed Lagrangian problem (I), (II), and (III) of multi-skill call center scheduling/staffing problem. The golden section search for (I) and the bisection search for (II) are very similar to the procedure in [59], whereas our local search for (III) is simulation-based and driven by pseudo-gradient.

For the out-most layer of optimization problem (I), we use the following golden section search algorithm. We obtain the lower and upper bound by an approximation based on the Erlang-C formula. The lower bound is calculated by treating the system as a single-skill call center with one call type and one agent group, and we take the period with the largest arrival rate and use the smallest service rate among all agents. The upper bound is calculated by having a separate call center for each call type, and summing up the number of slowest agents needed for each call center. Taking into account the effect of shift schedules, we scale the upper bound further by the minimum number of shifts needed to "cover" the business day.

Algorithm 10 Golden search for (I).

```
Input: Lower and upper bounds $n_L$ and $n_U$.
 1: Initialize $f^* = \infty$.
 2: while $n_U - n_L > 0$ do
 3:     Set $n_\lambda = \lfloor \gamma n_L + (1-\gamma) n_U \rfloor$ and $n_\rho = \lceil (1-\gamma) n_L + \gamma n_U \rceil$, where $\gamma = 2/(1+\sqrt{5})$ is the golden ratio.
 4:     Call Algorithm 11 with input $n_\lambda$. Record the outputs as $X_{n_\lambda}$ and $f^*_{n_\lambda}$.
 5:     Call Algorithm 11 with input $n_\rho$. Record the outputs as $X_{n_\rho}$ and $f^*_{n_\rho}$.
 6:     if $f^*_{n_\lambda} \leq f^*_{n_\rho}$ then
 7:         Set $n_U = n_\rho$ and reuse $f_{n_\lambda}$ as $f_{n_\rho}$ in the next iteration.
 8:         If $f^*_{n_\lambda} < f^*$, record $f^* = f^*_{n_\lambda}$ and $X^* = X_{n_\lambda}$.
 9:     else
10:         Set $n_L = n_\lambda$ and reuse $f^*_{n_\rho}$ as $f^*_{n_\lambda}$ in the next iteration.
11:         If $f^*_{n_\rho} < f^*$, record $f^* = f^*_{n_\rho}$ and $X^* = X_{n_\rho}$.
Output: $X^*$.
```

The second layer of optimization uses a bisection search (Algorithm 11) to find f^*_n in (II) while maintaining the feasibility of the corresponding X.

Algorithm 11 Bisection search for (II).

```
Input: Total number of agents $n$.
 1: Initialize lower bound $\beta_L = 0$ and upper bound $\beta_U = 1000$. Set $f^*_n = -\infty$.
 2: while $\beta_U - \beta_L > 0.01$ do
 3:     Set $\beta = \beta_L + (\beta_U - \beta_L)/z$.
 4:     Call Algorithm 8 with $f = f_n(\beta)$. Record the outputted solution as $X_n(\beta)$, and store the corresponding objective and estimated service level as $f_n(\beta)$ and $\hat{g}(X_n(\beta))$.
 5:     if  then
 6:         Set $f^*_n = f_n(\beta)$ and $X^*_n = X_n(\beta)$.
 7:     if $\hat{g}(X_n(\beta)) < \alpha$ then
 8:         Set $\beta_L = \beta$.
 9:     else
10:         Set $\beta_U = \beta$.
Output: $X^*_n$ and $f^*_n$.
```

Note that in each iteration of the bisection search step, we update the bounds β_L and β_U so that $X_n(\beta)$ remains feasible for the values of β in between. With such modification, a bisection can be used to optimize (II) due to the following observation that the objective is monotone in the region where $X_n(\beta)$ is feasible.

Observation 1. *Suppose there exists set Θ such that for every $\beta \in \Theta$, $\beta \geq 0$ and $X_n(\beta) := \arg\min_{\substack{\sum X = n, \\ X \in \mathbb{N}^{I \times Q}}} f_n(\beta, X)$ satisfies $g(X_n(\beta)) \geq \alpha$. Then $f_n(\cdot)$ is monotone in Θ.*

Proof. Choose $\beta_1, \beta_2 \in \Theta$ such that $\beta_1 > \beta_2$. Then by optimality of $X_n(\beta_2)$, we have

$$\begin{aligned} f_n(\beta_2) &= f_n(\beta_2, X_n(\beta_2)) \\ &= p(X_n(\beta_2)) + \beta_2(\alpha - g(X_n(\beta_2)) \\ &\leq p(X_n(\beta_1)) + \beta_2(\alpha - g(X_n(\beta_1))) \\ &< p(X_n(\beta_1)) + \beta_1(\alpha - g(X_n(\beta_1))) \\ &= f_n(\beta_1, X_n(\beta_1)) = f_n(\beta_1). \end{aligned}$$

□

Algorithms 10 and 11 act as a wrapper outside Algorithm 8 that uses a pseudo-gradient search for the final step (III) of the optimization. Each iteration of Algorithm 10 requires two calls of Algorithm 11, and each iteration of Algorithm bisection 11 requires one call of Algorithm 8, in which the trial solution is evaluated using simulation. Thus, to reduce the number of simulation runs, when Algorithm 10 calls the bisection with input n, we first run an initial local search with $\beta = \beta_U = 1000$ to obtain an upper bound on the service level. If this upper bound is below α, we set $n_L = n$ and skip the rest of the golden section iteration. Also, we take $z = \max\{2, \log_2(\beta_U - \beta_L)\}$ so that the size of $\beta_U - \beta_L$ reduces on a logarithmic scale.

4.5.4 Generating Trial Solution Using Pseudo-gradient

Throughout this section, we use f as a shorthand of $f_n(\beta)$ in (III), corresponding to the objective function $f(\cdot)$ in Algorithm 8.

The only remaining part is to define a pseudo-gradient that drives Algorithm 8 to solve (III). A well-defined pseudo-gradient for the multi-skill call center scheduling problem would approximate the change in the Lagrangian objective value in (III) with regard to a small change in the schedule X. Since the change can be positive or negative, we define the forward pseudo-gradient $\widehat{\nabla} f^+(X)$ with (i,q)-th component being the approximate change in the objective $f(X') - f(X)$ with $X'_{i,q} = X_{i,q} + 1$, keeping all other coordinates the same. Similarly, we define the backward pseudo-gradient $\widehat{\nabla} f^-(X)$ as the approximate change in the objective after removing an agent from group i and shift q, while keeping all other groups and shifts the same.

To calculate the forward pseudo-gradient $\widehat{\nabla} f^+(X)$, We use the list of *lateCalls* recorded from the simulation run that evaluates X. *lateCalls* contains the record of two kinds of calls. The first type is the calls that run out of patience before ever picked up, in which case *lateCalls* contains their arrival time, call type, and patience time. The second type is the calls that are picked up late ($>$ 20 seconds since arrival), and in addition to the same information for the first type, *lateCalls* also contains the group and shift number of the corresponding agent and the service time generated. Then the (i,q)-th component of $\widehat{\nabla} f^+(X)$ is calculated by "simulating" the lifetime of an additional agent added to group i and shift q. The procedure of generating $\widehat{\nabla} f^+(X)$ is described in the pseudo-code below, where N_{calls} is the total number of calls that entered the system in the simulation.

Procedure 4.1: The procedure to generate the forward pseudo-gradient $\widehat{\nabla} f^{+}(X)$.

```
Set time = beginning of shift $q$.
Set $\widehat{\nabla} f^{+}(X)_{i,q} = c_{i,q}$ for all $i = 1, \ldots, I$ and $q = 1, \ldots, Q$.
**while** time < end of shift $q$ **do**
    **for** every call in *lateCalls* **do**
        **if** the call can be picked up by agent of group $i$ **then**
            Increment $\widehat{\nabla} f^{+}(X)_{i,q}$ by $-\beta/N_{\text{calls}}$.
            Increment time by the service time (from *lateCalls* or otherwise generated).
```

The backward pseudo-gradient $\widehat{\nabla} f^{-}(X)$ is generated from the list of *lastCalls*, for which the (i, q)-th component is the number of calls handled by the "last" agent in group i and shift q. An agent is labeled "last" if he/she is the remaining one when number of available agents in the same group and shift drops down to 1 for the first time during the day. The intuition is that if this agent is removed from the system, the calls picked up by him/her will now be answered late or dropped, and the service level will suffer. Thus the (i, q)-th component of $\widehat{\nabla} f^{-}(X)$ is calculated as

$$\widehat{\nabla} f^{-}(X)_{i,q} = -c_{i,q} + \beta(\textit{lastCalls}_{i,q}/N_{\text{calls}}),$$

if $X_{i,q} >= 1$; otherwise set $\widehat{\nabla} f^{-}(X)_{i,q} = \infty$.

However, a caveat in this logic is that in a multi-skill setting, one type of call can be picked up by several groups of agents. Thus the removal of the last agent from one group may not necessarily harm the service level if the calls can be picked up by available agents in other groups. Therefore, our backward pseudo-gradient $\widehat{\nabla} f^{-}(X)$ overestimates the true effect of agent removals and may result in overstaffing. Nevertheless, the substitution effect is too complicated to capture in the simulation unless we store a complete trace of the run, which requires significant memory space and may slow down the simulation.

Finally, with $\widehat{\nabla} f^{+}(X)$ and $\widehat{\nabla} f^{-}(X)$, the trial solution X' is generated by

Heuristic 12. Since the total number of agents in the system needs to remain at n, we generate X' from the current solution X by moving agents between pairs of coordinates. Similar to Heuristic 9 for the bike-sharing example, we use a decreasing n_{match} to balance between the exploration of the search space and the exploitation of the neighborhood around the last best solution X. Apart from the decrease in size, the choice of n_{match} used to generate the new trial solution X' from X should also depend on the available room for improvement, indicated by the number of negative components in $\widehat{\nabla} f^{+}(X)$ and $\widehat{\nabla} f^{-}(X)$. For example, if there is only one pair of $(i1, q1)$ and $(i2, q2)$ with $\widehat{\nabla} f^{+}(X)_{i1,q1} + \widehat{\nabla} f^{-}(X)_{i2,q2} < 0$, the search radius n_{match} can only be as large as one.

Heuristic 12 Generation trial solution X' for the multi-skill call center example.

```
1: procedure Generate Trial Solution
Input: $\widehat{\nabla} f^{+}(X)$ and $\widehat{\nabla} f^{-}(X)$. Local randomization parameter $d_R$. Number of
    matches $n_{\text{match}}$ to generate.
2:     Initialize $X' = X'$.
3:     Sort $\widehat{\nabla} f^{+}(X)$ and $\widehat{\nabla} f^{-}(X)$ from smallest to largest.
4:     repeat
5:         Choose group $i1$ and shift $q1$ from the top $d_R$ largest components of
    $\widehat{\nabla} f^{+}(X)$.
6:         Choose group $i2$ and shift $q2$ from the top $d_R$ largest components of
    $\widehat{\nabla} f^{-}(X)$, such that $\widehat{\nabla} f^{+}(X)_{i1,q1} + \widehat{\nabla} f^{-}(X)_{i2,q2} < 0$.
7:         Increment $X'_{i1,q1}$ by 1 and decrement $X'_{i2,q2}$ by 1.
8:     until No pairs of $(i1, q1)$ and $(i2, q2)$ can be found with $\widehat{\nabla} f^{+}(X)_{i1,q1}$ +
    $\widehat{\nabla} f^{-}(X)_{i2,q2} < 0$ or the total number of pairs generated exceeds $n_{\text{match}}$.
```

4.5.5 Numerical Results

The pseudo-gradient search in Algorithms 10, 11, and Heuristic 12 is implemented in Matlab based on the simulation code for "Multi-period multi-skill call center" on `SimOpt.org` (see [36]). The code and data is available in the online repository

[39]. All test cases are run on a desktop with a 4-core Intel Core i7-3770 3.40 GHz processor with 16G memory, running Matlab R2013a on 64-bit Windows 7.

Staffing Problem

We take the three-skill call center problem from [59] as an example for a staffing problem. The call center has $K = 3$ types of calls and $I = 6$ agent groups. The routing matrix is

$$R = \begin{bmatrix} 1 & \infty & \infty \\ \infty & 1 & \infty \\ 2 & 2 & \infty \\ 3 & \infty & 1 \\ \infty & 3 & 2 \\ 4 & 4 & 3 \end{bmatrix},$$

where a call of type k will be assigned to the agent group i with the smallest R_{ik} first (e.g. call type 1 will be assigned in the order of $1 \rightarrow 3 \rightarrow 4 \rightarrow 6$). $R_{ik} = \infty$ indicates that the agents of group i cannot pick up calls of type k. The service rates μ_{ik} of agent group i for call type k are summarized in Table 4.2, and the arrival rates λ_k of call type k and the cost c_i of hiring an agent in group i are displayed in Table 4.3. All calls have infinite patience time and never abandon the system.

We run the simulation for 160 hours to evaluate each trial solution. In Algorithm 8, we stop before the consecutive number of fails reaches $n_{\text{maxFails}} = 5$ or there does not exist a pair of $(i1, q1)$ and $(i2, q2)$ with $\widehat{\nabla} f^{+}(X)_{i1,q1} + \widehat{\nabla} f^{-}(X)_{i2,q2} < 0$. In Heuristic 12, the starting n_{match} is set to be 32. Each time the stopping criteria 14 is met, n_{match} is reduced to the minimum between half of its previous value and the total possible pairs of $(i1, q1)$ and $(i2, q2)$ with $\widehat{\nabla} f^{+}(X)_{i1,q1} + \widehat{\nabla} f^{-}(X)_{i2,q2} < 0$.

We consider the following 5 cases from [59]. Based on Case 1, Case 2 and 3 have the service times associated with the call types 1 and 2 increased. Case 4 has arrival rate and service time of call type 1 both increased. Case 5 has arrival rate of type 1 and the service time of type 2 increased.

ID	μ_{11}	μ_{22}	μ_{31}	μ_{32}	μ_{41}	μ_{43}	μ_{52}	μ_{53}	μ_{61}	μ_{62}	μ_{63}
1	0.20	0.18	0.19	0.17	0.19	0.16	0.17	0.16	0.18	0.16	0.15
2	0.15	0.18	0.14	0.17	0.14	0.16	0.17	0.16	0.13	0.16	0.15
3	0.20	0.15	0.19	0.14	0.19	0.16	0.14	0.16	0.18	0.13	0.15
4	0.15	0.18	0.14	0.17	0.14	0.16	0.17	0.16	0.13	0.16	0.15
5	0.20	0.15	0.19	0.14	0.19	0.16	0.14	0.16	0.18	0.13	0.15

Table 4.2: Service rates μ_{ik} of agent group i for call type k.

ID	λ_1	λ_2	λ_3	c_1	c_2	c_3	c_4	c_5	c_6
1	1.0	1.5	2.0	1.0	1.0	1.1	1.1	1.1	1.2
2	1.0	1.5	2.0	1.0	1.0	1.1	1.1	1.1	1.2
3	1.0	1.5	2.0	1.0	1.0	1.1	1.1	1.1	1.2
4	2.0	1.5	2.0	1.0	1.0	1.1	1.1	1.1	1.2
5	2.0	1.5	2.0	1.0	1.0	1.1	1.1	1.1	1.2

Table 4.3: Arrival rates λ_k for call type k and cost c_i for agent group i.

Table 4.4 contains a comparison between the pseudo-gradient search, the local search using approximate SL from [59], and the cutting-plane method from [12]. The ending service levels for the pseudo-gradient search are all within ± 0.02 of the target level 0.8. The results for the other two methods are taken from [59]. Since we do not have the code for the search method or the simulation model from [59], the comparison of the optimal objective values in Table 4.4 is not exact. For the comparisons of the number of simulations and β steps, both our pseudo-gradient search and the local approximate search from [59] are based on Lagrangian relaxation, but [59] uses blocking probability to estimate the service level function in their local search, and only runs simulation when evaluating $f_n(\beta)$ for each β step in Algorithm 11. Our method uses simulation in each local search step in Algorithm 8, so we expect our number of simulations to be much larger, while the number of β steps to be comparable. Comparing to the cutting-plane method, the number of simulation runs used in the pseudo-gradient search is still much larger. Our intuition is that the cutting plane method requires solving integer programs, which is efficient for problems with small solution space like this, but might require more time when the problem dimension is larger.

ID	Solution	Cost	# Simulation runs	# β steps
1	Pseudo-gradient search	35.8	400	32
1	Local approximate search	34.6	31	31
1	Cutting planes	34.8	50	NA
2	Pseudo-gradient search	38.1	280	20
2	Local approximate search	36.8	18	18
2	Cutting planes	37.1	56	NA
3	Pseudo-gradient search	37.8	240	20
3	Local approximate search	36.7	19	19
3	Cutting planes	37.6	32	NA
4	Pseudo-gradient search	45.0	450	36
4	Local approximate search	43.8	34	34
4	Cutting planes	44.8	43	NA
5	Pseudo-gradient search	43.2	290	23
5	Local approximate search	42.0	22	22
5	Cutting planes	42.5	55	NA

Table 4.4: Results of the three-skill staffing problem over three different methods: our pseudo-gradient search, the local search based on SL approximation from [59], and their comparison to the cutting-plane method from [12]. Note that our method is using a different simulation model with different seed, so the cost comparison is not exact.

We expect the local search driven by simulation to yield more optimal solutions than the local search driven by approximated SL, but in our experiment this is not the case. Apart from the difference in our simulation models (e.g. random seed), we have mentioned in Section 4.5.4 that our backward pseudo-gradient overestimates the negative effect of removing agents, which may result in over-staffing.

Scheduling Problem

In this subsection, we take the example of a small call center with 2 call types and 2 agent groups, and a large call center with 20 call types and 35 agent groups. All the data used in this section are from the case of "Multi-period Multi-skill Call Center" on `SimOpt.org` (see [34]), except that we adapt a 285-shift scheme from [5] instead of the original 74 shifts on on `SimOpt.org`. The examples are very similar to the small and 36-period large call center cases from [5], but the arrival rates and the routing policy we take from [34] differ, making the results in this section not directly comparable. In addition, [5] constrains service level by each period (and sometimes also by call type), while we only have the overall constraint, so the problems being solved are different. Finally, [5] runs the numerical experiments in Java with a CPU time budget, so without being able to compare the solution quality, it is difficult to claim which method can achieve a better solution more quickly.

In both scheduling problems, the 8am-10pm business day is divided into 36 periods of 15 minutes. We use a total of 285 shifts, each with a 30-minute lunch break and two 15-minute coffee breaks, spanning the business day from 8am to 10pm. The details on the shift schedules are available from Table 1 of [5], hence omitted here. The cost of hiring an agent of group i in shift q is calculated by

$$c_{iq} = (1 + (\eta_i)\varsigma)l_q/30,$$

where η_i is the number of call types that agents of group i can pick up, $\varsigma = 0.1$ is a constant, and l_q is the length of shift q in number of 15-minute periods. Calls of each type arrive arrive according to a stationary Poisson process with piece-wise constant rate in each period. The service time is exponentially distributed with a rate of 8 per minute regardless of the agent group and call type. Customers abandon the system after a patience time that has a mixture distribution. The patience time is exponentially distributed with mean of 10 minutes with probability 0.999, and zero with probability 0.001.

We evaluate each trial solution using $m = 30$ days of simulation. Each simulated day takes around 0.9 seconds for the small call center, and 3.2 seconds for the large call center, with the simulation model implemented in Matlab. The configurations of n_{maxFails} and n_{match} in Algorithm 8 and Heuristic 12 are similar to those in the staffing problem.

Scheduling Problem: Small Call Center

In the small call center with two call types and two agent groups, agents of group 1 can only pick up calls of type 1, while agents of group 2 can pick up both but prefers calls of type 2. The decision variable is in the dimension of 570 (2 agent groups $\times$ 285 shifts). The arrival rates for each call type are plotted in Figure 4.3 and the data is available from [34].

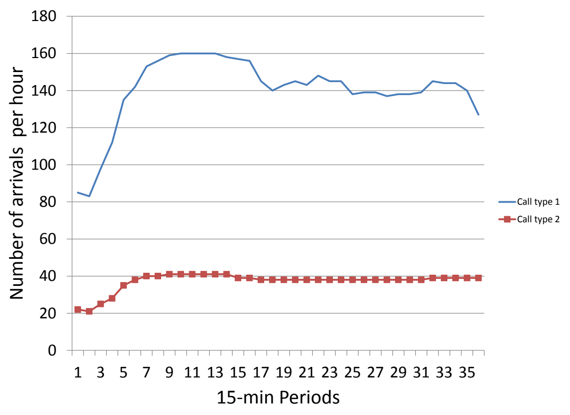


Figure 4.3: The arrival rates for the small call center.

Table 4.5 gives the mean and median cost and service level outputted by the final solution of Algorithm 8 over 5 independent trials of the small call center problem using different random seeds. We also give the average utilization of the agents in each group, and the average percent of abandonment for each call type, to gain insight on the optimality of the outputted solution. It seems that the search is able to output a solution with feasible service level most of the time (4 out of 5), and low call abandonment rate. The utilizations of both groups do not indicate over-staffing or under-staffing. Also, group 1 agents have higher utilization rate since calls of type 1 have higher arrival rates. The utilization of group 2 agents is not much lower because they are generalists that can also pick up type 1 calls. The average number of simulated days is 3320, indicating that there are about 110 trial solutions evaluated before the algorithm terminates.

Metric	Experiment output
mean cost	37.33
median cost	38.88
mean SL	0.81
min SL	0.78
mean # simulations	3320
mean CPU time (seconds)	3242
mean util. of group 1	0.75
mean util. of group 2	0.69
mean %abandonment of call type 1	0
mean %abandonment of call type 2	0.0015

Table 4.5: The results of pseudo-gradient search on the small call center with different random seeds.

Scheduling Problem: Large Call Center

In a large call center, there are 20 call types and 35 agent groups, giving a decision variable of $35 \times 285 = 9975$ dimensions. Figure 4.4 gives the arrival rate by call type in each of the 15-minute period in the day. The routing policy and agent groups are available from [34], hence omitted here.

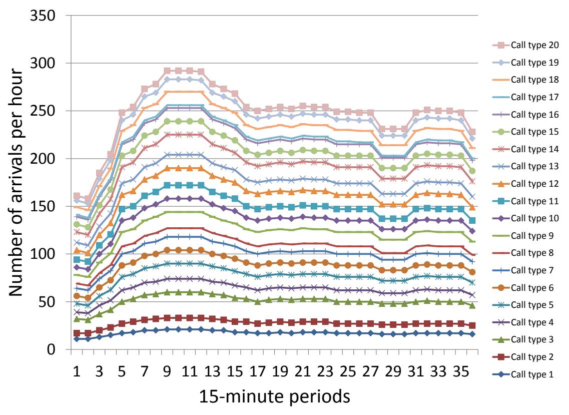


Figure 4.4: The arrival rates for the large call center.

Table 4.6 summarizes the results from 5 independent trials of the pseudo-gradient search, each with a different random seed.

Metric	Experiment output
mean cost	89.58
median cost	87.49
mean SL	0.80
min SL	0.78
mean # simulations	9,870
mean CPU time (seconds)	32,129

Table 4.6: The results of pseudo-gradient search on the large call center with different random seeds.

The search takes an average of around 9 hours to output the final solution. Although the input data for this experiment is different from that in [5], the solving time is smaller than their budget time of 10 hours, with each iteration of the simulation taking 3 times as long (due to the difference in the processing time of Matlab vs. Java). The outputted solution is feasible with an average server utilization of 0.55. The busiest agent groups have utilizations of around 0.79, and the most idle group has an utilization of 0.18 and specializes in the call type with the smallest arrival rates.

4.6 Remarks and Future Work

In this chapter, we propose the new concept of pseudo-gradient and use it to guide a search procedure for large-scale simulation optimization problems. Our pseudo-gradient search is implemented in the bike-sharing and multi-skill call center settings.

For the bike-sharing problem, the pseudo-gradient is calculated as the change in the number of failed trips with regard to the change by swapping bikes and/or docks between pairs of stations. The calculation is based on the assumption that the attempted sequence of bike pick ups and drop offs at each station remains the same with a small change in the bike or dock level. In the implementation, we find the pseudo-gradient search is able to make quick and effective progress and improve upon any starting solution. The method potentially eliminates the need for more sophisticated methods like CTMC.

For the call center problem, the pseudo-gradient is the total change in the hiring cost and the service level. To calculate the change in the service level, we use the lists of *lateCalls* and *lastCalls* to "simulate" what happens if we add or remove an agent from certain group and shift. *lateCalls* captures all the calls that are picked up late (or never picked up), which can be eliminated by hiring more agents. *lastCalls* records the calls picked up by the last agent in each shift and group, and those calls will become late if the agent is removed from the system. In our numerical experiment, compared to the cutting-plane methods, the pseudo-gradient search is slower in small problems and is comparable or faster in larger instances.

During the numerical experiments, we find that the pseudo-gradient search works better for the bike-sharing example because of two factors. The first one is the simplicity of how the system states are changed with regard to the future events. In the bike-sharing example we are able to record the list of bike pick ups and drop offs for all stations, and the effect of a change in the starting configuration of a station can be calculated relatively easily from that. However, for the call center example, the addition or removal of an agent changes the event list significantly, and its effect on the service level cannot be calculated in a straightforward way. The second factor is that the objective function in the bike sharing system can be easily decomposed by stations. This does not apply to the service level function in the call center example because of the complex interactions between agent groups and shifts. For example, our forward gradient and backward gradient calculated from the list of *lateCalls* and *lastCalls* cannot capture the substitution effect between agents of similar skill sets and overlapping shifts. Because of these two factors, the pseudo-gradient for the call center example does not approximate the true gradient as accurately as that for the bike-sharing example. This motivates future work in finding better pseudo-gradients in the subclass of simulation optimization problems with complex structure and large correlation between different coordinates of the decision variables.

APPENDIX A

APPENDIX TO CHAPTER 2

A.1 Characterizing Strictly Convex Vectors

Here we show that the interior of $\mathbb{C}$, denoted $\mathbb{C}^\circ$, is the set of all strictly convex vectors, where $\mathbb{C}$ is defined in Definition 2. We repeatedly exploit the fact that $\boldsymbol{g} \in \mathbb{C}$ if and only if the linear system (LS) is feasible. Throughout this section we take the points $\{\boldsymbol{x}_i : 1 \leq i \leq r\}$ as fixed. Denote the set of indices of the design points as $\mathcal{I} = \{1, 2, \ldots, r\}$. For each fixed $k \in \mathcal{I}$, consider the set $\mathcal{T}_k$ of real-valued coefficients that express $\boldsymbol{x}_k$ as a convex combination of the other points, i.e.,

$$\mathcal{T}_k = \{t \in \mathbb{R}^r : t_k = 0, t_i \in [0,1] \ i \neq k, \sum_i t_i = 1, \sum_{i=1}^r t_i \boldsymbol{x}_i = \boldsymbol{x}_k\}.$$

Lemma 1. *Given a function g, suppose that for each $k \in \mathcal{I}$, $\sum_{i=1}^r t_i g(\boldsymbol{x}_i) - g(\boldsymbol{x}_k) > 0$ for all $t \in \mathcal{T}_k$. Then there exists $\epsilon > 0$ such that $\sum_{i=1}^r t_i g(\boldsymbol{x}_i) - g(\boldsymbol{x}_k) \geq \epsilon > 0$ for all k and all $t \in \mathcal{T}_k$.*

Proof. Since there are only finitely many choices for $\boldsymbol{x}_k$, we just need to show that such a bound exists for each $k \in \mathcal{I}$. So fix $k \in \mathcal{I}$. If $\mathcal{T}_k$ is a finite set, then the required bound exists by taking the minimum over a finite number of positive terms. So suppose $\mathcal{T}_k$ contains an infinite number of terms, as arises for example when multiple $\boldsymbol{x}_i$ are linearly dependent. If a positive ϵ as stated does not exist, then there is a sequence $\{t^{(n)}, n \geq 1\}$ contained in $\mathcal{T}_k$ such that $D(n) := \sum_{i=1}^r t_i^{(n)} g(\boldsymbol{x}_i) - g(\boldsymbol{x}_k) \to 0$ as $n \to \infty$. The set $\mathcal{T}_k$ is compact, so by passing to a subsequence we can assume that $t^{(n)} \to t^* \in \mathcal{T}_k$ as $n \to \infty$. Also since $D(n) \to 0$, it follows by continuity that $D^* = \sum_{i=1}^r t_i^* g(\boldsymbol{x}_i) - g(\boldsymbol{x}_k) = \lim_{n\to\infty} D(n) = 0$, which contradicts the assumption of the lemma. $\square$

Proposition 2. *An r-dimensional vector $\boldsymbol{g} \in \mathbb{C}^\circ$ if and only if there exists a strictly convex function g whose values on $\boldsymbol{x}$ coincide with those of $\boldsymbol{g}$, i.e. $g(\boldsymbol{x}) = \boldsymbol{g}$ for $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_r)$.*

Proof. First, suppose there exists a strictly convex function g with $g(\boldsymbol{x}) = \boldsymbol{g}$. We want to show $\boldsymbol{g} \in \mathbb{C}^\circ$. Since g is strictly convex, the strict inequalities in the statement of Lemma 1 hold for all $k \in \mathfrak{I}$ and all $t \in \mathfrak{T}_k$. Lemma 1 then implies that there exists $\epsilon > 0$ such that

$$\sum_{i=1}^{r} t_i g(\boldsymbol{x}_i) \geq g(\boldsymbol{x}_k) + \epsilon \tag{A.1}$$

for all k and all $t \in \mathfrak{T}_k$. Consider the ball in $\mathbb{R}^r$ defined by $(\boldsymbol{y} \in \mathbb{R}^r : |\boldsymbol{y}_i| \leq \epsilon/2 \ \forall i)$, and the perturbed vector $\tilde{g} = \boldsymbol{g} + \boldsymbol{y}$ with ith component $g(\boldsymbol{x}_i) + \boldsymbol{y}_i$. Then for any $k \in \mathfrak{I}$ and any $t \in \mathfrak{T}_k$,

$$\begin{aligned}
\left(\sum_{i=1}^{r} t_i \tilde{g}(\boldsymbol{x}_i)\right) - \tilde{g}(\boldsymbol{x}_k) &= \left(\sum_{i=1}^{r} t_i (g(\boldsymbol{x}_i) + \boldsymbol{y}_i)\right) - (g(\boldsymbol{x}_k) + \boldsymbol{y}_k) \\
&= \sum_{i=1}^{r} t_i g(\boldsymbol{x}_i) - g(\boldsymbol{x}_k) + \left(\sum_{i=1}^{r} t_i \boldsymbol{y}_i\right) - \boldsymbol{y}_k \\
&\geq \epsilon - \max_{i=1}^{r} |\boldsymbol{y}_i| - |\boldsymbol{y}_k| \\
&\geq \epsilon - \epsilon/2 - \epsilon/2 = 0.
\end{aligned}$$

Thus all perturbed function values in the ball centered at $\boldsymbol{g}$ are convex, and it follows that $\boldsymbol{g} \in \mathbb{C}^\circ$.

Now suppose that $\boldsymbol{g} \in \mathbb{C}^\circ$. We want to show that there exists a strictly convex function, g^* say, that coincides with $\boldsymbol{g}$ on $\boldsymbol{x}$. Since $\boldsymbol{g} \in \mathbb{C}$ there exists a convex function g that coincides with $\boldsymbol{g}$ on $\boldsymbol{x}$. Let h be an arbitrary finite-valued strictly convex function, e.g., $h(x) = \|x\|_2^2$, and let $\boldsymbol{h}$ be the restriction of h to $\boldsymbol{x}$. For $\delta > 0$ sufficiently small, $\boldsymbol{g} - \delta \boldsymbol{h} \in \mathbb{C}$, since $g \in \mathbb{C}^\circ$ and $\delta \boldsymbol{h}$ is a finite vector with arbitrarily small norm for δ sufficiently small. Thus, there exists a convex function f that coincides with $\boldsymbol{g} - \delta \boldsymbol{h}$ on $\boldsymbol{x}$. But then $g^* = f + \delta h$ coincides with $\boldsymbol{g}$ on $\boldsymbol{x}$, and is the sum of a convex and strictly convex function so is strictly convex. $\square$

A.2 Alternatives to the Posterior Updates

When the covariance matrix Γ is known,the normal-normal conjugate prior (posterior) updates in (2.1) involve inverting the posterior covariance matrix Λ_n in every iteration. An alternative method is to use the Sherman-Morrison-Woodbury formula

$$\begin{aligned} \mu_n &= \mu_{n-1} + \Lambda_{n-1} s\Gamma^{-1}(I + \Lambda_{n-1} s\Gamma^{-1})^{-1}(\bar{y} - \mu_{n-1}) \\ \Lambda_n &= \Lambda_{n-1} - \Lambda_{n-1} s\Gamma^{-1}(I + \Lambda_{n-1} s\Gamma^{-1})^{-1}\Lambda_{n-1} \end{aligned} \tag{A.2}$$

In this version, only the inverse of $(I + \Lambda_{n-1} s\Gamma^{-1})$ needs to be updated in each iteration, and the inverse of Λ_{n-1} is not required. However, since Γ is full rank, this update is a full rank update on Λ_n, which means the calculation still requires $O(r^3)$ math operations.

Another way to avoid repeated matrix inversions is to update the precision matrix Λ_n^{-1} from iteration n to $n+1$ instead of the covariance matrix, and to use the Cholesky factorization when performing the updates. In each iteration, the update formula is

$$\begin{aligned} \Lambda_n^{-1} &= \Lambda_{n-1}^{-1} + s\Gamma^{-1} \\ L_n &= \text{Chol}(\Lambda_n^{-1}) \\ \mu_n &= \Lambda_n(\Lambda_{n-1}^{-1}\mu_{n-1} + s\Gamma^{-1}\bar{\boldsymbol{y}}) \\ &= L_n^T \backslash [L_n \backslash (\Lambda_{n-1}^{-1}\mu_{n-1} + s\Gamma^{-1}\bar{\boldsymbol{y}})]. \end{aligned} \tag{A.3}$$

Here $\text{Chol}(\Lambda_n^{-1})$ is the lower-triangular Cholesky factor of Λ_n^{-1}, which is unique when Γ_n^{-1} positive definite, and $x = A\backslash b$ means to solve the linear system $Ax = b$. In Algorithm 2, when samples with distribution $N(\mu_n, \Lambda_n)$ are required, we can generate them by computing $\mu_n + L_n^T\backslash\boldsymbol{Z}$, where $\boldsymbol{Z} \sim N(\mathbf{0}, I)$. This is based on the fact that $\text{Chol}(\Lambda_n) = L_n^{-T}$. We are not able to find a way to directly update the Cholesky factor, so the Cholesky decomposition step is required in each iteration, which means that this method also has $O(r^3)$ complexity at each iteration.

Although these proposals do not reduce the computational complexity of the update, they allow us to do fewer matrix inversions, which makes them numerically more robust.

BIBLIOGRAPHY

[1] Abrevaya, J. and Jiang, W. A nonparametric approach to measuring and testing curvature. *Journal of Business & Economic Statistics*, 23:1–19, 2005.

[2] Allon, G., Beenstock, M., Hackman, S., Passy, U., and Shapiro, A. Nonparametric estimation of concave production technologies by entropic methods. *Journal of Applied Econometrics*, 22(4):795–816, 2007.

[3] Atlason, J. *A Simulation-Based Cutting Plane Method for Optimization of Service Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 2004.

[4] Atlason, J., Epelman, M. A., and Henderson, S. G. Call center staffing with simulation and cutting plane methods. *Annals of Operations Research*, 127:333–358, 2004.

[5] Avramidis, A. N., Chan, W., Gendreau, M., LEcuyer, P., and Pisacane, O. Optimizing daily agent scheduling in a multiskill call center. *European Journal of Operational Research*, 200(3):822 – 832, 2010.

[6] Avramidis, A. N., Chan, W., and L'Ecuyer, P. Staffing multi-skill call centers via search methods and a performance approximation. *IIE Transactions*, 41(6):483–497, 2009.

[7] Baraud, Y., Huet, S., and Laurent, B. Testing convex hypotheses on the mean of a Gaussian vector. application to testing qualitative hypotheses on a regression function. *The Annals of Statistics*, 33(1):214–257, 2005.

[8] Bauschke, H. H. and Combettes, P. L. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[9] Bernardo, J. M. and Smith, A. F. M. *Bayesian Theory*, pages 240–376. John Wiley & Sons, Inc., 2008.

[10] Bhulai, S., Koole, G., and Pot, A. Simple methods for shift scheduling in multiskill call centers. *Manufacturing &amp; Service Operations Management*, 10(3):411–420, July 2008.

[11] Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, New York, 2004.

[12] Cezik, M. T. and L'Ecuyer, P. Staffing multiskill call centers via linear programming and simulation. *Management Science*, 54(2):310–323, 2008.

[13] Chemla, D., Meunier, F., and Calvo, R. W. Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120 – 146, 2013.

[14] Chen, X., Ankenman, B. E., and Nelson, B. L. The effects of common random numbers on stochastic kriging metamodels. *ACM TOMACS*, 22(2):Article 7, 2012.

[15] CitibikeNYC. Accessed Apr. 28, 2016, http://www.citibikenyc.com/.

[16] DeGroot, M. *Optimal Statistical Decisions*. McGraw-Hill, New York, NY, 1970.

[17] Diack, C. A. T. and Thomas-Agnan, C. A nonparametric test of the non-convexity of regression. *Nonparametric Statistics*, 9:335–362, 1998.

[18] Feng, M. and Staum, J. Green simulation designs for repeated experiments. In *Proceedings of the 2015 Winter Simulation Conference*, WSC '15, pages 403–413, Piscataway, NJ, USA, 2015. IEEE Press.

[19] Forma, I. A., Raviv, T., and Tzur, M. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. Submitted for publication, 2015.

[20] Fricker, C. and Gast, N. Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. Manuscript, 2014.

[21] Fricker, C., Gast, N., and Mohamed, H. Mean field analysis for inhomogeneous bike sharing systems. In *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 365–376, 2012.

[22] Fu, M. C. Gradient estimation. In Henderson, S. G. and Nelson, B. L., editors, *Simulation*, Handbooks in Operations Research and Management Science, pages 575–616. Elsevier, Amsterdam, 2006.

[23] Fu, M. C. *Handbook of Simulation Optimization*. International Series in Operations Research & Management Science. Springer New York, 2015.

[24] Fu, M. C. Stochastic gradient estimation. In *Handbook of Simulation Opti-*

mization, International Series in Operations Research & Management Science. Springer New York, 2015.

[25] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. *Bayesian Data Analysis*. Chapman & Hall/CRC Texts in Statistical Science. Chapman and Hall/CRC, 2nd edition, July 2003.

[26] Glasserman, P. *Gradient Estimation Via Perturbation Analysis*. Kluwer, The Netherlands, 1991.

[27] Glasserman, P. *Monte Carlo methods in financial engineering*. Springer, New York, 2004.

[28] Glynn, P. W. and Infanger, G. Simulation-based confidence bounds for two-stage stochastic programs. *Mathematical Programming*, 138(1-2):15–42, 2013.

[29] Glynn, P. W. and Whitt, W. The asymptotic efficiency of simulation estimators. *Oper. Res.*, 40(3):505–520, May 1992.

[30] Golub, G. H. and Van Loan, C. F. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. The Johns Hopkins University Press, 3rd edition, October 1996.

[31] Grant, M. and Boyd, S. Graph implementations for nonsmooth convex programs. In Blondel, V., Boyd, S., and Kimura, H., editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[32] Grant, M. and Boyd, S. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[33] Gurobi Optimization, I. Gurobi optimizer reference manual, 2016.

[34] Gutierrez, G. Simopt: Scheduling for multi-skill call center. `http://simopt.org/wiki/index.php?title=Multi-period_multi-skill_call_center` Accessed March 15, 2017.

[35] Hannah, L. A. and Dunson, D. B. Multivariate Convex Regression with Adaptive Partitioning. *ArXiv e-prints*, May 2011.

[36] Henderson, S. G. and Pasupathy, R. Simulation optimization library, 2011. http://www.simopt.org Accessed March 26, 2011.

[37] Henderson, S. G., O'Mahony, E., and Shmoys, D. B. (citi)bike sharing. Submitted for publication, 2015.

[38] Ho, Y. C. and Cao, X. R. *Perturbation Analysis of Discrete-Event Systems*. Kluwer Academic, Boston MA, 1991.

[39] Jian, N. Matlab code for multi-skill call center optimization. Created May. 16, 2016. https://github.com/njian/MPMultiSCC, 2017.

[40] Jian, N. Matlab package for convexity detection. Created Feb. 17, 2017. https://github.com/njian/convexity, 2017.

[41] Jian, N. Python code for citibike optimization. Created Dec. 22, 2016. https://github.com/njian/bikesharSimOpt, 2017.

[42] Jian, N., Henderson, S. G., and Hunter, S. R. Sequential detection of convexity from noisy function evaluations. In Tolk, A., Diallo, S. D., Ryzhov, I. O., Yilmaz, L., Buckley, S., and Miller, J. A., editors, *Proceedings of the 2014 Winter Simulation Conference*, pages 3892–3903, Piscataway, NJ, 2014. Institute of Electrical and Electronics Engineers, Inc.

[43] Jian, N., Freund, D., Wiberg, H. M., and Henderson, S. G. Simulation optimization for a large-scale bike-sharing system. In *Proceedings of the 2016 Winter Simulation Conference*, WSC '16, pages 602–613, Piscataway, NJ, USA, 2016. IEEE Press.

[44] Jian, N. and Henderson, S. G. An introduction to simulation optimization. In Yilmaz, L., Chan, W. K. V., Roeder, T. M. K., Macal, C., and Rosetti, M., editors, *Proceedings of the 2015 Winter Simulation Conference*, pages 1780–1794, Piscataway NJ, 2015. IEEE.

[45] Jian, N. and Henderson, S. G. Convexity detection in noisy function evaluations. *arXiv*, 1703.04185, 2017.

[46] Judge, G. G. and Takayama, T. Inequality restrictions in regression analysis. *Journal of the American Statistical Association*, 61(313):pp. 166–181, 1966.

[47] Juditsky, A. and Nemirovski, A. On nonparametric tests of positivity/monotonicity/convexity. *The Annals of Statistics*, 30(2):498–527, 2002.

[48] Kim, S., Pasupathy, R., and Henderson, S. G. A guide to SAA. In Fu, M., editor, *Encyclopedia of Operations Research and Management Science*, Hillier and Lieberman OR Series. Elsevier, 2014.

[49] Lau, L. J. Testing and imposing monoticity, convexity, and quasi-convexity constraints. *Electronic Journal of Statistics*, 1:409–453, 1978.

[50] Lim, E. and Glynn, P. W. Consistency of multidimensional convex regression. *Operations Research*, 60(1):196–208, 2012.

[51] Mathworks. Documentation for linprog. Accessed Feb. 17, 2017. https://www.mathworks.com/help/optim/ug/linprog.html, 2016.

[52] Meyer, M. C. Constrained penalized splines. *Canadian Journal of Statistics*, 40(1):190–206, 2012.

[53] Murty, K. G. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann Verlag, Berlin, 1988.

[54] Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic, 2004.

[55] O'Mahony, E. *Smarter Tools for (Citi)bike Sharing*. PhD thesis, Cornell University, Ithaca NY, 2015.

[56] O'Mahony, E. and Shmoys, D. B. Data analysis and optimization for (citi)bike sharing. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 687–694. AAAI, 2015.

[57] Pasupathy, R. and Henderson, S. G. Simopt : A library of simulation optimization problems. In Jain, S., Creasey, R. R., Himmelspach, J., White, K. P., and Fu, M., editors, *Proceedings of the 2011 Winter Simulation Conference*, pages 4080–4090, Piscataway NJ, 2011. IEEE.

[58] Pasupathy, R. and Henderson, S. G. Ambulance bases. Accessed May. 15, 2014. http://simopt.org/wiki/index.php?title=Ambulances_in_a_square, 2007.

[59] Pot, A., Bhulai, S., and Koole, G. A simple staffing method for multiskill call centers. *Manufacturing & Service Operations Management*, 10(3):421–428, 2008.

[60] Rainer-Harbach, M., Papazek, P., Hu, B., and Raidl, G. R. Balancing bicycle sharing systems: A variable neighborhood search approach. In Middendorf, M. and Blum, C., editors, *EvoCOP*, volume 7832 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2013.

[61] Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2005.

[62] Raviv, T. and Kolka, O. Optimal inventory management of a bike-sharing station. *IIE Transactions*, 45(10):1077–1093, 2013.

[63] Raviv, T., Tzur, M., and Forma, I. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229, 2013.

[64] Rubinstein, R. Y. and Shapiro, A. Optimization of static simulation models by the score function method. *Mathematics and Computers in Simulation*, 32:373–392, 1990.

[65] Schuijbroek, J., Hampshire, R., and van Hoeve, W.-J. Inventory rebalancing and vehicle routing in bike sharing systems. *Tepper School of Business*, Paper 1491, 2013.

[66] Seijo, E. and Sen, B. Nonparametric Least Squares Estimation of a Multivariate Convex Regression Function. *ArXiv e-prints*, March 2010.

[67] Shu, J., Chou, M. C., Liu, Q., Teo, C.-P., and Wang, I.-L. Models for effective deployment and redistribution of bicycles within public bicycle-sharing systems. *Operations Research*, 61(6):1346–1359, 2013.

[68] Silvapulle, M. J. and Sen, P. K. *Constrained Statistical Inference: Order, Inequality, and Shape Restrictions*, chapter 3, pages 59–141. John Wiley & Sons, Inc., 2001.

[69] Szechtman, R. and Yücesan, E. A bayesian approach to feasibility determination. In *Proceedings of the 2016 Winter Simulation Conference*, WSC '16, pages 782–790, Piscataway, NJ, USA, 2016. IEEE Press.

[70] Vogel, S. Stability results for stochastic programming problems. *Optimization*, 19(2):269–288, 1988.

[71] Wang, H., Pasupathy, R., and Schmeiser, B. W. Integer-ordered simulation optimization using R-SPLINE: Retrospective search using piecewise-linear interpolation and neighborhood enumeration. *ACM TOMACS*, 23(3), 2013.

[72] Wang, J. C. and Meyer, M. C. Testing the monotonicity or convexity of a function using regression splines. *Canadian Journal of Statistics*, 39(1):89–107, 2011.

[73] Williams, D. *Probability with Martingales.* Cambridge mathematical textbooks. Cambridge University Press, 1991.

[74] Xu, J., Nelson, B. L., and Hong, J. L. Industrial strength compass: A comprehensive algorithm and software for optimization via simulation. *ACM Trans. Model. Comput. Simul.*, 20(1):3:1–3:29, February 2010.