The Edge of Large-Scale Optimization in Transportation and Machine Learning

by

Sébastien Martin

M.S. in Applied Mathematics, École polytechnique (2015)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author . . . .
Sloan School of Management
May 15, 2019

Certified by . . . .
Patrick Jaillet
Dugald C. Jackson Professor
Department of Electrical Engineering and Computer Science
Co-director, Operations Research Center
Thesis Supervisor

Accepted by . . . .
Dimitris Bertsimas
Boeing Professor of Operations Research
Sloan School of Management
Co-director, Operations Research Center
Thesis Supervisor

The Edge of Large-Scale Optimization in Transportation and Machine Learning

by

Sébastien Martin

Submitted to the Sloan School of Management
on May 15, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

This thesis focuses on impactful applications of large-scale optimization in transportation and machine learning. Using both theory and computational experiments, we introduce novel optimization algorithms to overcome the tractability issues that arise in real world applications. We work towards the implementation of these algorithms, through software contributions, public policy work, and a formal study of machine learning interpretability. Our implementation in Boston Public Schools generates millions of dollars in yearly transportation savings and led to important public policy consequences in the United States.

This work is motivated by large-scale transportation problems that present significant optimization challenges. In particular, we study the problem of ride-sharing, the online routing of hundreds of thousands of customers every day in New York City. We also contribute to travel time estimation from origin-destination data, on city routing networks with tens of thousands of roads. We additionally consider the problem of school transportation, the scheduling of hundreds of buses to send tens of thousands of children to school everyday. This transportation problem is related to the choice of school start times, for which we also propose an optimization framework.

Building on these applications, we present methodological contributions in large-scale optimization. We introduce state-of-the-art algorithms for scheduling problems with time-window (`backbone`) and for school bus routing (`BiRD`). Our work on travel time estimation tractably produces solutions to the inverse shortest path length problem, solving a sequence of second order cone problems. We also present a theoretical and empirical study of the stochastic proximal point algorithm, an alternative to stochastic gradient methods (the de-facto algorithm for large-scale learning).

We also aim at the implementation of these algorithms, through software contributions, public policy work (together with stakeholders and journalists), and a collaboration with the city of Boston. Explaining complex algorithms to decision-makers is a difficult task, therefore we introduce an optimization framework to decomposes models into a sequence of simple building blocks. This allows us to introduce formal measure of the "interpretability" of a large class of machine learning models, and to study tradeoffs between this measure and model performance, the *price of interpretability*.

Thesis Supervisor: Dimitris Bertsimas
Boeing Professor of Operations Research
Sloan School of Management
Co-director, Operations Research Center

Thesis Supervisor: Patrick Jaillet
Title: Dugald C. Jackson Professor
Department of Electrical Engineering and Computer Science
Co-director, Operations Research Center

Acknowledgments

As the exciting journey of my PhD comes to an end, what will remain are the fantastic people I got to share it with.

I had the privilege to work with two research advisors, Dimitris Bertsimas and Patrick Jaillet, who also served as role models. I wish I only had a fraction of the energy and optimism of Dimitris. We shared many adventures, including public school committee meetings, testifying at the Rhodes Island State House and many successful Thanksgiving at his house! Dimitris' dedication to his students is truly impressive and I cannot help but share his dreams of good impact that operations research and analytics can have on our society. Witnessing his love of his job was instrumental in my decision to start a academic career. Patrick's wisdom and humanity also made him the perfect advisor. I could always come to him for advice, and not only about research. On top of being an extremely sharp mathematician, Patrick can stimulate and encourage research curiosity. He gave me a freedom of research I could not have dreamt of.

I would also like to thank my thesis committee members Asu Ozdaglar and Vivek Farias for their valuable feedback. I am very humbled that Asu accepted to join the committee of one of her game theory students, and for her help with my proximal work. And I thank Vivek for reviewing my research in the context on the ORC student paper competition, as well as his help with my interpretabibility work.

An important milestone of this PhD was the unexpected collaboration with Boston Public Schools (BPS). Discovering this new environment gave me the most precious gift: purpose. None of this would have been possible without the courage and kind-heartedness of John Hanlon, COO of BPS. His tireless commitment to Boston and its children is both humbling and contagious. Will Eger's endless supply of will and energy also fueled this whole collaboration (who else can make a full slide deck in less than a hour?). I was lucky to gain both a research partner and a friend. I think that John and Will's devotion to the public service should be an example for the world. Working with John and Will had its moments: the 2am pizzas at BPS, the moments of pure joy when we overcame obstacles, the moments of despair after the hardest school committee meetings, the Edelman Award... I would do everything again a thousand times. I also want to thank David Scharfenberg from the Boston Globe, Kade Crockford from the ACLU and Joi Ito from the MIT media lab for believing with us in the importance of the choice of school start times.

Arthur Delarue is a major contributor to the research of this thesis. With his incredible learning ability, mastery of words and humanity, Arthur could become a famous writer, charismatic politician, brilliant scholar, or maybe all of the above. I have worked with Arthur since the second year of my PhD, when he was an undergraduate student at MIT transitioning from physics to operations research. Four years later, I have gained a formidable research partner and friend who also started a PhD. The (too) many Domino's pizzas, the sleepless nights spent building last minute routes for the school buses of Boston, and the few times we thought we had a (maybe) "world-changing" research breakthrough in one of these tiny whiteboard rooms.

The Operations Research Center (ORC) has been a formidable environment, and I want to thank my friends Alex, Arthur, Emma, Jerry and Joel from the REFS (Resources for Easing Friction and Stress) for their work to make the ORC even better. The ORC is also a tight-knit community, and I thank all my fellow students for these five amazing years. In particular, my lab-mates and friends Ali, Anna, Andrew, Andrew, Cécile, Colin, Elisabeth, Emily, Jessica, Joey, max, Max, Mathieu, Rim and Yeesian. I also thank Profs. Ali Aouad, Antoine Désir, Joey Huchette, Andrew Li and Will Ma for their mentorship during the academic job market. I would share with my friend Ilias Zadik my hardest probability and life questions, discuss recurring sequences with Jean Pauphilet and exchange crazy research ideas with Max Biggs. ORC mornings could not have started without a high-five from Nico. Miles Lubin introduced me to Julia, the programming language used for all chapters of this thesis. The ORC could not run without Laura Rose and Andrew Carvalho, and I thank them for always helping me with my thousand daily questions. And none of this would have happened without my friend Jack Reilly and Pr. Alexandre Bayen convincing me to start this PhD, and I would not even have applied without the advice of Maximilien Burq. And I still follow Maximilien's advice, thank you for our many long conversations.

Life in Cambridge would not have been the same without my roommates and friends: Max, Maximilien, Sébastien, Lucie, Max, Hugh, Spencer, Anaëlle and Alena.

Khadija enlightened the last year of my PhD, and I hope many more years.

Being far away from France was not easy for me and my family, and I want to thank my parents Carlos and Véronique, and my siblings Ludovic and Marilou for their extraordinary love and support.

This work has been supported by the the National Research Foundation of Singapore Grant No. 015824-00078, by the Office of Naval Research grants N00014-12-1-0999 and N00014-16-1-2786, and by CIBC (Canadian Imperial Bank of Commerce).

To my parents, Carlos and Véronique

Contents

List of Figures

List of Tables

Chapter 1

Introduction

What time should schools start?

We had to answer this seemingly simple question for a research project with Boston Public Schools, one of the oldest public school districts in the United States. The surprising challenges hidden behind this question are representative of the work presented in this thesis. [1]

Schools start times could simply be chosen to comply with the preferences of teachers, children and parents. But this choice is much more complicated. Perhaps the greatest obstacle to adjusting school start times is the effect of changes on school transportation. Over 50 percent of U.S. schoolchildren rely on an army of half a million yellow school buses to travel to and from school every day. In Boston, transportation accounts for over 10% of the district's billion dollars budget. To reduce transportation spending, school districts such as Boston stagger the start and end times of different schools, allowing vehicles to be re-used several times throughout the day. Therefore, choosing when schools start requires to think through the optimization of their complex school bus system.

Transportation systems This example shows how deeply transportation systems are intertwined in many aspects of our day-to-day life. Being able to carefully design and control these systems using optimization can therefore lead to positive societal

[1] We will explore this question in details in Chapter 4.

impact. And examples are legion. Millions of schoolchildren hop on a school bus everyday We rely on our smartphones travel time estimations for our daily commutes. Lyft and Uber depend on complex optimization systems to dispatch their ride-sharing fleets. And the promise of self-driving vehicles could only increase this recent trend. This thesis will focus on such applications.

Large-scale optimization Optimizing the school start times and the school bus system of a large city is not an easy task, and has been mostly done by hand in Boston, as existing software could not provide a satisfying answer. Indeed, the problem of school bus routing is a NP-hard vehicle routing and scheduling problem, whose objectives, decisions and constraints typically change year to year. With more than a hundred schools and tens of thousands of students, we cannot hope to find optimal solutions. Typical routing software therefore has to resort to simple optimization heuristics such as greedy route construction or local improvement, which is not dissimilar to what is done by hand in Boston. This raises the more general question of the computational tradeoffs of large-scale optimization. The availability of massive data sets makes it possible to define new real-world applications that are larger than what state-of-the-art optimization algorithms and solvers can handle. Therefore, problems that matter the most are sometimes solved using simplistic optimization methods, potentially resulting in sub-optimal solutions. This thesis will explore this tradeoff in details. We will introduce optimization algorithms that manage to have both the flexibility of integer optimization, and the tractability required for transportation problems of this scale. We will then broaden this study by adding a theoretical and experimental study of the stochastic proximal point algorithm. This alternative to stochastic gradient methods for large-scale learning will also allow us to illustrate the computational tradeoffs of large-scale optimization.

Implementation Creating an algorithm that can efficiently optimize the school start times and bus routes is only the first step towards a successful implementation and real world impact. Many other steps are needed to go from an algorithm to a change in the status quo. Or collaboration with Boston Public Schools led to the implementation of a new school bus routing algorithm that saved the district millions of dollars. And our study of school start time choice had national coverage in the U.S., and led to the first policy change on this subject in thirty years in Boston. Such implementation does not only rely on accurate modeling and a tractable optimization algorithm. For example, building robust software and visualizations was also a significant part of the work. This is also why most chapters of this thesis are also associated with extensive software contributions. But what is the use of software if people do not want to use it? Policy work, and interactions with policy makers, stakeholders and sometimes journalists is also what can lead to a successful implementation, and was a key part of our implementation in Boston. Nonetheless, these interactions rely on our capacity to explain our models and algorithms. Building on this fact, this thesis also explores, conceptualizes and optimizes various facets of "interpretability" for a large class of models in machine learning.

Section 1.1 introduces the impactful applications of optimization in transportation that motivated our work. Section 1.2 connects these applications to the tradeoffs of large-scale optimization, and positions our methodological contributions within this space. Section 1.3 describes all the steps that are needed to bridge the gap between a successful optimization algorithm and a successful implementation, a key focus of this thesis. Finally, Section 1.4 is a detailed outline of the chapters of this thesis while stating its contributions to the field.

1.1 The Impact of Optimization in Transportation

Urban transportation is going through a rapid and significant evolution. In the recent past, the emergence of the Internet and of the smart-phone technologies has made us increasingly connected, able to plan and optimize our daily commute while large amounts of data are gathered and used to improve the efficiency of transportation systems. Today, real-time ridesharing companies like Uber or Lyft are using these technologies to revolutionize the taxi industry, laying the ground for a more connected and centrally controlled transportation structure, and building innovative systems like car-pooling. Tomorrow, self-driving and electrical vehicles will likely be the next transportation revolution. A major positive impact on the economy and the environment can be achieved when improving vehicle routing efficiency, using this newly available data and connectivity to its full extent. We present here the applications of transportation optimization that will be explored in this thesis.

1.1.1 Online Vehicle Routing for Ride-Sharing

We consider the decision problem that companies like Uber and Lyft face everyday: which customer should be matched to which vehicle. This is an important problem: in New York City, there are more than 500,000 Yellow Cab, Uber or Lyft rides everyday [102].

One simple solution is to dispatch the closest available vehicle when a customer requests a ride. But given the massive amount of available data, including traffic estimations, demand forecasts and anticipated requests, these simple myopic strategies can leave a lot on the table [2] and increase congestion and carbon emissions.

An alternative is to use the available information to "plan ahead" and make better decisions through a more involved optimization approach. This relates to the literature of *dynamic vehicle routing*, which is the general problem of dispatching vehicles to serve a demand that is revealed over time. We formulate the ride-sharing dispatch problem as a special case of dial-a-ride problem [9], where we assign each customer with a pickup time-windows and vehicles can transport only one customer at a time.

With the recent interest in real-time ride-sharing, several large-scale online decision systems for taxi scheduling have been proposed and implemented, though these applications focus more on managing large-scale decision systems than optimizing vehicle actions. [96] balances supply and demand in a discretized space and time, but does not consider microscopic routing decisions. [105, 104, 92, 122] implement large-scale systems of taxi-pooling, in which vehicles can transport several customers at the same time. These approaches focus on how to best match different requests,

[2]Chapter 2 shows that we can use 15% less vehicles using optimization to assign the requests.

but less on scheduling vehicles from request to request. [148] studies taxi routing with autonomous vehicles, taking into account congestion in a network-flow formulation.

This online problem exhibits an interesting tradeoff: on one hand finding the right vehicle for each customer given some information about the future is an optimization problem for which optimized solutions can significantly outperform simple heuristics. On the other hand, this problem requires to dynamically compute, within seconds, decisions involving tens of thousands of vehicles. This prevents the use of most vehicle routing optimization algorithms. We will show in Chapter 2 that we can scale the traditional optimization approaches through the use of the `local-backbone` algorithm, and outperform the state-of-the-art methods.

1.1.2 Optimizing Schools' Start Time and Bus Routes

Maintaining a fleet of buses to transport students to school is a major expense for U.S. school districts. Bus routes need to be designed every year to transport the children to and from school everyday, finding routes that minimize costs and maximize service quality is the school bus routing problem. In order to reduce costs by reusing buses between schools, many districts spread start times across the morning. However, assigning each school a time involves estimating the impact on transportation costs and reconciling additional competing objectives. Facing this intricate optimization problem, school districts must resort to ad hoc approaches, which can be expensive, inequitable, and even detrimental to student health. For example, medical research consensus is that early high school starts are impacting the development of an entire generation of students and constitute a major public health crisis [32, 34, 43, 44, 58, 106].

The problem of school bus routing has been addressed extensively [50, 109]. It is typically decomposed into three main subproblems: stop assignment, i.e., choosing locations where students will walk from their homes to get picked up; bus routing, i.e., linking stops together into bus trips; and bus scheduling, i.e., combining bus trips into a route that can be served by a single bus. State-of-the-art optimization algorithms exist for these subproblems in isolation [123, 61]. However, the literature on optimally combining subproblem solutions is less extensive. Approaches typically involve formulating the school bus routing problem as a large combinatorial optimization problem which can be solved using metaheuristics, including local search [128], simulated annealing [35], and special-purpose vehicle routing heuristics [26, 25]. Special-purpose algorithms have also been designed to address variants of the school bus routing problem, allowing "mixed loads" – students from different schools riding the bus together [128, 25, 110], bus transfers [23], or arrival time windows [61, 128, 35, 110].

Unfortunately, many tractable general-purpose algorithms do not consider additional constraints (fleet heterogeneity, student-specific needs) and thus lack portability. We will introduce `BiRD` in Chapter 4, a new algorithm for school bus routing that both outperforms the state-of-the-art on large instances, and has flexibility needed for practical problems. The School Time Selection Problem, or the problem of choosing school start time has been very little studied from an optimization point of view [138]. No existing algorithms address bell time selection in conjunction with bus routing [61], or take into account the multiple policy consequences of the choice of start times. We will also bridge this gap in Chapter 4.

1.2 Algorithms for Large-Scale Optimization

These applications are all associated with large-scale optimization problems. The ride-sharing problem in New York City requires to deal with hundreds of thousands of requests every day. School bus routing in Boston is a vehicle routing and scheduling problem that has two orders of magnitude more students than what could be solved to optimality. One alternative typically seen in these examples is the use of simple optimization heuristics like greedy insertions or local improvement [41, 128]. But these heuristics do not have the flexibility of with traditional optimization frameworks such as linear optimization, second order cone optimization and mixed integer optimization. Indeed, using such general frameworks offer the possibility to quickly implement and update the multiple objectives and constraints that arise in practice. They can also leverage the powerful optimization solvers that build on decades of research, as discussed in Section 1.2.1. But many of these formulations and solvers do not scale to the large optimization problems we consider. Ideally, we would like to design algorithms that will retain the modeling flexibility of mixed integer formulations while conserving tractability. This thesis explores this tradeoff for the special cases of ride-sharing and school district transportation optimization we introduced, but also in the more general examples of travel time estimation (Section 1.2.2) and the stochastic proximal point algorithm (Section 1.2.3).

1.2.1 Optimization Solvers in Practice

The work of chapters 2, 3, 4 and 6 of this thesis relies heavily on the use of state-of-the-art optimization solvers like *Gurobi* [65] and *Mosek* [2]. We cannot understate how useful they were in our large-scale optimization applications. First, even if finding provably optimal solutions to mixed integer optimization problems can be NP-hard, in many instances solvers can actually scale well to the typical problem sizes seen in practice. This could be surprising given the notorious difficulty of integer optimization and it has not always been the case. Actually, combining the computer performance increase and the solver progress between 1991 and 2015, the overall speedup factor for mixed integer optimization is approximately *2 trillion*[55]! This impressive improvement is unfortunately not enough to find optimal solutions for most of the problem we faced in this thesis. But we nonetheless use mixed-integer formulations as a building block of our algorithms, as they provide many advantages.

First, the layer of mathematical abstraction provided by these solvers allows to quickly modify objectives or add constraints without breaking the whole code. As an example, our work with Boston Public Schools taught us the challenges of modeling a real-world complex optimization problem. Indeed, the stakeholders do not necessarily know the exact objectives and constraints they need, and a long communication process is needed between the researcher and the decision-maker. The ability of solvers to quickly handle additional constraints or objectives proved to be tremendously important for the applications presented in this thesis.

Furthermore, we noticed that using these solvers significantly improved the speed of the coding process, and several factors contribute to this speedup. First, the use of the programming language *Julia* [20] and its optimization package JuMP [91] proved to be extremely efficient. This high-performance language has the ability to handle a large collection of optimization solvers in a seamless way. We can also easily interface solvers input and output with traditional code and algorithms, which simplify the creation of hybrid optimization algorithms that rely on extensive communication between the code and the solver.

Additionally, the debugging process, which takes a significant fraction of the coding time for large research projects, is significantly simplified by the use of solvers. Indeed, their ability to understand high level math formulations that avoid the burden of low-level coding, as well as enforcing constraints and identify the broken ones in the case of infeasibility reduces significantly the need for code debugging.

1.2.2 Scalable and Practical Travel Time Estimation

Chapter 3 will present an example of large-scale optimization algorithm for travel time estimation, which was a major need of our transportation projects in NYC and Boston.

The problem of inferring traffic patterns from diverse measurements is indeed a fundamental step behind the resolution of many complex questions in transportation and logistics. A simple cost function on the individual arcs of the network can often form a building block of a more complex network study, such as recent work by [114] presenting a novel understanding of resilient networks. Furthermore, many network problems specifically require a travel time estimate for each arc: for instance, [101], who develop a new model for traffic assignment that takes into account network uncertainty, present an approach starting from a prior estimate of the expected travel times of individual arcs in the network. Even in examples such as the aforementioned work or that of [77], both of which generally consider travel time to be a stochastic quantity, a good estimate for the network travel times is a valuable asset in order to define a prior or an uncertainty set for this uncertain quantity, laying the groundwork to answer more complex questions about the network.

To estimate travel times, we will use easily gatherable “origin-destination" (OD) data, that only records the time and location at the beginning and at the end of a trip, as, for example, collected by taxis or cellphone towers. Logging this data instead of high-density floating-car data increases the privacy of the taxi driver and passenger because the details of the followed route are not recorded. It also treats the network as a black box, only making measurements when the user enters and exits. OD data can be gathered for different purposes, and the methods we develop here in the context of vehicle traffic can be extended to other types of networks, including railways, subways, and bicycle and pedestrian networks (see recent studies such as [70]), or combinations of such networks. Nevertheless, this generality makes the travel time estimation harder: the problem of simultaneously determining paths and travel times based on origin-destination data only is close to the Inverse Shortest Path Length Problem (ISPL), an NP-hard problem which has also received some attention by [76].

Chapter 3 will introduce a solver-based large-scale optimization approach, solving a sequence of second order cone problems to fit large amounts of travel time data (up to millions of data-points) from various sources to a network model.

1.2.3 The Stochastic Proximal Point Algorithm

A good example of the tradeoffs of large-scale optimization is the wide-spread use of stochastic gradient methods for large-scale machine learning.

We consider the following convex stochastic optimization problem

$$\min_{x \in \mathcal{X}} F(x) = \mathbb{E}_S\left[f(x, S)\right] \tag{1.1}$$

where $\mathcal{X} \subset \mathbb{R}^d$ is a closed convex set, S is a random variable, $f(., S)$ are closed convex functions for each value S. These problems have many applications in statistical learning [72, 149], but also stochastic optimization and simulation. In the context of machine learning, $f(x, S)$ can represent the loss function of a model parametrized by x on a uniformly random data point represented by S, and (1.1) is equivalent to minimizing the expected loss over the data-set. For example, an ordinary least square problem would correspond to

$$f(\beta, S) = (X'_S\beta - y_S)^2$$

where X_S and y_S represent the feature and label of a uniformly sampled data-point.

Instead of knowing the distribution of S, we consider the case where we have the possibility to draw in dependent random samples and use them iteratively to solve (1.1). Lately, the large amounts of data that are available for machine learning applications make this stochastic setting particularly relevant for large-scale learning [24].

In the unconstrained case ($\mathcal{X} = \mathbb{R}^d$), the de facto algorithms for solving (1.1) are the stochastic (sub)gradient methods [24, 98, 115, 100], first introduced by [119] for smooth problems. Starting from a solution x_0, each iteration $k \geq 1$ draws an independent sample $S_k \sim S$ and updates the previous solution using a sub-gradient step of size $\mu_k \geq 0$ on $f(., S_k)$:

$$x_k = x_{k-1} - \mu_k g_k \quad \text{where } g_k \in \partial f(x_{k-1}, S_k) \tag{1.2}$$

These methods has guarantees of convergence, and it is used successfully in large-scale stochastic optimization [24, 149]. But the choice of step schedule can be challenging [98], and strong assumptions are needed for convergence.

Interestingly, the sub-gradient step is also the minimizer of the following optimization problem:

$$\begin{aligned} x_k = \operatorname{argmin}_x \left(f(x_{k-1}, S_k) + g'_k(x - x_{k-1})\right) + \frac{1}{2\mu_k}\|x - x_{k-1}\|^2 \\ \text{with } g_k \in \partial f(x_{k-1}, S_k) \end{aligned} \tag{1.3}$$

The term in parenthesis is a first order approximation of $f(., S)$ around x_{k-1}. If this approximation with the function itself, we get the *stochastic proximal point algorithm*

[121, 21, 132, 3, 112]:

$$x_k = \operatorname{argmin}_{x \in \mathcal{X}} f(x, S) + \frac{1}{2\mu_k} \|x - x_{k-1}\|^2 \tag{1.4}$$

This algorithm presents an interesting tradeoff. It enjoys better non-asymptotic convergence and stability than stochastic gradient [3], but is computationally expensive. We will study it in Chapter 5 and show that the optimization edge it provides can compensate for its computational limits, and make it a competitive alternative to stochastic gradient methods.

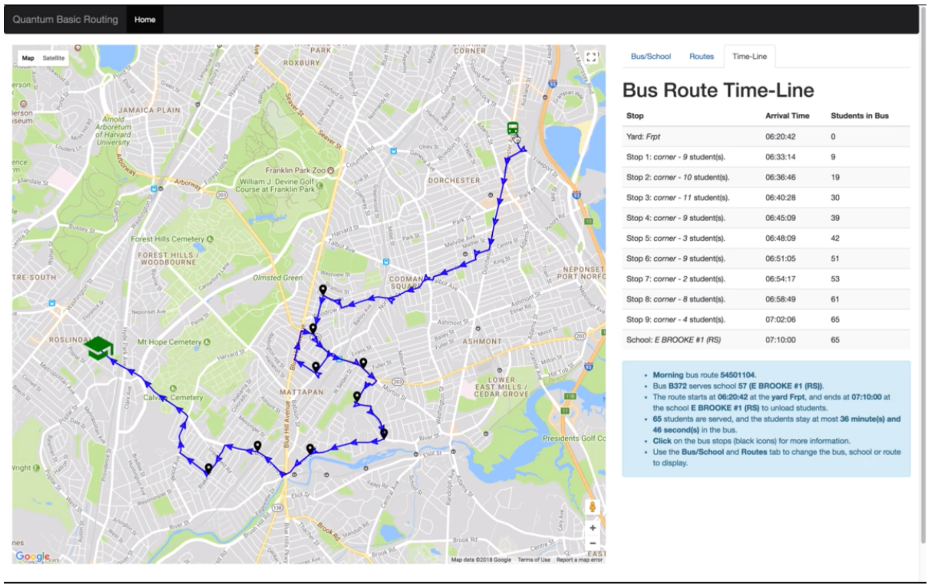
1.3 Bridging the Gap Between Optimization and Implementation

Once a positive use of analytics is found, and tractable optimization algorithms are created, the journey to their implementation can still be long. Nonetheless, researchers have many ways to make this step easier, and we explore a few examples in this thesis.

1.3.1 The Impact of Visualization and Software

Visualizations and good software can be tremendously important for applied research. The benefits are twofolds: it makes it easier to share ideas and convince people, but also helps a lot develop a deeper understanding of the research and generate new ideas.

Visualizations and software are amongst the key contributions of this thesis. We share an extensive micro-simulation software for vehicles in large cities [16, 13]. We also built a piece of software (mostly open-sourced) that is used every year by Boston Public Schools to decide the routes of their buses [14], and is illustrated in Figure 1-1. In Spring 2017, our ability to share our routing solutions with Boston Public School's transportation team in a visual way was instrumental in their decision to use them.

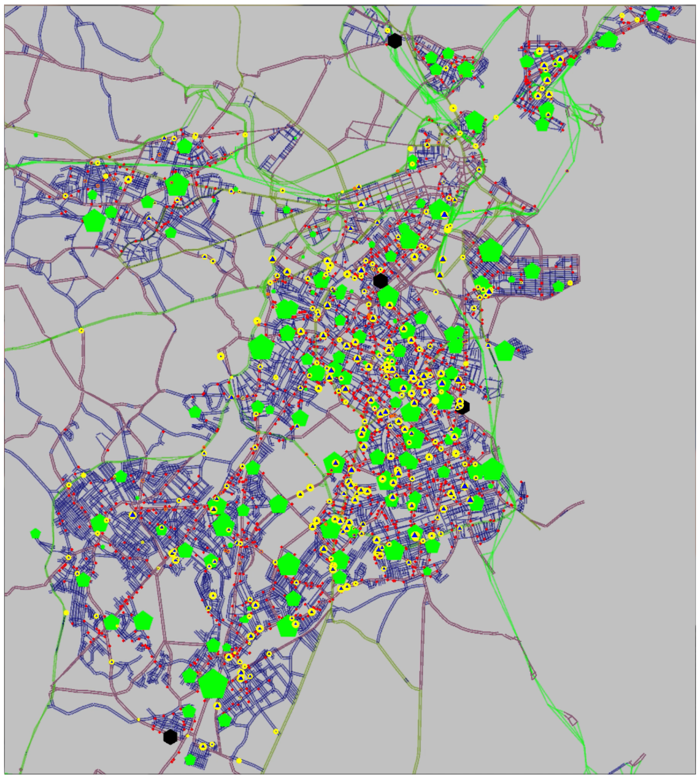


Bus Route Time-Line

Stop	Arrival Time	Students in Bus
Yard: *Frpt*	06:20:42	0
Stop 1: *corner* - *9* student(s).	06:33:14	9
Stop 2: *corner* - *10* student(s).	06:36:46	19
Stop 3: *corner* - *11* student(s).	06:40:28	30
Stop 4: *corner* - *9* student(s).	06:45:09	39
Stop 5: *corner* - *3* student(s).	06:48:09	42
Stop 6: *corner* - *9* student(s).	06:51:05	51
Stop 7: *corner* - *2* student(s).	06:54:17	53
Stop 8: *corner* - *8* student(s).	06:58:49	61
Stop 9: *corner* - *4* student(s).	07:02:06	65
School: *E BROOKE #1 (RS)*	07:10:00	65

- **Morning** bus route **54501104**.
- Bus **B372** serves school **57 (E BROOKE #1 (RS))**.
- The route starts at **06:20:42** at the **yard Frpt**, and ends at **07:10:00** at the school **E BROOKE #1 (RS)** to unload students.
- **65** students are served, and the students stay at most **36 minute(s) and 46 second(s)** in the bus.
- **Click** on the bus stops (black icons) for more information.
- Use the **Bus/School** and **Routes** tab to change the bus, school or route to display.

(a) Routing web-application.



(b) Real time school bus simulations

Figure 1-1: The visualization software developed for Boston Public Schools. The first picture is a screenshot of the web application we developed to share the outputs of our school bus routing algorithm with the city of Boston. Each student, school, route and bus schedules can be explored individually on the map (courtesy of Google Maps. The second picture is a screenshot of the real-time school bus simulation engine we built to have a representation of the buses moving around the city (buses are yellow and schools are in green).

This would later save millions of dollars to be reinvested in the education of the children of the district.

1.3.2 The Interface with Policy

Modeling a real world problem is an important part of any optimization application. But there is a limit to any model, and for the implementation to be successful, it is important to get a constant feedback from stakeholders and policymakers.

In the case of the optimization of Boston's schools start times, the policy component was extremely important, as these decisions affect most parents in the city. It was unanimously voted that these times should be chosen in a way that is equitable and healthy for the students. The intense public debate that followed in December 2017 in Boston had us exchange a lot with the stakeholders and media, including the Wall Street Journal [95] and the Boston Globe [125]. We noticed first hand how our algorithms and statistics were being used for various political means. Research that interacts with policy should strive for excellence in clarity, as researchers that can participate to these public debates and be understood can do a lot of good.

1.3.3 Model Interpretability

When exchanging with policymakers, stakeholders and journalists, the interpretability of the optimization models and algorithms is a necessity. The models that are implemented or viewed positively by the public were in our experience always the most interpretable. This motivated a new line of research, and we explore in this thesis the meaning and tradeoffs of interpretability in the context of machine learning models.

Model interpretability, and its tradeoff with predictive accuracy, are of significant interest to the machine learning community [59]. However, a major challenge in this line of research is that the very concept of interpretability is hard to define and even harder to quantify [89]. Many definitions of interpretability have a "know it when you see it" aspect which makes quantitative analysis difficult, though several recent works [54, 63] have introduced new paradigms that could help overcome the ad hoc nature of existing approaches.

Fortunately, the absence of a universally agreed-upon definition of interpretability has not prevented extensive research in the development of interpretable models. Decision trees [28, 15] are considered interpretable for their discrete structure and graphical visualization, as are close relatives including rule lists [86, 144], decision sets [84], and case-based reasoning [80]. Other approaches include generalized additive models [90], i.e. linear combinations of single-feature models, and score-based methods [133], where point values for each feature can be summed up into a final "score".

In the case of (generalized) linear models, interpretability often comes down to sparsity, a topic of extensive study over the past twenty years [73]. Sparse models are characterized by a small number of nonzero coefficients, meaning that the target variable only depends on a small subset of the input features. Training sparse models can be done through regularization, e.g. using LASSO [131], or by using special algorithms including forward/backward stepwise regression [130] and least-angle regression [57]. Very recently, mixed-integer optimization approaches have been used to solve the sparse regression problem exactly [18, 19].

In Chapter 6, we will introduce a framework that generalizes these examples and provide a way to quantify and optimize the "interpretability" of a large class of models.

1.4 Thesis Outline and Main Contributions

This thesis consists of two parts. Chapters 2, 3 and 4 describe our work on large-scale optimization for transportation applications. Chapters 5 and 6 study broader methodological tools with applications in machine learning. Chapter 7 concludes our work and discusses potential extensions.

1.4.1 Chapter 2 - Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications

With the emergence of ride-sharing companies that offer transportation on demand at a large-scale and the increasing availability of corresponding demand data-sets, new challenges arise to develop routing optimization algorithms that can solve massive problems in real time. In this chapter, we develop an optimization framework, coupled with a novel and generalizable backbone algorithm, that allows us to dispatch in real time thousands of taxis serving more than 25,000 customers per hour. We provide evidence from historical simulations using New York City routing network and yellow cabs data to show that our algorithms improve upon the performance of existing heuristics in such real-world settings.

This research was done in collaboration with Dimitris Bertsimas and Patrick Jaillet, and is published in *Operations Research* [16].

The contributions of this work include:

1. Introducing the `backbone` and `local-backbone` algorithms, novel algorithms for scheduling problems with time-windows that outperform the existing alternatives. This algorithm can solve large-scale scheduling problems with thousands of customers in under a second, which was not possible before, and its core idea generalizes to a large class of optimization problems.

2. The modeling of a complex real-time ride-sharing problem.

3. An open-source simulation, optimization and visualization engine to run the online algorithms with real demand and city data.

1.4.2 Chapter 3 - Travel Time Estimation in the Age of Big Data

Twenty-first century urban planners have identified the understanding of complex city traffic patterns as a major priority, leading to a sharp increase in the amount and the diversity of traffic data being collected For instance, taxi companies in an increasing number of major cities have started recording metadata for every individual car ride. In this chapter, we show that we can leverage network optimization insights to extract accurate travel time estimations from such origin-destination data, using information from a large number of taxi trips to reconstruct the traffic patterns in an entire city on a variety of timescales. We develop a method that tractably exploits origin-destination data, and draws from its optimization framework the flexibility needed to take advantage of other sources of traffic information. Using synthetic data, we establish the robustness of our algorithm to uncertainty, and display its ability to significantly reduce input variance. We show empirically that our algorithm can leverage any available amount of data, even in a high-variance environment, to provide insights about urban traffic patterns on different scales, leading to accurate travel time estimations throughout the network.

This research was done in collaboration with Dimitris Bertsimas, Arthur Delarue and Patrick Jaillet, and is published in *Operations Research* [13].

The contributions of this work include:

1. A probabilistic modeling of a network-based travel time estimation problem with simplistic assumptions and traffic rules. This orthogonal take on the traffic estimation literature sacrifices modeling accuracy for tractability, which allows us to handle large amounts of origin-destination data and obtain accurate and interpretable travel time estimates.

2. An approximation algorithm for the Inverse Shortest Path Length Problem, involving solving a sequence of Second Order Cone Problems.

3. Software to load real city networks and estimate travel times from origin destination data.

1.4.3 Chapter 4 - Optimizing Schools' Start Time and Bus Routes

Maintaining a fleet of buses to transport students to school is a major expense for school districts. To reduce costs by reusing buses between schools, many districts spread start times across the morning. However, assigning each school a time involves estimating the impact on transportation costs and reconciling additional competing objectives. Facing this intricate optimization problem, school districts must resort to ad hoc approaches, which can be expensive, inequitable, and even detrimental to student health. For example, there is medical evidence that early high school starts are impacting the development of an entire generation of students and constitute a major public health crisis. We present an optimization model for the school time selection problem (STSP), which relies on a school bus routing algorithm that we call bi-objective routing decomposition (BiRD). BiRD leverages a natural decomposition of the routing problem, computing and combining sub-problem solutions via mixed integer optimization. It significantly outperforms state-of-the-art routing methods, and its implementation in Boston has led to $5 million in yearly savings, maintaining service quality for students despite a 50-bus fleet reduction. Using BiRD, we construct a tractable proxy to transportation costs, allowing the formulation of the STSP as a multi-objective generalized quadratic assignment problem. Local search methods provide high-quality solutions, allowing school districts to explore tradeoffs between competing priorities and choose times that best fulfill community needs. In December 2017, the development of this method led the Boston School Committee to unanimously approve the first school start time reform in 30 years.

This research was done in collaboration with Dimitris Bertsimas and Arthur Delarue, and is published in *Proceedings of the National Academy of Sciences of the United States of America* [14].

The contributions of this work include:

1. A new algorithm (BiRD) for school bus routing that outperforms the state-of-the-art on standard benchmarks.

2. The implementation of this routing algorithm in Boston, that currently saves the city five million dollars each year.

3. The modeling of the school start time selection problem.

4. A multi-objective optimization framework based on generalized quadratic assignment to optimize the school start times.

5. Policy and media impact that raised national awareness of the importance of the choice of schools start times.

6. Open source software and benchmark instances.

1.4.4 Chapter 5 - The Benefits of the Stochastic Proximal Point Algorithm

In the context of solving stochastic convex optimization problems, we present a detailed theoretical and experimental comparison study of the stochastic proximal point algorithm and the popular stochastic (sub)gradient methods. Asymptotic convergence as well as stability have been studied, but our focus is to understand when proximal point should be used in practice. This requires being able to precisely characterize the differences in the transient phase of convergence, as well as taking into account the computational requirements. To this effect, we study the simple example of stochastic quadratic convex unconstrained problems. In one dimension, this allows us to define an optimal step schedule and therefore to be able to compare the algorithms in a way that does not depend on the choice of step size. This approach highlights how the commonly use technique of mini-batching (variance reduction) favors the proximal algorithm. We experimentally generalize these insights in higher dimensions, and compare the algorithms from a computational point of view, which reveals that proximal steps can be surprisingly efficient. We finally introduce an approximation algorithm to solve the proximal problem using conjugate gradient, that significantly outperforms both stochastic gradient and stochastic proximal point steps.

This research has been done in collaboration with Dimitris Bertsimas and Patrick Jaillet.

The contributions of this work include:

1. An theoretical study of the behavior of stochastic proximal point and gradient steps on one-dimensional convex-quadratic cost functions using optimal step schedules.

2. An experimental generalization of these findings to higher dimensions.

3. A study of the error of the stochastic proximal point algorithm focusing on computational time and mini-batching, which are typically overlooked in the literature.

4. The `approx` algorithm that uses conjugate gradient methods to make proximal steps computationally competitive in high-dimensions.

1.4.5 Chapter 6 - The Price of Interpretability

Interpretability is often a desirable characteristic of quantitative models, such as machine learning models, especially in the context of human-in-the-loop analytics. When predictive models are used to support decision-making on complex and important topics, understanding a model's "reasoning" can increase trust in its predictions, expose hidden biases, or reduce vulnerability to adversarial attacks. However, the concept of interpretability remains loosely defined and application-specific. In this chapter, we introduce a mathematical framework in which models are constructed in a sequence of *interpretable steps*. We show that for a variety of models, a natural choice of interpretable steps recovers standard interpretability proxies (e.g. sparsity in linear models). We then generalize these proxies to yield a parametrized family of consistent measures of model interpretability. This formal definition allows us to quantify the "price" of interpretability, i.e. the tradeoff with predictive accuracy. We demonstrate practical algorithms to apply our framework on real and synthetic data-sets.

This research has been done in collaboration with Dimitris Bertsimas, Arthur Delarue and Patrick Jaillet.

The contributions of this work include:

1. The modelling of an incremental model framework to capture many aspects generally associated with interpretability in the literature.

2. A formal and general definition of model "interpretability".

3. An novel application of this framework to linear models.

Chapter 2

Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications

2.1 Introduction

Urban transportation is going through a rapid and significant evolution. In the recent past, the emergence of the Internet and of the smart-phone technologies has made us increasingly connected, able to plan and optimize our daily commute while large amounts of data are gathered and used to improve the efficiency of transportation systems. Today, real-time ridesharing companies like Uber or Lyft are using these technologies to revolutionize the taxi industry, laying the ground for a more connected and centrally controlled transportation structure, and building innovative systems like car-pooling. Tomorrow, self-driving and electrical vehicles will likely be the next transportation revolution. A major positive impact on the economy and the environment can be achieved when improving vehicle routing efficiency, using this newly available data and connectivity to its full extent.

A field that can make such important contributions is *vehicle routing*, i.e., the optimization of each vehicle actions to maximize the system efficiency and throughput.

In the special case of *taxi routing*, we decide which taxi or ride-sharing vehicle to assign to each ride request. This setting is typically online, as there is little prior demand information available and the vehicle actions have to be decided in a dynamic way. There is more and more central control of these vehicle actions, allowing the design of strategies that surpass myopic agent behaviors. Furthermore, real-world applications are generally at a decidedly large-scale: everyday, there are more than 500,000 Yellow Cab, Uber or Lyft rides in New York City — see [102].

In this chapter, we present a tractable rolling-horizon optimization strategy for online taxi routing that can be adapted to a variety of applications. Our formulation is guided by the increased degree of control and prior information available in today's ride-sharing dispatching systems. We introduce a novel approach to make vehicle routing optimization formulations tractable at the largest practical scales, involving tens of thousands of customers per hour. This approach is general and can be extended to a variety of vehicle routing problems. We implement these online strategies on real taxi demand data in New York City, dispatching thousands of vehicles in real time and outperforming state-of-the-art algorithms and heuristics, thus showing the edge of optimization.

2.1.1 Related Work

Dynamic vehicle routing is the general problem of dispatching vehicles to serve a demand that is revealed in real time.

In the *pick-up and delivery* problem, vehicles have to transport goods between different locations. When vehicles are moving people, the routing problem is referred to as *dial-a-ride* in [9]. Taxi routing is a special case of dial-a-ride problem with time-windows, where vehicles can transport only one customer at a time, with pick-up time windows but no destination time windows. Customers are also associated with a pick-up time window, which is a typical model of customer flexibility in diverse applications of vehicle routing as in [49, 4, 36]. This constraint can be relaxed, and vehicle routing with soft time windows, for example in [71], penalizes a late pick-up instead of disallowing it. We use "hard" time windows in this work, though our approach can be extended to the soft time window case.

[113] argues that taxi routing has received relatively small attention in the field of vehicle routing, as the practical problem sizes are typically large, and last-minute requests leave "limited space for optimization". Nonetheless, with the emergence of ride-sharing smart-phone applications, it has become easier to request for a last-minute ride even when the available vehicles are far away, and to book a ride in advance. Such possibilities can be modeled in vehicle routing formulation using pick-up time windows and prior request times, and have been studied for different applications in [75, 145]. We will demonstrate that the additional prior information they provide can be leveraged when optimizing the routing decisions, even at the largest scale.

With the recent interest in real time ride-sharing, several large-scale online decision systems for taxi scheduling have been proposed and implemented, though these applications focus more on managing large-scale decision systems than optimizing vehicle actions. [96] balances supply and demand in a discretized space and time, but does not consider microscopic routing decisions. [105, 104, 92, 122] implement large-scale systems of taxi-pooling, in which vehicles can transport several customers at the same time. These approaches focus on how to best match different requests, but less on routing vehicles from request to request. [148] studies taxi routing with autonomous vehicles, taking into account congestion in a network-flow formulation.

Several strategies have been proposed for dynamic vehicle routing. Simple online routing algorithms can be studied in a worst-case approach using competitive ratios as in [78]. However, when some prior information is available, practical approaches are re-optimization and rolling-horizon algorithms — see [9, 141]: a "static" (or "offline") solution is constantly updated as new demand information becomes available, and this solution is used to decide the vehicles actions in real-time. This strategy has been applied with success, and [145] show that the quality of the online decision depends on the quality of the offline solutions that are used at each iteration. Together with the rolling-horizon, [7] uses a scenario-based approach with consensus, [8] successfully adds a waiting and relocation strategy, and [97] uses a double-horizon to take into account long-term goals. The offline decision problems can then be translated into well-defined optimization formulations, and sometimes solved to optimality. These formulations are typically solved using column generation as in [4], which is also used in the online setting in [36]. Custom branching algorithms, as found in [66, 49], are used for specific routing problems. [10] formulates the decision problems using constraint programming. For problems similar to taxi routing, with identical vehicles and paired pick-up and delivery, variants on network flow formulations have been proposed, for example [145] uses such a formulation to optimize truckload pickup and delivery. Unfortunately, these exact algorithms rarely scale past a few dozens or hundreds of customers and vehicles, depending on the application. We will use such formulations in this chapter, though applying them on problems with tens of thousands of customers and thousands of vehicles.

A classical way to solve these static vehicle routing problems at a larger scale is the use of heuristics, discussed in the survey [27]. Existing solutions can be locally improved by exploring their neighborhood: for example, 2-OPT is a famous general-purpose heuristic that was first introduced for the traveling salesman problem (TSP) in [41]. In practice, combinations of insertions-based greedy construction heuristics, local-improvement and exploration heuristics are used, as in [142, 27, 97]. For sizable problems such as taxi routing, we found that using a first-accept local-improvement heuristic similar to the 2-OPT* algorithm presented in [116] was a good trade-off when limited computational time is available, and we use it as a benchmark for our work. Unfortunately, these heuristics are usually special-purposed and have to be adapted to each particular new problem. State-of-the-art algorithm of vehicle routing include popular meta-heuristics such as Tabu Search applied in [62], Evolutionary Algorithms, Ant Colony Algorithms, Simulated Annealing and hybrid algorithms that combine the advantages of different methods, as in [10]. In this chapter, we were not able to successfully apply any of these algorithms, because of the size of our problem and the very small time available for computations.

To the best of our knowledge, there is no large-scale benchmark for dynamic vehicle routing, as emphasized in [113]. To test our algorithms, we chose the New York City Taxi and Limousine Commission data-set, available at [102] and frequently used in the literature. This massive data-set contains all ride-sharing and taxi trips in NYC, starting from 2009, for a total of more than 1 billion rides. A comprehensive description of this data-set is available in [143]. It has been used for vehicle routing decisions in [104, 148, 122]. We used the OpenStreetMap map data [69] to reconstruct the real city network, together with the work in [18] to infer link travel times from the taxi data. Furthermore, we used Julia, a programming language with a focus on numerical computing introduced in [20], in combination with the optimization modeling library described in [91], to create a large-scale simulation and visualization for real-world routing, and support our experiments.

2.1.2 Our Contributions

This chapter explores taxi routing, in the contemporary context of increased connectivity and prevalent data: we formulate, solve, scale and apply optimization formulations to real-world settings. Overall, we show that some seemingly intractable optimization formulations in vehicle routing can be scaled to the largest problem sizes. This is desirable, as these formulations generalize much better than special-purpose heuristics to the various operational constraints of real-world applications.

Motivated by the centralization and modernization of taxi routing in the ride-sharing industry, we formulate the online taxi problem as a pick-up and delivery problem, using re-optimization and an efficient network flow mixed-integer optimization formulation similar to [145], to leverage any prior information of ride requests to make better decisions. In an extensive empirical study with synthetic data, we show that in situations of high demand, optimal solutions to the offline taxi routing problem are usually significantly superior to the output of common local-improvement and greedy routing algorithms. This confirms the edge of optimization formulations on simple heuristics for the taxi routing problem, and outlines what practical situations make these formulations easy or difficult to solve in practice.

To scale our formulations to real-world applications with thousands of taxis and tens of thousands of customers, we use the specific structure of taxi-routing applications with high demand to dramatically reduce the problem size. We additionally

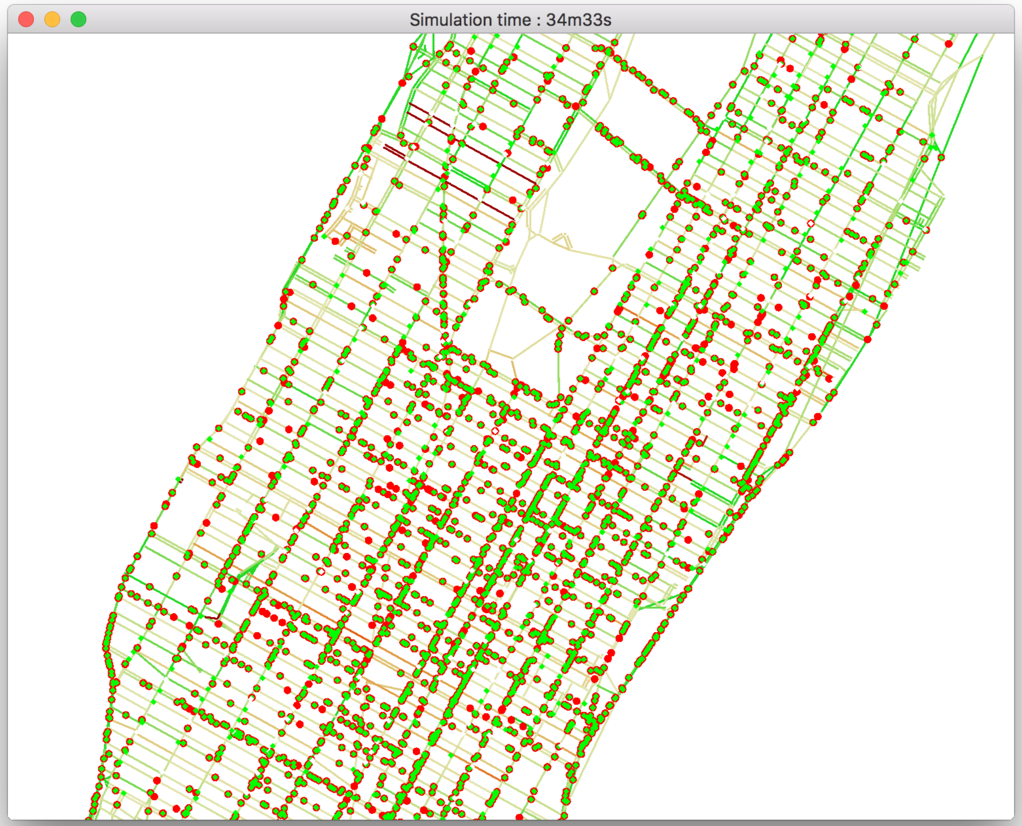


Figure 2-1: Our taxi simulation software, displaying online taxi routing in Manhattan with 5000 taxis and 26,000 customers. The red circles represent taxis, and the green squares represent customers being transported or waiting. The software shows the taxi movements in real or accelerated time, and implements all the online and offline algorithms we discuss in this chapter. It has been designed to run on a standard laptop.

introduce a novel “backbone” algorithm, that first computes a restricted set of candidate actions that are likely to be optimal, allowing us to efficiently solve a much sparser problem. On a very time-constrained re-optimization schedule, with only 15 seconds to solve a vehicle routing problem involving thousands of taxis, we show that our new algorithm performs significantly better than other popular large-scale routing heuristics.

We created an open-source simulation software (available in the TaxiSimulation Julia package [94]) using state-of-the-art technologies, allowing us to simulate and visualize taxi-routing in real-world setting, and presented in Figure 2-1. We use New York City taxi ride data and the complete Manhattan routing network to apply our algorithms in practical settings, leading us to confirm the results we got from synthetic data. The insights we get from such applications are relevant to the current and future taxi and ride-sharing industry, and our models have the potential to be extended to a variety of other applications. For reproducibility, the input, output and all the code of our experiments are shared with the published version of this chapter.

Section 2.2 introduces and defines the online taxi routing problem, and we study in Section 2.3 its offline counterpart, formulating it using mixed integer optimization and comparing it to established heuristics on synthetic data. In Section 2.4, we demonstrate how to scale this formulation to the applications of interest. We finally apply the offline algorithms to large online taxi problems in Section 2.5, using re-optimization, and with real demand data in NYC.

2.2 The Online Taxi Routing Problem

In this section, we introduce the online taxi routing problem and the notations we will use throughout this chapter. This model captures any prior information we may have on customer requests, due to prior booking or customer pick-up flexibility.

2.2.1 Model and Data

We consider the *online taxi routing problem*, a special case of the *online dial-a-ride problem with time windows*. In this application, vehicles are only allowed to serve one customer at a time.

Let $\mathcal{C}$ be the set of all customers. A customer $c \in \mathcal{C}$ is associated with a pick-up time window (t_c^{min}, t_c^{max}), corresponding to its minimal and maximal possible pick-up times. In the online setting, we also introduce a confirmation time t_c^{conf}, at which customer c is provided with a guarantee to be picked up (or is rejected), and a request time $t_c^{request}$, at which the customer's information becomes available. Note that $t_c^{request} \leq t_c^{conf}$, as the confirmation of future pick-up can only happen after the pick-up request.

We represent each customer as a node in a directed graph $\mathcal{G}$. An arc (c', c) in $\mathcal{G}$ represents the possibility for a vehicle to pick-up customer c immediately after servicing customer c'. Each arc (c', c) is associated with a travel time $T_{c',c}$ such that we must have $t_{c'} + T_{c',c} \leq t_c$, where $t_{c'}$ and t_c are the respective pick-up times of customers c' and c. $T_{c',c}$ typically represents the time for a taxi to serve customer c' and to drive to the pick-up location of c. Each arc is also associated with a profit $R_{c',c}$, that represents the quantity we want to maximize. In this work, we use it to represent the profit of the taxi company, and set $R_{c',c}$ to the fare paid by customer c minus the cost of driving from the drop-off point of c' to the pick-up point of c and to its destination.

We restrict ourselves to the case where $\mathcal{G}$ is acyclic. In other words, the pickup time windows can only allow one customer to be picked-up before or after another one, but never both. There is an arc $c \to c'$ in $\mathcal{G}$ if and only if:

$$t_c^{min} + T_{c,c'} \leq t_{c'}^{max} \tag{2.1}$$

There exists a cycle of length 2 if and only if Equation (2.1) is verified from c to c'

and also from c' to c. By combining the two equations we obtain:

$$(t_c^{max} - t_c^{min}) + (t_{c'}^{max} - t_{c'}^{min}) \geq T_{c,c'} + T_{c',c} \tag{2.2}$$

Negating Equation (2.2) gives us a sufficient condition for the absence of any 2-cycle:

$$(t_c^{max} - t_c^{min}) + (t_{c'}^{max} - t_{c'}^{min}) < T_{c,c'} + T_{c',c} \quad \forall c, c' \in \mathcal{C} \tag{2.3}$$

In other words, the condition states that the sum of the lengths of the pick-up time windows within the cycle should be less that the total cycle travel time, and the exact same reasoning shows that this condition works with any cycle length. A stronger sufficient condition to avoid any cycle is therefore that each pick-up time window is smaller than the following ride time:

$$(t_c^{max} - t_c^{min}) < T_{c,c'} \quad \forall c, c' \in \mathcal{C} \tag{2.4}$$

This condition is a little too extreme, but it nonetheless holds for our taxi routing application: the time windows we use in this chapter are of the order of 5 minutes, and the vast majority of taxi trips considered in this chapter take more than 5 minutes (and most of them a lot longer). The few trips that are smaller than 5 minutes are still satisfying Equation (2.3). Taxi routing is not the only application where this assumption holds: any dynamic vehicle routing application with time windows that are no bigger than the typical trip length will have no or few cycles. For in-between applications that just have a few cycles, a simple pre-processing step can remove the cycles, or else adding sub-tour elimination constraints will be very fast and tractable. Nonetheless, for applications with larger (or infinite) time windows and thus a large number of cycles in $\mathcal{G}$, the algorithms of this chapter would have to be adapted.

Let $\mathcal{K}$ be the set of all taxis, that are supposed to be identical and whose initial positions are represented as additional nodes in $\mathcal{G}$. Each taxi k is parametrized by a initial time of service t_k^{init} at which it becomes available. For each customer c that can be the first pick-up of taxi k from its original position, we add the arc (k, c) to

$\mathcal{G}$. This arc is associated with a travel time $T_{k,c}$, typically the time for taxi k to go to c's pick-up location, and a profit $R_{k,c}$, typically the fare paid by c minus the driving costs.

2.2.2 Decisions

A solution to the taxi routing problem is a subset of arcs of $\mathcal{G}$ that designate the sequences of customers assigned to each taxi. Each customer must only be picked-up by at most one taxi, while respecting its pick-up time window constraint: if arc (c', c) is in the solution (a taxi serves the two customers sequentially), then we must have $t_{c'} + T_{c',c} \leq t_c$.

The goal is to maximize the total profit of the solution, as described by the parameters $R_{c',c}$ and $R_{k,c}$. Additionally, the problem is solved in an online setting, where the information of the existence of customer c is only revealed at time $t_c^{request}$: the node appears in the graph, together with its arcs to and from other known nodes. In this setting, the decision to add the arc (c', c) to the solution has to be made early enough, when the taxi that is serving customer c' can still pick-up customer c on time. Moreover, the decision whether to pick-up or reject customer c must be made before t_c^{conf}.

2.2.3 Interpretation

This formulation is general enough to model many optimization objectives. For example we can minimize total empty driving time instead of profit by modifying the R parameters accordingly. Setting $R_{c',c} = 1 \; \forall c \in \mathcal{C}$ will maximize the throughput: the total number of customers that we can serve. Also, setting $t_c^{request} = 0 \; \forall c \in \mathcal{C}$ corresponds to the full-information offline problem, and $t_c^{request} = t_c^{min} \; \forall c \in \mathcal{C}$ corresponds to the fully online problem without any prior information.

Note that travel times T are deterministic once revealed. Also, the destination of customer c is revealed at time $t_c^{request}$, as it allows us to plan the next moves as we know the travel times to the next customers. As seen in the introduction, we focus on well-connected taxi systems, that already ask customers for their destination when requesting a ride.

A typical setting to compute the travel times T is to consider a routing network, where each customer will be associated with a pair of origin and destination nodes. Assuming stationary travel times for each edge of the network and some additional routing rules such as "taxis use the fastest path", we can compute the times T, possibly including additional constant service time to pick-up and drop-off each new customer. As we will discuss in Section 2.5, these travel time forecasts can be adjusted in a dynamic way along with the re-optimization process.

2.3 Offline Routing: the Edge of Optimality

In this section, we introduce the offline taxi routing problem, that naturally appears when using a re-optimization strategy for the online taxi routing. We present a mixed integer optimization (MIO) formulation of the offline problem, along with a set of heuristics. These different algorithms are compared on synthetic data, and we develop an empirical intuition about the effect of a variety of practical settings on the problem difficulty and the algorithms performances. We show that in situations of high demand, provably optimal solutions to the routing problem outperform their heuristic counterpart by a large margin.

2.3.1 A Re-Optimization Approach to Online Taxi Routing

When all the demand information is known beforehand, the problem is called *offline* (or static). In the offline problem, there is no uncertainty about future customers. This is equivalent to setting all the request times to the beginning of the instance, i.e., $t_c^{request} = 0$ for each customer c. In this case, the taxi routing problem introduced in Section 2.2 becomes the *Offline Taxi Routing Problem.* It is a well-defined optimization problem, and the feasible solutions maximizing profit are offline optimal. While most real-world taxi routing problems are not offline, the profit of an offline optimal solution is an upper bound to the profit of any strategy applied to the corresponding online problem.

If an efficient solution method for the offline problem is available, it can be used to solve the online problem through *re-optimization*. This online strategy repeatedly solves the offline problem with the known customers, and implements the first taxi actions as time goes by. Formally, given a re-optimization rate Δt^{update} (in our case we will always use $\Delta t^{update} = 30$ seconds), iteration k of the strategy solves the offline problem with all unserved customers known at time $t = k\Delta t^{update}$, i.e., the set of customers $\left\{c \in \mathcal{C},\ t_c^{request} \leq k\Delta t^{update}\right\}$. We then implement all the taxis actions that take place between $t = k\Delta t^{update}$ and $t = (k+1)\Delta t^{update}$ in the previously computed offline solution. A detailed description of our implementation of the re-optimization strategy is presented is Section 2.5.1.

The re-optimization online strategy has been shown to work well in practice, see [9] and [145]. However, its efficiency relies on the quality of the available offline solution methods, and their ability to give good solutions in a time that needs to be less than Δt^{update}. In the examples of this chapter we set a limit of 15 seconds for the computation of an offline solution within a re-optimization iteration.

In the rest of this section, we introduce and compare different offline algorithms, focusing on their tractability and the quality of the solutions they produce.

2.3.2 Offline Solution Methods

We formulate the offline taxi-routing problem using MIO, so that we can use a MIO solver to compute provably optimal solutions. We also introduce a set of heuristics that provide good feasible solutions and can serve as a benchmark.

MIO formulation. We translate the taxi problem decisions, objective and constraints from Section 2.2 into a linear mixed-integer optimization formulation.

Each arc of graph $\mathcal{G}$ is associated with a binary decision variable (x or y), representing whether this arc is used in the solution. For each customer c, we also add the binary variables p_c to represent them being picked-up or rejected, together with the continuous decision variable t_c to represent their pick-up time.

$$y_{k,c} = \begin{cases} 1, & \text{if customer } c \text{ is picked-up by taxi } k \text{ as a first customer,} \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{c',c} = \begin{cases} 1, & \text{if customer } c \text{ is picked-up immediately after customer } c' \text{ by a taxi,} \\ 0, & \text{otherwise.} \end{cases}$$

$$p_c = \begin{cases} 1, & \text{if customer } c \text{ is picked-up by a taxi,} \\ 0, & \text{if } c \text{ is rejected.} \end{cases}$$

$$t_c = \quad \text{pick-up time of customer } c.$$

We maximize the total profit, which is the sum of the profits associated with each arc of $\mathcal{G}$ in the solution.

$$\text{maximize} \sum_{k\in\mathcal{K},\, c\in\mathcal{C}} R_{k,c} y_{k,c} + \sum_{c',c\in\mathcal{C}} R_{c',c} x_{c',c} \tag{2.5}$$

To enforce that each taxi is associated with a unique sequence of customers to pick-up, we implement a set of network-flow constraints on the variables x, y and p.

$$p_c = \sum_{k\in\mathcal{K}} y_{k,c} + \sum_{c'\in\mathcal{C}} x_{c',c} \qquad \forall c\in\mathcal{C} \tag{2.6}$$

$$\sum_{c\in\mathcal{C}} x_{c',c} \le p_{c'} \qquad \forall c'\in\mathcal{C} \tag{2.7}$$

$$\sum_{c\in\mathcal{C}} y_{k,c} \le 1 \qquad \forall k\in\mathcal{K} \tag{2.8}$$

$$x_{c',c} \in \{0,1\} \qquad \forall c', c\in\mathcal{C} \tag{2.9}$$

$$y_{k,c} \in \{0,1\} \qquad \forall k\in\mathcal{K}, c\in\mathcal{C} \tag{2.10}$$

$$p_c \in \{0,1\} \qquad \forall c\in\mathcal{C} \tag{2.11}$$

Eq. (2.6) defines p_c: a customer c is picked up if and only if a (unique) taxi k serves her as a first customer (variable $y_{k,c}$) or after another customer c' (variable $x_{c',c}$). Eq. (2.7) guarantees that each customer c' is either picked-up and followed by at most one other customer c ($\sum_{c\in\mathcal{C}} x_{c',c} \leq p_{c'} = 1$) or not picked up and thus not followed by any customers ($\sum_{c\in\mathcal{C}} x_{c',c} \leq p_{c'} = 0$). Eq. (2.8) states that each taxi k has at most one first customer. Together these constraints can be interpreted as "flow constraints" on the network $\mathcal{G}$, and guarantee that each feasible solution is a set of edges in $\mathcal{G}$ that corresponds to a set of non-intersecting paths starting from taxis nodes. Our assumption that there is no cycle in the graph plays an important role here: it allows us to avoid cycle-breaking constraints that usually appear in vehicle routing with large time windows.

We add the pick-up time window constraints:

$$t_c^{min} \leq t_c \leq t_c^{max} \qquad \forall c \in \mathcal{C} \tag{2.12}$$

$$t_c - t_{c'} \geq \left(t_c^{min} - t_{c'}^{max}\right) + \left(T_{c',c} - \left(t_c^{min} - t_{c'}^{max}\right)\right) x_{c',c} \qquad \forall c, c' \in \mathcal{C} \tag{2.13}$$

$$t_c \geq t_c^{min} + \left(t_k^{init} + T_{k,c} - t_c^{min}\right) y_{k,c} \qquad \forall c \in \mathcal{C},\ k \in \mathcal{K}. \tag{2.14}$$

Eq. (2.12) bounds the pick-up times to the customer time windows. Eqs. (2.13) and (2.14) are two strengthened *Big M* sets of constraints that make sure that the sequence of customers assigned to each taxi is compatible with their respective pick-up times. For example, if customer c' is served by a taxi immediately before customer c (i.e., $x_{c',c} = 1$), Eq. (2.13) becomes $(t_c - t_{c'}) \geq T_{c',c}$, which is exactly the meaning of the travel time $T_{c',c}$ as defined in Section 2.2. Conversely, if $x_{c',c} = 0$, Eq. (2.13) becomes $(t_c - t_{c'}) \geq t_c^{min} - t_{c'}^{max}$, which is always true given the time window constraint (2.12).

The MIO formulation (2.5)-(2.14) has $O(|\mathcal{K}|.|\mathcal{C}| + |\mathcal{C}|^2)$ constraints and decision variables. Nonetheless, not all variables $x_{c',c}$ and $y_{k,c}$ need to be defined. For example, we do not need the decision variable $x_{c',c}$ if $t_{c'}^{min} + T_{c',c} \geq t_c^{max}$, because the pick-up time constraint (2.13) will force $x_{c',c} = 0$. It is therefore sufficient to only consider the decision variables corresponding to the actions that are compatible with the pick-up time windows, which correspond by definition to the arcs of graph $\mathcal{G}$. Let $N = |\mathcal{K}| + |\mathcal{C}|$ be the number of vertices in graph $\mathcal{G}$ and E be the number of arcs. We obtain $O(E+N)$ constraints and decisions variables, which is why this formulation is particularly efficient, owing to the fact that decision variables $x_{c',c}$ are not indexed by the taxi k that serves customers c and c'. We can then use a MIO solver to get an optimal integer solution to the offline taxi-routing problem. We call this optimal algorithm `MIOoptimal`.

Max Flow Heuristic. In the previous MIO formulation, the constraints (2.6)-(2.11), together with the objective (2.5) represent a max-flow problem with integer bounds on the flow variables. Thus, extreme points of the formulation are integral, and we can use the simplex algorithm to get an optimal integral solution. Unfortunately, time window constraints (2.12)-(2.14) break this integrality property.

However, in the special case where the pick-up times are fixed, we obtain the following integrality result:

Theorem 1. *If each customer has a fixed pick-up time, i.e., the time windows are limited to one unique pick-up time $t_c^{min} = t_c^{max} = t_c^*$, $\forall c \in \mathcal{C}$ then the mixed-integer formulation (2.5)-(2.14) is integral.*

Proof. First, replacing $t_c^{min} = t_c^{max} = t_c^*$ into Constraint (2.12), we obtain $t_c = t_c^*$. By substituting the decision variable t_c with its value t_c^* in Constraint (2.13), we obtain

$$(t_c^* - t_{c'}^*) \ \geq \ (t_c^* - t_{c'}^*) + (T_{c',c} - (t_c^* - t_{c'}^*))\, x_{c',c},$$

which is equivalent to

$$(T_{c',c} - (t_c^* - t_{c'}^*))\, x_{c',c} \leq 0. \tag{2.15}$$

If $T_{c',c} - (t_c^* - t_{c'}^*) > 0$, then we must have $x_{c',c} = 0$ and the formulation is equivalent to a formulation in which variable $x_{c',c}$ is removed.

If $T_{c',c} - (t_c^* - t_{c'}^*) \leq 0$, then Equation (2.15) is always true no matter what the value of $x_{c',c}$ is. As a consequence, Constraint (2.13) is inactive on the decision variables x, y and p.

The same reasoning applies to Constraint (2.14). Therefore, the feasibility region of the decision variables x, y and p is the same as the one defined by the network-flow constraints (2.6)-(2.11). This formulation is integral. $\square$

When pick-up times are fixed, the integrality result means that we can solve the offline problem efficiently, for example using the simplex algorithm. We use Theorem 1 to design a heuristic for the offline taxi-routing problems with time windows. If we assign to each customer c a fixed pick-up time t_c^* such that $t_c^{min} \leq t_c^* \leq t_c^{max}$, then the optimal solution for the max-flow problem with fixed pick-up times t_c^* is feasible for the general formulation with time windows. Note that these solutions are often sub-optimal, as they do not use the time windows flexibility to pick-up customers more efficiently. Empirically, setting $t_c^* = t_c^{max}$ will yield good solutions, as taxis have more time to go from their first position to the first customers. On the other hand, when time windows are small, the solutions are often near-optimal or even optimal on some problems. We call this heuristic `maxflow`.

Baseline Heuristic: Greedy Insertion. A simple approach to the offline taxi routing problem is to assign the customers to taxis in a greedy way. We iterate through the customers by order of t_c^{min} (earliest customers first), and we assign them to the closest available taxi or reject them if no taxi is available. This heuristic is related to insertion-based solutions construction algorithm as presented in [27]. We name this algorithm `greedy`, and its formal implementation is detailed in Appendix 2.7.1. Because of its simplicity, tractability and wide-spread use, `greedy` will be our baseline for the offline taxi routing problem.

Local-Improvement with 2-OPT. Traditional solution methods for large-scale vehicle routing include heuristics that locally improve a feasible solution in an iterative way. The 2-OPT algorithm is a popular local-improvement heuristic, first introduced for the TSP in [41]. We implement an optimized version of the 2-OPT* algorithm presented in [116], in order to compare our MIO formulation to state-of-the-art fast heuristics. We initialize it with the `greedy` solution, and stop it when it reaches a locally optimal solution. We name this algorithm `2-OPT`, and its details are presented

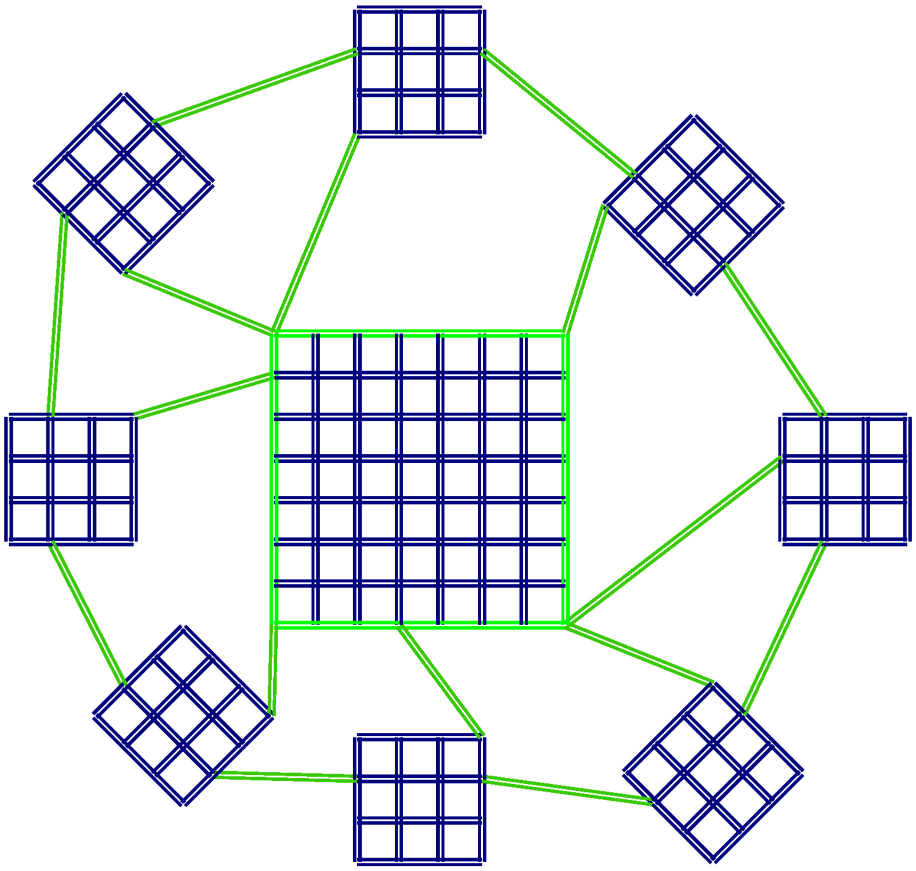


Figure 2-2: Routing network used to generate the synthetic instances of taxi routing. Each road is two-way, and the green connecting arcs have faster travel times. The travel times are chosen such that the mean trip-time between two vertices in the routing graph is around 10 minutes. The network has been designed to represent commuting effect between residential area and city-center. Customers trips are randomly generated as Poisson processes on each intersection of the routing network.

in Appendix 2.7.2.

We chose not to use more complex meta-heuristics with exploration, such as Tabu-Search presented in [62]. While we acknowledge these meta-heuristics avoid local optima and are common in vehicle routing, we could not find a way to implement a version of Tabu Search that worked with the limited time budget of a few seconds of online decision making and the large problem size with tens of thousands of customers.

2.3.3 Application on Synthetic Data

We generate random synthetic instances of offline taxi routing to evaluate the algorithms presented in Section 2.3.2. We compare the running time and the quality of solutions in different scenarios, in order to gain insights that we will use to solve large-scale real-world problems.

Routing in synthetic city. To compare the solutions of `MIOoptimal` to the solutions provided by the other algorithms, we have designed a way to generate synthetic routing problems. We need these synthetic problems, as real-world problems are rarely small enough to be solved to optimality by state-of-the-art commercial solvers. We have built synthetic instances that can be solved to optimality while being large and complex enough to provide interesting insights.

The synthetic routing network represents a simplified city and its suburbs. The graph has 192 nodes and 640 bi-directional arcs. The downtown area is represented by a 8 times 8 square, and the 8 suburbs are represented by 4 times 4 squares. Travel times are slower inside the city and suburbs, and faster in connecting links and around the city. The routing network is represented in Figure 2-2. This network and all the algorithms are implemented using our open-source simulation and visualization framework in Julia.

On this network, we create a random one-hour instance of taxi-routing. We choose the actions of a fleet of 20 taxis with uniformly distributed initial locations. Customers are randomly generated as a Poisson process with a fixed rate, with the origin and destination of each trip uniformly drawn across the network nodes. The fares are set to be proportional to the distance of the trips, which are defined as the paths that minimize the total travel time. We compute the time parameters of the taxi routing problems $T_{c',c}$ using the total time of these shortest paths, to serve customer c' and then go to customer c origin. The profit parameters $R_{c',c}$ are set equal to the profit, i.e., the difference between the fare paid by c and a cost proportional to the driving time.

The travel times $T_{c',c}$ are selected so that the highway links (green in Figure 2-2)

Time Window	Demand	Algorithms Increase in Profit		
		`MIOoptimal`	`2-OPT`	`maxflow`
1 min.	low	1.94%	1.13%	1.27%
	medium	8.24%	3.70%	5.75%
	high	15.92%	8.19%	**12.82%**
3 min.	low	1.73%	1.22%	1.10%
	medium	9.00%	5.36%	2.75%
	high	14.11%	7.24%	4.98%
6 min.	low	1.42%	0.96%	-0.86%
	medium	9.38%	5.84%	-2.60%
	high	**19.93%**	**11.64%**	3.17%

Table 2.1: Comparing the offline solvers to the `greedy` baseline. Each row corresponds to a different setting of synthetic customer data: we vary the customers time windows (flexibility in the pick-up time), and the level of demand (number of customers). We show the improvement in profit of our algorithms compared to the `greedy` heuristic, averaged across 20 randomly generated simulations. Low demand corresponds to 40 customers, and taxis are typically idle half of the time. Medium demand is 70 customers, which represents a balanced supply and demand. High demand is 140 customers and at least half of the customers are typically rejected. We represent in bold the most favorable situation for each algorithm.

are twice as fast as the other links, and so that a taxi can serve up to 3-4 customers per hour. The profit $R_{c',c}$ are computed such that there is a cost of \$5 per hour of driving and \$1 per hour of waiting, and a customer fare of \$80 per hour of driving.

Results We study the influence of the time windows and the level of demand on the behavior of our algorithms. We select three levels of demand, setting the rates of the customer Poisson processes so that the expectation of the total number of customers is respectively 40, 70 and 140 customers. 40 customers (low demand) corresponds to optimal solutions in which taxis are idle half the time and are able to serve all customers. 70 customers (medium demand) corresponds to a matched supply and demand: almost all customers are accepted and taxis are driving most of the time. 140 customers (high demand) represents a surge scenario, where taxis can only accept 50% of the customers on average. We give all customers a fixed time window around their preferred pick-up time, from 1 to 6 minutes. We average our results across 20 random simulations for each set of parameters. Table 2.1 compares the profit of

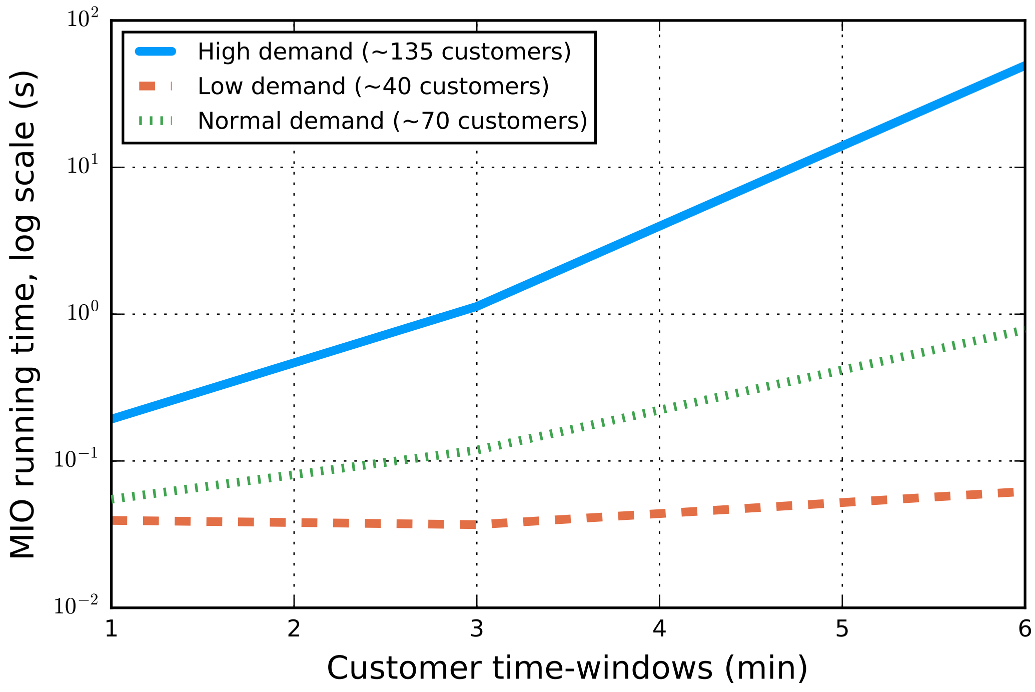


Figure 2-3: Time for `MIOoptimal` to find the optimal solutions. The times are averaged across 20 simulations. The three curves represent the different levels of demand, and the x-axis is the length of the customers time windows in minutes. Generally, a larger time windows and more customers lead to an exponentially higher computational time. Note the logarithmic scale of the y-axis.

the solutions of each algorithm to the `greedy` baseline, and Figure 2-3 shows the computational time needed by the commercial solver Gurobi to compute the optimal solution (`MIOoptimal`).

2.3.4 The Edge of Optimality

The results of the simulations on synthetic data, presented in Table 2.1 and Figure 2-3, allow us to split the level of demand and the length of the time windows into three main categories of vehicle routing problems. These settings represent fundamentally different optimization problems, and we study how the solution methods compare in each one of them.

First, when demand is low, the `greedy` heuristic performs almost optimally, always within 2% of the optimum in our simulations. This situation is intuitive: most taxis being free, assigning the nearest free taxi to a customer performs well in practice. In this situation, optimization is not extremely useful and we recommend using `greedy`, which is the fastest and the most interpretable.

When demand is medium to high, taxis are mostly busy and `greedy` is typically far from optimality. There is an edge in using optimization: taking into account future customers and using time window flexibility allows for better solutions. We identify two main settings in this situation.

When the time windows are small, `maxflow` is very close to optimality. Indeed, we have seen in Section 2.3.2 that this solution method is actually optimal when pick-up times are fixed. As commercial linear optimization solvers are typically very fast and scale well, we recommend using this heuristic in practice. It performs significantly better than `greedy` and `2-OPT` while being close to the optimal solution provided by `MIOoptimal`.

The most interesting case is when demand is medium to high and time windows are not small. This situation is the most useful in practice, as a 3-6 minutes time window is a fair estimation of customer patience. Indeed, at the time we write this article, the media reports a median Uber customer waiting time of 2-10 minutes, depending on the city, though we could not find any official statistics. High demand scenarios are also typical in taxi routing, with peak-hours everyday. In this case, optimal solutions outperform the locally-optimal solutions provided by `2-OPT` and have a strong edge on `greedy` solutions. `maxflow` can perform poorly when the time windows are large. We recommend using `MIOoptimal` when possible, but the problem can be significantly harder to solve to optimality, as shown in Figure 2-3. The mixed-integer optimization solver takes an exponential time, in the level of demand and in the time window length, to converge to provable optimality and also to provide near-optimal feasible solutions. When `MIOoptimal` is too slow to be used in practice, the locally optimal solution provided by `2-OPT` is a reasonable alternative, and is widely used for large-scale vehicle routing, as stated in [27]. Scaling the optimal formulation to real-world applications and outperforming the local-optimization methods is the objective of the next section.

2.4 Scaling Optimization to Real-World Applications

Using optimal solvers for the offline taxi routing problem leads to a significant improvement in the solution quality, particularly when customer demand matches or exceeds the vehicle supply. `MIOoptimal` nevertheless becomes quickly intractable when the number of customers and taxis increases: to obtain a proof of optimality in less than one hour with a typical laptop, the limit is around 150 customers for a 5 minutes time window.

In this section, we show how to leverage the structure of real-world vehicle routing applications to make mixed-integer optimization formulations tractable. We construct a large-scale and real-world taxi routing problem in New York City using Yellow-Cab demand data. This problem involves thousands of taxis and customers, and each iteration of the re-optimization process corresponds to a mixed-integer optimization formulation with more than 10 million binary decision variables, and needs to be solved in seconds. We propose an algorithm that is tractable at this scale of taxi routing, outperforms the state-of-the-art and combines the advantages of local-search and global optimization to get near-optimal results within the allowed computational time.

2.4.1 Sparsifying the Flow Graph

One way to increase the tractability of `MIOoptimal` is to decrease the number of binary decision variables in the mixed integer optimization formulation presented in Section 2.3.2. Equivalently, removing some well-chosen arcs from the flow graph $\mathcal{G}$ will reduce the solution space and make optimization easier. Nonetheless, we risk removing some arcs that are in the optimal solution and hence decrease the quality of the result. If we find arcs that are less likely to be optimal than others, removing them can increase tractability without decreasing the optimal solution quality too much, and, given the limited computational time, lead to better practical solutions.

Our results on synthetic data in Section 2.3.4 show that high demand scenarios are the hardest offline taxi routing problems and are the most favorable and interesting for optimization-based algorithms. As a result, we focus on scenarios in which demand matches or exceeds supply. In the optimal solution for such a problem, a taxi is unlikely to wait or drive empty for too long before getting a customer. Indeed, if we have a large number of taxis spread throughout the city and a high demand, taxis will probably pick-up customers that are nearby in space and time. When closer customers are available, we do not expect a taxi to drive empty and wait a long time to pick-up a far-away customer: we can safely remove the corresponding arcs from $\mathcal{G}$.

Formally, we define a cost function between nodes in graph $\mathcal{G}$. For nodes representing customers c' and c, we define the cost $C(c', c)$ to be the shortest possible "lost time" that a taxi will have to spend waiting or driving empty when serving customer c' and c sequentially:

$$C(c', c) = \max\left(T_{c',c},\, t_c^{min} - t_{c'}^{max}\right) - T_{c'},$$

where $T_{c'}$ is the time to transport customer c' from its origin to its destination. For nodes representing a taxi k and a customer c, we define $C(k, c)$ to be the minimal time (including wait) it takes for the taxi to reach and pick-up c as a first customer:

$$C(k,c) = \max\left(T_{k,c},\, t_c^{min} - t_k^{init}\right).$$

We want to keep the arcs in $\mathcal{G}$ that have the lowest cost, as they represent actions of picking up "nearby" customers, and thus more likely to be optimal. These costs are just used as an indicator of the quality of each edge, and will be used to remove the edges that are really unlikely to be in an optimal solution.

For a given sparsity parameter K, we prune $\mathcal{G}$ to create a sparser graph $K\mathcal{G}$ by only keeping the K-lowest cost incoming and outgoing arcs for every node in $\mathcal{G}$. We name this flow graph pruning technique `K-neighborhood`, and its steps are detailed in Algorithm 1.

Algorithm 1 `K-neighborhood`

```
Require:
    The flow-graph $\mathcal{G}$ and a sparsity parameter $K$.
Ensure:
    A sparser flow-graph $K\mathcal{G}$ by pruning $\mathcal{G}$.

1: Initialize graph $K\mathcal{G}$ with the same nodes as $\mathcal{G}$ and without any arcs.
2: for all node $n$ in $\mathcal{G}$ do
3:     Select the $K$ incoming arcs to node $n$ in $\mathcal{G}$ with lowest cost $C(\cdot, n)$ and add
       them to $K\mathcal{G}$.
4:     Select the $K$ outgoing arcs out of node $n$ in $\mathcal{G}$ with lowest cost $C(n, \cdot)$ and
       add them to $K\mathcal{G}$.
```

The new formulation associated with the graph $K\mathcal{G}$ (for fixed K) has $O(|\mathcal{C}|+|\mathcal{K}|)$ decision variables and constraints instead of $O(|\mathcal{C}|^2 + |\mathcal{C}||\mathcal{K}|)$ for $\mathcal{G}$, which allows us to solve problems at a much larger scale. In practice,for the real-world problems we have tried, we noticed that $K = 20$ usually provides near optimal solutions, and $K = 50$ optimal solutions. We have also empirically found that the choice of K should only depend on the balance between supply and demand. When demand is lower than supply, we have seen in Section 2.3.4 that taxis tend to be idle and the closest taxi is likely to pick-up a customer in an optimal solution. Low values of K are then enough to get near-optimal solutions, as the closest taxis will have generally the lowest values of $C(\cdot, \cdot)$. As demand grows and matches or exceeds supply, we have empirically found that the best solutions correspond to higher values of K, up to $K = 50$. Furthermore, the size of the city and the total number of taxis and customers typically do not influence the choice of K: the choices for one given taxi are typically local, in the taxi's neighborhood, and are not influenced by the total size of the city.

When time windows are small, results from Section 2.3.4 indicate that `maxflow` provides near-optimal solutions. Additionally, using `K-neighborhood` with $K = 50$, problems with thousands of taxis and customers are solved in seconds using `maxflow`. When time windows are larger, typically 3-6 minutes, we have shown that the offline taxi routing problem becomes much harder, and that we need to use `MIOoptimal` to get good solutions. Unfortunately, when using `MIOoptimal` and `K-neighborhood` for our large-scale applications, the problem is typically intractable for $K \geq 4$ and low values of K reduce the quality of the solutions. These observations motivate the ideas in the next section.

2.4.2 The Backbone Algorithm

In order to make `MIOoptimal` tractable for large instances of the taxi routing problem, we need to remove a lot of arcs from $\mathcal{G}$. We cannot set a value of K that is too low, as `K-neighborhood` would remove too many arcs that participate in optimal solutions and correspondingly decrease the quality of the solution. Nonetheless, even within a limited neighborhood around a taxi's position, there are customers that are better than others, given the positions of other taxis. If we identify these potentially good arcs of $\mathcal{G}$, we could reduce the number of arcs even more and make `MIOoptimal` tractable.

In Theorem 1 we have shown that for fixed pick-up times the `maxflow` algorithm solves the problem optimally. Furthermore, the `maxflow` algorithm scales for very large problems. If we randomly select a pick-up time within each time window and solve the fixed-pickup time problem with `maxflow` in the graph $K\mathcal{G}$, we get a solution that is feasible for the problem with time windows, as seen in Section 2.3.2. If we resolve several times the fixed pick-up time problem with the tractable `maxflow` and random pick-up time within the time windows, and collect all the optimal arcs across the different solutions, we obtain a set of arcs that are likely to be optimal. This set of arcs represents a very sparse sub-graph of $\mathcal{G}$, a "backbone" for our optimization problem, on which we can use `MIOoptimal` to compute an optimal solution within the backbone network, which is near-optimal for the original graph $\mathcal{G}$. This is the `backbone` algorithm, described formally in Algorithm 2.

Algorithm 2 `backbone`

```
Require:
    The flow-graph $\mathcal{G}$.
    A sparsity parameter $K$ such that maxflow is tractable on $K\mathcal{G}$.
    a limit $E_{max}$ on the number of arcs such that MIOoptimal stays tractable.
Ensure:
    A backbone flow-graph $B\mathcal{G}$ that is a sparser version of $K\mathcal{G}$ with a maximum of
    $E_{max}$ arcs.

1: Step 1: initialize the backbone graph $B\mathcal{G}$ by removing all the arcs in $\mathcal{G}$.
2: while $B\mathcal{G}$ has less than $E_{max}$ arcs do
3:     for each customer $c \in \mathcal{C}$ do
4:         Step 2: generate a uniformly random pick-up time $t_c \in [t_c^{min}, t_c^{max}]$.
5:     Step 3: use maxflow on $K\mathcal{G}$ with the fixed pick-up times $t_c$.
6:     Step 4: add all the optimal arcs of the computed solution to $B\mathcal{G}$.
```

We typically choose $K = 20$ for the large-scale instances of this chapter: this choice of K creates a sparse graph while rarely sacrificing optimality. This algorithm gives good results in practice, especially if the time windows are small (less that 2 minutes in our applications). Steps 2-3 of Algorithm 2 can be executed in parallel, which allows us to assign most of the available computational time to solve the mixed-integer optimization problem on the backbone $B\mathcal{G}$. For wider time windows, the `maxflow` solutions with random pickup times create too many arcs and therefore `MIOoptimal` is not tractable at the largest scale. This motivates the need to improve the `backbone` algorithm, which we do next.

2.4.3 The Local Backbone Algorithm

When using the re-optimization strategy presented in Section 2.3.1, we solve the offline taxi routing problem with all the available future demand information at every time-step of length Δt, typically 30 seconds. The offline problem at time t is very similar to the next problem at time $t + \Delta t$ as we only add and remove a few requests. Therefore, a good solution to the offline problem at time t can be used to construct a good solution to the problem at time $t + \Delta t$. More specifically, we adapt the previous solution by removing the customers that have just been served at time t and adding the new requests of time $t + \Delta t$ as "rejected" to make the solution feasible for the new problem (we do not know yet if we can accept them). We can then use this solution as a warm-start for the new problem.

In Steps 2-3 of the `backbone` algorithm in Section 2.4.2, the fixed pick-up time is selected uniformly randomly within the customers time windows. The idea of the local backbone algorithm is to update the customers time windows so that the solution s at time t is feasible with these pick-up times.

For each customer c served in s, we define $[t^{min}_{c,s}, t^{max}_{c,s}]$ to be the interval of possible pick-up times t_c such that s is still feasible. In other words, all taxis can still serve the same sequence of customers as prescribed by solution s, while respecting the pick-up time t_c for customer c. We have $[t^{min}_{c,s}, t^{max}_{c,s}] \subset [t^{min}_{c}, t^{max}_{c}]$. We compute $t^{min}_{c,s}$ and $t^{max}_{c,s}$ next. Suppose that in solution s, a taxi has to pick-up customers c^- , c and c^+, in this order. Then

$$t^{min}_{c,s} = \max\left(t^{min}_{c}, t^{min}_{c^-,s} + T_{c^-,c}\right), \tag{2.16}$$

$$t^{max}_{c,s} = \min\left(t^{max}_{c}, t^{max}_{c^+,s} - T_{c,c^+}\right). \tag{2.17}$$

Equation (2.16) states that the minimal pick-up time $t^{min}_{c,s}$ for customer c either corresponds to t^{min}_{c}, the beginning of its time window, or to the earliest possible time to pick-up customer c^- plus the travel time between c^- and c : $t^{min}_{c^-,s} + T_{c^-,c}$. Equivalently Equation (2.17) defines $t^{max}_{c,s}$ to either be equal to t^{max}_{c} or to the latest possible time to pick-up c^+ minus the travel time between c and c^+, whichever is the earliest.

Additionally, if c^{first} and c^{last} are the first and last customers to be picked-up by taxi k, there are no propagating constraints on their earliest and latest pick-up times, respectively, which leads to:

$$t^{min}_{c^{first},s} = \max\left(t^{min}_c, t^{init}_k + T_{k,c}\right), \tag{2.18}$$

$$t^{max}_{c^{last},s} = t^{max}_c. \tag{2.19}$$

Using (2.16) and (2.18), $t^{min}_{c,s}$ can be computed for each customer c by forward induction on each taxi's sequence of customers. Similarly, $t^{max}_{c,s}$ can be computed by backward induction using Equations (2.17) and (2.19). These forward and backward computations are similar to the *Lazy* and *Eager Scheduling Algorithms* introduced in [10] to build solutions for the dial-a-ride problem, and are linear in the number of pick-ups in the route.

The `local-backbone` algorithm. We use these new time windows as a guide for our exploration process: instead of selecting random pick-up times within $[t^{min}_c, t^{max}_c]$ in the `backbone` algorithm, we select them within $[t^{min}_{c,s}, t^{max}_{c,s}]$. All the arcs generated by `maxflow` will therefore be in a "neighborhood" of solution s, allowing us to improve on the solution while building on the quality of s to limit the search space. This process can be boot-strapped to improve on itself iteratively: we name `local-backbone` this variant of `backbone`, as described in Algorithm 3.

`local-backbone` is an algorithm that combines the advantages of a local-improvement and global optimization. It aims to avoid local-minima by using an MIO solver, and usually provides near-optimal solutions. Its main strength is when the problem is hard to solve or when we have tight constraints on computational time: the difficulty of the problem can be limited by using a very sparse graph $B\mathcal{G}$, and compensate for the corresponding decrease in the solution quality by doing more iterations of `local-backbone` to keep improving the solution as with any local-improvement method. We empirically found that the starting solution does not significantly influence the quality of the convergence: we could not find unsatisfactory local minima

Algorithm 3 `local-backbone`

```
Require:
    The flow-graph $\mathcal{G}$.
    A limit $E_{max}$ on the number of arcs such that MIOoptimal stays tractable.
    A starting solution $s$, that can be empty if none is known.
Ensure:
    A solution $s'$ for the offline taxi routing problem that improves upon solution $s$.

 1: while time is available do
 2:     Compute the values $[t^{min}_{c,s}, t^{max}_{c,s}]$ for each customer $c$ using the solution $s$.
 3:     Create an empty "backbone" graph $B\mathcal{G}$ by removing the arcs of $\mathcal{G}$.
 4:     Add all the arcs of $s$ to $B\mathcal{G}$.
 5:     while $B\mathcal{G}$ has less than $E_{max}$ arcs do
 6:         for each customer $c \in \mathcal{C}$ do
 7:             Generate a uniformly random pick-up time in $[t^{min}_{c,s}, t^{max}_{c,s}]$.
 8:         Use maxflow on $K\mathcal{G}$ with the fixed pick-up times $t_c$.
 9:         Add all the optimal arcs of the computed solution to $B\mathcal{G}$.
10:     Use MIOoptimal to solve the offline taxi routing problem on $B\mathcal{G}$, with $s$ as a
    warm-start.
11:     Update $s$ to be this new solution $s'$.
```

in our applications. Furthermore, we adapted the algorithm to get out of any local optimum, as described in the Remark below.

Remark 1. If we modify slightly `local-backbone` to also add some uniformly random pick-up time (not local) in addition to the local ones, we obtain an algorithm that converges to the optimum. Indeed, at each iteration of Algorithm 3, the non-local pick-up times have a positive probability of being compatible with the optimal solution. If they are, `maxflow` will add the arcs of the optimal solution to $B\mathcal{G}$, and the optimal solver will find the optimal solution.

For large-scale online routing problems, we found that `local-backbone` leads to stronger solutions than `backbone`, as it is able to make the most out of a warm-start when using re-optimization. Furthermore, we have found in our experiments that `local-backbone` outperforms all the other methods we tried by a large margin. We next present computational results and compare `local-backbone` to other offline solvers.

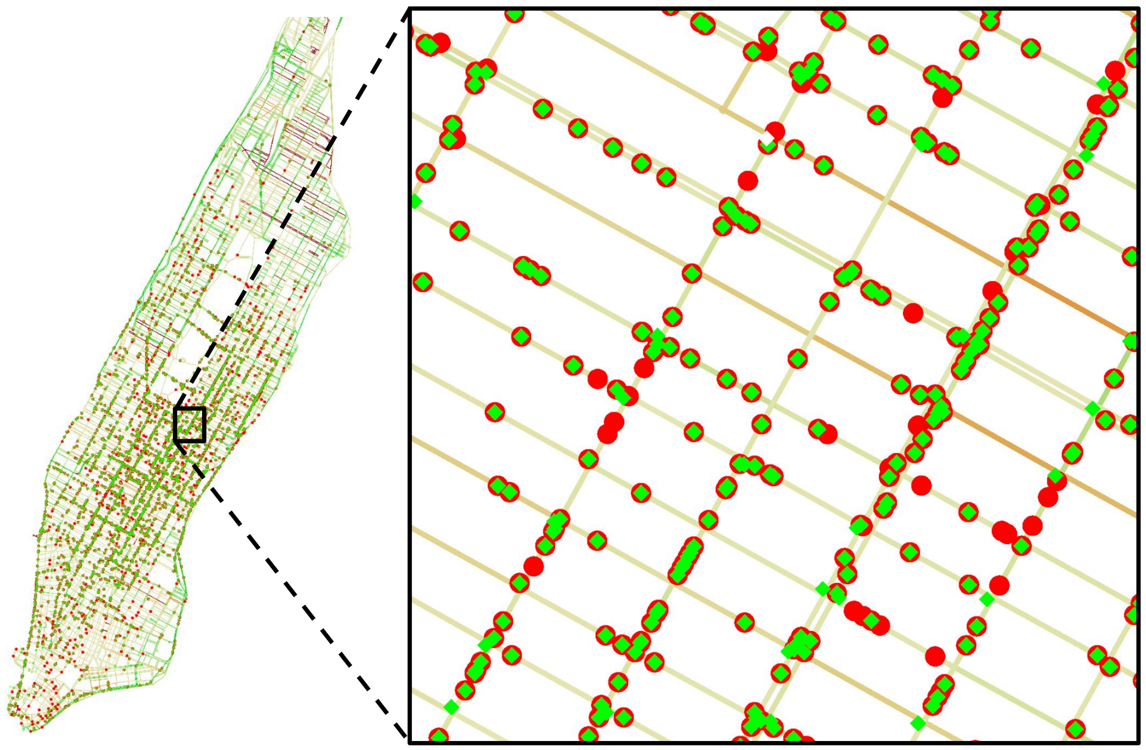


Figure 2-4: A taxi-routing simulation on the Manhattan routing network in NYC. On the left, we show a general view of the routing network, with 4324 intersections and 9518 directed arcs, built using OpenStreetMap data. On the right, we zoom on a detail of the online taxi routing simulation. Each red circle is a taxi, and the green squares represent customers, either waiting or being transported by a taxi. There were more than 26,000 customers and 4,000 taxis for 1.5 hours of simulation.

2.4.4 Taxi Routing in NYC

When studying large-scale vehicle routing problems, synthetic data is not enough to represent complex real-world demand and networks. Therefore, we reconstructed the exact Manhattan routing network in New York City, and used real demand data from NYC Yellow Cabs to build accurate real-world online taxi-routing problems.

Using OpenStreetMap data, presented in [69], we extracted the complete routing network of the island of Manhattan, as represented in Figure 2-4. This large network, with 4324 intersections and 9518 directed arcs, was chosen because taxis and on-demand ride-sharing vehicles are an extremely popular mean of transportation in this city, with more than 500,000 trips everyday. Interestingly, a large fraction of the rides stay within Manhattan from origin to destination: taxi demand data in [102] shows that around 80% of the rides that have a pick-up location in Manhattan stay within Manhattan.

The New York City Taxi and Limousine Commission has released a large taxi-trip data-set that is freely available online, see [102]. We have access to more than a billion trips, all the Yellow and Green Cabs trips for the years 2009-2016. The available information includes their pick-up and drop-off points and times, the fare and tip paid, the number of passengers and more. The volume of the demand is large, with generally more than 400,000 trips a day for Yellow Cabs alone, more than 12 million per month. We focus in this chapter on the yellow-cab rides of Friday 04/15/2016 12-1:30pm from Manhattan to Manhattan, which represent exactly 26,109 customers after removing data errors. This date was chosen purely arbitrarily, though we selected a time of high demand. We adapt the trips to our routing network by projecting the origin and destination of the customers on the nearest intersection. The fare paid by each customer c is used to create the profit parameters $R_{c',c}$, and we set the beginning of the pick-up time window t_c^{min} to be the real pick-up time of the customers. The values of t_c^{max}, $t_c^{request}$, and t_c^{conf} are chosen for each simulation to represent the situations we want to model.

Simulated taxis are added to the network, and we change the number of taxis while keeping the demand constant to control the balance between supply and demand. We typically need a lot less taxis than the real number of Yellow Cabs to serve the same demand, because of our optimized solutions, more centralized control and future planning. Furthermore, we used the very same yellow-cab taxi data to estimate the travel time on all arcs of the routing network, running the algorithm described in [18] on the same demand data we used to create the rides. Therefore, these travel times match the congestion and traffic patterns of the same precise day and time: Friday 04/15/2016 12-1:30pm. Under the assumption that taxis use the fastest route, which is verified in practice for ride-sharing companies as drivers paths are suggested and monitored in real-time by the driver's phone application, we can compute the travel time $T_{c',c}$ for each arc of the graph $\mathcal{G}$. We also subtract a cost of \$5 per hour of driving so that $R_{c',c}$ represents the profit.

Algorithm	Increase in profit, compared to greedy			
	$K = 2$	$K = 4$	$K = 8$	$K = 20$
MIOoptimal	-5.05%	**5.41%**	3.75%	2.28%
local-backbone	-5.74%	4.74%	8.27%	**9.13%**
2-OPT	**4.03%**			

Table 2.2: Comparing offline algorithms on a big routing problem. These results are averaged on five distinct simulations for an offline taxi routing problem in Manhattan with 2700 taxis and more than 6000 customers. The computational time is limited to 5 minutes for each algorithm. To make the problem tractable for the optimization-based methods, we apply `K-neighborhood` to shrink the flow graph $\mathcal{G}$. We show the results for different values of K, highlighting the most favorable case for each method. Our version of `2-OPT` does not use $\mathcal{G}$ and is therefore not sensitive to K.

We created a micro-simulation software able to simulate, optimize and visualize online vehicle routing on real-world networks. This software has provided us a much finer control and better speed than existing software like MATsim. It also enables us to easily interact with any free or commercial solver like CBC or Gurobi, through the Julia for Mathematical Programming (JuMP) interface. Figure 2-4 shows an example of such a simulation in Manhattan.

2.4.5 Offline Results for Large-Scale Taxi Routing

We have introduced new algorithms to scale the offline MIO formulation of taxi routing to real-world demand scenarios. The flow-graph shrinking heuristic `K-neighborhood` implements a trade-off between the tractability of the solution methods and the quality of the solutions. And the `local-backbone` algorithm is our most scalable optimization-based algorithm. In the high-demand scenario with time windows, we want to compare our algorithms to the baseline `greedy` and `2-OPT`. These algorithms are meant to be used in a re-optimization setting when solving the online problem. We study in this section an offline taxi routing problem in NYC that represents a typical iteration of re-optimization for the online problem.

We create an offline scenario, using the online taxi-routing problem presented in Section 2.4.4. We assign each customer a 5 minutes time window and a random request time that is on average 15 minutes prior to their first desired pick-up time, generated randomly uniformly between 0 and 30 minutes. This 15 minutes prior time was chosen to represent a situation with some reasonable prior information available, and we will study in Section 2.5.3 the influence of this prior request time on the quality of the solutions. We add 2700 taxis on the routing network, at random locations following the distribution of customer pick-up location in Manhattan at this time of the day. This number is chosen to represent situations with slightly exceeding demand, for which optimization algorithms are useful. In this context, the greedy heuristic is able to serve 80% of the demand on time.

We consider the offline taxi routing problem corresponding to one step of the re-optimization process: at 12:30pm, with all the customers c such that $t_c^{request} \leq$ 12:30pm. This gives roughly 6000-6500 customers, depending on the random values chosen for $t_c^{request}$. We generate five such random problems by re-generating the request times and the taxi initial positions and average the profits generated by our different algorithms, with a computational limit of 5 minutes for each. The actual time available to solve this problem in practice is 15 seconds, but we give the algorithms more time to compensate for the fact that we do not provide a warm-start and to be able to compare the optimization power of each algorithm. In the next section, we will show how to limit the computational time. The numerical results are presented in Table 2.2.

The optimization based algorithms `MIOoptimal` and `local-backbone` perform better than the nearest-taxi baseline `greedy` and the state-of-the-art local improvement algorithm `2-OPT`: our algorithms scale to real-world taxi routing. `MIOoptimal` managed to find the provable optimal solution within 5 minutes only for $K = 2$. Note that this solution is worse than `greedy`, as $K = 2$ is too small and `greedy` does not operate on the pruned flow graph, and thus gives a better solution. It did not give an optimal solution in the other cases, but generally yielded a good feasible solution. For $K = 2$ and $K = 4$, `MIOoptimal` performs slightly better than `local-backbone`, because of the loss due to the backbone structure. But for larger values of K, `local-backbone` continues to improve and provides higher quality solutions, whereas `MIOoptimal` becomes intractable and fails to find better solutions in the allowed time.

`local-backbone` manages to do better than all the algorithms we tried on offline taxi routing. More than the extra 5% of profit this algorithm generates, we have demonstrated that mixed integer formulations can be used in practice for large-scale vehicle routing by leveraging the "locality" of the decisions. On the other hand, we have only solved one particular iteration of the re-optimization strategy for one particular offline routing problem. We show in the next section how our algorithm performs in the full online setting, and what situations are more favorable for optimization.

2.5 Online Taxi Routing in NYC

In Section 2.3.1, we have introduced a re-optimization strategy to solve online taxi routing. This iterative algorithm requires to be able to solve large-scale offline taxi routing problems within a limit of 15 seconds, which is the limit we have chosen for our applications. We have demonstrated in the previous section that `local-backbone` can be used to get near-optimal solutions to these large offline problems in a tractable way, though not yet respecting this strong time limit. We now show how to respect the 15 seconds limit in practice, and compare our optimization-based algorithms to other online strategies. These algorithms are tested on the New York City taxi routing problem defined in Section 2.4.4 to gain insights on how the increasing connectivity, central control and knowledge of the future demand can be used to better optimize online routing decisions.

2.5.1 Re-optimization and Warm-Starts

Re-optimization involves re-solving the offline taxi routing problem with all the known future customers periodically, at every time-step of length $\Delta t^{update} = 30$ seconds. This frequent re-optimization can be leveraged to reduce the computational time needed at each iteration. We present here our approach to re-optimization in a large-scale real-time setting.

Accelerating Re-Optimization. In the re-optimization strategy, the solution of the offline problem at one iteration can be used to provide the solver with an initial solution feasible for the next iteration. We have discussed in Section 2.4.3 that `local-backbone` and `2-OPT` can improve on any provided initial solutions; our relatively high re-optimization frequency provides good warm-start at each step, which leads to better results when a limited time is available.

Moreover, the previous solution is not the only thing we can build on from the past iterations. Our `local-backbone` algorithm uses the flow graph $\mathcal{KG}$ to represent the problem to solve. Unfortunately, it takes time to construct the graph $\mathcal{G}$ at each iteration, to prune it with `K-neighborhood` as presented in Section 2.4.1, and to convert the resulting problem into a sparse matrix to give to a commercial solver. It actually takes us 10 to 40 seconds to go through these preliminary steps for a problem of the scale of taxi routing in Manhattan. Thankfully, the graph $K\mathcal{G}$ is not too different from one iteration to the next. As new customers appear, we perform an online update on $K\mathcal{G}$, adding new arcs and removing the obsolete ones. This online update is particularly useful because we never have to construct and store the full graph $\mathcal{G}$. To make such an update possible, we keep track of the cost $C(c', c)$ (as introduced in Section 2.4.1) of each arc of the graph $K\mathcal{G}$, and we use a heap data structure that allows us to efficiently keep and update the K-best arcs when new requests come or old requests become obsolete. Thus, we update the pruned flow graph $K\mathcal{G}$ in-place at each iteration, without reconstructing and pruning the full graph $\mathcal{G}$. This in-place update of the graph and of the corresponding sparse matrix that we send to the solver, is what we call a *formulation warm-start*. In practice, formulation warm-start allows us to create $K\mathcal{G}$ in one to two seconds when the formulation of the previous iteration is available, instead of half a minute at each iteration.

Parallelization is also useful in practice, particularly to accelerate `local-backbone` and when using a solver to perform a branch-and-bound on the MIO formulation. Indeed, the exploration phase of the backbone algorithm can be computed in parallel, as discussed in Section 2.4.2.

The Online Re-Optimization Strategy The online re-optimization strategy periodically re-optimizes its assignments of future customers to taxis, sends the taxi routing decisions to the vehicles, receives the vehicles status, and processes the customer requests. We list all the steps of one iteration of our implementation of re-optimization:

1. Gather the new taxi actions since the last update, and all the new customer requests.

2. Compute the new pruned flow graph $K\mathcal{G}$: we update the one from the previous iteration to the new situation. More specifically, we add the new requests and we remove the completed picked-ups and the rejected customers. This is done while maintaining the K-sparsity property of $K\mathcal{G}$. This step corresponds to the "formulation warm-start" discussed above.

3. Update the offline solution of the previous iteration to make it feasible for the new formulation. Specifically, mark the new customers as being rejected and remove the decisions that have already been implemented.

4. Solve the optimization problem with `local-backbone`, using the formulation and the warm-start constructed in Steps 2 and 3. A solution must be provided in less than $\Delta t^{update} = 30$ seconds, but we use 15 seconds to keep a security margin and leave time to broadcast the actions to the fleet.

5. If we have reached the confirmation time t_c^{conf} of a customer c, look at the customer status in the current solution. If the customer is rejected, communicate this information to the customer, or offer her to wait for another confirmation time. In the examples of this chapter, we will reject the customer. If the customer is accepted, make sure that she will be accepted in all future iterations. A simple way to do so is to add the constraint $p_c = 1$ to the MIO formulation presented in Section 2.3.2. This does not break the network-flow structure of the problem and makes sure that customer c is picked-up in all feasible solutions.

6. Send the taxis all the routing actions that occur before the next update. Specifically, we dispatch a taxi to a customer pick-up location so that it reaches the customer at the earliest pick-up time compatible with the new solution, which is $t_{c,s}^{min}$ as defined in Section 2.4.3. Taxis are not aware of the full offline schedule, as it can change in the next re-optimization iterations.

7. Idle taxis are instructed to wait at their current position. Note that other behaviors could be chosen instead, for example using forecast demand to route the idle taxis, or just let them move as they want. We do not study these choices in this chapter.

2.5.2 Online Solution Methods

To evaluate the performance of our re-optimization strategy with `local-backbone`, we created a set of reference large-scale online algorithms that will serve as a baseline to evaluate our work.

Pure online algorithm. Our simplest algorithm, `pure-online`, does not use the customer prior request information, pretending that $t_c^{request} = t_c^{min}\ \forall c \in \mathcal{C}$. At time t_c^{min}, this algorithm will send the nearest available taxi to the customer. The taxi needs to be able to pick-up c before t_c^{max}, but does not have to be idle at t_c^{min}. This myopic algorithm is not too different from real taxi behavior, that will look for a customer in their neighborhood, or from ride-sharing for-hire vehicles, that will be matched with nearby requests. Therefore, `pure-online` will be used as a baseline to outline the extra efficiency other algorithms gain from more optimization and prior knowledge of the demand.

Planning with no re-optimization. `no-reopt` is a greedy algorithm that uses prior request knowledge to plan ahead and find better solutions, but does not re-optimize. We maintain a list of future assignments for each taxi. When a new customer requests a ride for a specific pick-up time window, we check if we can insert it in the lists of customers assigned to each taxi. If it is not possible, we reject the customer. If we can, we assign it to a taxi chosen such that this new assignment maximizes the total profit, using the efficient insertion algorithm described in Appendix 2.7.1. Therefore, `no-reopt` takes into account the future positions of the taxis when making these decisions, though the decision cannot be changed once a customer is assigned to a taxi. This is different from the re-optimization process described in Section 2.5.1, that can re-assign a customer to another taxi as new information about the future is revealed, and only decides the final action when it is time for pick-up.

Optimization-based updates. The `backbone` online algorithm is the re-optimization process described in Section 2.5.1. This algorithm uses `local-backbone` to perform the updates and is limited to 15 seconds of computation per iteration.

Heuristic-based updates. The `2-OPT` online algorithm is the adaptation of its offline counterpart to the online setting. We use a re-optimization process similar to the one presented in Section 2.5.1, removing the flow-graph computations and replacing the offline solution method `local-backbone` by `2-OPT`. We use the warm-start solution from the previous iteration, and we limit the algorithm to 15 seconds of computation. This algorithm uses all available prior information, allows for re-optimization and performs typically well in practice.

2.5.3 Experiments and Results

We apply our online algorithms to the taxi routing problem presented in Section 2.4.4. The confirmation time t_c^{conf} for each customer c is chosen to be a maximum of 3 minutes after the request time $t_c^{request}$. To study the impact of prior customer knowledge, we vary the customer request time. Let $T^{request}$ be the desired average time of prior request. We assign each customer c with a random request time $t_c^{request}$ drawn uniformly within the interval $[t_c^{min} - 2T^{request}, t_c^{min}]$. The randomness of the request times is important: for example, if each customer c were to request a ride at the non-random time $t_c^{request} = t_c^{min} - T^{request}$, the request times would be ordered by pick-up times, which is not real-world behavior. The customer time window length is the same for each customer: we assign each customer with a time window of length T^{wait}, with $t_c^{max} = t_c^{min} + T^{wait}$. To control the supply-demand balance, we vary the number of taxis while keeping the customers constant.

As discussed in Section 2.4.4, our algorithms are implemented in the Julia language, with a special care for computational speed and visualizations. Their parameters were all optimized to get the best results. We created a framework allowing us to test the different online strategies in the same environment, making sure that we only share the requests information in real time. All simulations are run on identical machines, using 2 CPUs and 8GB of memory. Each simulation presented in this section was done over a time period of 1.5 hours, as we simulated vehicle routing in Manhattan for the real yellow cab demand of Friday 04/15/2016 12:00-1:30pm. Figure 2-4 is an example of visualization created by our simulation software, during an online simulation. These visualizations have proved to be extremely helpful to understand the algorithms behavior, to compare their results and to develop a good intuition of the problem, and ultimately to design the `backbone` online algorithm.

Figure 2-5 shows how prior information influences the different online algorithms, in a high demand scenario with 5 minutes time windows. `backbone` performs significantly better than `2-OPT`, and the similarity of the two curves confirms the similarity of the two re-optimization approaches. The extra percents of profit, from 1% to 3.5% between these two methods are significant in practice, as they represent hundreds of additional customers that have been served thanks to optimization. The sharp increase of profit for the first few additional minutes of prior request time at the beginning of the curve is experienced by all online methods using prior information. It is explained by the additional time available to dispatch taxis to customers that are further away, and that `pure-online` cannot pick-up because the 5 minutes time window is too short. Nonetheless, `no-reopt` plateaus when more information is available, and cannot use the increasing prior request time to make better decisions. This is typically the situation in which re-optimization is important: due to high demand, all the `no-reopt` taxis are assigned to customers and we cannot accept new ones.

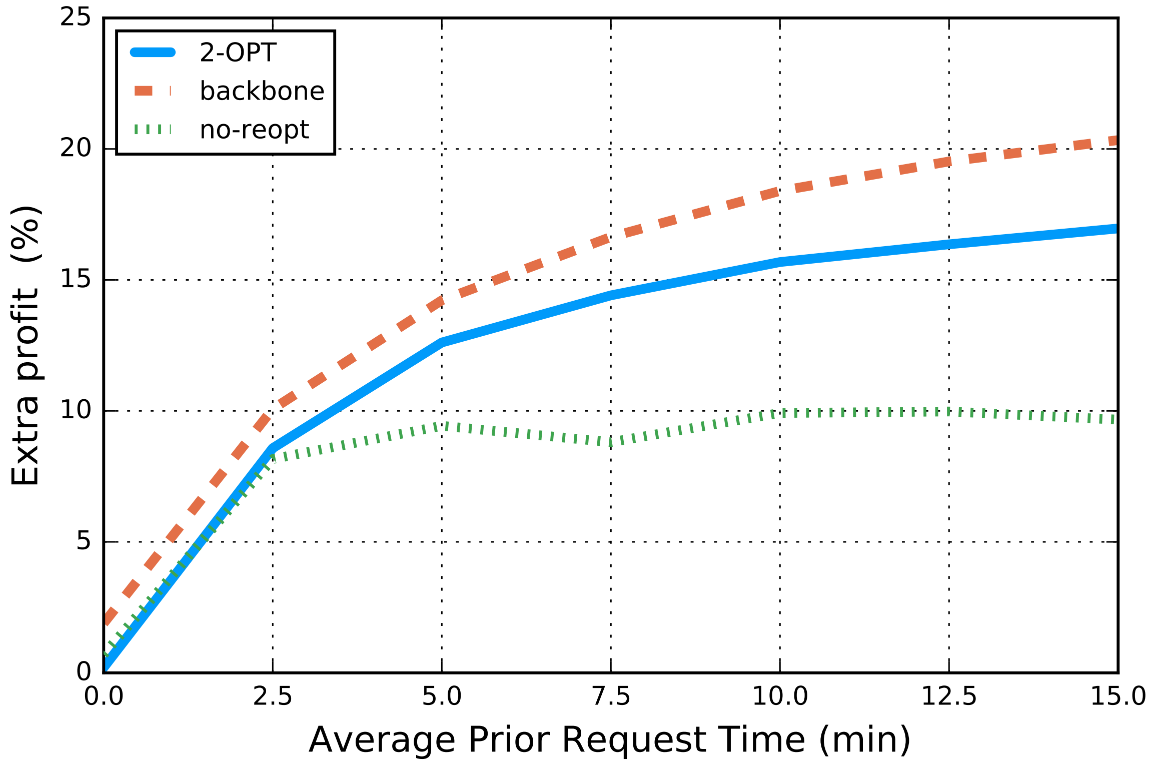


Figure 2-5: Varying the mean prior request time. Increase in profit of each online taxi routing algorithm compared to `pure-online`. We vary the mean prior request time $T^{request}$ from 0 (pure online situation) to 15 minutes. Each customer is assigned to a $T^{wait} = 5$ minutes time window. We control 4000 taxis, which corresponds to a high demand scenario as 80% of the demand is served in the best case.

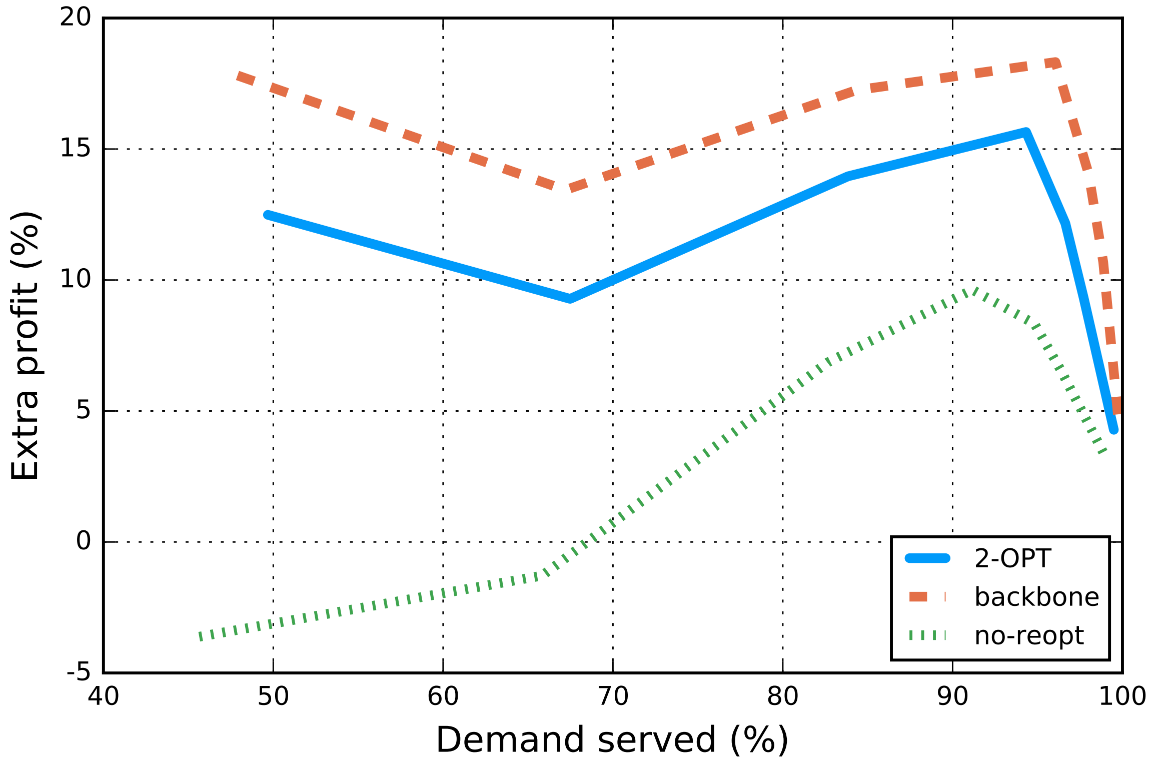


Figure 2-6: Varying the supply-demand balance. Increase in profit of each online taxi routing algorithm compared to `pure-online`. We vary the number of taxis from 2000 to 10000, and represent on the x-axis the corresponding fraction of customers served by the different algorithms. The time windows have a length of $T^{wait} = 5$ minutes and the mean prior request time is $T^{request} = 15$ minutes.

On the other hand, re-optimization allows the option to reorganize the assignment of customers to taxis in order to be able to pick-up more customers, more efficiently. The surprising finding is that not a lot of prior information is needed in order to make better decisions: asking customers to request for a ride 10 minutes beforehand already allows for an 18% increase in profits.

Figure 2-6 shows how the balance between supply and demand influences the results of our algorithms. We have showed in Section 2.3.4 that optimization-based algorithms and `2-OPT` have a strong edge on their greedy counterpart when demand was high. These results confirm this observation in an online setting: when the served demand is below 95%, we do not have enough taxis to serve the high demand. Thus `2-OPT` and `backbone` perform significantly better than the greedy algorithms `no-reopt` and `pure-online`, and `backbone` clearly outperforms `2-OPT`. We have found

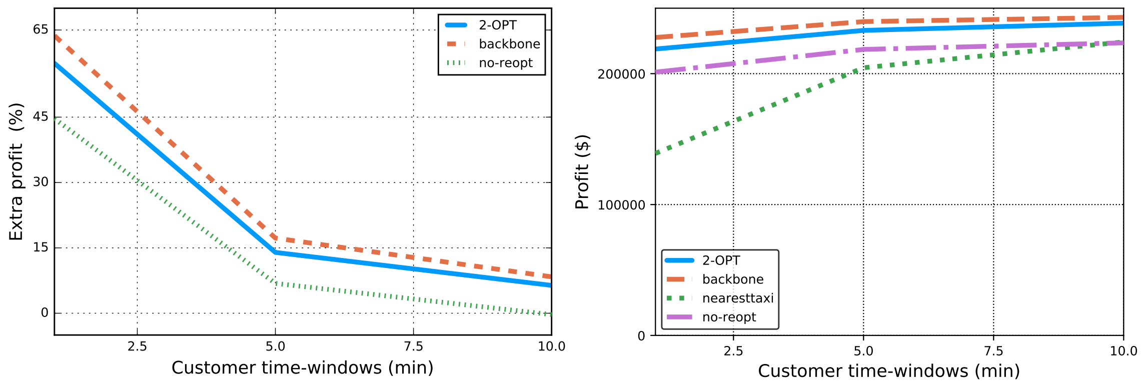


Figure 2-7: Varying the time windows size. Increase in profit of each online taxi routing algorithm compared to `pure-online` (left) and in absolute profit values (right). We vary the size T^{wait} of the customer time windows: $T^{wait} = 1$ minute, $T^{wait} = 5$ minutes and $T^{wait} = 10$ minutes. There are 4000 taxis, which corresponds to a high demand scenario, and the mean prior request time is $T^{request} = 15$ minutes.

that problems with more demand than available taxis (in this case with less taxis) are generally harder to solve by offline solution methods in the re-optimization process. Given our limited computational time, this difficulty reduces the quality of the solutions found in the allowed time. This phenomenon is illustrated by the loss of performance at around 70% of served demand: this problem was the hardest to solve and there was not enough time given to the solution methods to find near-optimal solutions. When demand is low, and the served demand is close to 100%, taxis are generally mostly idle, and a greedy algorithm like `no-reopt` performs almost as well as the optimization-based algorithm. This confirms the insight we gained in Section 2.3.4 that problems with low demand are easy to solve and do not require re-optimization.

Figure 2-7 shows the impact of the time window length on the quality of the different solution methods. We also represented the profit values on the right to give a sense of scale. The sharp decrease in relative profit of the online algorithms in comparison to `pure-online` is actually due to an increase in quality of the `pure-online` solutions, which masks the fact that all strategies give better results with larger time windows. For $T^{wait} = 1$ minute, `pure-online` does not manage to pick-up customers when there is no free taxi in the close vicinity, and performs really poorly. Interestingly Figure 2-7 illustrates that `no-reopt` is no better than `pure-online` for very large time windows, which makes sense as the two greedy heuristics are almost equivalent in this setting. The prior information accessed by `no-reopt` is not useful when the time windows are long enough. As a consequence, the extra 10% of profit obtained by `backbone` for $T^{wait} = 10$ minutes is only due to the edge of re-optimization over greedy algorithms, as revealed in Section 2.3.4. Even with large time windows, re-optimization methods are significantly better than `pure-online` when some prior request information is available. Moreover, even if `pure-online` manages to use the large time windows to pick-up more customers, these pick-ups are generally later than the three other algorithms, giving them a strong edge in practice for customer satisfaction. In general, large time windows are better represented using soft time windows constraints, penalizing the delay.

We have compared all our results to the `pure-online` profit, as it is representative of typical taxi system greedy strategies. This empirical study shows that using optimization based strategies on today's relevant large-scale vehicle transportation systems can have a serious impact on their performance, particularly in the daily situations of peak demand. Furthermore, our experiments suggest that these systems should give incentives to customers to request their trips few minutes in advance. Customers flexibility in pick-up time should also be used as much as possible, and time windows could be personalized for each customer, with an incentive to accept a larger one.

2.6 Conclusions

2.6.1 Extensions

Using historical data, it is possible to accurately forecast the demand in large-scale settings and use it to route idle taxis to areas of popular demand. We did not use this in our application, but such an extension can improve the system efficiency, especially when there is a large cyclical demand in far-away locations like airports. Another way to use historical and real-time data is to provide online estimate of the travel times.

In our applications in NYC, we have estimated the travel times from data, under the assumption that they are stationary for the time of the experiment. In practice, travel times can be re-estimated at each step of the re-optimization process.

We used the assumption of full control of the vehicles, as we expect that vehicle control will become increasingly centralized in the future. However, the re-optimization framework can be adapted to be more of a recommendation system, suggesting customers to drivers, and updating the planning at each iteration given the vehicles actual moves. More generally, this framework is also suitable to other real-time vehicle routing applications. As our algorithm use a mixed-integer optimization at the core, we could add extra operational constraints to represent situations as diverse as cargo ship routing, on-demand private jets, bus renting, electric vehicles, self-driving taxis, car-pooling and more.

2.6.2 Impact

Our contributions surpass the scope of this chapter in two ways:

First, the core ideas of our main algorithms `K-neighborhood`, `backbone` and `local-backbone` are not specific to taxi routing and can be applied to other large-scale decision problems of vehicle routing and operations research. The core idea of a "backbone" is that some decision variables do not vary too much across almost all near-optimal solutions, and that identifying them can significantly accelerate the optimization process. This idea can be applied in a variety of situations, and is much more general than taxi routing. For example, [124] presents a backbone algorithm for the TSP, though formulated as a greedy heuristic. The part of a backbone algorithm that depends on the application is how to generate "good" and varied feasible solutions in a cheap way: we use `maxflow` for our taxi routing application. `local-backbone` goes even one step further: if it is too expensive to construct the problem backbone, one can do it iteratively, at each step constructing a local backbone around the current best solution to improve on it. This general algorithm has the advantage of combining global-optimization to avoid local extrema and local-improvement for tractability.

Additionally, the software we have built and released (see [94]) is able to simulate and visualize online and offline vehicle routing problems with synthetic or real-world routing data, using real or generated demand data. Being able to simulate real-world vehicle routing, our framework and algorithms can solve problems that are relevant to the industry. For example, the insights we get about the value of future information can be of immediate practical interest for current urban transportation companies.

2.7 Appendix

We present in this appendix the details of our offline greedy and local improvement heuristics. We use these algorithms as a baseline to evaluate the effectiveness of our optimization-based algorithms.

2.7.1 Insertions and Greedy Heuristic

Given a solution s to the offline taxi routing problem and a customer c that is rejected in this solution, an *insertion* of c into s is the process of finding a taxi that is able to pick-up c without modifying the rest of the solution. For example, if a taxi in s is supposed to serve customers $c_1, c_2, \dots, c_n$ in this order, inserting customer c at position k corresponds to modify s so that the taxi serves customer $c_1, \dots, c_{k-1}, c, c_k, \dots, c_n$, under the condition that the solution is still feasible.

The most important thing when inserting a customer is to be able to check the feasibility of the insertion, given the pick-up time window constraints. It is possible to do this in a very efficient way: given a solution s and a customer c, let $[t_{c,s}^{min}, t_{c,s}^{max}]$ be the interval of possible pick-up times t_c such that s is still feasible. These times are defined in Section 2.4.3 and can be computed quickly using forward induction. We can use them to quickly check the feasibility of inserting a customer. If we want to insert customer c within taxi k's schedule, we can compute the feasible time windows using the induction equations (2.16)-(2.19). For example, to insert customer c between c_{k-1}

and c_k, we first compute its values $t_{c,s}^{min}$ and $t_{c,s}^{max}$ using equations (2.16) and (2.17):

$$t_{c,s}^{min} = \max(t_c^{min}, t_{c_k,s}^{max} - T_{c,c_k}) \tag{2.20}$$

$$t_{c,s}^{max} = \min(t_c^{max}, t_{c_{k-1},s}^{min} + T_{c_{k-1},c}) \tag{2.21}$$

and the insertion is only feasible if the pick-up time window is non-empty, i.e, $t_{c,s}^{min} \leq t_{c,s}^{max}$.

For each possible insertion, we can compute the difference in profit ΔR in the new solution after insertion. For example, when inserting c between c_{k-1} and c_k, we have:

$$\Delta R = R_{c_{k-1},c} + R_{c,c_k} - R_{c_{k-1},c_k} \tag{2.22}$$

We can now use these insertions in an iterative way to describe the `greedy` heuristic introduced in Section 2.3.2:

1. Create an “empty” solution s, in which all customers are rejected and all taxis idle.

2. Order all the customers by minimum pick-up time t_c^{min}, and apply the next steps to each one, sequentially.

3. Given a customer c to insert, try to insert it in each taxi using the feasibility rules described in this section with the values $[t_{c,s}^{min}, t_{c,s}^{max}]$ of the other customers.

4. If no inserting position is feasible, reject the customer.

5. If inserting the customer is feasible, select the taxi and the position that yield the highest difference in profit ΔR, and insert the customer.

6. Update the values $[t_{c,s}^{min}, t_{c,s}^{max}]$ for all the customers that are assigned to the taxi chosen for the insertion, using equations (2.16)-(2.19).

Inserting the customers by order of t_c^{min} performs typically really well, and is very close to the nearest-taxi strategy, as each customer will be inserted at the end of a taxi’s schedule, usually the taxi that is the closest to the customer.

2.7.2 Local-Improvement and 2-OPT

Let s be a solution to the offline taxi routing problem, a local improvement is a solution s' that is in a "neighborhood" of s, such that the total profit of s' is higher than the profit of s. A simple yet powerful definition of such a neighborhood is the *2-OPT* neighborhood. We perform a swap between two nearby taxis, exchanging their assigned customers. For example if taxi 1 is picking up customers c_1^1, c_2^1, c_3^1 and taxi 2 is picking-up customers c_1^2, c_2^2, c_3^2, swapping customer c_2^1 and c_2^2 (together with the subsequent customers) could result in assigning c_1^1, c_2^2, c_3^2 to taxi 1 and c_1^2, c_2^1, c_3^1 to taxi 2. Formally, we execute the following algorithm:

1. Given a solution s, choose a customer c that is already assigned to taxi k, let $c_1^k, \ldots, c_n^k$ be the customers assigned to k whose pick-up times are after customer c. Let also c_{-1}^k be the customer coming immediately before c in taxi k's schedule.

2. Select another taxi k'. Let customer c' be the first customer of k' such that $t_{c',s}^{min} + T_{c',c} \leq t_{c,s}^{max}$. In other words, c' is the first customer assigned to k' such that k' can serve all its customer preceding c', followed by $c', c, c_1^k, \ldots, c_n^k$.

3. Let $c_1^{k'}, \ldots, c_n^{k'}$ be the customers assigned to k' whose pick-up times are after customer c' in solution s. Remove these customers from k', and assign $c, c_1^k, \ldots, c_n^k$ to k' after c' instead.

4. Find the first customer $c_i^{k'}$ of the sequence $c_1^{k'}, \ldots, c_n^{k'}$ such that $t_{c_{-1}^k,s}^{min} + T_{c_{-1}^k, c_i^{k'}} \leq t_{c_i^{k'},s}^{max}$. In other words, find the longest sub-sequence $c_i^{k'}, \ldots, c_n^{k'}$ such that all these customers can be inserted at the end of taxi k's schedule, immediately after customer c_{-1}^k, while respecting the pick-up time windows.

5. Assign customers $c_i^{k'}, \ldots, c_n^{k'}$ to taxi k. And reject the customers $c_1^{k'}, \ldots, c_{i-1}^{k'}$ that we could not insert.

6. At this point of the swap, taxi k schedule is now $\ldots, c_{-1}^k, c_i^{k'}, \ldots, c_n^{k'}$ and taxi k' schedule is now $\ldots, c', c, c_1^k, \ldots, c_n^k$. Customers $c_1^{k'}, \ldots, c_{i-1}^{k'}$ are rejected.

7. Use the insertion algorithm described in Section 2.7.1 to try to insert all the customers that were rejected in s into k and k' schedules, the only two taxis that we have modified.

8. Also use the insertion algorithm to try to insert the newly rejected customers $c_1^{k'}, \dots, c_{i-1}^{k'}$ in all taxis schedule.

9. We have built our final solution s'. Compute its profit and compare it with the previous one.

This construction of a new solution may seem elaborate, because of its need to respect the time windows feasibility. However, it is in practice very fast as it only modifies a small sub-part of the solution. Steps 1. and 2. are the two most important, as we choose the two taxis and customers on which we will perform the swap. To make it tractable on a large-scale such as our application in Manhattan in Section 2.5.3, we use the costs described in Section 2.4.3 to smartly choose good potential swaps. In practice, we were able to perform 10,000 swaps per minute in the large-scale online taxi problem in NYC introduced in Section 2.4.4.

We use these swaps to perform a local-improvement descent, only accepting a 2-OPT swap when the profit is improved, as described here:

1. Begin with a solution s as given by `greedy`.

2. Perform a 2-OPT swap on s. If the profit is improved, update s to be this new solution.

3. If there is time left, go back to Step 2.

We call this offline algorithm `2-OPT`. Note that all solutions s in this algorithm share the invariant that no customer rejected in s can be inserted in s. Indeed, `greedy` respect this invariant, and steps 7. and 8. make sure that we try all new insertion possibilities at each swap. On small instances of taxi routing (less than a few hundred customers), we have noticed that `2-OPT` tends to converge very fast to a locally optimal solution. In large cities with thousands of customers, we usually do not have enough time to reach a locally optimal solution. The algorithm is slowed down by the high dimensionality of the routing problem, though it manages to significantly improve the solutions quality. This is a sign that more complex local-improvement algorithms, like `3-OPT` modifying 3 taxi's schedules at a time, could not really help with large-scale problems, as we do not even have enough time to sufficiently explore the `2-OPT` neighborhood. The same applies for more complex global-local algorithms like *Tabu-search*.

Chapter 3

Travel Time Estimation in the Age of Big Data

3.1 Introduction

In today's increasingly dense urban landscapes, traffic congestion is an ever more prevalent issue. As flows of goods and people increase, billions of dollars in potential savings are at stake, making the understanding of traffic patterns a major urban planning priority.

A main goal of traffic studies is *travel time estimation*, which in the broadest sense consists of evaluating the time necessary to travel from any origin O to any destination D. This goal is difficult to achieve because travel times depend on a range of effects at different timescales, from the structure of the network (number of lanes on each road, speed limit, etc.) over the long term, to the state of congestion of the network over the medium term, to a host of small random events (missed lights, etc.) over the very short term. Because of the sheer number and diversity of these sources of uncertainty, most general approaches to travel time estimation consist of finding parameters that describe a distribution or set of distributions from which the travel time from O to D is sampled.

Travel time estimation is often combined with the related goal of *routing*, especially in the short to medium-term planning case. In this case, the goal is to evaluate the time necessary to travel from O to D and find at least one path that drivers could use to achieve this estimate, relating the travel time estimate to interpretable network properties. In this chapter, we present a novel method to estimate typical travel times for each road in a city network using taxi data, thus providing reasonable paths and total trip time estimates for any origin and destination in the network.

3.1.1 The Need for a Generalized Approach to Travel Time Estimation

The problem of inferring traffic patterns from diverse measurements is a fundamental step behind the resolution of many complex questions in transportation and logistics. A simple cost function on the individual arcs of the network can often form a building block of a more complex network study, such as recent work by [114] presenting a novel understanding of resilient networks. Furthermore, many network problems specifically require a travel time estimate for each arc: for instance, [101], who develop a new model for traffic assignment that takes into account network uncertainty, present an approach starting from a prior estimate of the expected travel times of individual arcs in the network. Even in examples such as the aforementioned work or that of [77], both of which generally consider travel time to be a stochastic quantity, a good estimate for the network travel times is a valuable asset in order to define a prior or an uncertainty set for this uncertain quantity, laying the groundwork to answer more complex questions about the network.

In a real-world setting, there are different ways to obtain traffic data in a network, each leading to different travel time estimation methods. A popular approach uses fixed detectors that provide information about traffic at particular points in the network, most commonly loop sensors as in [38], or more advanced methods as in [87] that exploit communication between sensors to identify the same vehicle at different locations. Another popular approach, as in [79], uses so-called “floating-car" data, where GPS-equipped vehicles record their location and speed at fixed time intervals, which can range from a few seconds to a few minutes. The path followed by the vehicle between “pings" of the GPS device can be inferred in a variety of ways, from probabilistic models in [74] to tensor decomposition in [136].

A third area of study involves easily gatherable “origin-destination" (OD) data, that only records the time and location at the beginning and at the end of a trip, as, for example, collected by taxis or cellphone towers. Logging this data instead of high-density floating-car data increases the privacy of the taxi driver and passenger because the details of the followed route are not recorded. It also treats the network as a black box, only making measurements when the user enters and exits. OD data can be gathered for different purposes, and the methods we develop here in the context of vehicle traffic can be extended to other types of networks, including railways, subways, and bicycle and pedestrian networks (see recent studies such as [70]), or combinations of such networks. Nevertheless, this generality makes the travel time estimation harder: the problem of simultaneously determining paths and travel times based on origin-destination data only is close to the Inverse Shortest Path Length Problem (ISPL), an NP-hard problem which has also received some attention by [76].

In recent years, the New York City Taxi and Limousine Commission ([102]) has maintained a complete public record of yellow cab rides within the city. The database contains relevant metadata such as the origin, destination, fare, distance and time traveled for over 170 million rides per year, and has been exploited for a variety of purposes, as shown in [143]. Despite the data's size and availability, however, it has not been used very much for travel time estimation. [134] develop a machine learning method based on k-nearest neighbors matching, while [122] describe a very simple smoothing heuristic. Meanwhile, [147]'s more model-oriented approach develops a full probabilistic path selection scheme.

3.1.2 Our Contributions

The main contribution of this chapter is a tractable methodology to solve the travel time estimation and routing problem in a real-world setting on a large network, which has a number of desirable properties.

First, we use very few assumptions about the data: to provide travel time estimates, we only ask for a set of trips within a known network for which an origin, a destination and a travel time are recorded. In particular, we do not require information about the demand structure in the network. We design a simple static model of traffic based on shortest path theory. This simplicity allows us to develop a multipurpose network optimization method that can leverage large amounts of high-variance origin-destination data to build an estimate of city travel times that is accurate both in and out of sample. Moreover, this method is general enough to be able to handle other sources of data, including floating cars and loop sensors.

Furthermore, the method also recovers interpretable city traffic and routing information from this potentially noisy and incomplete data. We estimate a single parameter for each edge, which enhances the interpretability of the results (see Figure 3-1 for an example). In order to avoid overfitting, particularly in regions of the city where little data is available, we add a simple regularization term to the model. The method provides insight on traffic patterns at the scale of a few city blocks, as well as at the scale of the entire network, and also allows us to quickly find viable paths associated with our travel time estimates.

Solving this estimation problem to optimality at an impactful scale is generally intractable. For this reason, we develop a novel iterative algorithm that provides good solutions, by solving a sequence of large second-order cone problems (SOCPs), which modern solvers can tractably handle. We verify the accuracy of this algorithm in a variety of settings and show that it provides high-quality solutions. The method is tractable, determining the typical paths and travel times in the 4300-node Manhattan network over a three-hour time window in under 20 minutes.

In Section 3.2, we formulate an optimization problem that gives both accurate origin-destination travel time estimates and interpretable link travel times and routing paths. In Section 3.3, we introduce an iterative algorithm that can compute solutions to this optimization in large-scale settings. In Section 3.4, using synthetic data we show that the solutions of this algorithm are near-optimal and that the simplifications we made for tractability did not impact accuracy and interpretability. In Section 3.5, we show that this also extends to real-world situations and we present results on

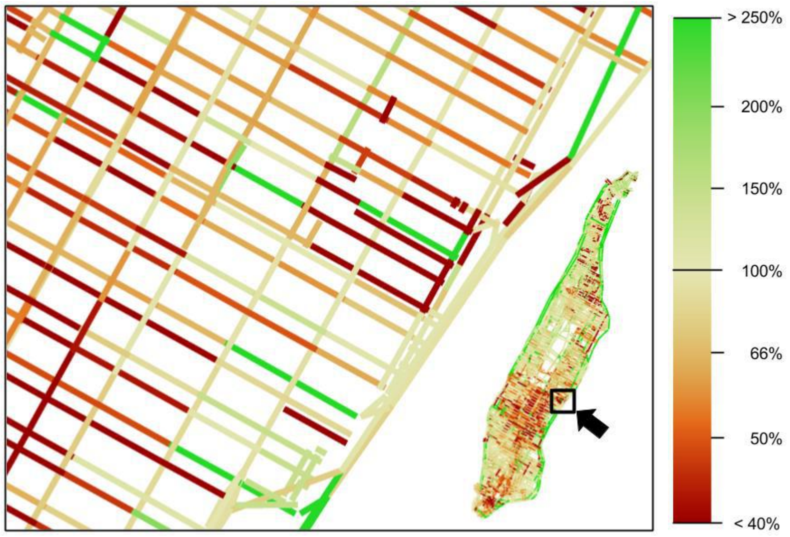


Figure 3-1: Close-up of Manhattan with the arc travel times estimated by our method between 9 and 11 AM. The color of each arc represents the speed along that arc as a percentage of the reference velocity $v_0 = 13.85$ kph (average velocity in Manhattan on weekdays). We can identify traffic effects at the scale of the city (Midtown congestion) and at the scale of a single street (the ramp onto the highway on the eastern shore of Manhattan is congested).

Manhattan taxi data.

3.2 Methodology

In this section, we define the probabilistic setting of travel time estimation, and introduce a simple traffic model that leverages the knowledge of the routing network to represent travel time estimates in the lower dimensional space of network arc travel times. This allows us to create an optimization formulation that uses origin-destination data to build an interpretable image of the network travel times, and at the same time provide accurate travel time estimates.

3.2.1 Problem Statement: Estimating Travel Times From Data

Data. We consider a road network, represented as a directed graph $G = (V, E)$. On this graph, we are given a data set of origin-destination travel time values in the network, of the form (o, d, T) with $(o, d) \in V \times V$ the origin and destination nodes and $T \in \mathbb{R}^+$ the corresponding observed travel time. Data corresponding to this general description can be obtained in many different ways. For example, the set of observed travel times for taxi trips that started between 12pm and 1pm on 2016 Wednesdays in Manhattan would be a valid example of such a data set, as would the set of stop-to-stop travel times for Boston school buses in the academic year 2016-17.

Some origin-destination pairs have several travel time observations in the data set, while others have none. We can therefore define $W \subset V \times V$ as the subset of origin-destination pairs for which we have data: for each $(o, d) \in W$ we are given travel times $\{T_{od}^i\}_{i=1}^{n_{od}}$, realized on n_{od} distinct trips from o to d.

Probabilistic Setting. We would like to estimate the times of trips that are "similar" to the trips that are given in the data set, but may not have been observed in the data. In other words, given any origin-destination pair $(o, d) \in V^2$, we would like to provide a point estimate $\hat{T}_{od}$ of the time it takes to go from o to d. To properly define these estimates for all origin-destination pairs, we describe a simple probabilistic setting.

Each observation of the data-set is assumed to be independently sampled from the same probability distribution. This sampling process goes as follows: the origin and destination nodes (o, d) are sampled from a discrete distribution $\mathcal{D}$ in V^2. Then, conditioned on having an origin o and a destination d, the observed travel times $\{T_{od}^i\}_{i=1}^{n_{od}}$ are assumed to be sampled independently from the distribution $\mathcal{D}_{od}$. Note that $\mathcal{D}_{od}$ can be different for each (o, d). We assume that our data set was built by following this sampling process, but that we do not know the distributions $\mathcal{D}$ or $\mathcal{D}_{od}$. We will hold these probabilistic assumptions to be true throughout this chapter, including our experiments on synthetic data in Section 3.4. In Section 3.5, we will show that our results extend to real-world data that does not necessarily verify our probabilistic assumptions.

We want to obtain a point estimate $\hat{T}_{od}$ of the distribution $\mathcal{D}_{od}$ for every pair $(o, d) \in V \times V$. Specifically, we would like to estimate the geometric mean of the distribution $\mathcal{D}_{od}$: $\exp(\mathrm{E}_{T_{od} \sim \mathcal{D}_{od}}[\log(T_{od})])$. We choose the geometric mean instead of the standard mean because we think that the quality of travel time estimations is perceived on a multiplicative rather than an additive scale, as we discuss in the next paragraph.

To understand the choice of estimating the geometric mean, note that the geometric means of all the distributions $\mathcal{D}_{od}$ are estimates that minimize the overall mean squared log error (MSLE) :

$$\begin{aligned}
\mathrm{MSLE}((\hat{T}_{od})_{(o,d)\in V^2}) &= \mathbb{E}_{(o,d)\sim\mathcal{D},T_{od}\sim\mathcal{D}_{od}}\left[\left(\log(\hat{T}_{od})-\log(T_{od})\right)^2\right] && (3.1)\\
&= \mathbb{E}_{(o,d)\sim\mathcal{D},T_{od}\sim\mathcal{D}_{od}}\left[(\log(T_{od})-\mathbb{E}_{T_{od}\sim\mathcal{D}_{od}}\left[\log(T_{od})\right])^2\right] && (3.2)\\
&+ \mathbb{E}_{(o,d)\sim\mathcal{D}}\left[\left(\log(\hat{T}_{od})-\mathbb{E}_{T_{od}\sim\mathcal{D}_{od}}\left[\log(T_{od})\right]\right)^2\right]. && (3.3)
\end{aligned}$$

Note that the expectations are taken with respect to the distributions $\mathcal{D}$ and $\mathcal{D}_{od}$. The MSLE decomposes into the mean log variance of the data (3.2) which is independent of our estimates, and the mean squared log bias (MSLB) (3.3) which is a measure of the distance of each estimate $\hat{T}_{od}$ from the geometric mean of $\mathcal{D}_{od}$. Using the MSLE implies that an estimate that is twice an observed value is equally bad as an estimate that is half of it. Additionally, a 30-second estimation error is a lot worse for a trip that last 2 minutes than for a trip that lasts 15 minutes. This is what we want and why we chose the log scale and geometric mean estimates.

Model. In practice, we do not observe all the possible (o, d) pairs, which makes it hard to estimate the geometric mean of $\mathcal{D}_{od}$ using only the data that has origin o and destination d. Nonetheless, the estimates of the distributions $\mathcal{D}_{od}$ are typically related: for example, a trip from o to d and a trip from o' to d where o and o' are geographically close will have similar travel time estimates. Therefore, we leverage the network structure by introducing parameters t_{ij} that represent the typical travel time along any arc $(i, j) \in E$, and use them to compute our estimates $\hat{T}_{od}$.

We define a path P_{od} from o to d as a series of consecutive arcs (without cycles), starting at o and ending at d, and $\mathcal{P}_{od}$ to be the finite set of all possible paths from o to d. For each possible path $P_{od} \in \mathcal{P}_{od}$, we model the point estimate of the total travel time along this path to be $\hat{T}_{P_{od}} = \sum_{(i,j)\in P_{od}} t_{ij}$. Because our data set provides no information as to which path was followed to realize a given travel time, we assume that drivers use the fastest paths available. We thus select $\hat{P}_{od} = \operatorname{argmin}_{P_{od}\in\mathcal{P}_{od}} \sum_{(i,j)\in P_{od}} t_{ij}$, and define our point estimate to be $\hat{T}_{od} = \hat{T}_{\hat{P}_{od}} = \sum_{(i,j)\in\hat{P}_{od}} t_{ij}$. As a consequence, given the parameters t_{ij}, our model chooses the point estimates $\hat{T}_{od}(\mathbf{t}) = \min_{P_{od}\in\mathcal{P}_{od}} \sum_{(i,j)\in P_{od}} t_{ij}$, where $\mathbf{t}$ is simply shorthand for the vector $(t_{ij})_{(i,j)\in E}$ (following standard boldfaced vector notation).

To use this model, we must only provide $|E|$ parameters, which is generally much less than the $|V|^2$ estimates we want to obtain. The model is also interpretable, as we expect the values t_{ij} to be representative of the typical travel times along arc $(i,j) \in E$. We acknowledge that the shortest-path assumption itself can be questioned. From a behavioral standpoint, taxi drivers may have other objectives in mind, such as maximizing revenue or minimizing fuel consumption; in addition, [51] showed that shortest paths can be sensitive to changes in travel time. However, we find that despite this modeling assumption, our results on real data are interpretable and reasonably accurate.

Parameter Estimation. We want to use the observed travel times T^i_{od} to estimate the model parameters t_{ij}. Following our goal to have estimates as close as possible to the geometric mean of $\mathcal{D}_{od}$, we want to find the values of t_{ij} that minimize the MSLE of the estimates $\hat{T}_{od}$. Because the distributions $\mathcal{D}_{od}$ and $\mathcal{D}$ are unknown, we approximate them with the empirical distribution of our observations and we obtain the following minimization problem:

$$\min_{\mathbf{t}} \sum_{(o,d)\in W} \sum_{i=1}^{n_{od}} (\log \hat{T}_{od}(\mathbf{t}) - \log T^i_{od})^2, \tag{3.4}$$

which is equivalent to

$$\min_{\mathbf{t}} \sum_{(o,d)\in W} n_{od} (\log \hat{T}_{od}(\mathbf{t}) - \log T_{od})^2, \tag{3.5}$$

where $T_{od} = \left(\prod_{i=1}^{n_{od}} T_{od}^{i}\right)^{1/n_{od}}$, the geometric mean of all the observed travel times from o to d.

Regularization. In order to generalize well out of sample, we need to add a regularization term to the empirical MSLE. This is important because we may not have sampled enough data from $\mathcal{D}$ and $\mathcal{D}_{od}$, and the empirical MSLE (3.4) may not be a good approximation of the MSLE (3.1). Leveraging our knowledge of the city network, we hypothesize that two similar intersecting or consecutive roads should have similar traffic speeds by default. Two arcs (i, j) and (k, l) are called *neighboring* when they represent consecutive or intersecting roads with the same "type". These types are defined through our knowledge of the routing network, and differentiate highways, major arteries and smaller roads. The neighboring relationship is written as $(i, j) \leftrightarrow (k, l)$. This regularization is somewhat unusual in traffic studies, but it is effective in practice and will only influence our estimation when we do not have enough data. Adding the regularization term to our objective yields:

$$\sum_{(o,d)\in W} n_{od}\left(\log \hat{T}_{od} - \log T_{od}\right)^2 + \lambda \sum_{(i,j)\leftrightarrow(k,l)} \left|\frac{t_{ij}}{d_{ij}} - \frac{t_{kl}}{d_{kl}}\right| \frac{2}{d_{ij} + d_{kl}}, \tag{3.6}$$

where d_{ij} corresponds to the length in meters of the arc (i, j) in the routing network, $\sum_{(i,j)\leftrightarrow(k,l)}$ represent the sum over all pairs of neighboring arcs (i, j) and (k, l) and the parameter λ represents the strength of the regularization. In other words, we minimize the difference in speed of neighboring roads, with the weighting factor $2/(d_{ij} + d_{kl})$ ensuring that continuity is more important in shorter neighboring roads (where constant velocity is a better approximation) than in longer ones.

3.2.2 MIO Formulation

We can now estimate the parameters t_{ij} from the data by solving the following mixed-integer formulation with linear constraints and a non-linear objective:

$$
\begin{aligned}
\min_{\hat{\mathbf{T}},\mathbf{t},\mathbf{z}} \quad & \sum_{(o,d)\in W} n_{od}\left(\log \hat{T}_{od} - \log T_{od}\right)^2 + && \lambda \sum_{(i,j)\leftrightarrow(k,l)} \left|\frac{t_{ij}}{d_{ij}} - \frac{t_{kl}}{d_{kl}}\right| \frac{2}{d_{ij}+d_{kl}} && \text{(3.7a)}\\
\text{s.t.} \quad & \hat{T}_{od} \le \sum_{(i,j)\in P^{\ell}_{od}} t_{ij} && \forall (o,d)\in W,\ P^{\ell}_{od}\in\mathcal{K}_{od} && \text{(3.7b)}\\
& \hat{T}_{od} \ge \sum_{(i,j)\in P^{\ell}_{od}} t_{ij} - M(1-z^{\ell}_{od}) && \forall (o,d)\in W,\ P^{\ell}_{od}\in\mathcal{K}_{od} && \text{(3.7c)}\\
& \sum_{\ell} z^{\ell}_{od} = 1 && \forall (o,d)\in W && \text{(3.7d)}\\
& z^{\ell}_{od}\in\{0,1\} && \forall (o,d)\in W, \ell\in\{1,\dots,|\mathcal{K}_{od}|\} && \text{(3.7e)}\\
& t_{ij}\ge a_{ij} && \forall (i,j)\in E. && \text{(3.7f)}
\end{aligned}
$$

The objective (3.7a) is the parameter estimation cost introduced in (3.6). For each $(o,d)\in W$, the constraints enforce that $\hat{T}_{od} = \min_{P_{od}\in\mathcal{K}_{od}} \sum_{(i,j)\in P_{od}} t_{ij}$, i.e. $\hat{T}_{od}$ is the time of the shortest path from o to d out of all the paths in $\mathcal{K}_{od}$. This non-linear shortest path constraint is enforced using the binary variables z^{l}_{od} that represent which path $P^{\ell}_{od}\in\mathcal{K}_{od}$ is the shortest path, together with the constraints (3.7b), (3.7d) and the big-M constraints (3.7c). Typically, $\mathcal{K}_{od}=\mathcal{P}_{od}$ is the set of all paths from o to d, but the formulation generalizes to any other subset $\mathcal{K}_{od}\subset\mathcal{P}_{od}$. Finally, the constraints (3.7f) introduce the bounds a_{ij} to enforce a speed limit on the arc travel times t_{ij}.

3.2.3 Iterative Path Generation

For each $(o,d)\in W$, formulation (3.7) requires one binary variable for each path going from o to d. The number of paths is typically exponential in the size of the graph, so we need to reduce the number of paths to consider if we want to be able to solve (3.7). It turns out the formulation is naturally suited for an iterative approach. Assume we start with a small set of paths $\mathcal{P}^0_{od}$ for every origin-destination pair in the data-set. We can solve the problem in (3.7) by considering the set of paths $\mathcal{K}_{od}=\mathcal{P}^0_{od}$ instead of the much larger $\mathcal{K}_{od}=\mathcal{P}_{od}$. This yields values of t_{ij}, for which we can recompute new shortest paths in the network using any shortest-path algorithm. If for a given (o, d), the new shortest path has length less than $\hat{T}_{od}$, then we know that the minimum path length computed over $\mathcal{P}_{od}^0$ is not equal to the minimum path length over $\mathcal{P}_{od}$. In this case, we add the new shortest path P_{od}^1 to our set of paths, obtaining the set $\mathcal{P}_{od}^1 = \mathcal{P}_{od}^0 \cup \{P_{od}^1\}$. We can then re-solve (3.7) using $\mathcal{P}_{od}^1$ instead of $\mathcal{P}_{od}^0$, and iterate this process. If instead the new shortest path for each (o, d) has length equal to $\hat{T}_{od}$, then we know we have already found reasonable paths, reaching a stopping point for the algorithm. The algorithm thus generates an increasing list of path candidates $\mathcal{P}_{od}^k$ for each iteration k and $(o, d) \in W$, so that the shortest paths P_{od}^k are added to the path candidates of the next iteration, e.g. $\mathcal{P}_{od}^{k+1} = \mathcal{P}_{od}^k \cup \{P_{od}^k\}$.

This iterative approach is inspired by cutting plane algorithms in linear optimization. In practice, most paths between o and d are not remotely close to being the shortest and would never even be considered by drivers looking to travel from o to d. Although this iterative method does not necessarily converge to the global optimum of (3.7) with $\mathcal{K}_{od} = \mathcal{P}_{od}$, we will show empirically that it yields good results for large problems, does not exhibit pathological local optima when used with appropriate regularization and typically converges in a few steps. Additionally the algorithm is always interpretable: the solution at any iteration k corresponds to the optimal solution of the problem if the drivers only consider the paths in $\mathcal{P}_{od}^k$.

3.3 Solving Large-Scale Problems

Even with the iterative path generation presented in 3.2.3, the optimization problem (3.7) cannot be tractably solved for most problems of interest. The main reasons are that the objective is non-convex, and that there are at least $O(|W|)$ binary variables, which makes it impossible for state-of-the-art solvers to give interesting solutions in a reasonable time for problems with more than 1000 data-points and routing networks that represent real cities. Actually, solving this problem to optimality relates to the problem of path reconstruction in a graph (sometimes called the Inverse Shortest Path Length problem), an NP-hard problem, as discussed in [76]. We present a tractable approach that produces good solutions, allowing us to handle hundreds of thousands of data points in networks with tens of thousands of arcs.

3.3.1 Adapting the shortest path constraint

In order to handle a large number of data points, we need to discard the binary variables z_{od}^{ℓ} introduced in (3.7e). One way to do this is to modify the constraint $\hat{T}_{od} = \min_{P_{od} \in \mathcal{K}_{od}} \sum_{(i,j) \in P_{od}} t_{ij}$. An interesting solution can be built by fixing the values of the binary variable, i.e., choosing which path should be the shortest for each $(o,d) \in W$. Indeed, if the shortest path in $\mathcal{K}_{od}$ is chosen to be P_{od}^*, then the shortest path constraints (3.7b)-(3.7e) trivially become:

$$\hat{T}_{od} = \sum_{(i,j) \in P_{od}^*} t_{ij} \qquad \forall (o,d) \in W, \tag{3.8a}$$

$$\hat{T}_{od} \geq \sum_{(i,j) \in P_{od}} t_{ij} \qquad \forall (o,d) \in W,\ P_{od} \in \mathcal{K}_{od}. \tag{3.8b}$$

For this formulation to become useful, we need a clever way to choose P_{od}^* for each $(o,d) \in W$. Our iterative path generation algorithm introduced in Section 3.2.3 provides a good candidate. At iteration k, the algorithm computes the shortest path P_{od}^k for each $(o,d) \in W$. This path can be viewed as our "best estimate" of the true path at iteration k, and is one of the paths we consider at iteration $k+1$. For this reason, we choose to use this path as the chosen shortest path for the next iteration $k+1$, setting $P_{od}^* = P_{od}^k$.

In the end, the results on synthetic data in Section 3.4 and on real data in Section 3.5 show that this method, appropriately regularized, yields interpretable high-quality solutions and empirically converges. Our intuition is the following: this path estimation may not seem perfect, but the tractability gains allow us to use orders of magnitude more data, which will improve the accuracy of the t_{ij} parameters and the $\hat{T}_{od}$ estimates, thus allowing us to compute better paths at each iteration.

3.3.2 Towards a Convex Objective

The left term in the minimization objective (3.7a) is nonconvex and not easily optimized by traditional optimization solvers. We want to find a surrogate that is convex, tractable, and a good approximation of the original squared log cost. More specifically, we want to find a convex loss function ℓ such that $\ell(\hat{T}_{od}, T_{od}) = (\log(\hat{T}_{od}) - \log(T_{od}))^2 + o((\hat{T}_{od} - T_{od})^2)$, and such that ℓ is also unbiased in the multiplicative space, i.e. $\ell(aT_{od}, T_{od}) = \ell(\frac{T_{od}}{a}, T_{od})$ for any scalar $a > 0$.

A good candidate is the maximum ratio loss: $\ell(\hat{T}_{od}, T_{od}) = \left(\max\left(\frac{T_{od}}{\hat{T}_{od}}, \frac{\hat{T}_{od}}{T_{od}}\right) - 1\right)^2$. It is a convex function of the variable $\hat{T}_{od}$ that has all the desired properties. Our objective thus becomes:

$$\sum_{(o,d)\in W} n_{od} \left(\max\left(\frac{T_{od}}{\hat{T}_{od}}, \frac{\hat{T}_{od}}{T_{od}}\right) - 1\right)^2 + \lambda \sum_{(i,j)\leftrightarrow(k,l)} \left|\frac{t_{ij}}{d_{ij}} - \frac{t_{kl}}{d_{kl}}\right| \frac{2}{d_{ij} + d_{kl}} \tag{3.9}$$

We want to be able to solve the corresponding optimization with hundreds of thousands of data-points. To the best of our knowledge, only state-of-the-art LP and SOCP solvers are able to handle formulations with hundreds of thousands of variables and constraints. As a consequence, we would like to slightly modify our formulation to be able to formulate it as an SOCP. All we need to do is replace the squared losses by absolute values, yielding the modified objective:

$$\sum_{(o,d)\in W} n_{od} \max\left(\frac{T_{od}}{\hat{T}_{od}}, \frac{\hat{T}_{od}}{T_{od}}\right) + \lambda \sum_{(i,j)\leftrightarrow(k,l)} \left|\frac{t_{ij}}{d_{ij}} - \frac{t_{kl}}{d_{kl}}\right| \frac{2}{d_{ij} + d_{kl}} \tag{3.10}$$

This new objective allows us to reformulate each iteration as an SOCP:

$$\min_{\hat{\mathbf{T}},\mathbf{t},\mathbf{x}} \quad \sum_{(o,d)\in W} n_{od}x_{od} + \lambda \sum_{(i,j)\leftrightarrow(k,l)} \left|\frac{t_{ij}}{d_{ij}} - \frac{t_{kl}}{d_{kl}}\right| \frac{2}{d_{ij}+d_{kl}} \tag{3.11a}$$

$$\text{s.t.} \quad \hat{T}_{od} = \sum_{(i,j)\in P^*_{od}} t_{ij} \qquad \forall (o,d)\in W, \tag{3.11b}$$

$$\hat{T}_{od} \geq \sum_{(i,j)\in P_{od}} t_{ij} \qquad \forall (o,d)\in W,\ P_{od}\in \mathcal{K}_{od}, \tag{3.11c}$$

$$x_{od} \geq \frac{\hat{T}_{od}}{T_{od}} \qquad \forall (o,d)\in W, \tag{3.11d}$$

$$x_{od} \geq \frac{T_{od}}{\hat{T}_{od}} \qquad \forall (o,d)\in W, \tag{3.11e}$$

$$t_{ij} \geq a_{ij} \qquad \forall (i,j)\in E. \tag{3.11f}$$

where $x_{od} = \max\left(\frac{T_{od}}{\hat{T}_{od}}, \frac{\hat{T}_{od}}{T_{od}}\right)$. The objective can be formulated as linear, and all the constraints are linear except (3.11e), which can be reformulated as the following second-order cone constraint:

$$\left(x_{od} + \hat{T}_{od}\right) \geq \left\| \begin{pmatrix} \hat{T}_{od} - x_{od} \\ 2\sqrt{T_{od}} \end{pmatrix} \right\|. \tag{3.12}$$

Replacing the squared losses by absolute values makes our new formulation more robust to outliers and more tractable, but weakens the case for replacing the observations T^i_{od} that share the same (o,d) with their geometric mean T_{od}. Once more, we trade some modeling rigor for the ability to use more data, and we will show that this choice is empirically justified.

3.3.3 A Tractable Algorithm

We now summarize our tractable algorithm for large-scale static travel time estimation.

1. Choose a regularization parameter λ and an initial set of arc travel times:

$(t_{ij}^0)_{(i,j)\in E}$. We will show in the next sections that our results are not sensitive to these choices. For each $(o,d) \in K$, start with an empty set of paths $\mathcal{P}_{od} = \emptyset$. Then start Step 2 with iteration $k = 1$.

2. For each iteration k, do the following:

3. Use an efficient, parallelized shortest-path algorithm to compute all the shortest paths $(P_{od}^k)_{(o,d)\in W}$, using the arc travel times $(t_{ij}^{k-1})_{(i,j)\in E}$. Add these paths to the previous set of paths $\mathcal{P}_{od}^k = \mathcal{P}_{od}^{k-1} \cup \{P_{od}^k\}$. If there is a limit Π on the number of paths we can store (for memory or tractability reasons), remove the path of $\mathcal{P}_{od}^k$ with the longest travel time to make sure that $|\mathcal{P}_{od}^k| \leq \Pi$.

4. Solve the optimization problem (3.11), using the newly computed shortest paths $P_{od}^* = P_{od}^k$, to obtain the new arc travel times $(t_{ij}^k)_{(i,j)\in E}$.

5. If a convergence criterion is met, stop the algorithm and return the times $(t_{ij}^k)_{(i,j)\in E}$. Else, start iteration $k+1$ and go to Step 2.

In the end, our algorithm returns a set of arc travel times, that can be used to compute shortest paths and travel time estimations $\hat{T}_{od}$ for any origin-destination pair in the network. We propose a convergence criterion based on path differences.

Definition 1 (Path difference)**.** Given a node pair (o,d) and two paths P_{od}^A and P_{od}^B, the path difference $d(P_{od}^A, P_{od}^B)$ is defined as the average of the number of arcs in P_{od}^A that are not in P_{od}^B and the number of arcs in P_{od}^B that are not in P_{od}^A.

At each iteration k, we can compute the path difference between the new path P_{od}^k and the path of the previous iteration P_{od}^{k-1} for each $(o,d) \in W$. We stop our algorithm when the mean path difference across all $(o,d) \in W$ is less than a threshold δ, i.e. $\frac{1}{|W|}\sum_{(o,d)\in W} d(P_{od}^k, P_{od}^{k-1}) < \delta$. In this chapter, we fix δ to a small value: $\delta = 0.5$. We chose this value because we noticed that our estimates $\hat{T}_{od}$ were not improving in subsequent iterations, for the specific applications of this chapter. In this situation, the algorithm tends to converge in less than 10 iterations.

3.3.4 A General Model

The ability to solve the travel time estimation and routing problem using only origin-destination data is useful because it makes minimal assumptions on the format of the data. However, in some cases more data is available, for instance from loop sensors or floating car probes (see Section 3.1.1). Due to its optimization-based framework, our method is flexible enough to handle many additional forms of data.

The method presented in the previous section is designed under the assumption that for every (o, d) in the set of input node pairs $W \subset V \times V$, we are only given a finite number of sample travel times, from which we compute a geometric mean T_{od}, with no information about the path taken by the drivers. Constraint (3.11b) reflects the algorithm's attempt to guess the correct path, assuming that the drivers are trying to minimize driving times. If we assume now that for some $(o, d) \in W$, we are given not only a travel time T_{od}^{obs}, but also the used path P_{od}^{obs}, then we can add a term in the objective penalizing the distance between the observation T_{od}^{obs} and the sum $\sum_{(i,j)\in P_{od}^{\text{obs}}} t_{ij}$ of link travel times along the path P_{od}^{obs}.

Another form of traffic data that is commonly available comes from loop sensors/traffic cameras, which can sometimes measure traffic velocity on a given set of arcs $L \subseteq E$. For example, [137] shows that a single loop detector on a highway is enough to provide accurate speed estimates. A velocity measurement on arc (i, j) is easily integrated into our model, by adding a term in the objective that penalizes the distance of t_{ij} from its measurement.

Thus, though our method is designed with minimal data in mind, it can easily incorporate additional information about the network. In a world where more and more data is available, but formats may differ greatly from source to source, an optimization-based approach allows for the easy integration of complementary information, yielding a multipurpose method to solve the problem of travel time estimation and routing.

3.4 Performance on Synthetic Data

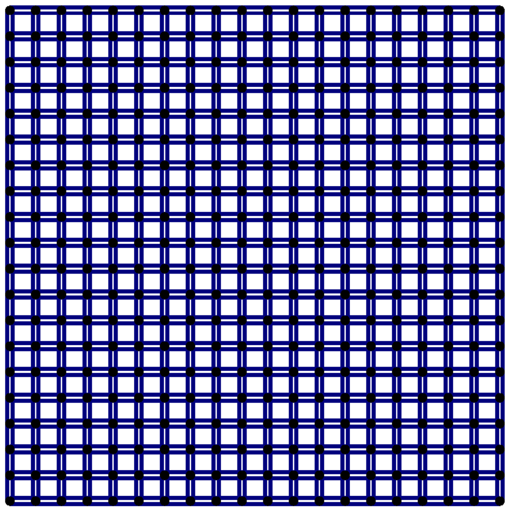
When developing a tractable algorithm in Section 3.2 and 3.3, we made several simplifying assumptions about driver behavior and network properties, and the complexity of our optimization formulation led to several heuristic simplifications. It is hard to verify if the tractable iterative algorithm presented in Section 3.3 provides good solutions to our original problem presented in Section 3.2.2 using real-world travel time data. Indeed, real data does not always follow our model's assumptions. This is why we first use synthetic data verifying our model's assumptions to study the convergence of our tractable algorithm as an approximation of the original formulation presented in Section 3.2, and then show in Section 3.5 that our model generalizes well to real-world data in terms of interpretability and accuracy.

Therefore, the goal of this section is twofold: first, we show that despite its heuristic steps, our approach to solving the optimization problem in Section 3.3 converges to a good estimate $\hat{T}_{od}$ of $\mathcal{D}_{od}$ in the log space, while recovering interpretable parameters t_{ij} that represent the local congestion states in the city. Second, we show that even with high variance travel time distributions $\mathcal{D}_{od}$ and very incomplete observations ($|W| << |V|^2$), we are still able to generalize well and recover good estimates $\hat{T}_{od}$ for all $(o, d) \in V^2$, when the synthetic data is generated following our modeling assumptions.

3.4.1 Synthetic Networks and Virtual Data

In order to test our method on synthetic data, we create simple model networks in which we attempt to reconstruct traffic patterns. One model reproduces some features of a city, with a central “downtown" area (8×8 square grid), surrounded by suburbs (4×4 square grids) and circled by highways (with higher speed limits) that connect each suburb to the central area and to the two closest neighboring suburbs. This network is shown in Figure 3-2b. For more advanced testing, we use a larger 20×20 square grid, with which we investigate a range of traffic patterns.

Once we have constructed the routing graphs, we create the synthetic travel time


(a) Simple square network, with 400 nodes and 1520 arcs. There are two arcs between any adjacent nodes (one in each direction). All arcs are of the same type, and consequently they all have the same maximum speed.

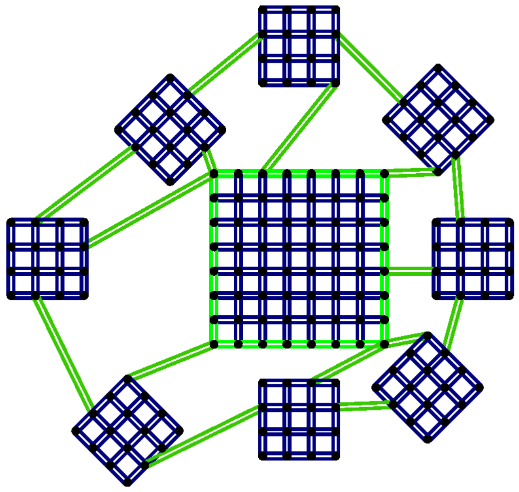

(b) Toy model of a city, with 192 nodes and 640 arcs. The green roads are highways, with much higher speed limits (and consequently lower travel times proportional to their length).

Figure 3-2: Model networks used to test our travel time estimation and routing algorithm.

distributions $\mathcal{D}$ and $\mathcal{D}_{od}$. Each observation (o, d, T) is generated as follows: first, the distribution $\mathcal{D}$ is chosen to be uniform over all origin-destination pairs in V^2. In practice, taxi trips are not uniformly distributed over the city network; however, we will see in the Section 3.5 that our model performs well with real-world observations that are far from being uniformly distributed. Then, $\mathcal{D}_{od}$ is chosen to be lognormal with log-mean parameter $\mu = \log(T_{od}^{\text{real}})$ and a second parameter σ that controls the randomness of travel times from o to d. For context, $\mathcal{D}_{od}$ has geometric mean T_{od}^{real}, and a value of $\sigma = \log 2 \approx 0.7$ means that a sampled time T_{od}^{i} is within one geometric standard deviation of the T_{od}^{real} if it is between $0.5T_{od}^{\text{real}}$ and $2T_{od}^{\text{real}}$. The values T_{od}^{real} are chosen to follow our shortest-path model. Therefore, we set a deterministic value of the parameter t_{ij}^{real} for each arc (i, j), and define $T_{od}^{\text{real}} = \min_{P_{od} \in \mathcal{P}_{od}} \sum_{(i,j) \in P_{od}} t_{ij}^{\text{real}}$. As a consequence, T_{od}^{real} are the best estimates of the distributions $\mathcal{D}_{od}$ given our shortest path model and the estimation loss introduced in Section 3.2.1.

We then use this process to sample N observations of origin-destination travel times. For each (o, d) independently, we would need several samples to be able to estimate T_{od}^{real} (because of the randomness σ), but the routing network model of our algorithm allows us to be able to use much less samples to provide accurate point estimates for $\mathcal{D}_{od}$ (i.e. close to T_{od}^{real}), even when $(o, d) \notin W$.

3.4.2 Results

We evaluate the quality of our estimation using the Root Mean Squared Log Error (RMSLE) of our estimates $\hat{T}_{od}$, defined as the square root of the MSLE (3.1). Interestingly, the formulation simplifies when using the lognormal distributions:

$$\text{RMSLE}(\hat{T}_{od}) = \sqrt{\sigma^2 + \mathbb{E}_{(o,d)\sim\mathcal{D}}\left[\left(\log(\hat{T}_{od}) - \log(T_{od}^{real})\right)^2\right]}. \tag{3.13}$$

To make it easier to compare our estimations across different values of σ, we focus on the square root of the MSLB (RMSLB), removing the contribution of the log variance σ^2 of the data (see (3.3)).

$$\text{RMSLB}(\hat{T}_{od}) = \sqrt{\sum_{(o,d)\in V^2}\left(\log(\hat{T}_{od}) - \log(T_{od}^{real})\right)^2}, \tag{3.14}$$

where we used that $\mathcal{D}$ is uniform over V^2. Note that RMSLB $= 0$ means that we recover the geometric expectation of the travel times exactly.

We present the effects of our method when run on the city models described in the previous section, with a few different travel time functions and data generated as described above. We begin by studying the toy model of a city introduced in Figure 3-2b. The values t_{ij}^{real} are chosen by road type, with one speed for regular streets and another for the highways. We sample N travel time observations as described in Section 3.4.1, and we start with random arc travel times to define the initial path P_{od}^0 for each (o, d) in W. In Table 3.1 we present results of our method for different values of N and σ.

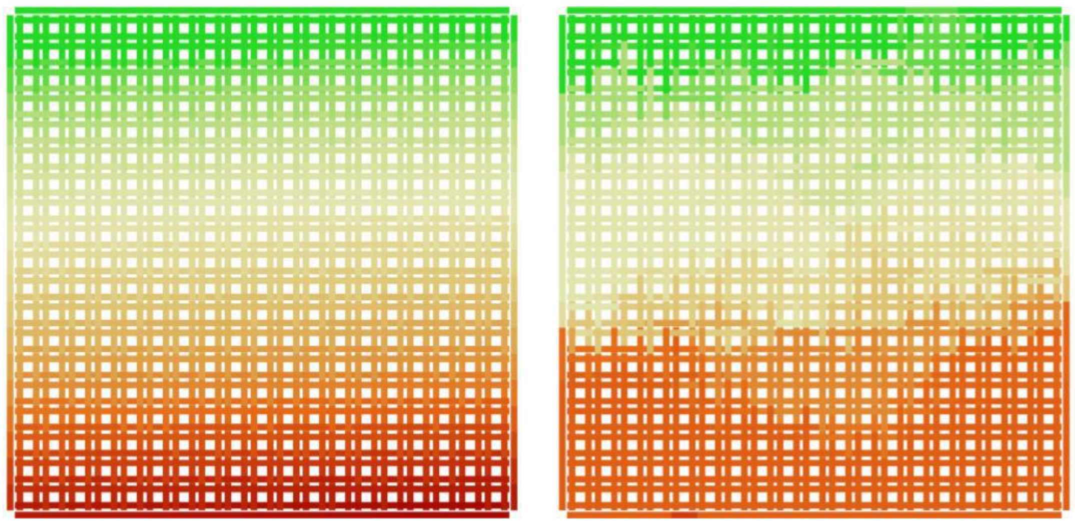
For all values of σ, when setting $\delta = 0.5$ we find that the method tends to converge in under 10 iterations. Each iteration on this small network (192 nodes, 640 arcs) takes

	Available amount of data			
	$N = 100$	$N = 500$	$N = 1,000$	$N = 10,000$
Randomness of input data	RMSLB of estimation			
$\sigma = 0.0$	0.08	0.03	0.01	0.01
$\sigma = 0.1$	0.09	0.06	0.03	0.02
$\sigma = 0.5$	0.23	0.12	0.15	0.05
$\sigma = 1.0$	0.48	0.22	0.20	0.07
$\sigma = 2.0$	0.62	0.43	0.30	0.15

Table 3.1: RMSLB (Root Mean Squared Log Bias) of estimation for a varying amount of data and randomness σ. RMSLB of estimation for a varying amount of data N and a varying amount of travel time randomness on the toy city model (see Fig. 3-2b). The toy city used has just under 37,000 node pairs (192 nodes), but notice that we need very little data to create an estimate with small bias.

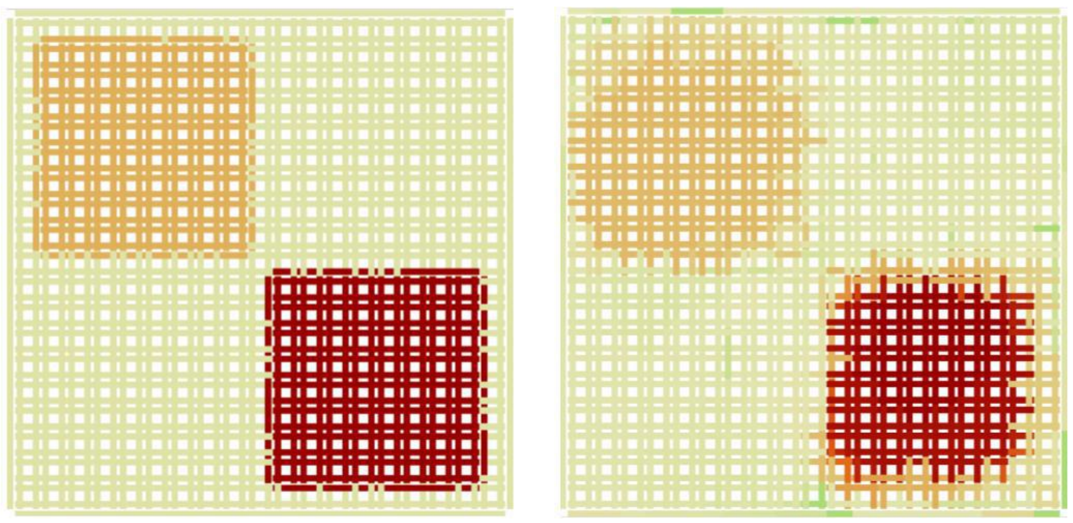
less than 10 seconds for a total run-time of less than two minutes. We noticed that the regularization term in the objective greatly speeds up convergence by reducing the relevance of tiny path differences. In addition, Table 3.1 confirms the rather obvious fact that results are more accurate with more data and less randomness in the data (top left corner). However, it also reveals that when the input has a high log variance σ^2 it is possible to obtain an estimate with comparatively small bias with very little data. For example, when $\sigma = 2$, it is possible to obtain an estimation bias that is smaller by more than a factor of two with only 100 observations, i.e., less than 2% of the total origin-destination pairs.

The results in Table 3.1 should be taken with a grain of salt, however, as the toy model in Fig. 3-2b is intentionally suited to the assumptions with which we developed our model (especially the velocity continuity assumption of our regularization). The goal of this experiment is simply to confirm that the method converges as intended and produces sensible results. The next step is to consider a model that does not follow our traffic assumptions as closely. We therefore focus on the square network (400 nodes), which we associate with two traffic configurations, presented in Figures 3-3 and 3-4, corresponding to semi-realistic scenarios, including a gradual north-to-south increase in travel time (Fig. 3-3a), and two congested neighborhoods where arc travel times are doubled and quadrupled (Fig. 3-4a) as compared to the rest of



(a) True congestion. (b) Reconstruction.

Figure 3-3: Results of our algorithm on a square network for a congestion gradient. Green arcs have a higher velocity and thus a lower travel time, while red arcs are more congested and thus have a lower velocity and a higher travel time. All arcs are of the same type, with a maximum velocity of 50 kph. The estimates $\hat{T}_{od}$ are computed using $N = 5,000$ observations. The Root mean squared log bias (RMSLB) of the estimates is 0.041, which is over eight times smaller than the input log standard deviation $\sigma = 0.35$. The algorithm effectively reconstructs high-level traffic patterns and provides accurate travel time estimates despite extremely noisy data. For arcs in each of the four quarters from the top, the true velocity is respectively 60%, 30%, 20%, and 15% of the maximum velocity. The gradient from Figure 3-3a is clearly visible despite some noise.



(a) True congestions (b) Reconstruction

Figure 3-4: Results of our algorithm on a square network with two congested neighborhoods. Green arcs have a higher velocity and thus a lower travel time, while red arcs are more congested and thus have a lower velocity and a higher travel time. All arcs are of the same type, with a maximum velocity of 50 kph. The true velocity is 30% of the maximum velocity for arcs in the upper-left neighborhood, 15% for arcs in the lower-right neighborhood, and 60% for arcs outside these neighborhoods. The estimates $\hat{T}_{od}$ are computed using $N = 5,000$ observations. The RMSLB of the estimates is 0.069, which is over eight times smaller than the input log standard deviation $\sigma = 0.35$.

the city. Readers will note that these scenarios break our road velocity continuity assumption in different ways: the first because the velocities of neighboring vertical arcs are never equal, and the second because the borders between congested and uncongested areas are strongly discontinuous in terms of velocity.

For each of the two scenarios thus described, we sample $N = 5,000$ observations as described in Section 3.4.1, for $\sigma = 0.35$ (we choose this value because it is approximately our estimate of the log standard deviation of Manhattan taxi travel time data).

The arc travel times found by our algorithm are shown in Figures 3-3b and 3-4b. The algorithm does a remarkable job reconstructing the travel times in the network given limited data. As noted above, the data provided was noisy ($\sigma = 0.35$), yet the RMSLB for the estimated travel times T_{od} over all (o, d) in W is 0.07 in one case and 0.04 in the other. Therefore, the algorithm not only produces accurate travel times estimates for the origin-destination pairs for which no data was available, it does so with minimal bias when compared to the high randomness and sparsity of the travel time data. Notice that the regularization term in the objective, though based on a questionable traffic assumption, does not preclude us from reconstructing the arc travel times as desired in both cases, though it does make it difficult to find the exact border of the congested neighborhoods in one case, and the precise velocity gradient in the other.

All in all, the method developed in this chapter is able to extract useful information from high-variance inputs, and produces a network cost function, in the form of arc travel times, that is interpretable and can in turn be used for other applications in the network. In the following section, we show that all the algorithm properties displayed in this section, namely low estimation bias despite inputs with high randomness, and the production of an interpretable final solution, also hold in a real-world setting at much larger scales.

3.5 Performance on Real-World Data

So far, we have described, implemented and tested a methodology to solve the travel time estimation and routing problem. We use a network formulation because we assume the only allowed origins and destinations are nodes in the graph. However, real origin-destination data records vehicles' starting and ending points using GPS coordinates, which are continuous variables.

In this section, we first provide a bridge between the continuous and discrete problems in order to apply our method to real-world OD data, and present the results on data from New York City taxis. We then display the results of our method for varying amounts of available data, showing that our method provides both accurate travel time estimations throughout the network and sensible routing information, for an interpretable understanding of traffic in the city.

3.5.1 A Large-Scale Data Framework

A major contribution of this chapter is the ability to exploit a large data-set to solve the travel time estimation and routing problem for the real-world network of a large city. Providing a tractable method at this scale requires the construction of a substantial framework to handle large amounts of network information and origin-destination data, allowing us to leverage big data insights in solving a complex problem.

In order to solve the travel time estimation and routing problem in a real-world setting, it is necessary to overcome two major challenges. The first difficulty is to extract a network structure from a complex urban landscape, and specifically to identify a graph that is elaborate enough to capture most of the details of the city under study, but simple enough to tractably support our network optimization methods. For this purpose, we use open-source geographical data from the OpenStreetMap project. Its database provides a highly-detailed map of New York City, which we simplify by excluding walkways and service roads, and removing nodes that do not represent the intersection of two or more roads. For the island of Manhattan, to which we restrict our problem, we obtain a strongly connected graph with 4324 nodes and

9518 arcs. This network is quite large, and the tractability of our method on a map of this size is itself a significant contribution of this chapter: readers will realize that an algorithm seeking to estimate travel times in this network must consider over 18 million origin-destination pairs of nodes and at least that many shortest paths.

The second challenge is obtaining and cleaning real OD data. Data from the New York City Taxi and Limousine Commission for the years 2009-2016 is freely available from [102]. A month's worth of data (approximately 2GB) contains information for over 12 million taxi trips (over 400,000 a day). We perform all computations, network and data handling using the Julia programming language. Our method's tractability is enhanced by the use of the cutting-edge Julia for Mathematical Programming (JuMP) interface by [91], which enables us to take advantage of top-of-the-line linear and second-order programming methods, as implemented by the commercial solvers Gurobi and Mosek. Therefore, our framework can handle problem instances considering hundreds of thousands of data points in the entirety of Manhattan.

The results presented in Sections 3.5.3 and 3.5.4 use taxi data from weekdays in May 2016, in the time windows 9-11AM (morning), 6-8PM (evening), and 3-6AM (night). We restrict the data to a single month to reduce the taxi trip variance. Smaller time windows also guarantee less noisy data, but at the cost of fewer data points, and so we opt for a medium-sized window of a few hours. Our method therefore seeks to capture network patterns that are averaged over the considered time window.

In order to eliminate extreme outliers, we ignore trips shorter than 30s and longer than 3 hours, trips connecting points that are less than 250m or more than 200km away, and trips which would require an average speed greater than 110 kph or less than 2 kph to make sense. The existence of such unrealistic trips is a consequence of the imperfection of the GPS sensors inside the taxi meters. After this filtering step, we split the data into a training set containing about 415,000 trips and a testing set containing about 275,000 trips. In the next section, we explain how we adapt this taxi data to our discrete network-based framework.

3.5.2 Applying a Discrete Model to Real-World Data

The model described in Section 3.2 is discrete in space and static in time: it considers fixed traffic patterns during a given time window in a network where the only possible start and end locations are intersections. In contrast, real-world data is continuous in time and space: a given taxi trip is associated with a start time and an end time recorded by a clock within the taxi meter, and with start and end locations that are recorded using often noisy GPS sensors. We therefore need to process the data a bit further for it to work with our model.

In the database, each taxi trip is represented as a 6-tuple $(x_O, y_O, x_D, y_D, t_{\text{start}}, tt_{OD})$, where x_O and y_O are the GPS coordinates of the origin, x_D and y_D are the GPS coordinates of the destination, t_{start} is the date and time of the beginning of the ride, and tt_{OD} is the travel time of the taxi from its origin to its destination. We use t_{start} to assign taxi trips to time intervals of length τ, and consider only this time window, discarding all taxi trips that do not start inside this interval. For taxi trips that do start within the interval, we do not differentiate between different t_{start} values, so each trip is reduced to the 5-tuple $(x_O, y_O, x_D, y_D, tt_{OD})$.

The length τ of the time interval should be chosen based on the scope of the application. If the goal is static planning, we can select a large value of τ (from a few hours to a few months), which will allow us to consider a large amount of data, and estimate fixed travel time parameters over the interval as accurately as possible. If the goal is short-term dynamic planning, we can pick a small value of τ, say 5 minutes, and use the small amount of data in this interval to quickly estimate the travel time parameters, which we will only assume to be valid for the next time interval.

In the discrete formulation presented in the previous sections, the input of the method is a set of node pairs W, with a known (but possibly noisy) travel time T_{od} for each (o, d) in the set W. In the continuous problem, the inputs are position vectors (x_O, y_O, x_D, y_D) and associated travel times tt_{OD}. We therefore need to project the continuous origin (x_O, y_O) and destination (x_D, y_D) onto the network to be able to use our discrete methods in this real-world setting.

There exist many methods of projecting continuous data onto a discrete network model (see recent work by [117], [33]); all our results were obtained by projecting each continuous origin-destination pair (x_O, y_O, x_D, y_D) to the nearest node pair (o, d) using the Euclidean metric in $\mathbb{R}^4$. We can now apply our algorithm to real taxi data in Manhattan.

3.5.3 Evaluating Results at the Scale of the City

Accuracy. We have explained in this chapter that real-world OD data has significant variance, originating from several main sources: the imprecision of the data-gathering protocol, including potent “urban canyoning" effects in GPS data, as well as the inherent variance of traffic patterns (see Section 3.2.1). The latter source is especially important when the time window is long, as a consequence of our static traffic modeling assumption.

As seen in Section 3.2, the mean squared log error (MSLE) decomposes into the sum of the mean log variance of the data and the mean squared log bias of our estimate. On empirical data, we can only evaluate the MSLE using (3.4). In order to be able to evaluate the performance of our estimation, we need to estimate the log variance of the travel time observations (3.2). Indeed, it is a lower bound on the MSLE of our estimate, and a low-bias estimate must have an MSLE as close as possible to this lower bound.

For this purpose, we simply compare each taxi trip in the data to the average of the k trips closest to it, and compute the empirical log variance between the two values. This gives us an upper bound on the log variance term of the MSLE of our estimate. Because the data-set is quite large, this bound is indicative, especially when we choose the value of k that minimizes it. For instance, this approach yields an input log variance of $0.31^2 = 0.10$ for the time window 9-11 AM. To understand the magnitude of this variance, consider a mean time of 20 minutes. A trip within one standard deviation of the mean could last any amount of time between $20e^{-0.31} = 14.7$ minutes and $20e^{0.31} = 27.3$ minutes.

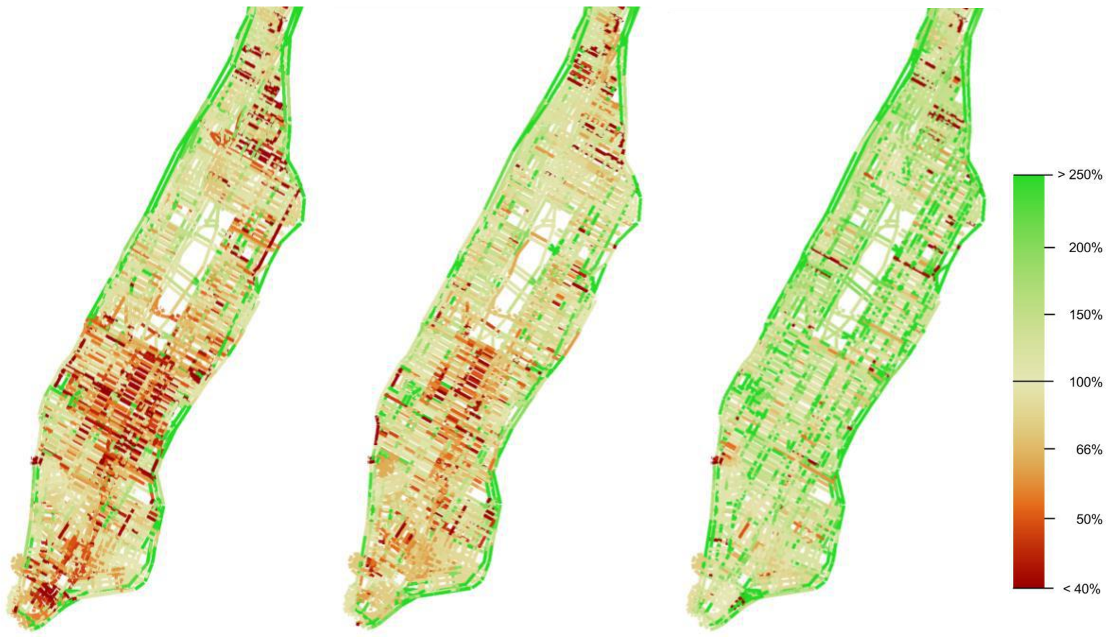
We will measure the accuracy of our method by how close the MSLE of our estimate is to the log variance (3.2) of the data. This is a proxy for minimizing the mean squared log bias (3.3), which is what we did in Section 3.4 when we new the distributions $\mathcal{D}_{od}$. If the difference between the MSLE of our estimations and the input log variance of the data is small, it means that our estimations are very close to the geometric mean of the network travel times, and most of our error comes from the inherent variability of the taxi trips.

For tractability reasons, we restrict the size of the input node pairs set W to 100,000 (o, d) pairs. With $|W| = 100,000$, the total computation at the scale of Manhattan takes less than 2 hours. We choose the regularization parameter $\lambda = 1000$, which is the value that minimized the MSLE in cross-validation. We note that the algorithm converges in 10 iterations without showing noticeable cycling (the out-of-sample improves at each iteration).

It turns out that between 9 and 11 AM, the out-of-sample RMSLE of our estimations is just over 0.36. This result means that our travel time estimation error is barely worse than the inherent noise in the data, and our estimated travel times must therefore be very close to the geometric expectation of the travel times throughout the network.

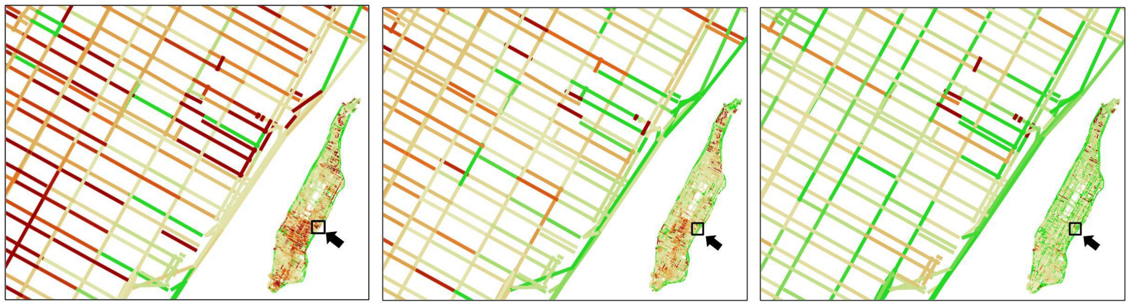
Interpretability. In addition to its accuracy, we argue that our method provides global insights about traffic patterns in New York. To show this, we compare our results in Manhattan in the morning (9-11AM), in the evening (6-8PM) and at night (3-6AM). We show the edge travel times for these time windows in Figures 3-5 and 3-6. The overall traffic patterns are easily identifiable in Figure 3-5, in particular the effect of the morning (and to a lesser extent, evening) commute in Midtown, as well as the congestion in the northern part of the island near the bridges connecting it to the mainland.

The results of our algorithm provide insights at a variety of scales: in addition to displaying citywide effects such as the daily commute, they also reveal more subtle realities about traffic in New York: For example, when looking at Figures 3-6a and 3-6b, it is clear that crosstown (east-west) travelers are much more exposed to



(a) Morning (9-11AM) (b) Evening (6-8PM) (c) Night (3-6AM)

Figure 3-5: Edge travel times in Manhattan estimated by our algorithm on weekdays in May 2016 in the morning, evening and at night. The color of each edge represents the speed along that edge as a percentage of the reference velocity $v_0 = 13.85$ kph (average velocity in Manhattan on weekdays). At the scale of the city, the algorithm clearly identifies morning commute congestion in Midtown and the Financial District, while at the scale of individual city blocks, it confirms the empirically known fact that crosstown (east-west) traffic in Manhattan is much more congested than uptown-downtown (north-south) traffic.



(a) 9-11AM (close-up) (b) 6-8PM (close-up) (c) 3-6AM (close-up)

Figure 3-6: Zoom on edge travel times in Manhattan estimated by our algorithm on weekdays in May 2016 for 9-11AM and 3-6AM. Detail of Figure 3-5. The color of each edge represents the speed along that edge as a percentage of the reference velocity $v_0 = 13.85$ kph (average velocity in Manhattan on weekdays). The gap in congestion between crosstown traffic and uptown-downtown traffic is also visible at this much smaller scale.

Time of the day	Training trips	Mean trip time	Out-of-sample RMSLE
09-11 AM	415,106	13m54s	0.31
06-08 PM	545,965	11m54s	0.30
03-06 AM	75,339	07m38s	0.28

Table 3.2: Effect of the time of the day on the taxi-trip data-set and the estimation power of our method. Note that the number of trips available and the root mean squared log error (RMSLE) depends on the time of the day. As a consequence, the time-window choice plays an important role in the quality of our estimation. Figures 3-5 and 3-6 represents the corresponding network travel times.

congestion than uptown-downtown (north-south) travelers, and that the highways on Manhattan's eastern and western shores (FDR Drive and Riverside Drive) are much faster routes than Manhattan's inner streets.

More detailed error results regarding the morning, evening and night time windows are available in Table 3.2.

A Robust and Sensible Path Estimation. At each iteration of the algorithm, the total path difference decreases, which means the algorithm finds a stable solution to the travel time estimation and routing problem. Moreover, we also note that the algorithm always converges to a similar choice of path, independently of the choice of initial paths and arc travel times. To support this claim, we show in Figures 3-7 and 3-8 the evolution of a path between a given origin and destination over random restarts of the algorithm. Specifically, we initiate the algorithm with random times: each edge has a velocity drawn randomly between 1 and 130 kph. This results in the random initial paths shown in panes 3-7a, 3-7b, and 3-7c. After 5 iterations, we consider the paths obtained by our method, in panes 3-8a, 3-8b, and 3-8c.

We see that in all three cases, the algorithm made the justifiable decision of using the freeway on the western edge of Manhattan. In addition, despite stark differences in the starting point, the final paths found by the method are eerily similar. One can quibble about the exact level of similarity between these final paths, but it is wise to remember that our method does not seek to obtain the "optimal" path between an origin o and a destination d (and indeed [51] questions the existence of such an

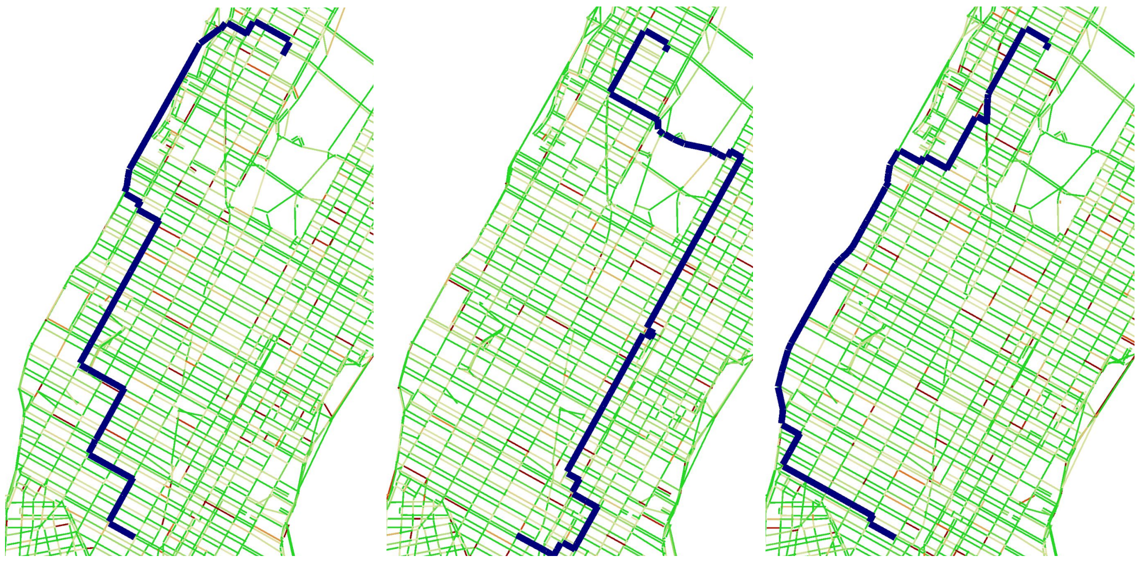

(a) Random initial path 1. (b) Random initial path 2. (c) Random initial path 3.

Figure 3-7: Evolution of paths studied by our algorithm : original paths. Three random starting point paths are presented. See Figure 3-8 for the results of the algorithm.

optimal path in a noisy environment), but simply a reasonable path that achieves the estimated travel time. Figure 3-8 is an example of our method accomplishing this stated purpose.

To provide intuition for why our routes seem sensible, note that we have empirically established that the travel time estimation accuracy in Manhattan has low bias, as we showed that the MSLE was close to our estimate of the mean log variance of the data. Additionally, the regularization allows us to generalize well to parts of the city with few observations. Furthermore, the obtained arc travel time parameters t_{ij} are good estimations on synthetic data and seem reasonable in NYC. All these observations lead us to hypothesize that the obtained paths are sensible.

This result means that in a network with almost ten thousand nodes, with only a few OD pairs relative to the possible 18 million pairs, using high variance data, we are able to reconstruct the all-pairs shortest paths that minimize the error between the shortest path lengths and the input data. Our optimization-based algorithm thus exhibits a certain number of important properties: it is tractable at the scale of a large and complex city, produces accurate travel time estimations and sensible routing

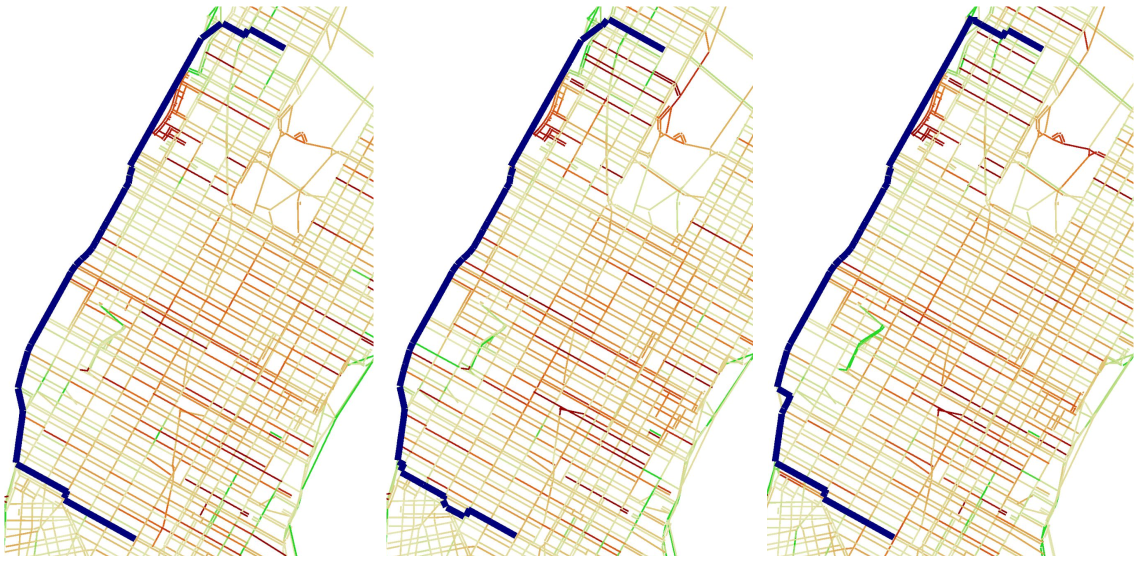

(a) Path 1 after 5 iterations. (b) Path 2 after 5 iterations. (c) Path 3 after 5 iterations.

Figure 3-8: Evolution of paths studied by our algorithm : path convergence. Shows the resulting path to which the algorithm converges after 5 iterations, starting from the initial paths and times presented in Figure 3-7. The reader will notice that despite strong differences in the starting paths, the algorithm eventually converges to a very reasonable solution, a path that makes use of the freeway on the western shore of Manhattan.

information despite high variance data, and produces an overall traffic map of the city that can be used for numerous other applications. These results are obtained with a large number of data points, and in fact we operate at the limit of what our solvers can handle. In the following section, we explore the effect of reducing available data on our method's accuracy.

3.5.4 Impact of Data Density and Comparison with Data-Driven Methods

The results presented in Section 3.5.3 show that, when run with a large number of data points, our method tractably estimates travel times in Manhattan. Given this performance with a wealth of data, it is natural to wonder how our algorithm fares when the data is much more sparse. Good performance in data-poor environments is important for two reasons: first, few cities have as much demand for taxis as New York, so extending the method to other networks would necessitate good behavior with only minimal amounts of data. Second, taxis do not necessarily explore networks in a uniform manner: even in cities such as New York where they represent a significant fraction of traffic, taxis only seldom visit certain neighborhoods, creating data-rich and data-poor settings within a single network.

Nearest Neighbor Travel Time Estimation. In this section, we choose to compare the performance of our method to simple purely data-driven schemes, which are expected to work very well in a data-rich setting and comparatively less well in a data-poor setting. Indeed, the formulation of the real-world travel time estimation problem as the estimation of tt_{OD} as a function of the four variables x_O, y_O, x_D, and y_D suggests simple solution approaches based solely on the data. With no knowledge of the network or the underlying behavior of taxi drivers, it is possible to use machine learning to infer travel times.

For instance, a simple k-nearest neighbors scheme would match an input origin and destination $(x_O, y_O, x_D, y_D) \in \mathbb{R}^4$ to the k taxi trips in the database closest to it (for some choice of metric) and compute the geometric average of their travel times to produce an estimate for the travel time between the provided origin and destination.

This scheme has the advantage of being extremely simple and allowing for quick travel time estimations. In addition, it is easy to see that the bias of this travel time estimate would converge to zero as the number of observation increase (if k is scaled appropriately). Indeed, this approach is not limited by the low-dimensional model assumption of our algorithm. However, in practice it has several drawbacks: first, it is not particularly well suited to travel time estimation for origins and destinations for which little data is available. This is a particularly damaging flaw because, as stated before, origin-destination data is not very complete and is concentrated in regions with more taxi traffic.

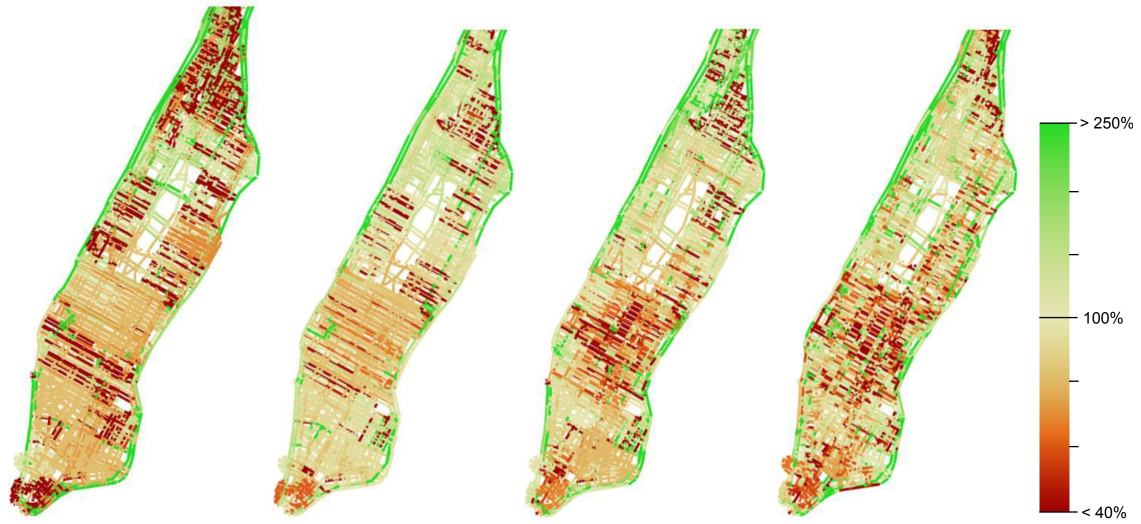
Second, this pure data-driven approach does not address the routing aspect of the problem: with no knowledge of the network it cannot possibly provide information as to which path should be used. These two drawbacks justify the use of our more complicated network optimization approach, but the k-nearest neighbors scheme remains

	k-NN		Optimization		[147]
Training trips	Best k	RMSLE	Best λ	RMSLE	RMSLE
100,000	16	0.3243	1e3	0.3595	-
10,000	11	0.3636	1e3	0.3775	-
1,000	7	0.4296	1e3	0.4019	0.8228
100	6	0.5556	1e3	0.4495	0.8822

Table 3.3: Effect of data density on k-nearest neighbors (k-NN), our optimization method, and [147]'s method on the out-of-sample RMSLE. The best values of k and the continuity parameter λ are chosen. As before, the convergence threshold δ is set to 0.5. For large amounts of data, k-nearest neighbors is unsurprisingly more accurate than our optimization-based method (although not by much), but it performs much worse in a low-data environment. [147]'s method is 10-20 times slower than ours (untractable for 10,000 trips or more), and is less accurate.

a useful benchmark of our performance. Of course, we do not expect to produce more accurate estimates than a k-nearest neighbors scheme when a wealth of taxi trips is available. With a good method, however, we should be able to obtain more accurate travel times than k-nearest neighbors in zones without much data, and only slightly less accurate in zones where data is plentiful.

High Accuracy in Data-Poor Environments. To evaluate the impact of the data-set size on our method, we compute travel times and paths for varying amounts of training data and compare the obtained RMSLE values on the testing set with those produced by the k-nearest neighbors approach. We also compare our results to those produced by the travel time estimation method in [147] (for the small amounts of data where it is tractable). The results are presented in Table 3.3: though, as expected, the k-nearest neighbors scheme outperforms the optimization method for high amounts of data, it is significantly less accurate in a data-poor setting. Meanwhile, [147]'s method is less accurate and also untractable for more than a small number of trips, as the runtime was 10-20 times longer than ours (due to the much larger size of the network as compared to the one used by the authors). Looking at the results, it seems that with our method, simply recording the origin, destination and travel time of 100 taxi trips is enough to accurately estimate the traffic patterns in an entire city.



(a) $N = 10^2$ (b) $N = 10^3$ (c) $N = 10^4$ (d) $N = 10^5$

Figure 3-9: Edge travel times in Manhattan (9-11AM) estimated by our algorithm for an increasing number of input taxi trips N. The color of each edge represents the speed along that edge as a percentage of the reference velocity $v_0 = 13.85$ kph (average velocity in Manhattan on weekdays). With just 100 taxi trips, the algorithm is able to identify that Midtown is generally congested, especially in the area around Times Square and Penn Station, and that the shoreline highways are very fast. As N increases, congestion patterns become more precise, and smaller congested areas become apparent, for example around freeway ramps. For $N = 100,000$ (the largest size that allows our algorithm to converge in less than 2 hours), we obtain a detailed, edge-by-edge description of Manhattan traffic, without losing sight of global congestion patterns.

The accuracy gap between our method and k-nearest neighbors is noticeable, especially when you consider that our method also provides a path for any (o, d) pair in the network, which a simple k-nearest neighbors scheme can never provide since it has no knowledge of the network. Therefore, in a data-poor environment, our scheme is superior to a purely data-driven one in terms of accuracy and routing, and both methods have a running time that is appropriate for the application (a few seconds for k-nearest neighbors, a few minutes for our method). In a higher-data environment we pay for the added routing information with a decrease in accuracy of just fractions of a minute and an increase in computational time.

3.6 Conclusions

The method proposed in this chapter leverages a simple approach to tractably yield accurate solutions to the travel time estimation and routing problem in a real-world setting. Given trip times for any number of origin-destination pairs, from a few hundred to a few hundred thousand, we can estimate the travel time from any origin to any destination, as well as provide a sensible path associated with this travel time. Furthermore, our algorithm is robust to a high degree of input uncertainty, successfully exploiting very noisy data to provide results characterized by their accuracy.

Providing travel times for each arc in the city effectively augments the network with a cost function based on real traffic information, which can be of use both for city planners and for further network-based research. Using our optimization-based framework, we can estimate traffic patterns in a real-world network, providing insight at every scale, from a few blocks to the entire city, and extracting global meaning from the observed data.

Chapter 4

Optimizing Schools' Start Time and Bus Routes

In the twenty-first century, school districts across the US face a wide array of challenging problems on a daily basis, from adjusting to the digital age to educating an increasingly diverse and multicultural student body. Yet perhaps the most complicated decision that administrators face is seemingly the most innocuous: determining what time each school in the district should start in the morning and end in the afternoon.

The issue of choosing appropriate school "bell times" has received increased attention in recent years, as too-early start times have been linked to a wide array of health issues among teenagers, including diminished academic achievement [32] and cognitive ability [43, 107], and increased rates of obesity [34], depression [58], and traffic accidents [44]. Indeed, changes in the body's circadian clock during puberty effectively prevent adolescents from getting adequate sleep early in the night [42]. While the American Academy of Pediatrics recommends that teenagers not start their school day before 8:30AM, a recent CDC report found that just 17.7% of U.S. high schools comply [139]. Some experts estimate that over the next ten years, the dire public health implications of early high school start times could impact the U.S. economy by over $80 billion [67].

Moreover, research suggests that these repercussions disproportionately affect the most economically disadvantaged students [56]. As achievement gaps between students from different backgrounds remain stark [140], research has consistently found systematic biases, largely on racial lines [37], that partially explain these gaps. For example, school bell times can suffer from such biases, as is the case in Boston [126].

For decades, school districts across America have considered ways to adjust their bell times and solve these issues in a fair way. However, the sheer complexity of the problem is a major obstacle to change. School districts typically struggle with balancing many competing objectives, including student health, special education programs, parent and staff schedules, state and federal regulations, and public externalities [93].

Perhaps the greatest obstacle to adjusting school bell times is the effect of changes on school transportation. Over 50 percent of U.S. schoolchildren rely on an army of half a million yellow school buses to travel to and from school every day. In Boston, where specialized programs draw students from all over the city and traffic is often at a standstill, transportation accounts for over 10% of the district's $1 billion budget. To reduce transportation spending, school districts such as Boston stagger the start and end times of different schools, allowing vehicles to be re-used several times throughout the day. Because many school districts construct bus routes by hand, it is exceedingly difficult for them to evaluate the impact of bell time changes on bus costs, let alone find a set of bell times that satisfies all of the district's objectives without inflating the budget. No matter how unpalatable, the status quo is often the only viable option. In addition, because of the impossibility of systemwide change, districts may experiment with a piecemeal approach to bell time change, where the most vocal and best connected schools may benefit the most.

The problem of school bus routing has been addressed extensively [50, 109]. It is typically decomposed into three main subproblems (see Fig. 4-1D-F): stop assignment, i.e., choosing locations where students will walk from their homes to get picked up; bus routing, i.e., linking stops together into bus trips; and bus scheduling, i.e., combining bus trips into a route that can be served by a single bus. State-of-the-art optimization algorithms exist for these subproblems in isolation [123, 61]. However, the literature on optimally combining subproblem solutions is less extensive.

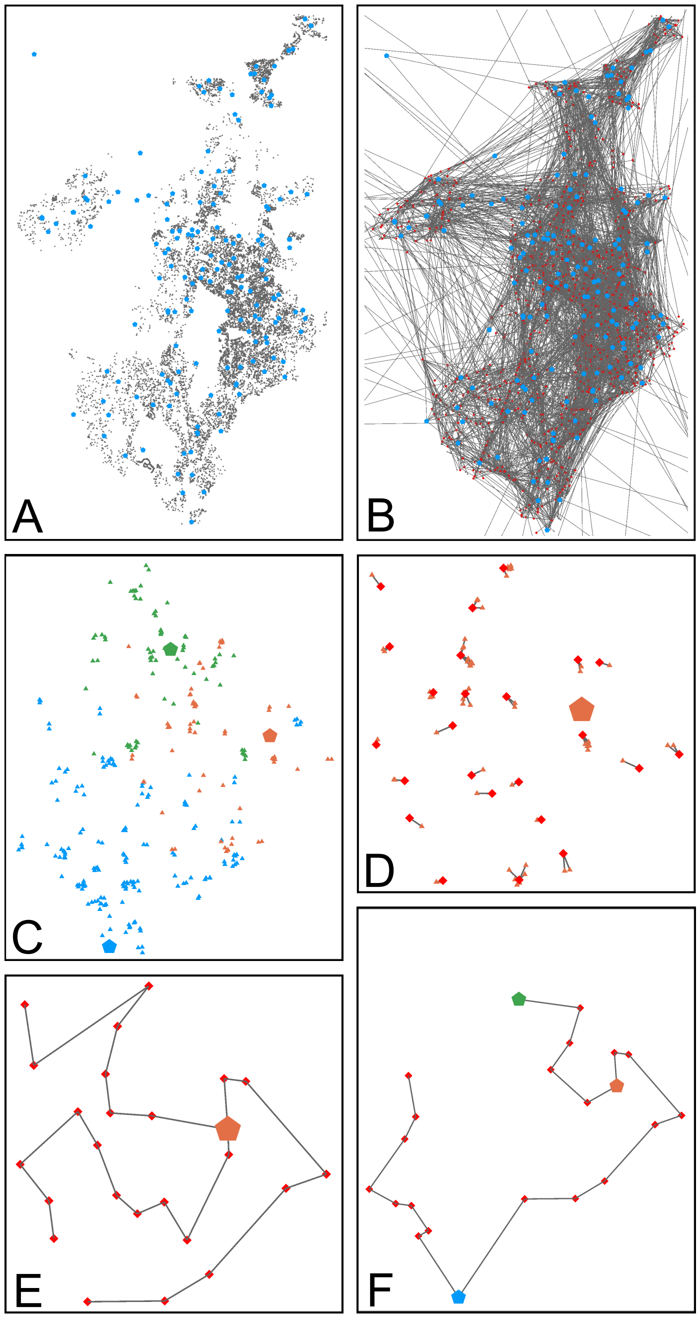


Figure 4-1: Geographic visualization of the school bus routing problem (and subproblems). **(A)** BPS 2017-18 data (anonymized) with gray triangles representing students and blue pentagons representing schools. **(B)** Sample BPS routing solution, with schools as blue pentagons, bus stops as red squares, and lines connecting bus stops that are served in sequence by the same bus, illustrating the complexity of Boston school transportation. **(C)** Small synthetic district (3 schools); students (triangles) are the same color as their assigned schools (pentagons). **(DEF)** Example of 3 main routing steps in this district: stop assignment **(D)**, where students (triangles) attending the orange school (pentagon) are shown connected to their assigned stops (red squares); one-school routing **(E)**, where all bus stops for the orange school are connected into bus trips; and bus scheduling between multiple schools **(F)**, where three trips (one from each school) are connected into a single bus itinerary.

Approaches typically involve formulating the school bus routing problem as a large combinatorial optimization problem which can be solved using metaheuristics, including local search [128], simulated annealing [35], and special-purpose vehicle routing heuristics [26, 25]. Special-purpose algorithms have also been designed to address variants of the school bus routing problem, allowing "mixed loads" – students from different schools riding the bus together [128, 25, 110], bus transfers [23], or arrival time windows [61, 128, 35, 110].

Unfortunately, many tractable general-purpose algorithms do not consider additional constraints (fleet heterogeneity, student-specific needs) and thus lack portability. Though an optimization framework to the School Time Selection Problem has been proposed [138], no existing algorithms address bell time selection in conjunction with bus routing [61].

This work presents a new model for the STSP, allowing the joint optimization of school bell times with school bus routes. We first develop a new school bus routing algorithm called BiRD (Bi-objective Routing Decomposition) which bridges the gap between standard subproblems to find better solutions. We then propose a mathematical formulation of the STSP, a multi-objective approach that can model any number of community objectives as well as transportation costs using BiRD.

BiRD outperforms state-of-the-art methods by 4% to 12% on average on benchmark data sets, and allowed Boston Public Schools (BPS) to take 50 buses off the road and save almost $5 million in the fall of 2017, without increasing the average student's walking or riding times. Our modeling approach to the STSP, along with the successful implementation of BiRD, led the Boston School Committee to reconsider start time policies for the first time since 1990, unanimously approving a comprehensive reform prioritizing student health in December 2017. Our STSP model was used by BPS to evaluate the impact of many different scenarios and ultimately propose new bell times for all 125 BPS schools. These start times are being reviewed at the request of parent groups, and our approach remains central to Boston start time policy.

We first present general overview of our work, and we group most of the technical content and experiment in the later "Technical Details" sections for better readability.

4.1 School Transportation: A BiRD's Eye View

Solving the school bus routing problem means assigning students to stops near their homes, selecting which bus will pick them up and in what order (keeping in mind that a bus only carries students for one school, but can serve several schools in succession thanks to staggered bell times), in a way that minimizes the overall number of buses, or another objective of interest. We show an example of a school district (BPS) in Fig. 4-1A and of a model school district that mimics the real setting in Fig. 4-1C and in the SI Appendix, Fig. S2.

The BiRD algorithm consists of several steps (see Fig. 4-2) for which we develop optimization-based approaches, implemented with modern software tools [20, 91] and available online [48]. For clarity, we focus on the morning problem, but our algorithm generalizes to the afternoon (see SI Appendix). Because problem details often vary between districts, it may be advantageous to adjust some steps to changes in the problem setting. BiRD's defining feature is thus the decomposition of the problem, and in particular the scenario selection step which bridges the gap between the single-school and multi-school subproblems.

4.1.1 Single-School Problem

To assign students to stops (Fig. 4-1D), we use an integer optimization formulation of the assignment problem, with maximum walking distance constraints. We minimize the overall number of stops because (i) it simplifies bus trips and (ii) the minimum pickup time at a stop is typically high, even if the stop has few students. When long bus routes span the entire city, as in Boston (see Fig. 4-1B), stop assignment has a negligible effect on the macroscopic quality of the routing solution. Our formulation can include additional objectives, such as the total student walking distance, and can exclude stop assignments that require students to cross major arteries or unsafe areas (see SI Appendix, Stop Assignment).

We then use an insertion-based algorithm to connect sequences of stops into feasible bus trips (Fig. 4-1E). We use integer optimization to combine these feasible trips

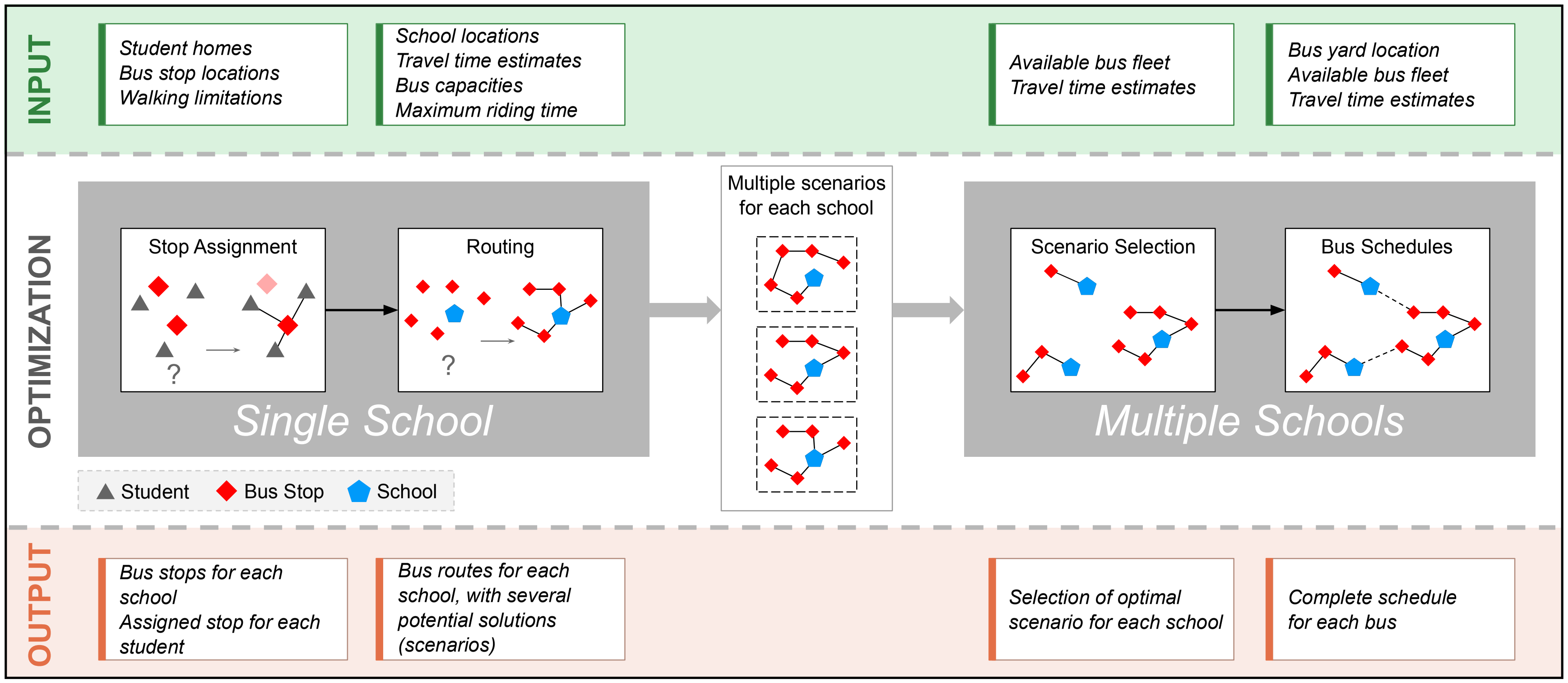


Figure 4-2: Overview of BiRD algorithm. On the left, the single-school problem can be divided into the two subproblems of stop assignment and single-school routing; on the right, the multi-school problem can be divided into the two subproblems of scenario selection and bus scheduling. The generation of not one, but several routing scenarios for each school, and the subsequent joint selection of a single scenario for each school, bridge the divide between the single-school and multi-school problems.

with a minimum number of buses, with a set cover formulation reminiscent of crew scheduling problems [127] (see SI Appendix, Single-School Routing). Our method has the flexibility to handle practical modifications in the routing problem, from vehicles with different capacities to student-bus compatibility restrictions (e.g. students in a wheelchair need a bus with a special ramp/lift). In principle, the modularity of the overall algorithm means that the single-school routing algorithm can be replaced with any state-of-the-art vehicle routing method.

4.1.2 Routing Multiple Schools

We use the single-school routing method to generate not one, but several varied optimized routing scenarios for each school, in order to select the best one for the system. In particular, we consider several scenarios on the Pareto frontier of two objectives (hence the name of Bi-objective Routing Decomposition), number of buses and average riding time. This tradeoff makes sense because shorter routes are more easily connected into bus schedules.

Then, we first jointly select one scenario for each school in a way that favors maximal re-use of buses from school to school (Fig. 4-2), by formulating an integer optimization problem with network flow structure that seeks to minimize the number of buses at the scale of the entire district (see SI Appendix, Scenario Selection). Given one routing scenario for each school, we can then solve another integer optimization problem to identify a trip-by-trip itinerary for each bus in the fleet (Fig. 4-1F). In this final subproblem, we optimize the number of buses jointly in the morning and in the afternoon (see SI Appendix, Bus Scheduling).

4.1.3 Evaluating the Routing Algorithm

We compare BiRD's ability to minimize the total number of buses to existing methods [25, 35], on 32 published benchmarks [110] and on 20 of our own synthetically-generated examples. We outperform all other methods on all but one instance, with an average improvement of 4% on the instances from [110] and 12% on our instances.

The scenario selection step is key to this improvement: computational experiments (see SI Appendix, Routing Experiments) indicate that BiRD's performance improves by 20% when we compute two different routing scenarios for each school and select the best one by considering the whole system, as opposed to using the best scenario for each school. Intuitively, what is optimal for one school may not be optimal for the entire system, motivating the bi-objective decomposition approach.

4.1.4 Application in Boston

BPS has the highest transportation expenditure per student in the U.S., with rising costs due in part to narrow streets and infamous rush-hour traffic, a large fraction of special education students, and a complicated history of school desegregation. In addition, over the last decade BPS has adopted a "controlled choice" approach to school selection, which gives parents greater latitude in selecting a public school while promoting fairness across the district [1, 111]. As a result of this policy, some schools may draw students from far across the city, further complicating the school transportation problem and driving up costs.

Before we started working with BPS, bus routes for 125 public schools and over 80 private and charter schools were computed and maintained manually. BiRD's ability to incorporate district-specific constraints (including four different bus types, and only one compatible with wheelchairs) was essential in producing a practical solution. In the end, we solved the Boston school bus routing problem using only 530 buses, against 650 for the manual solution. This represents an 18% reduction, with estimated cost savings in the range of $10 to $15 million. To ensure a smooth transition, BPS decided to only take 50 buses off the road in the first year of implementation, still amounting to a hefty $5 million in cost savings [103]. Despite the smaller number of buses, the average student ride time stayed constant from 2016-2017 (around 23 minutes).

4.2 Formulating the STSP

Selecting bell times is a complex policy problem with many stakeholders. We first focus on the interplay with transportation, since computing school bus routes is a necessary component of bell time selection. For instance, it is of interest to evaluate transportation costs when each school S is assigned a particular bell time t_S. However, there are too many possibilities to explore in practice (exponential in the number of schools). Instead, we develop a general formulation for the STSP, which contains a tractable proxy for transportation cost constructed using BiRD. We show how to include other community objectives in the next section.

4.2.1 Transportation Costs

A key factor in an optimized school bus routing solution is the "compatibility" of pairs of trips, i.e., how easy it is for a single bus to serve them with minimum idle time in between. We define a trip compatibility cost that trades off (a) the feasibility of a bus serving the two trips sequentially, and (b) the amount of idle or empty driving time involved, with tradeoff parameters that depend on characteristics of the school district, and can be found using cross-validation. Then, for any pair of schools S and S', we can define a routing pairwise affinity cost $c^{\text{routing}}_{S,t,S',t'}$ that is the sum of the compatibility costs between every trip in every routing scenario for S at time t and S' at t' (see SI Appendix, Transportation Costs).

4.2.2 Optimizing

Because its objective function only includes pairwise affinity costs, our model of the STSP is a special case of the Generalized Quadratic Assignment Problem (GQAP) [68]. When different GQAP formulations for the STSP were investigated in [138], even small instances could be intractable. We therefore develop a simple local improvement heuristic that works well in practice. Given initial bell times, we select a random subset of schools. The problem of finding the optimal start times for this subset, while fixing all other schools' start times, is also a GQAP.

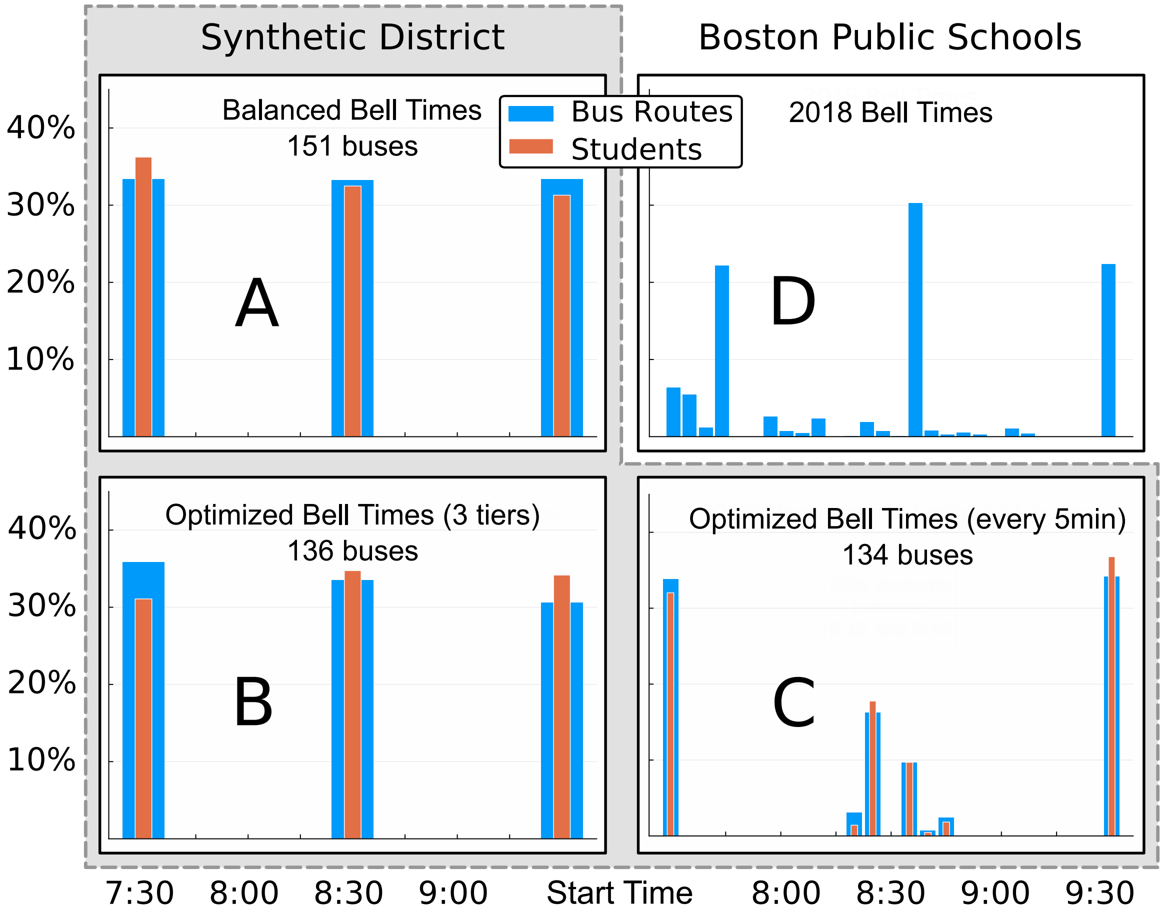


Figure 4-3: Bell time optimization. Comparison of 3 bell time optimization strategies on a synthetic district. When only three bell times are allowed, balancing the number of bus routes across bell times **(A)** works well, but is typically beaten by routing compatibility optimization **(B)**. Even better solutions can be obtained by allowing more bell times in the middle tier **(C)**. In comparison, BPS bell times are not even balanced **(D)**.

We can then solve this restricted GQAP problem using mixed-integer optimization to obtain a new set of bell times, in seconds for small enough subsets. We repeat the operation with new random subsets until convergence. Results on synthetic data suggest that a subset size of one gives near-optimal results, if the local improvement heuristic is run several times with random starting points. We note that the heuristic is interpretable: with a subset size of n, a solution obtained after convergence can only be improved by changing the bell times of at least $n+1$ schools.

4.2.3 Evaluating three-tier systems

In many districts, such as Boston (Fig. 4-3D), start times are separated into three equally spaced "tiers" (e.g. 7:30AM, 8:30AM and 9:30AM). Such a system allows each bus to serve up to three schools every morning [106], so districts will typically try to balance the number of bus trips across all three tiers. Our method allows us to quantify the empirical behavior of this intuitive idea.

Simulations suggest that optimizing three-tier bell times using our algorithm (Fig. 4-3A) yields an 11% cost improvement over simply balancing the number of bus routes across tiers (Fig. 4-3B), which is already better than what school districts typically do (Fig. 4-3D). Distributing schools across tiers without accounting for geography/routing compatibility is suboptimal.

Furthermore, a three-tier system is not necessarily the right answer *per se*. Fig. 4-3C shows that allowing many possible start times for the middle tier (5-minute intervals between 8:00AM and 9:00AM) can yield a 1-2% improvement over the standard three-tier optimized solution (Fig. 4-3B). Interestingly, no school starts at 8:30AM in this system. Though tiered bell times are popular because of their simplicity, algorithmic tools such as ours suggest that better solutions exist. For instance, in Boston, we can find a bell time solution that requires just 450 buses, which represents a 15% improvement over the number of buses obtained without changing the bell times, and a 31% improvement over the number of buses used by BPS in the 2016-2017 school year.

4.3 Bell Times in Practice

In a real district, bell time selection goes far beyond minimizing the number of buses, as we found in our work with BPS. For context, Boston's existing bell time policy, enacted in 1990, split the public schools into three tiers, with start times of 7:30AM, 8:30AM, and 9:30AM, stipulating that tiers would rotate through the start times every 5 years. Unfortunately, this policy was never enforced, and the bell times assigned in 1990 mostly remain today.

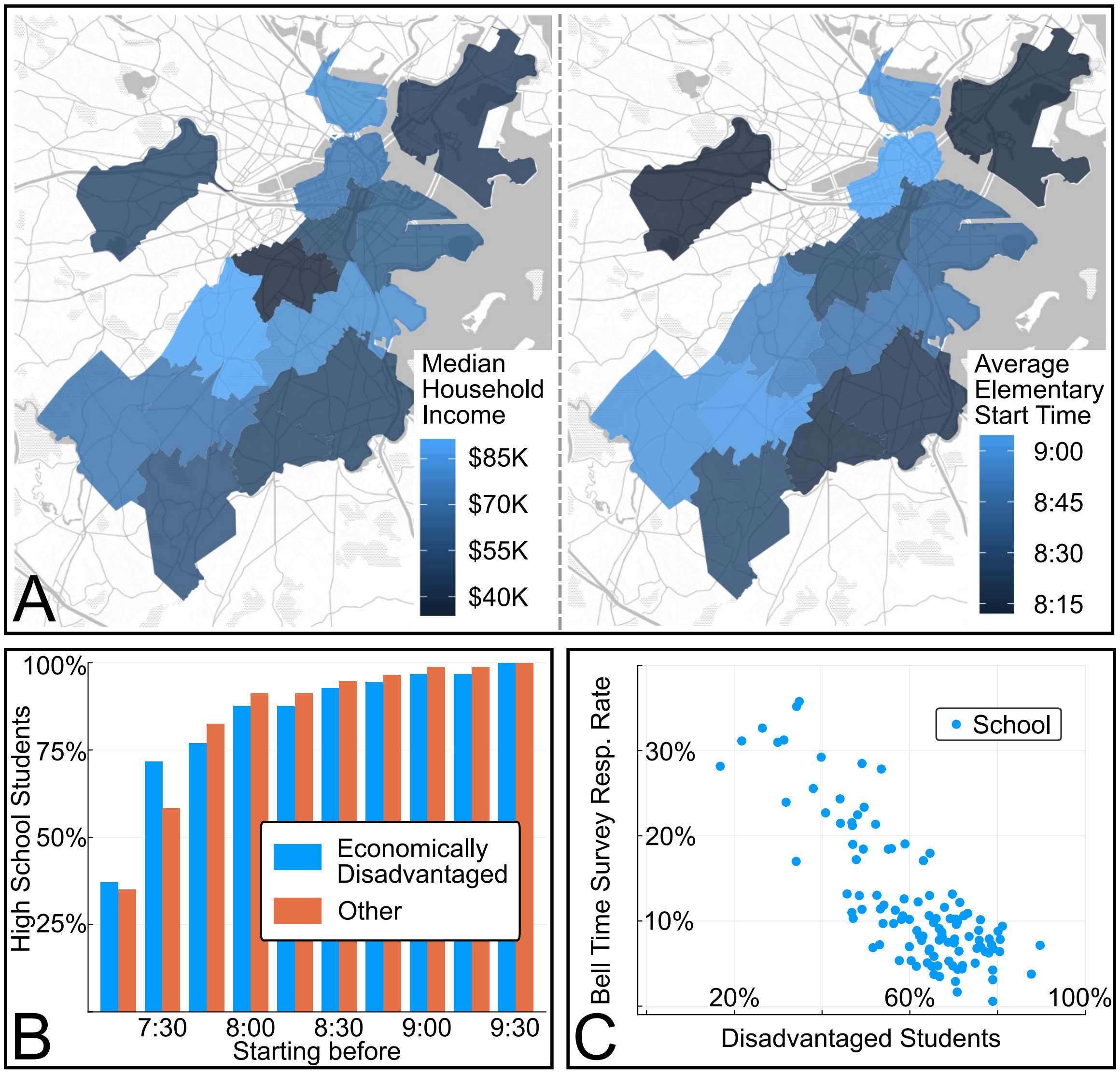


Figure 4-4: Equity and current start times in Boston. **(A)** Maps of Boston, with neighborhoods shaded by median household income (ACS) and average elementary start time. Elementary students start later in wealthier neighborhoods (0.78 correlation between household income and start time). **(B)** Proportion of high school students starting before each time in the morning (comparing economically disadvantaged and other students). Start times skew early for economically disadvantaged high school students (χ^2 homogeneity p-value $< 10^{-5}$). **(C)** BPS Community Survey response rate by school, shown against fraction of disadvantaged students attending the school. Economically fragile populations have a lower bell time survey response rate.

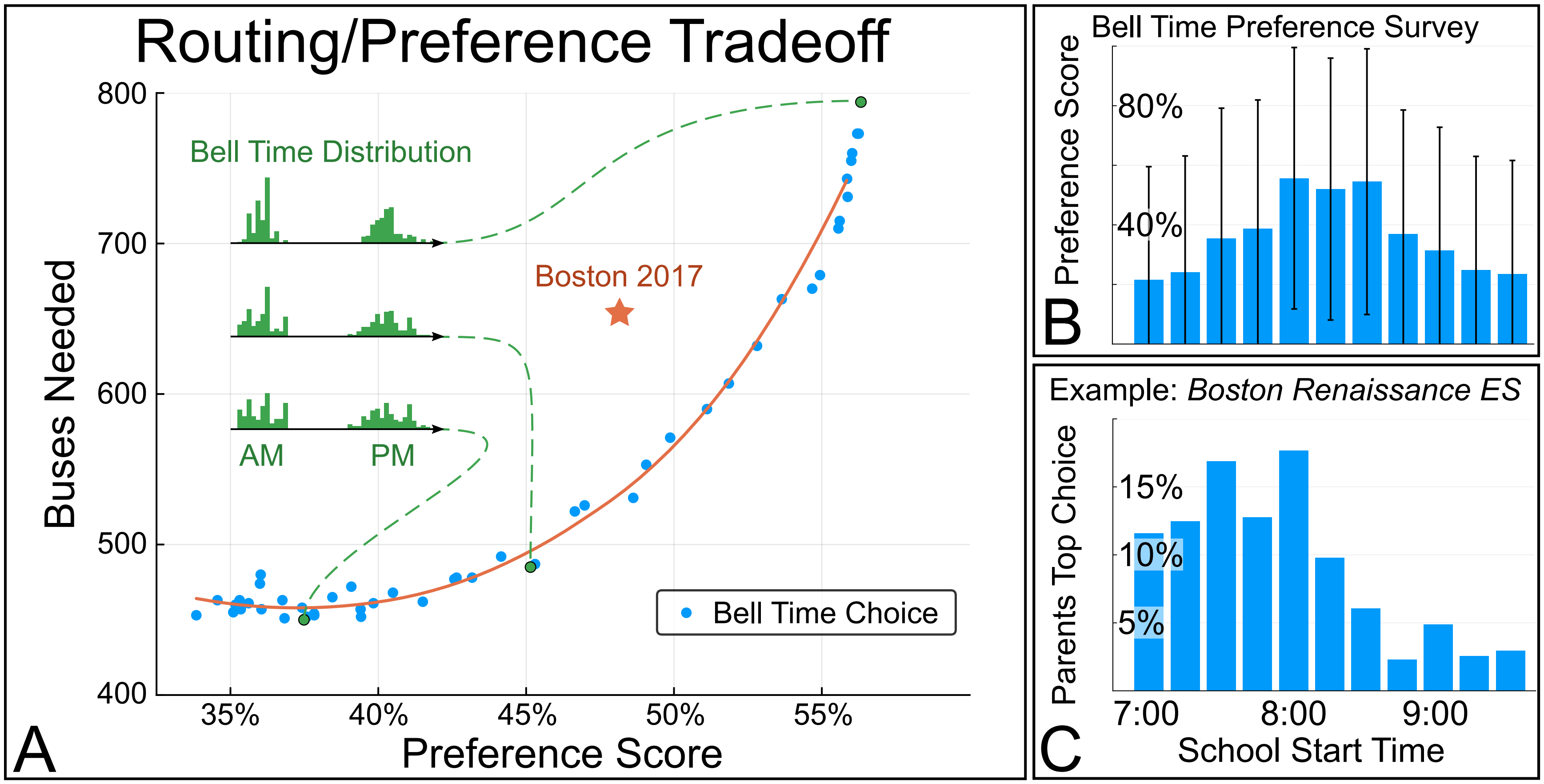


Figure 4-5: Optimizing preferences is hard. (A) Tradeoff curve derived by our algorithm between preference score (metric of community satisfaction – see the SI Appendix, Boston Community Survey) and transportation cost, along with three sets of bell times along the curve. Even a slight improvement in satisfaction comes at a high cost. (B) District-wide preference score of each bell time, showing that parents typically prefer 8:00AM to 8:30AM start times, with high variance. (C) Distribution of parents' top bell time choice at a particular school (Boston Renaissance). Even within a single school, agreement is hard to come by: even though the school's current start time is 7:30AM, only 17% of parents list this time as their favorite.

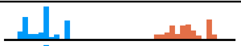
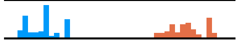
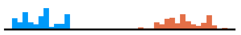
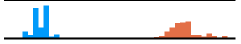
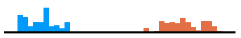
	Buses	Early HS	Late ES	Survey score	Bell time distribution
Current	650	74%	33%	48%	
NewRoutes	530	74%	33%	48%	
LowCost	450	43%	27%	37%	
MaxSurvey	934	0%	8%	56%	
Optimal	481	6%	15%	40%	

Figure 4-6: Bell time selection tradeoffs. Sample of a few scenarios considered by BPS. Current start times (with or without new routes) have many high school students starting before 8:00AM (Early HS) and elementary school students ending after 4:00PM (Late ES), mediocre community satisfaction (survey score), and a suboptimal bell time distribution both in the morning and in the afternoon (histogram weighted by students – blue AM, orange PM). The three other scenarios present different tradeoffs between the bell time objectives – BPS chose the "Optimal" scenario.

These bell times are flawed. First, because they have remained static while school demographics have evolved, they have contributed to the steady rise of the BPS transportation budget over the last decade. Second, over 74% of high school students currently start school before 8:00AM. Many studies have shown that the negative effects of early high school starts are magnified in economically fragile students [56]. However, in Boston such students have worse bell times, on average, than economically advantaged students [126]. In Fig. 4-4, we see for example that economically disadvantaged high school students are more likely to start before 7:30AM than other high school students.

4.3.1 Gridlock

The Boston status quo has persisted for decades despite its shortcomings. Indeed, bell time selection is intrinsically difficult because stakeholders cannot agree on what is best for everyone. Figs. 4-5B and 4-5C show community preferences for different start times across all public schools, obtained through a BPS survey. Though families and school staff tend to favor start times between 8:00AM and 8:30AM, the displayed preferences are mostly characterized by broad disagreement, even within a single school (Fig. 4-5C). Any bell time for any school is sure to have both fervent supporters and vehement critics.

School districts have no hope of satisfying all, or even most, of their constituents. Moreover, the cost of even trying to satisfy the individual preferences of parents and staff can be prohibitive: Fig. 4-5A shows that each additional point of community satisfaction in Boston can cost dozens of additional buses and tens of millions of taxpayer dollars.

For BPS, the tradeoff curve in Fig. 4-5A represented a paradigm shift, the first time the district could visualize, or even quantify, any of the tradeoffs of bell time policymaking. The curve illustrates our model's first use: providing a district the quantitative support necessary to understand the problem and make the best decision.

4.3.2 The Greater Good

Though stakeholders have many competing personal priorities, they often agree on broader goals, such as having fair and equitable bell times or reinvesting saved transportation costs into schools. Starting in 2016, BPS led an engagement process aiming to understand broad community values. The results suggested four main objectives: to maximize how many high school students start after 8:00AM, minimize how many elementary school students end after 4:00PM, prioritize schools with high special education needs, and reinvest transportation savings into classrooms, while achieving these objectives in an equitable manner.

In the general case, solving the STSP in practice means optimizing a set of several objectives, such as the ones outlined above. We call an objective GQAP-representable if it can be represented using only single affinity costs $c_{S,t}$ (representing the aversion of school S for bell time t) and pairwise affinity costs $c_{S,t,S',t'}$ We find that the GQAP framework has sufficient modeling power to represent all the objectives and constraints that interest school districts in general (see SI Appendix, GQAP-Representable Objectives) and Boston in particular.

Typically, school districts will wish to balance multiple GQAP-representable objectives, including transportation costs. As is usual in multi-objective optimization, we consider that the final cost function to optimize is a weighted average of the district's different (GQAP-representable) objectives, with weights indicating policy makers' priorities.

We explored tens of thousands of tradeoffs for BPS, such as those presented in Fig. 4-6. We notice that in Boston, reducing both the number of high school students starting too early and the number of elementary school students ending too late can be done at little to no cost.

4.3.3 Application in Boston

In December 2017, the Boston School Committee unanimously approved a new policy [129], stipulating that all future bell time solutions should optimize the verifiable criteria described above, paving the way for algorithmic bell time selection. Our flexible methodology allowed us to take into account a number of very specific constraints, e.g. preventing large neighboring high schools to dismiss at the same time (which could create unsafe situations at neighboring MBTA stops). In the end, the proposed bell times (see Fig. 4-6) reduced the number of high school students starting before 8:00AM from 74% to 6%, and the number of elementary school students dismissing after 4:00 from 33% to 15%. The plan also led to an estimated reinvestment of up to $18 million into classrooms. Due to the significant amount of change under this new plan, and in response to protests by some families, BPS delayed the plan's implementation to allow more time to adjust the objective weights and constraints. As BPS continues to gather community input, the legitimate concerns raised by these families can be modeled as objectives within our general formulation and integrated within our framework.

Ultimately, using an algorithm for bell time selection at the scale of a city allows leaders to thoroughly evaluate their options, and empowers them to make decisions based not on the political whims of special interest groups, but on an objective standard agreed upon by the community.

In the following technical details sections, we dive into the algorithms, experiments and mathematical formulations. We introduce the mathematical notation and formalism needed to formulate and solve different optimization subproblems. We first describe the BiRD school bus routing algorithm in full detail, then we specify the setting of our computational experiments, in particular with respect to synthetic data. Subsequently, we present our mathematical formulation of the School Time Selection Problem (STSP), explain the details of our optimization algorithm and how it interfaces with school bus routing, before detailing our computational work, on both synthetic and real data.

4.4 Technical Details: BiRD Routing Algorithm

We begin by giving a complete mathematical description of the BiRD (Bi-objective Routing Decomposition) algorithm for school bus routing. In the first sections, we decompose the overall problem of school transportation into a single-school problem and multi-school problem. The single-school problem can be further decomposed into the two subproblems of stop assignment and single-school routing, while the multi-school problem can be further decomposed into the two subproblems of scenario selection and bus scheduling (the overall decomposition is detailed diagrammatically in Fig. 2). In this section, we detail the four subproblems of stop assignment, single-school routing, scenario selection and bus scheduling in order. Throughout the section, the (mixed-)integer optimization problems that we formulate are solved using a mixed-integer optimization solver.

4.4.1 Stop Assignment

Call $\mathcal{S}$ the set of schools, and $\mathcal{P}_s$ the set of pupils (students) attending each school $S \in \mathcal{S}$. In addition, call $\mathcal{C}$ the set of all locations that can serve as bus stops. Each student $p \in \mathcal{P}_S$ is associated with a set of allowed bus stops $\mathcal{C}_p \subseteq \mathcal{C}$. The walking distance from the home of student p to a stop $c \in \mathcal{C}_p$ is denoted as $d_{p,c}$. This general setting reflects a variety of student-specific needs. For example, the allowed bus stops for younger students may be closer to their home or accessible without crossing major arteries. In addition, a pupil p with special needs may require a "door-to-door" pickup: in this case, the set $\mathcal{C}_p$ is a singleton $\{c\}$, where $d_{p,c} = 0$.

The problem of stop assignment has received attention in recent years, with the development of innovative new modeling approaches and algorithms as in Zeng, Chopra and Smilowitz [146]. We propose a simple integer optimization approach, where we seek to minimize the number of stops for each school and the total student walking distance. Similarly to a facility location problem [39], we solve the following integer program for each school S.

$$
\begin{aligned}
\min \quad & \sum_{c \in \mathcal{C}} z_c + \beta \sum_{p \in \mathcal{P}_S} \sum_{c \in \mathcal{C}_p} d_{p,c} y_{p,c} && && \text{(4.1a)} \\
\text{s.t.} \quad & y_{p,c} \le z_c && \forall p \in \mathcal{P}_S, c \in \mathcal{C}_p && \text{(4.1b)} \\
& \sum_{c \in \mathcal{C}_p} y_{p,c} = 1 && \forall p \in \mathcal{P}_S && \text{(4.1c)} \\
& y_{p,c} \in \{0,1\} && \forall p \in \mathcal{P}_S, c \in \mathcal{C}_p && \text{(4.1d)} \\
& z_c \in \{0,1\} && \forall c \in \mathcal{C} && \text{(4.1e)}
\end{aligned}
$$

The binary variable z_c indicates whether stop c is selected for school S, and the binary variable $y_{p,c}$ indicates whether student p is assigned to stop c. (4.1b) ensures that student p is assigned to stop c only if stop c is selected for school S, and (4.1c) certifies that each student is assigned to one stop. The first term in the objective corresponds to the total number of stops for school S, while the second term corresponds to the total walking distance for students attending school S. The parameter β controls the tradeoff between these two priorities. For large values of β, students will be assigned to the nearest stop to their home; as β tends to 0, students may walk further from their home in order to consolidate several stops (though never to an unacceptable stop that is not in $\mathcal{C}_p$). We explore this tradeoff on synthetic data in Fig. 4-9a.

When applying this process in Boston, we added an additional constraint preventing the same stop from having too many students at the same time. This correlates the stop assignment solutions of each school, requiring the concurrent optimization of stop assignments for all schools.

4.4.2 Single-School Routing

Given that each student p has been assigned a bus stop $c_p \in \mathcal{C}_p$, we now turn to the problem of connecting stops into bus routes (or bus trips). Since several students may be assigned to the same stop, we can denote as $\mathcal{C}_S \subseteq \mathcal{C}$ the set of bus stops with at least one student from school S, and call $n_{c,S}$ the number of students from school S at a stop $c \in \mathcal{C}_S$.

We consider that the bus fleet is composed of several bus types, and denote the set of bus types as $\mathcal{B}$. All buses of a given type $b \in \mathcal{B}$ are considered identical, with a fixed number of seats (capacity) Q_b, and a fixed number of wheelchair spots W_b. Let $\mathcal{Y}$ designate the set of bus depots (or bus yards) where buses are stored during the night and the middle of the day.

We let $t^{\text{pickup}}_{c,S}$ designate the length of time needed to pick up every student for school S at stop c, and $t^{\text{drop-off}}_{S}$ the length of time needed to drop off every student on a bus at school S. Note that the former is a function of the number of students at the stop $n_{c,S}$, while the latter is independent of the number of students on the bus. For any two locations $\ell_1, \ell_2 \in \mathcal{S} \cup \mathcal{C} \cup \mathcal{Y}$, and given a particular time of day τ, let $t^{\text{drive}}_{\ell_1,\ell_2,\tau}$ designate the driving time from location ℓ_1 to location ℓ_2, when departing location ℓ_1 at time τ. Finally, we consider that all buses serving school S must arrive at school at time τ_S to drop off their students, and are therefore free to leave the school at time $\tau_S + t^{\text{drop-off}}_{S}$. We also consider that students cannot spend more than a fixed duration $T^{\max}$ on the bus.

Throughout this chapter, we consider that the travel times $t^{\text{drive}}_{\ell_1,\ell_2,\tau}$ are deterministic and known. We note that this modeling choice makes it more difficult to account for unforeseen traffic events such as accidents. These travel times can be obtained through a commercial service like the Google Maps API, or estimated from data. In practice, school bus routes are typically constructed under a static, deterministic travel time model, and stakeholders understand that this results in buses sometimes arriving late, especially in poor weather or traffic conditions. As a simple way to mitigate the impact of traffic in our work with BPS, we artificially increased the drop-off times

$t_S^{\text{drop-off}}$ at each school, e.g. requiring buses to arrive at school 10 or 15 minutes before the beginning of school even though physically unloading students may only require 3 minutes. This "buffer" both reduces the chance that students will be late to class, and makes it less likely that delays will be compounded onto the bus's next trip. However, no matter what travel time estimates are used, it is practically impossible for buses to always be on time.

For school S, there are many methods from the vehicle routing literature to find bus trips to serve all the stops in $\mathcal{C}_S$ [50, 109, 123, 22]. A trip (or route) is simply an ordered sequence of stops visited, or served, by a bus. We assume that a bus serving a trip will pick up every student at each stop in the trip. A trip is considered feasible if (a) it verifies that no student spends more than a time $T^{\max}$ between pickup and drop-off, and (b) there exists a type of vehicle in the fleet with enough capacity to transport all students assigned to the stops served by the trip. Given a feasible trip R, we let $\mathcal{B}_R \subseteq \mathcal{B}$ designate the set of types of buses that have the necessary capacity to serve the trip, and we denote by T_R the service time of the trip, i.e. the time between arrival at the first stop and arrival at the destination school.

We use a randomized greedy heuristic (Algorithm 1) similar to Braca et al. [25, 26] to generate a set of feasible trips $\mathcal{T}_S$, making sure that each stop $c \in \mathcal{C}_S$ is served by a nonempty set of feasible trips $\mathcal{T}_c \subset \mathcal{T}_S$. More precisely, the heuristic returns a set of trips $\mathcal{T}$ that covers each stop exactly once, and we run it N times to build a set of feasible trips where each stop is covered by several trips. The heuristic returns N different solutions and each stop will be covered by N different feasible trips.

We index the trips in $\mathcal{T}_S$ by $1, \ldots, m_S$ and for a given set of trips $\mathcal{T} \subseteq \mathcal{T}_S$ we let $I(\mathcal{T})$ designate the subset of $\{1, \ldots, m_S\}$ corresponding to trips in $\mathcal{T}$. We can then

Algorithm 4 Randomized greedy insertion heuristic, assuming only one type of bus with capacity Q. Input: a school S and a time of day τ

```
1: function GreedyRandomized($S$, $\tau$)
2:     $\mathcal{T} \leftarrow \emptyset$                                   ▷ Initialize empty set of trips
3:     while $\mathcal{C}_S \neq \emptyset$ do
4:         $c$ randomly selected from $\mathcal{C}_S$
5:         $\mathcal{C}_S \leftarrow \mathcal{C}_S \backslash \{c\}$
6:         $R \leftarrow [c]$                          ▷ Trip initialized with selected stop $c$
7:         $N^{\text{students}} \leftarrow n_{c,S}$              ▷ Number of students currently on trip $R$
8:         while $\mathcal{C}_S \neq \emptyset$ do
9:             $T^{\text{new}}, c^{\text{new}}, i^{\text{new}} \leftarrow$ BestInsertion($R$, $\mathcal{C}_S$, $\tau$)
10:            if $T^{\text{new}} \leq T^{\max}$ and $N^{\text{students}} + n_{c^{\text{new}},S} \leq Q$ then
11:                $R \leftarrow$ Insert($R$, $c^{\text{new}}$, $i^{\text{new}}$)
12:                $\mathcal{C}_S \leftarrow \mathcal{C}_S \backslash \{c^{\text{new}}\}$
13:            else
14:                break
15:        $\mathcal{T} \leftarrow \mathcal{T} \cup \{R\}$
16:    return $\mathcal{T}$
17: function BestInsertion($R$, $\mathcal{C}_S$, $\tau$)
18:     Let $R := [c_1, c_2, \dots, c_n]$           ▷ Name the $n$ stops in $R$ for clarity of notation
19:     $T^{\text{best}} \leftarrow \infty$; $c^{\text{best}} \leftarrow \infty$; $i^{\text{best}} \leftarrow \infty$ ▷ Total time, inserted stop, insertion index
20:     for $c \in \mathcal{C}_S$ do
21:         for $i \leftarrow 0$ to $n$ do
22:             $T \leftarrow t^{\text{pickup}}_{c,S}$                          ▷ Include pickup time first
23:             if i = 0 then
24:                 $T \leftarrow T + t^{\text{drive}}_{c,c_1,\tau} + \sum_{j=1}^{n-1} t^{\text{drive}}_{c_j,c_{j+1},\tau} + t^{\text{drive}}_{c_n,S,\tau}$      ▷ $c$ inserted before $c_1$
25:             else if i=n then
26:                 $T \leftarrow T + \sum_{j=1}^{n-1} t^{\text{drive}}_{c_j,c_{j+1},\tau} + t^{\text{drive}}_{c_n,c,\tau} + t^{\text{drive}}_{c,S,\tau}$      ▷ $c$ inserted after $c_n$
27:             else
28:                 $T \leftarrow T + \sum_{j=1}^{i} t^{\text{drive}}_{c_j,c_{j+1},\tau} + t^{\text{drive}}_{c_i,c,\tau} + t^{\text{drive}}_{c,c_{i+1},\tau} + \sum_{j=i+1}^{n-1} t^{\text{drive}}_{c_j,c_{j+1},\tau} + t^{\text{drive}}_{c_n,S,\tau}$
    ▷ $c$ inserted between $c_i$ and $c_{i+1}$
29:             if $T < T^{\text{best}}$ then
30:                 $T^{\text{best}} \leftarrow T$; $c^{\text{best}} \leftarrow c$; $i^{\text{best}} \leftarrow i$
31:     $T^{\text{best}} \leftarrow T^{\text{best}} + \sum_{j=1}^{n} t^{\text{pickup}}_{c_j,S}$
32:     return $T^{\text{best}}$, $c^{\text{best}}$, $i^{\text{best}}$
33: function Insert(R, c, i)
34:     Let $R := [c_1, c_2, \dots, c_n]$
35:     if i = 0 then
36:         $R \leftarrow [c, c_1, \dots, c_n]$
37:     else if i=n then
38:         $R \leftarrow [c_1, \dots, c_n, c]$
39:     else
40:         $R \leftarrow [c_1, \dots, c_i, c, c_{i+1}, \dots, c_n]$
41:     return R
```

find the best set of routes by solving the following minimum cover problem [127].

$$\begin{aligned}
\min \quad & \lambda \sum_{i=1}^{m_S} r_i + \sum_{i=1}^{m_S} r_i \Theta_i && && \text{(4.2a)}\\
\text{s.t.} \quad & \sum_{i \in I(\mathcal{T}_c)} r_i \geq 1 && \forall c \in \mathcal{C}_S && \text{(4.2b)}\\
& r_i \in \{0, 1\} && \forall i \in \{1, \ldots, m_S\} && \text{(4.2c)}
\end{aligned}$$

The binary variable r_i indicates whether trip i is selected. Constraint (4.2b) imposes that every stop must be served by at least one trip. For each trip i in $\mathcal{T}_S$, we have $\Theta_i = \sum_{p:c_p \in C_i} \theta_p^{(i)}$, where C_i represents the list of stops served by the i-th trip i, and $\theta_p^{(i)}$ corresponds to the time spent by student p on the bus during trip i. Thus, the parameter λ controls the importance of the number of trips relative to the total time students spend on the bus.

In the optimal solution of the problem above, some stops may be served by more than one trip. We consider the set of trips $\mathcal{T}_c^{\text{opt}} \subset \mathcal{T}_S$ serving each such stop c. For each $i \in I(\mathcal{T}_c^{\text{opt}})$, we compute the improvement δ_i to the objective of (4.2) that would be obtained by removing stop c from trip i. We let c remain in trip j, such that $j = \arg\min_{i \in I(\mathcal{T}_c^{\text{opt}})} \delta_i$, and remove it from all other trips in $\mathcal{T}_c^{\text{opt}}$.

To gain tractability at the expense of full optimality, it is easy to split up this trip selection phase into K phases, where the first phase selects the best trips among the first N/K greedy solutions, then the second phase selects the best trips among the trips from the next N/K routing solutions and the optimal trips from the first phase, etc.

The output of this algorithm is a set of trips $\mathcal{T}_S^*$ that covers every stop in $\mathcal{C}_S$ exactly once. We can perform this iterative optimization algorithm several times for different values of the tradeoff parameter λ to obtain an array of varied routing scenarios for each school. Low values of λ will lead to scenarios with more buses but shorter trips, while high values of λ will produce scenarios with longer trips but fewer buses.

For each school $S \in \mathcal{S}$, we therefore end up with a set of routing scenarios $\mathcal{R}_S =$

$\{\mathcal{T}_S^h\}_{h=1}^{h=h_S}$, where each scenario is a complete set of trips $\mathcal{T}_S^h$ that covers every stop in $\mathcal{C}_S$ exactly once. Each scenario is located in a different region of the Pareto front between two objectives, the number of buses and the average student travel time, hence the name of Bi-objective Routing Decomposition (BiRD).

Because our overall school bus routing approach is modular, the methods we propose to solve the single-school routing problem can easily be replaced, e.g. by heuristic approaches [35]. The only requirement on such substitute approaches is that they be able to produce several different solutions for each school, trading off between the average time students spend on the bus and the number of bus trips.

4.4.3 Scenario Selection

We develop a bus scheduling algorithm that bridges the gap with the bus routing problem. The algorithm takes as input a set of scenarios $\mathcal{R}_S$ of size h_S for each school S. Because what is optimal for one school may not be optimal for the entire system, our goal is to jointly select one scenario for each school in a way that minimizes the desired objective (e.g. number of buses) across the whole district.

In order to select a single-school solution for each school in a way that minimizes the total number of buses, we formulate an integer network flow problem on a graph where nodes represent bus trips, which are served when they are traversed by a unit of flow. To simplify notation, we assume at first there exists only one type of bus; we then create the following "scenario selection graph" $(\mathcal{N}, \mathcal{A})$. The set of nodes $\mathcal{N}$ consists of (i) a single depot node y, (ii) a "trip node" $\rho_{S,h,R}$ for each school S, each scenario index $1 \leq h \leq h_S$, and each trip $R \in \mathcal{T}_S^h$, and (iii) an "availability node" $a_{S,h}$ for each school S and for each routing scenario $1 \leq h \leq h_S$. The set of arcs $\mathcal{A}$ includes an arc from y to each trip node $\rho_{S,h,R}$, from each trip node $\rho_{S,h,R}$ to the corresponding availability node $a_{S,h}$, and from each availability node $a_{S,h}$ back to the depot y. We also include an arc from each availability node $a_{S,h}$ to each trip node $\rho_{S',h',R'}$ where trip $R' \in \mathcal{T}_{S'}^{h'}$ is time-compatible with a bus starting from school S. By time-compatible we mean that there is enough time for the bus to drive from school S to the first stop $c_{R'}^{\text{start}}$ of trip R' and then make it to school S' on time, which can

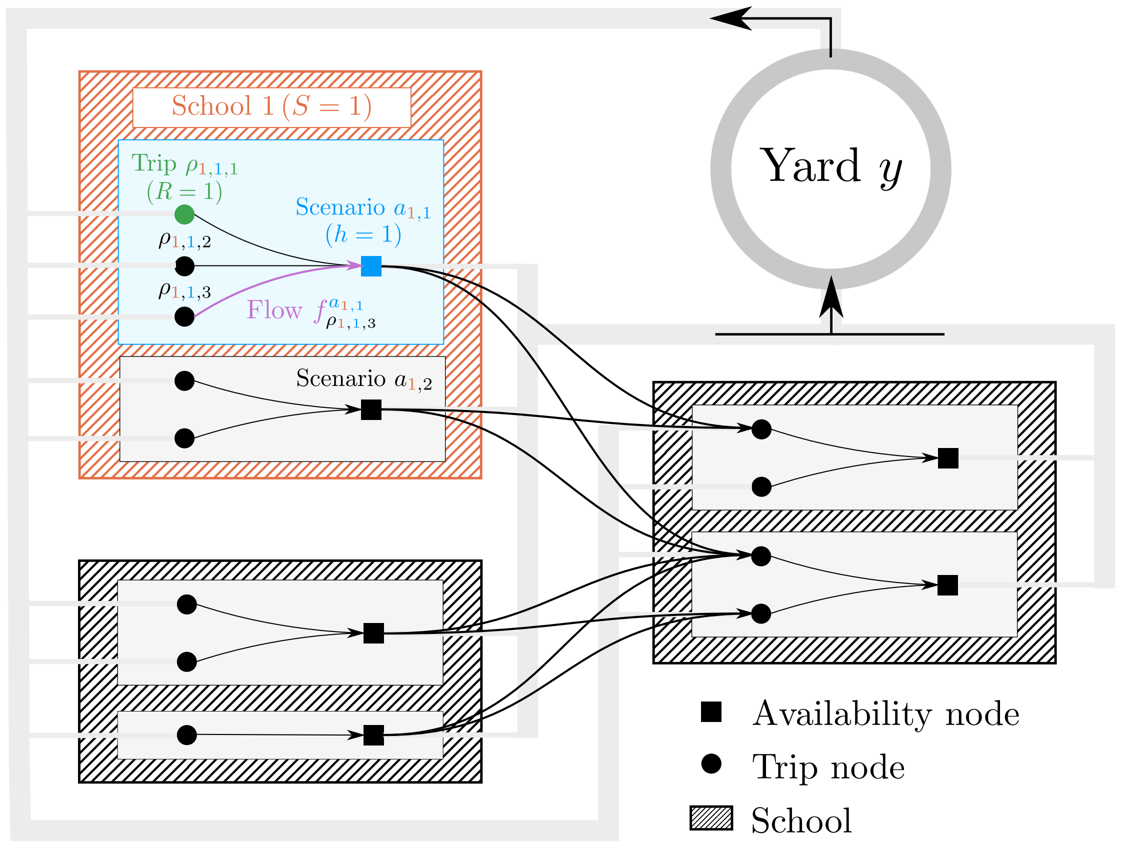


Figure 4-7: Diagram of the scenario selection graph for a small example of a school bus routing problem with three schools. Each school is represented by a diagonally striped rectangle, with two associated routing scenarios, represented by lightly shaded rectangles within the larger rectangle of the school. Note that for the two schools on the left, the two scenarios do not have the same overall number of trips (e.g. 3 trips vs. 2 trips for School 1), while this is not the case for the school on the right (both scenarios consist of two trips). Black arrows represent edges from trip nodes to availability nodes (within each school), and from availability nodes to trip nodes (modeling bus reuse between schools). Light gray arrows represent edges between trip/availability nodes and the yard/depot node.

be expressed as $\tau_S + t_S^{\text{drop-off}} + t_{S,c^{\text{start}}_{R'},\tau_S}^{\text{drive}} + T_R \le \tau_{S'}$. For a node $i \in \mathcal{N}$, let $\mathcal{I}(i) \subseteq \mathcal{N}$ be the in-neighborhood of node i, and $\mathcal{O}(i) \subseteq \mathcal{N}$ designate the out-neighborhood of i. We display a diagram of the scenario selection graph for a small example with three schools in Fig. 4-7.

Given the graph described above, we consider that a unit of flow traversing a series of trip nodes corresponds to a bus serving the corresponding trips in order. Therefore, minimizing the total number of buses corresponds to minimizing the total flow out of the yard node y subject to the constraints that (a) flows along all arcs must be integral and (b) given that a particular single-school routing solution $\mathcal{T}_S^h$ is selected for school S, every trip $R \in \mathcal{T}_S^h$ must be served by exactly one bus (i.e. each node $\rho_{S,h,R}$ is traversed by exactly one unit of flow). We therefore formulate the following network flow problem with integer flow variables f_i^j for each arc (i,j).

$$\begin{aligned}
\min \quad & \sum_{S\in\mathcal{S}} \sum_{h=1}^{h_S} \sum_{R\in\mathcal{T}_S^h} f_y^{\rho_{S,h,R}} && && (4.3a)\\
\text{s.t.} \quad & f_{\rho_{S,h,R}}^{a_{S,h}} = z_{S,h} && S\in\mathcal{S}, 1\le h\in h_S, R\in\mathcal{T}_S^h && (4.3b)\\
& \sum_{j\in\mathcal{I}(i)} f_j^i = \sum_{j\in\mathcal{O}(i)} f_i^j && \forall i\in\mathcal{N} && (4.3c)\\
& \sum_{h=1}^{h_S} z_{S,h} = 1 && \forall S\in\mathcal{S} && (4.3d)\\
& z_{S,h}\in\{0,1\} && \forall S\in\mathcal{S}, 1\le h\le h_S && (4.3e)\\
& f_i^j \in \mathbb{Z}_+ && \forall (i,j)\in\mathcal{A}. && (4.3f)
\end{aligned}$$

The binary variable $z_{S,h}$ is 1 if $\mathcal{T}_S^h$ is the selected set of trips for school S, and constraint (4.3d) ensures that exactly one set of trips is selected for each school. Constraint (4.3c) ensures conservation of flow f_i^j at each node i, with the following interpretation at each node. At the depot node y, it means that buses leaving the depot must eventually come back. At a trip node $\rho_{S,h,R}$, it means that a bus serving trip R must then become available at school S at time $\tau_S + t_S^{\text{drop-off}}$. At an availability node $a_{S,h}$, it means that a bus that is available at a school after serving a trip must either return to the yard or serve another trip. Constraint (4.3b) guarantees that a particular set of trips is selected if and only if each trip is assigned to exactly one bus.

The formulation above has a large number of integer variables, on the order of 2 million for a problem with 200 schools and 5 scenarios per school. However, commercial solvers such as Gurobi can solve it to optimality in less than two hours. Intuitively, the network flow formulation is quite strong, allowing the relaxation-based techniques and other heuristics implemented in modern MIO solvers to tackle it successfully.

The formulation above can easily be modified if there is more than one type of bus available ($|\mathcal{B}| > 1$). All we need to do is create a new graph such that each trip node $\rho_{S,h,R}$ maps to the set of nodes $\{\rho_{S,h,R,b}\}_{b\in\mathcal{B}_R}$ in the new graph (one for each bus type), and similarly map the yard node y and the availability nodes for each school $a_{S,h}$ to sets of nodes $\{y_b\}_{b\in\mathcal{B}}$ and $\{a_{S,h,b}\}_{b\in\mathcal{B}}$. For every arc (i,j) in the original arc set $\mathcal{A}$, we create a new arc (i_b, j_b) in the new graph, and modify constraint (4.3b) such that only one of the trip nodes $\{\rho_{S,h,R,b}\}_{b\in\mathcal{B}_R}$ can be traversed, effectively selecting the bus type that will serve trip R.

4.4.4 Bus Scheduling

The last step of our routing methodology involves determining final bus schedules given exactly one routing scenario for each school. We use a similar approach to the one described above for scenario selection. We define a "bus selection graph" $(\bar{\mathcal{N}}, \bar{\mathcal{A}})$, where the set of nodes $\bar{\mathcal{N}}$ consists of (i) a node $y_{b,\ell}$ for each bus type $b \in \mathcal{B}$ and each physical bus depot location $\ell \in \mathcal{L}$, (ii) a trip node $\rho^{\text{AM}}_{S,R,b,\ell}$ (respectively $\rho^{\text{PM}}_{S,R,b,\ell}$) for each school S, each morning trip R in the selected scenario $\mathcal{T}^{\text{AM}}_S$ (respectively each afternoon trip $R \in \mathcal{T}^{\text{PM}}_S$), each bus type $b \in \mathcal{B}_R$, and each depot location $\ell \in \mathcal{L}$.

The set of arcs $\bar{\mathcal{A}}$ includes (i) an arc to and from the depot node $y_{b,\ell}$ for each trip node $\rho^{\text{AM}}_{S,R,b,\ell}$ and $\rho^{\text{PM}}_{S,R,b,\ell}$, (ii) an arc from each trip node $\rho^{\text{AM}}_{S,R,b,l}$ (resp. $\rho^{\text{PM}}_{S,R,b,l}$) to each trip node $\rho^{\text{AM}}_{S',R',b,l}$ (resp. $\rho^{\text{PM}}_{S',R',b,l}$) where trip R' for school S' is time-compatible with a bus starting from school S. For a node $i \in \bar{\mathcal{N}}$, let $\bar{\mathcal{I}}(i) \subseteq \bar{\mathcal{N}}$ designate the in-neighborhood of node i, and $\bar{\mathcal{O}}(i) \subseteq \bar{\mathcal{N}}$ designate the out-neighborhood of i. For depot nodes $y_{b,\ell}$, define $\bar{\mathcal{O}}^{\text{AM}}(y_{b,\ell}) = \bar{\mathcal{O}}(y_{b,\ell}) \cap \{\rho^{\text{AM}}_{S,R,b,\ell} \in \bar{\mathcal{N}} : S \in \mathcal{S}, R \in \mathcal{T}^{\text{AM}}_S\}$, and define $\bar{\mathcal{O}}^{\mathrm{PM}}(y_{b,\ell})$ similarly. We assign a cost $c_{i,j}$ to each arc $(i,j) \in \bar{\mathcal{A}}$ to represent the objective we are trying to optimize (fuel consumption, driving distance, etc.), and introduce auxiliary variables to minimize the number of buses. Solving the bus scheduling problem corresponds to solving the following mixed-integer optimization problem.

$$\min \quad \gamma \sum_{b\in\mathcal{B}} \sum_{\ell\in\mathcal{L}} c_b K_{b,\ell} + (1-\gamma) \sum_{(i,j)\in\bar{\mathcal{A}}} c_{i,j} f_i^j \tag{4.4a}$$

$$\sum_{j\in\bar{\mathcal{I}}(i)} f_j^i = \sum_{j\in\bar{\mathcal{O}}(i)} f_i^j \qquad \forall i \in \bar{\mathcal{N}} \tag{4.4b}$$

$$\sum_{b\in\mathcal{B}} \sum_{\ell\in\mathcal{L}} \sum_{i\in\bar{\mathcal{I}}\left(\rho^q_{S,R,b,l}\right)} f_i^{\rho^q_{S,R,b,\ell}} = 1 \qquad \forall S \in \mathcal{S}, \forall R \in \mathcal{T}^q_S, q \in \{\mathrm{AM}, \mathrm{PM}\} \tag{4.4c}$$

$$\sum_{i\in\bar{\mathcal{O}}^q(y_{b,\ell})} f^i_{y_{b,\ell}} \le K_{b,\ell} \qquad \forall b \in \mathcal{B}, \ell \in \mathcal{L}, q \in \{\mathrm{AM}, \mathrm{PM}\} \tag{4.4d}$$

$$K_{b,\ell} \in \mathbb{Z}_+ \qquad \forall b \in \mathcal{B}, \forall \ell \in \mathcal{L} \tag{4.4e}$$

$$f_i^j \in \{0,1\} \qquad \forall (i,j) \in \bar{\mathcal{A}}. \tag{4.4f}$$

The flow variables indicate f_i^j select the bus type and origin depot for the bus serving each trip, as well as the sequence of trips served by each bus. Meanwhile, the variables $K_{b,\ell}$ represent the number of buses of type b coming from bus yard location l, and they are related to the variables f_i^j by constraint (4.4d). Constraint (4.4b) enforces flow conservation, which simply means that buses must leave the yard, serve at least one trip, and then return to the yard. Constraint (4.4c) enforces that each trip is served by exactly one bus, from one particular depot location and of one particular type. In the objective, $c_b > 0$ represents the cost of a bus of type b, $c_{i,j}$ designates the cost of a particular edge in the graph (driving distance, fuel consumption, etc.), and γ controls the relative importance of the two parts of the objective (number of buses and driving distance/fuel consumption). Note that the combination of constraint (4.4d) and the positive cost associated with $K_{b,\ell}$ in the objective ensure that $K_{b,\ell}$ is the maximum of the number of morning buses and the number of afternoon buses (of type b, from location ℓ).

While the previous subproblem of scenario selection has not been priorly studied to our knowledge, the bus scheduling subproblem has received significant attention, from Fügenschuh [61], Bögl et al. [23], Spada et al. [128], and others.

Readers will note that we only solve the morning and afternoon problems jointly in the last step (bus scheduling) of the BiRD algorithm. In principle, there is no reason not to solve morning and afternoon together at the scenario selection step as well. However, we found that treating morning and afternoon separately at the scenario selection step did not significantly affect the final solution while meaningfully improving tractability.

4.5 Technical Details: Routing Experiments

We now describe the setting for our computational experiments, and present a more thorough overview of the results from the first sections. We evaluate BiRD using our own synthetic examples, as well as sample problems from the literature. We first illustrate the way we generate synthetic experiments and the insights that they can provide. We then detail the process used to compare BiRD to existing approaches from the literature, using synthetic problems from Park, Tae and Kim [110].

4.5.1 Synthetic Experiments and Results

In order to build intuition about the BiRD algorithmic framework, we first study its performance on our own synthetically generated examples. We consider a school district as a square (30km by 30km) in the 2D plane, in which we sample $|\mathcal{S}|$ school locations at random. We fix the total number of students to $|\mathcal{P}|$, among which we

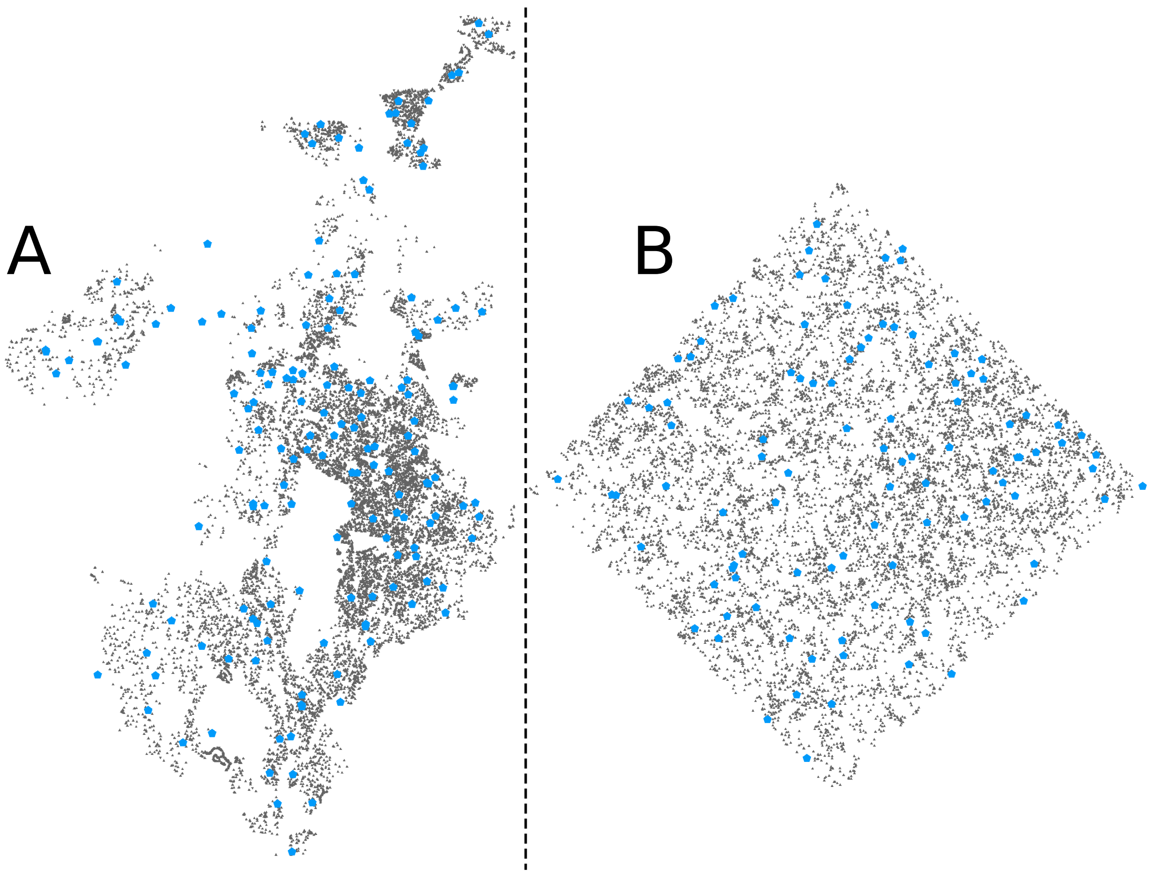
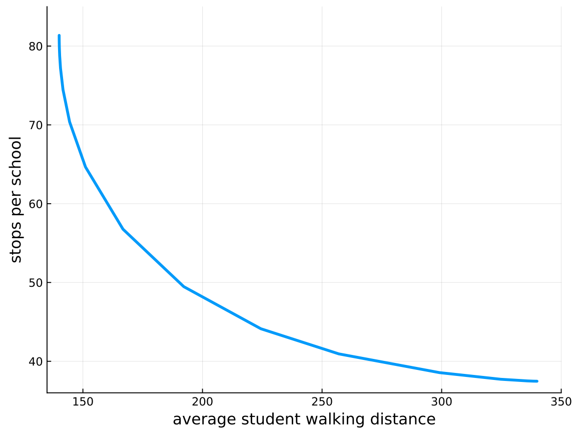
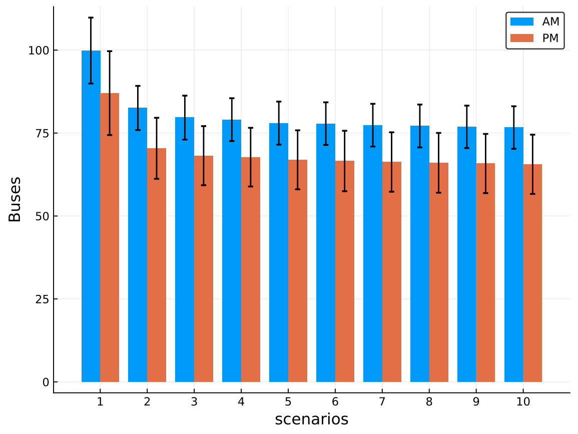


Figure 4-8: Geographic visualizations of two school districts. Small gray triangles represent students, while larger blue pentagons represent schools. (A) Boston Public Schools, 2017-18 school year (anonymized). (B) Synthetic school district with 100 schools, generated as described in the section 4.5.1 on Synthetic Experiments.

sample $|\mathcal{S}|-1$ students $p_{i_1}, \ldots, p_{i_{|\mathcal{S}|-1}}$ uniformly at random without replacement, and enforce WLOG that $0 = i_0 < i_1 < \ldots < i_{|\mathcal{S}|-1} < i_{|\mathcal{S}|} = |\mathcal{P}|$. Then we assign all students p_i such that $i_{k-1} < i \leq i_k$ to school k. For any point x and any positive real number ρ, let $\mathcal{U}_\rho(x)$ designate the uniform distribution over the disk of radius ρ centered at x.

For each school with location x_S, we select a radius $R_S \sim \mathcal{U}(\underline{R}_S, \overline{R}_S)$. The first student for that school is assigned a location sampled from $\mathcal{U}_R(x_S)$, and every subsequent student location is sampled with probability q from $\mathcal{U}_r(x_p)$, where x_p designates the location of the previous student, and with probability $1-q$ from $\mathcal{U}_R(x_S)$. This procedure creates small clusters of students which can be thought of as small "neighborhoods". Each school is randomly assigned a start time of 7:30, 8:30, or 9:30, and





(a) Tradeoff between average number of stops per school and average student-to-stop walking distance. Experiment on synthetic school district with 100 schools and 10,000 students. As the average walking distance increases, the number of stops decreases. The convexity of the tradeoff curve suggests diminishing returns in increasing the average walking distance.

(b) Effect of the number of scenarios on the total number of buses. As we increase the number of varied scenarios for each school, we create more room for optimization: the algorithm can select shorter routes for some schools, and longer routes for other schools, in a way that maximizes bus re-use. Overall, using several scenarios can yield as much as a 25% improvement in the number of buses. Using just two scenarios for school already improves the objective by 20%. Results are averaged over 100 random synthetic districts, with error bars corresponding to the standard deviation of the number of buses.

Figure 4-9: Analysis of performance of algorithm components on synthetic data.

a day length of 7, 8, or 9 hours. Finally, for each student with location x_p, we create a bus stop with location sampled from $\mathcal{U}_{r_s}(x_p)$. A large example of a synthetically generated district is shown in Fig. 4-8.

We begin with the stop assignment problem, exploring the tradeoff between the average student walking distance and the number of stops per school. We generate 100 synthetic school district instances with 100 schools and 10,000 students, and vary the stop assignment tradeoff parameter β. The results can be seen in Fig. 4-9a, showing that it is possible to reduce the average student walking distance by more than 25% without adding more than one or two stops per school.

Next, we generate 100 synthetic instances with 50 schools and 5,000 students to study the effect of the number of scenarios. For each school, we generate ten scenarios with different values of the single-school tradeoff parameter λ. For each scenario, we combine randomized routes such that each stop is covered by 400 routes, in 20 phases of 20 routes each. Then we solve the scenario-selection problem using ρ scenarios, for $\rho = 1, \ldots, 10$. We use cross-validation to select the specific subset of size ρ among the considered values of λ. The results, shown in Fig. 4-9b support the intuitive statement that "what is optimal for one school may not be optimal for the entire system", as choosing two routing options for each school yields a 20% improvement over choosing just one. In addition, the number of scenarios quickly yields diminishing returns, which is useful because it enables us to solve the scenario selection problem with two or three scenarios per school (increasing tractability) and obtain a solution that is almost as good as one computed with many more scenarios per school.

4.5.2 Comparison with Existing Methods

We have shown empirically that our approach of computing several single-school scenarios and jointly selecting the best option for each school gives very good results. In this section, we compare the performance of the BiRD algorithm to other methods from the literature.

We first compare the performance of the algorithm on benchmark synthetic data sets from Park, Tae and Kim [110]. As we mentioned before, the school bus routing problem has a large number of variants, including mixed loads and drop-off time windows. The benchmark data sets have a time window (between 10 and 30 minutes depending on the school) associated with each school corresponding to possible drop-off times, and buses serving the same school can thus arrive at different times. In contrast, our implementation of BiRD assumes that all buses for a given school must arrive at the same time. We therefore modify the benchmark data-sets such that all school start times are fixed at the beginning of the provided time window.

Having computed a single start time for each school, we are now ready to solve the school bus routing problem. We consider two categories of benchmarks, RSRB and CSCB, which are generated slightly differently (see the original paper for details). Each one assumes that stop assignment has been performed as a preprocessing step, and can be solved assuming the maximum ride time $T^{\max}$ is 45 minutes (2700 s) or 90 minutes (5400 s). The RSRB benchmarks have 8 different problems of varying size, while the CSCB benchmarks have 16 different problems of varying size (of which, following Chen et al. [35], we only consider the first 8).

For each instance, we compare the following solution approaches (all code is provided on Github [48]):

1. Our own implementation of the location-based heuristic of Braca et al. [25] (with the constraint that all buses for a given school arrive at the same time), with a number of buses denoted as Z_{LBH}.

2. The BiRD algorithm with eight scenarios for each school (using eight different values of λ: 10^2, $5 \cdot 10^2$, $5 \cdot 10^3$, 10^4, $5 \cdot 10^4$, 10^5, $5 \cdot 10^5$, 10^6 – note: for one instance, we compute fourteen scenarios for each school). We then perform the scenario selection and bus scheduling steps with the overall number of buses as the only objective. We denote this number of buses as Z_{BiRD}.

3. The BiRD algorithm with only one scenario for each school (using $\lambda = \infty$, i.e. effectively minimizing the number of buses for each school). We write this number of buses Z_{∞}.

						Number of buses				Improvement	
Instance	MRT (s)	N_{sch}	N_{stops}	N_{stud}	Z_{LBH}	Z_{∞}	Z_{Chen}	Z_{BiRD}	Z_{Hyb}	BiRD	Hybrid
RSRB01	2700	6	250	3409	38	31	31	31	31	0%	0%
RSRB02	2700	12	250	3670	34	31	30	30	29	0%	3%
RSRB03	2700	12	500	6794	64	56	56	55	55	2%	2%
RSRB04	2700	25	500	6805	77	63	62	63	62	-2%	0%
RSRB05	2700	25	1000	13,765	123	101	105	100	100	5%	5%
RSRB06	2700	50	1000	12,201	120	104	106	104	103	2%	3%
RSRB07	2700	50	2000	26,912	205	173	173	164	161	5%	7%
RSRB08	2700	100	2000	31,939	220	181	188	175	173	7%	8%
CSCB01	2700	6	250	3907	39	33	34	33	33	3%	3%
CSCB02	2700	12	250	3204	43	37	37	37	37	0%	0%
CSCB03	2700	12	500	6813	73	67	66	65	64	2%	3%
CSCB04	2700	25	500	7541	81	70	73	69	69	5%	5%
CSCB05	2700	25	1000	16,996	158	146	146	143	143	2%	2%
CSCB06	2700	50	1000	18,232	162	141	145	141	140	3%	3%
CSCB07	2700	50	2000	27,594	240	218	223	206	206	8%	8%
CSCB08	2700	100	2000	27,945	234	192	195	188	186	4%	5%
Average	2700				119.4	102.8	104.4	100.2	99.4	3%	4%

Table 4.1: Comparison of BiRD with existing methods on first set of synthetic benchmarks.

					Number of buses					Improvement	
Instance	MRT (s)	N_{sch}	N_{stops}	N_{stud}	Z_{LBH}	Z_{∞}	Z_{Chen}	Z_{BiRD}	Z_{Hyb}	BiRD	Hybrid
RSRB01	5400	6	250	3409	33	31	31	31	31	0%	0%
RSRB02	5400	12	250	3670	33	27	27	26	26	4%	4%
RSRB03	5400	12	500	6794	52	52	50	50	50	0%	0%
RSRB04	5400	25	500	6805	55	52	50	50	49	0%	2%
RSRB05	5400	25	1000	13,765	95	93	93	91	91	2%	2%
RSRB06	5400	50	1000	12,201	93	82	86	76	76	12%	12%
RSRB07	5400	50	2000	26,912	162	162	166	152	151	6%	7%
RSRB08	5400	100	2000	31,939	186	167	174	154	152	11%	13%
CSCB01	5400	6	250	3907	31	31	32	30	30	3%	3%
CSCB02	5400	12	250	3204	29	29	29	28	28	3%	3%
CSCB03	5400	12	500	6813	61	56	52	51	51	2%	2%
CSCB04	5400	25	500	7541	57	52	51	48	48	6%	6%
CSCB05	5400	25	1000	16,996	131	127	128	121	121	5%	5%
CSCB06	5400	50	1000	18,232	127	122	124	116	114	6%	8%
CSCB07	5400	50	2000	27,594	181	178	170	163	162	4%	5%
CSCB08	5400	100	2000	27,945	174	155	149	140	136	6%	9%
Average	5400				93.8	88.5	88.3	82.9	82.3	4%	5%

Table 4.2: Comparison of BiRD with existing methods on second set of synthetic data benchmarks.

4. Our own partial re-implementation of the method from Chen et al. [35]. More specifically, we use the single-school routes computed by Chen et al. and published along with their paper. We then solve the bus scheduling problem with the previously-discussed constraint that all buses for a particular school arrive at the school at the same time, to ensure consistency with the other methods. We denote this number of buses as Z_{Chen}.

5. A hybrid BiRD method where we add the routes from Chen et al. [35] to the eight scenarios for each school computed in approach number 2 above, and perform the scenario selection and bus scheduling steps with the resulting nine solutions for each school. We denote the number of buses obtained by this method as Z_{Hyb}. Note that the solutions computed by methods 2 and 4 are both feasible in this setting, meaning that Z_{Hyb} is guaranteed to be no worse than Z_{Chen} and Z_{BiRD}.

Results are presented in Tables 4.1 and 4.2, and can be replicated using our Julia package released on Github [48], which includes code for all the methods described above. We notice that BiRD with several scenarios matches or improves upon the best of LBH and Chen et al.'s combination of simulated annealing and CPLEX in all but one instance, with an average improvement of about 4%.

The results also suggest that BiRD's modularity allows it to benefit from the advantages of other methods. When the routes computed by Chen et al.'s method are considered in addition to other scenarios, our method outperforms Chen et al.'s by 5% on average. The idea of using multiple scenarios allows BiRD to leverage the strengths of other methods. Indeed, it seems that most of BiRD's improvement comes from the central idea of multiple scenarios for each school. Indeed, BiRD with a single scenario per school gives comparable results to Chen et al.'s method. This is to be expected: both methods aim to minimize the number of buses for each school before solving the bus scheduling subproblem using mixed-integer optimization. The only difference is that our single-school solutions are computed with the mixed-integer optimization-based heuristic described earlier, while Chen et al.'s are computed using local search and simulated annealing.

The benchmark data sets from Park, Tae and Kim [110] are useful because they have been used by several authors to compare their methods. However, there are only a few such benchmarks, and they do not necessarily reflect a wide range of potential school districts (for example, schools can start as early as 5AM and as late as 10AM, which corresponds to a much larger spread than in most US school districts including Boston). More generally, the school bus routing literature suffers from a lack of benchmark instances that can be used to compare different solution approaches. Therefore, another contribution of this work is the publication of open-source code that can generate and visualize synthetic examples as described in the previous section, to be used in future work on the school bus routing problem.

We can use these new synthetic examples to further study the performance of our method. In particular, we are interested in studying cases where the number of schools and the number of stops per school are both large. We consider 20 synthetically generated instances with varying parameters, described in Table 4.3. We compare solution approaches 1, 2 and 3 above.

The results, shown in Table 4.3, suggest that the BiRD algorithm with multiple scenarios per school significantly outperforms competing methods on large-scale instances, with a number of buses that is 12% lower, on average, than the next best solution. We notice also that in these instances, minimizing the number of buses for each school performs quite poorly (worse than LBH), because it makes bus re-use extremely difficult. Code to reproduce the results from Table 4.3 is also provided in our Github repository [48].

Since BiRD is a heuristic, it provides no guarantees as to the optimality of the solution. Furthermore, the question of finding lower bounds for the school bus routing problem has received very little attention and remains very much open. To our knowledge, only Park, Tae and Kim [110] have made a serious attempt at finding a lower bound, and they themselves concede that much improvement is still needed. Because solving the school bus routing algorithm exactly is intractable for large instances, and good lower bounds do not really exist, quantifying the effect of the problem

				Number of buses			
Instance	N_{sch}	N_{stops}	N_{stud}	Z_{LBH}	Z_{∞}	Z_{BiRD}	Improvement
1	50	3544	5000	117	128	104	11%
2	100	7785	10,000	229	246	195	15%
3	150	12,248	15,000	327	365	283	14%
4	200	16,875	20,000	438	465	385	12%
5	50	3683	5000	111	129	106	5%
6	100	7969	10,000	228	251	196	14%
7	150	12,492	15,000	330	366	281	15%
8	200	17,155	20,000	436	470	388	11%
9	50	3808	5000	113	130	105	7%
10	100	8,156	10,000	234	249	199	15%
11	150	12,729	15,000	334	363	286	14%
12	200	17,430	20,000	442	478	384	13%
13	50	3924	5000	115	129	107	7%
14	100	8336	10,000	236	251	202	14%
15	150	12,967	15,000	331	361	284	14%
16	200	17,699	20,000	447	479	383	14%
17	50	4042	5000	118	133	107	9%
18	100	8503	10,000	236	248	202	14%
19	150	13,184	15,000	335	366	286	15%
20	200	17,942	20,000	443	470	391	12%
Average				280.0	303.9	243.7	12%

Table 4.3: Comparison of BiRD with other methods on large-scale synthetic data benchmarks.

decomposition on the optimality of the final solution remains an open question.

4.6 Technical Details: Bell Time Selection

In this section, we present the details of our mathematical formulation for the School Time Selection Problem (STSP) and describe our synthetic experiments.

4.6.1 Transportation Costs

Given the complexity of the school bus routing problem when school start times are fixed, jointly optimizing bus routes and bell times is clearly a very intractable problem, which grows exponentially in size with the number of schools. The key idea of our approach is thus to find a reasonable proxy for the transportation cost of any start time assignment. We choose to define pairwise routing costs $c^{\text{routing}}_{S,t,S',t'}$ in a bid to balance tractability with expressivity (pairwise costs allow us to capture interaction between pairs of schools).

The main intuition behind the routing costs $c^{\text{routing}}_{S,t,S',t'}$ is the fact that costs are lower if individual buses can serve as many trips as possible. Therefore, the main factor in reducing the number of buses is the "compatibility" of groups of trips, i.e. how easy it is for a single bus to serve a certain set of trips without wasting time waiting or driving without passengers. Given two trips R and R', let Δt be the time between the end of R and the beginning of R'. We define a piecewise linear compatibility cost $c_{R,R'}$ that is low if it is profitable for a bus to serve the two trips sequentially. More precisely:

- $c_{R,R'} = 0$ if it is impossible for a bus to serve the two trips successively
- $c_{R,R'} = 0$ if $\Delta t \geq \bar{T}$ with $\bar{T}$ a compatibility parameter that defines the maximal time a bus can drive between two trips for them to be "compatible".
- $c_{R,R'} = -\frac{\bar{T}-\Delta t}{\bar{T}}$ otherwise, i.e. the cost is -1 when $\Delta t = 0$ and R' can be served immediately after R and then increases linearly to 0 as Δt increases to $\bar{T}$.

For each school and each year for which we enrollment data is available, we compute a set of varied bus routing scenarios as described earlier. The scenarios are selected so that they are likely to be used in the optimal school bus routing solution. For each school, we therefore obtain a list of scenarios that is the union of all the scenarios obtained from each year of data. Then, for two schools S and S', we can define a compatibility cost $c^{compat}_{S,t,S',t'}$ that is the sum of the compatibility costs $c_{R,R'}$ and $c_{R',R}$ for every trip R in every routing scenario for school S and every trip R' in every routing scenario for school S' when the schools bell times are respectively t and t'.

Our experiments show that the costs c^{compat} are good approximation of how the choice of bell time allows the routes of different schools to be "compatible" across the years. Choosing bell times that maximize this compatibility indirectly minimizes the future transportation costs incurred by the district. It turns out that maximizing the compatibility of different routes as described has the unwanted tendency to lead to a reduced number of schools with early and late start times, which has a negative impact on the number of buses in the solution. This is a consequence of using a simple pairwise affinity cost that only takes into account groups of two schools. In practice, we can counteract this adversarial effect by adding a cost that encourages bell times to be spread out over all allowed values: $c^{spread}_{S,t,S',t'} = -|t-t'|$. The final transportation costs are therefore defined as $c^{\text{routing}} = c^{compat}(\bar{T}) + \gamma c^{spread}$ where $\bar{T}$ and γ are the two parameters that depend on fundamental characteristics of the school district, and can be found using cross-validation. Ultimately, the transportation costs do not need to be perfect: year-to-year enrollment changes mean that directional information is more than enough in practice.

Given the routing costs defined above, and temporarily ignoring all other objectives, the STSP can be formulated as a Generalized Quadratic Assignment Problem

(GQAP), for example using integer optimization:

$$\begin{aligned}
\min \quad & \sum_{S\in\mathcal{S}} \sum_{t\in\mathcal{T}_S} \sum_{S'\in\mathcal{S}} \sum_{t'\in\mathcal{T}_{S'}} c^{\text{routing}}_{S,t,S',t'} z_{S,t,S',t'} && && \text{(4.5a)}\\
\text{s.t.} \quad & \sum_{t\in\mathcal{T}_S} a_{S,t} = 1 && \forall S\in\mathcal{S} && \text{(4.5b)}\\
& z_{S,t,S',t'} \geq a_{S,t} + a_{S',t'} - 1 && \forall S\in\mathcal{S}, t\in\mathcal{T}_S, S'\in\mathcal{S}, t'\in\mathcal{T}_{S'} && \text{(4.5c)}\\
& z_{S,t,S',t'} \leq a_{S,t} && \forall S\in\mathcal{S}, t\in\mathcal{T}_S, S'\in\mathcal{S}, t'\in\mathcal{T}_{S'} && \text{(4.5d)}\\
& z_{S,t,S',t'} \leq a_{S',t'} && \forall S\in\mathcal{S}, t\in\mathcal{T}_S, S'\in\mathcal{S}, t'\in\mathcal{T}_{S'} && \text{(4.5e)}\\
& z_{S,t,S',t'} \in \{0,1\} && \forall S\in\mathcal{S}, t\in\mathcal{T}_S, S'\in\mathcal{S}, t'\in\mathcal{T}_{S'} && \text{(4.5f)}\\
& a_{S,t} \in \{0,1\} && \forall S\in\mathcal{S}, t\in\mathcal{T}_S. && \text{(4.5g)}\\
& && && \text{(4.5h)}
\end{aligned}$$

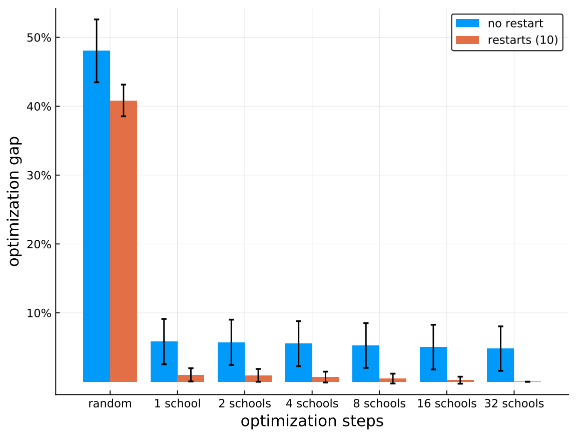
In the formulation above, the key decision variable $a_{S,t}$ is 1 when school S is assigned time t, and 0 otherwise. Similarly, the decision variable $z_{S,t,S',t'}$ is 1 when schools S and S' are respectively assigned times t and t', and 0 otherwise. The set $\mathcal{T}_S$ designates all bell times that are allowed for school S (this is a discrete, finite set, e.g. every 10 minutes between 7:30AM and 9:30AM). Constraint (4.5b) enforces that each school is assigned exactly one time, while constraints (4.5c), (4.5d) and (4.5e) enforce the relationship between the single and pairwise decision variables $a_{S,t}$ and $z_{S,t,S',t'}$. A similar formulation is proposed in [138].

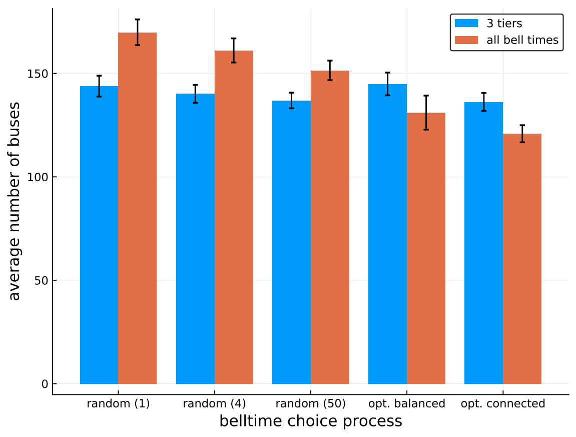
4.6.2 Bell Time Optimization on Synthetic Data

We have formulated the STSP as a GQAP, and we solve it using a simple local improvement heuristic, which randomly selects a smaller subset of schools and optimizes their start times, keeping the bell times of all other schools fixed. Given an initial bell time assignment $\{t_S^0\}_{S\in\mathcal{S}}$, and having selected a subset $\mathcal{S}_1 \subseteq \mathcal{S}$, it turns out that the problem of finding the optimal start times for this subset is still a GQAP. We can formulate this GQAP as an integer program as above, solve it using a commercial solver such as Gurobi, and iterate until a stopping criterion is met.

We now turn to synthetic data to examine the effect of the parameters of the local improvement heuristic, namely the size of the subset and the number of iterations. The first tradeoff we explore is the size of the optimized subset. Clearly, as the number of schools in the optimized subset increases, the local improvement heuristic will find better solutions, but each iteration will take more time. To understand this tradeoff, we run 100 randomly-generated experiments with 100 schools. For each one, we first select a random starting point (leftmost column), then we perform 1024 iterations one school at a time, followed by 512 iterations two schools at a time, etc. Each time, we double the number of schools in the optimized subset and halve the number of iterations, so that in expectation each school features in the optimized subset the same number of times. We show results in Fig. 4-10a. We notice that compared to a random solution, optimizing one school at a time already drastically improves the quality of the solution, and subsequently increasing the size of the optimized subset does not have a strong effect on the quality of the solution. In addition, using random restarts, i.e. running the experiment several times with different random starting points and keeping the best one, has a very significant effect on the optimization gap. In fact, it seems that using several random restarts has a much stronger effect on the solution quality than increasing the number of schools that are optimized at each iteration.

To model transportation costs, we introduced costs $c^{\text{routing}} = c^{compat}(\bar{T}) + \gamma c^{spread}$. We claim that the optimal bell time assignment given these costs indeed induces a routing solution with a small number of buses. We support this claim with the experiments described in the Fig. 4-3. We present another set of experiments here which also support the same point. We consider a problem instance with 100 schools, where the only objective is to minimize transportation costs, and the allowed bell times either follow 3 "tiers" (7:30, 8:30, 9:30) or encompass all 15-minute intervals between 7:15 and 9:30. We compare three optimization strategies. The first ("random") assigns each school a bell time uniformly at random across the universe of possibilities (possibly with some random restarts to improve solution quality). The second ("balanced") simply tries to balance the number of routes into evenly spaced tiers, with a parameter controlling the spacing between the tiers which we can choose





(a) Effect of subset size on quality of solution found by optimization-based heuristic, averaged over 100 synthetic experiments. As the subset size increases, the optimization gap decreases. In addition, random restarts have a stronger effect on the solution quality than increasing the size of the optimized subset.

(b) Comparison of different routing cost approximations for bell time optimization. Two settings are considered: three bell time tiers, i.e., schools may only start at 7:30, 8:30 or 9:30, and all bell times, when every 5-minute interval between 7:15 and 9:30 is allowed. The random assignment strategy performs well when there are only three allowed bell times, but poorly when all bell times between 7:15 and 9:30 are allowed. Approximating routing costs using the routing compatibility costs described in the section 4.6.1 on Transportation Costs (“connected” experiment) gives better results in all cases than simply making sure the number of routes is balanced across tiers without regard for the actual compatibility of these routes.

Figure 4-10: Results of bell time optimization algorithm on synthetic data.

by cross-validation. The third (“connected”) is the one described in the Section 4.6.1 “Transportation Costs”.

The results averaged over 100 random instances, which can be seen in Fig. 4-10b, show that our optimization strategy consistently and significantly outperforms the other two in both experimental settings. We notice that the random assignment strategy works quite well in the case when only three bell times are allowed, even better than the strategy that tries to balance routes into evenly spaced tiers. However, the random strategy is not able to make use of the additional allowed bell times in the second set of experiments, while both optimization strategies achieve significant improvements when the number of allowed bell times for each school increases.

4.6.3 GQAP-Representable Objectives

As discussed in the “Bell Times in Practice” section 4.3, the start time assignment problem typically involves many objectives beyond the simple optimization of transportation costs. We present many such objectives here, and show how they can be integrated within the GQAP framework.

Many real-world objectives can be represented using single affinity costs $c_{S,t}$. Here are a few of the possibilities we explored in collaboration with BPS:

- Limiting change from the current bell times can be achieved by setting $c_{S,t}$ to a positive value when t is different from the current time t_{current} for school S. The specific value can exhibit any functional dependence on t and t_{current}.

- Incorporating individual school preferences. Districts can easily quantify the preferences of various stakeholders at a particular school, from teachers and staff to parents and students, using surveys, focus groups, etc. These preferences can then easily be converted into aversion costs $c_{S,t}$.

- Favoring a particular school. Sometimes, a district may wish to prioritize the needs of a particular set of schools $\mathcal{S}_0 \subseteq \mathcal{S}$, in order to bolster academic achievements, support economically disadvantaged students, or provide a more auspicious environment for students with special needs. This objective can be achieved by penalizing less desirable times more for $S \in \mathcal{S}_0$ than for $S \notin \mathcal{S}_0$. For example, in Boston, we optimized a preference score for schools with a high number of special education students (weighted by the number of these students).

- Promoting later high school start times/earlier elementary school end times can be achieved by penalizing undesirable times for each school with a high cost.

- Interfacing with after-school programs/school-specific constraints. If a school must end before a certain time to leave time for a specific extracurricular activity, or to alleviate traffic congestion in the city, it is straightforward to compute the cost of such undesirable times and consider this as an objective.

Also allowing pairwise affinity costs $c_{S,t,S',t'}$ increases modeling power by allowing the representation of more complicated real-world objectives:

- Allowing school partnerships. Groups of schools often partner to offer joint extracurriculars or athletic competitions. Schools can also share all or part of their transportation. Partner schools may therefore require compatible bell times, i.e. bell times that are within a particular interval.

- Considering externalities. School districts may wish to separate the start and/or end time of some pairs of schools to prevent fights between rival students or reduce the strain on public transportation.

- Ensuring equity. Single costs allow a school district to model average school satisfaction, while pairwise affinity costs also allow it to model the variance in satisfaction across neighborhoods, communities, or the entire district. Consider two sets of schools $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{S}$, and let $\mu_S^{(i)}$ be $1/|\mathcal{S}_i|$ if $S \in \mathcal{S}_i$ and 0 otherwise. Consider a metric which assigns cost $\hat{c}_{S,t}$ to bell time t for school S. Then the squared difference $\delta_{\mathcal{S}_1,\mathcal{S}_2}(\mathbf{t})$ of the mean of $\hat{c}$ between the two considered subsets can be written as

$$\delta_{\mathcal{S}_1,\mathcal{S}_2}(\mathbf{t}) = \left(\sum_{S\in\mathcal{S}} \mu_S^{(1)} \hat{c}_{S,t_S} - \sum_{S\in\mathcal{S}} \mu_S^{(2)} \hat{c}_{S,t_S}\right)^2 = \left(\sum_{S\in\mathcal{S}} \left(\mu_S^{(1)} - \mu_S^{(2)}\right) \hat{c}_{S,t_S}\right)^2 \tag{4.6a}$$

$$= \sum_{S\in\mathcal{S}} \left(\mu_S^{(1)} - \mu_S^{(2)}\right)^2 \hat{c}^2_{S,t_S} + \sum_{S\in\mathcal{S}} \sum_{S'\in\mathcal{S}, S'\neq S} \left(\mu_S^{(1)} - \mu_S^{(2)}\right) \left(\mu_{S'}^{(1)} - \mu_{S'}^{(2)}\right) \hat{c}_{S,t_S} \hat{c}_{S',t_{S'}}. \tag{4.6b}$$

The above objective is GQAP-representable, if we define single affinity costs $\bar{c}_{S,t} = \left(\mu_S^{(1)} - \mu_S^{(2)}\right)^2 \hat{c}^2_{S,t}$ and pairwise affinity costs:

$$\bar{c}_{S,t,S',t'} = \left(\mu_S^{(1)} - \mu_S^{(2)}\right) \left(\mu_{S'}^{(1)} - \mu_{S'}^{(2)}\right) \hat{c}_{S,t_S} \hat{c}_{S',t_{S'}}$$

And this property generalizes to arbitrary weights μ, allowing districts to ensure equity across all communities and populations in a district.

We note that pairwise affinity costs are more general than single affinity costs since optimizing any single affinity cost $c_{S,t}$ is equivalent to optimizing the corresponding pairwise affinity cost $c_{S,t,S',t'}$, which equals $c_{S,t}$ when $S = S'$ and $t = t'$, and 0 otherwise. For ease of notation, we choose to represent all single costs in this manner.

Our complete approach to the STSP is thus a multi-objective formulation. Specifically, given a set of GQAP-representable objectives $\{c^\alpha\}_{\alpha=1}^{\alpha=A}$ (one of which could be the routing costs c^{routing} and corresponding priority weights η_α (both of which are determined by the school district), we replace the routing-only objective (4.5a) with the weighted sum of all of the district's objectives:

$$\min \sum_{S\in\mathcal{S}} \sum_{t\in\mathcal{T}_S} \sum_{S'\in\mathcal{S}} \sum_{t'\in\mathcal{T}_{S'}} \sum_{\alpha=1}^{\alpha=A} \eta_\alpha c^\alpha_{S,t,S',t'} z_{S,t,S',t'}. \tag{4.7}$$

Policymakers are free to vary the priority weights η_α to explore tradeoffs between competing objectives.

4.6.4 Boston Community Survey

A typical example of a real-world objective that school districts must take into account is community satisfaction. We describe the data collected by BPS to understand the preferences of parents, teachers and staff.

When BPS began exploring the idea of bell time adjustment in the fall of 2016, they launched a community survey in order to understand the preferences of various stakeholders, including parents, teachers and staff. The survey included both an online and phone component. Parents and school staff were asked to score all bell times between 7:00 and 9:30 (every 15 minutes) between 1 (worst) and 7 (best).

To reduce noise in the survey (e.g some parents rate all bell times as 1 or 2 while other parents rate all bell times as 6 or 7), we normalize these scores so that (a) they lie between 0 and 1, and (b) for any respondent, their favorite bell time is rated a 1 and their least favorite a 0. Then the preference score (or survey score) of a particular bell time assignment is the average (weighted by enrollment) of the preference scores of each school for their assigned bell time, where the preference score for a school S at a given bell time t is the average of the normalized preferences of all school S's parents and staff for bell time t, where parent preferences carry twice as much weight as staff preferences (the ratio was decided by BPS). To handle schools with too few responses, we add three "dummy parents" to all schools' responses, with preferences equal to the average of all parent preferences across the entire survey. The results in Fig. 5 and Table 1 rely on this community survey.

The main insight provided by the survey was the general disagreement of parents within each school. BPS realized that optimizing the average preference score was not very meaningful because every bell time would have both supporters and critics at every school. Therefore, they moved towards optimizing broader objectives (e.g. moving high schools later) rather than optimizing this particular preference score.

Because of the survey's low response rate in some schools, it did not provide conclusive answers to all of BPS's questions. For instance, it is not clear whether households of different income levels have different preferences for the start times of their younger children due to different work schedules.

4.7 Conclusion

Spreading start times allows school districts to reduce transportation costs by reusing buses between schools. However, assigning each school a time involves both estimating the impact on transportation costs and reconciling additional competing objectives. These challenges force many school districts to make myopic decisions, leading to an expensive and inequitable status quo. For instance, most American teenagers start school before 8:00 AM, despite evidence of significant associated health issues. We proposed an algorithm to jointly solve the school bus routing and bell time selection problems. Our application in Boston led to $5 million in yearly savings (maintaining service quality despite a 50-bus fleet reduction) and to the unanimous approval of the first school start time reform in 30 years. The tools we develop can be useful to school districts, to reduce overhead operational costs and invest directly into students, in a way that fits the priorities and needs of the community.

Chapter 5

The Benefits of the Stochastic Proximal Point Algorithm

5.1 Introduction

We consider the following convex stochastic optimization problem:

$$\min_{x \in \mathcal{X}} F(x) = \mathbb{E}_S \left[f(x, S) \right], \tag{5.1}$$

where $\mathcal{X} \subset \mathbb{R}^d$ is a closed convex set, S is a random variable, and $f(., S)$ are closed convex functions for each value S. These problems have many applications in statistical learning [72, 149], but also stochastic optimization and simulation. In the context of machine learning, $f(x, S)$ can model the loss function of a model parametrized by x with respect to a data-point S, uniformly sampled from a data-set. In this setting (5.1) is equivalent to minimizing the averaged loss over the data-set. For example, an ordinary least square problem (OLS) would correspond to

$$f(\beta, S) = \left(X_S' \beta - y_S\right)^2$$

, where X_S and y_S represent the feature and label of a uniformly sampled data-point.

We do not assume knowledge of the distribution of S. Instead, we consider the case where we have the possibility to draw independent random samples of S and use them iteratively to solve (5.1). Lately, the large amounts of data that are available for machine learning applications make this stochastic setting particularly relevant for large-scale learning [24].

In the unconstrained case ($\mathcal{X} = \mathbb{R}^d$), the de facto algorithms for solving (5.1) are the stochastic (sub)gradient methods [24, 98, 115, 100], first introduced by [119] for smooth problems. Starting from a solution x_0, each iteration $k \geq 1$ draws an independent sample $S_k \sim S$ and updates the previous solution using a sub-gradient step of size $\mu_k \geq 0$ on $f(., S_k)$:

$$x_k = x_{k-1} - \mu_k g_k \quad \text{where } g_k \in \partial f(x_{k-1}, S_k). \tag{5.2}$$

These methods have guarantees of convergence (see [100] for the last iterate and [115] for the averaged sequence) under an appropriate choice of step schedule $[\mu_k] = (\mu_1, \mu_2, \cdots) \in \mathbb{R}^{\mathbb{N}}$. It has been used successfully in large-scale stochastic optimization [24, 149]. But the choice of step schedule can be challenging [98], and strong assumptions are needed to guarantee convergence.

Interestingly, the sub-gradient step is also the minimizer of the following optimization problem:

$$\begin{aligned} x_k = \operatorname{argmin}_x \left(f(x_{k-1}, S_k) + g_k'(x - x_{k-1}) \right) + \frac{1}{2\mu_k} \|x - x_{k-1}\|^2 \\ \text{with } g_k \in \partial f(x_{k-1}, S_k). \end{aligned} \tag{5.3}$$

The term in parenthesis is a first order approximation of $f(., S)$ around x_{k-1}. If we replace the approximation with the function itself, we get the *stochastic proximal point algorithm* [121, 21, 132, 3, 112]:

$$x_k = \operatorname{argmin}_{x \in \mathcal{X}} f(x, S) + \frac{1}{2\mu_k} \|x - x_{k-1}\|^2. \tag{5.4}$$

This algorithm is derived from its deterministic counterpart, the proximal point algorithm [120], an optimization algorithm associated with an appealing theoretical framework and strong guarantees of convergence [108]. More broadly, proximal algorithms have encountered a lot of success in large-scale and distributed optimization [108], in particular with the successful use of the proximal gradient method [6, 108].

A disadvantage of the proximal operator (5.4) is its computational challenges. Indeed, computing a proximal point update requires to solve a convex optimization problem, which can be much harder than computing a gradient [108].

Nonetheless, a typical way to increase the tractability of stochastic optimization algorithms is to use distributed computing. Mini-batching can be used for this purpose [40, 47]. Instead of drawing samples of S one by one, we can instead sample n of them simultaneously: $[S_n] = (S_1, \cdots, S_n)$. We then replace $f(., S)$ in (5.2) and (5.4) by its empirical expectation:

$$f_n(x, [S_n]) = \frac{1}{n}\sum_{i=1}^{n} f(x, S_i)$$

Using mini-batching can lead to tractability gains. For example, mini-batching is standard in large-scale learning and neural-networks training. Leveraging mini-batching to make the stochastic proximal steps more efficient is a focus of this work.

5.1.1 Existing Work on Stochastic Proximal Point Methods

Although stochastic gradient methods are widely used and studied [119, 98, 115, 100], their shortcomings are also well known. THey require strong assumptions to guarantee appropriate convergence [98] (typically bounded or Lipschitz-gradients). They can also exhibit instability with respect to the choice of step schedule. Indeed, even when choosing a step schedule and problem that verify the conditions for convergence, [98] recently showed that stochastic gradient steps can experience a phase of exponential divergence before convergence. Given a step schedule $\mu_k = \frac{\mu_0}{k}$, too small choices of μ_0 (even slightly) lead to lower asymptotical rate of convergence, whereas higher values can lead to a non-asymptotical exponential divergence.

The stochastic proximal point methods have been analysed fairly recently, and present a good alternative to these shortcomings. [12] was to the best of our knowledge the first paper to have introduced the stochastic proximal point algorithm in 2011, albeit in an incremental context (finite number of samples, not necessarily drawn randomly). Since then, the focus of this literature has been on convergence and stability of the algorithm. Stochastic proximal point methods indeed share the asymptotic guarantees of their stochastic gradient counterparts without requiring bounded or Lipschitz gradients [3, 112]. They are also more stable with respect to the choice of step schedule and do not have the exponential transient behavior of stochastic gradient methods [3, 121]. Additionally, they are also particularly efficient on "noiseless" or "easy" problems, where the minimizers of F also minimize $f(., S)$ for each S, for which [3, 121] prove linear convergence.

[121] provides the first comparison study of the stability of stochastic gradient and proximal point steps, as well as asymptotical results, using ideas from monotone operator theory.[21] looks at a broader theoretical setting. [132] uses averaging, asymptotic and non-asymptotic analysis. [112] also provides an analysis of stochastic proximal point methods, in the case where the feasibility set is also defined in a stochastic setting, as the intersection of random sets $\mathcal{X}_S$, that is $\mathcal{X} = \cap_{S\in\Omega}\mathcal{X}_S$ where $(\Omega, \mathbb{P})$ is a probability space associated with S. [3] builds on the previous literature to generalize both the stochastic proximal and gradient methods in their *aProx* framework of model-based methods. Their theoretical work explores the conditions of stability and asymptotic convergence within this framework, and provides non-asymptotical bounds for the proximal steps. They also conduct an experimental study, including the case of least square problems, of the stability of various algorithms of their general class.

This literature suggests that a reason to use stochastic proximal point methods is their stability with respect to the choice of step schedule. But if the step schedules are chosen adequately ([24] discusses a practical way of choosing step sizes in large-scale learning), the advantages of stochastic proximal steps are not as obvious, at least given the existing non-asymptotical bounds. This is particularly problematic given the tractability challenges associated with the computation of the proximal operator [108], that are, to the best of our knowledge, not been addressed in the existing work on stochastic proximal point methods. And in many settings, computational time is extremely important. For example, [24] explains that given a limited computational time, faster iterations results in more samples being used, which can significantly outbalance the potential optimization limitations of fast algorithm like stochastic gradient methods.

5.1.2 Contributions

We take an orthogonal approach to the study of the stochastic proximal point algorithm. This chapter focuses on one question: should we use proximal point steps in practice? Existing positive answers [3] leverage their greater stability and easier choice of step schedule. We also show that even with the best possible step schedule for gradient steps, proximal point steps still have an edge and can be made tractable at a large-scale.

The existing literature studies asymptotic behavior and general non-asymptotic bounds. In order to get more precise results, we focus on the particular example of unconstrained quadratic cost functions to provide a much more precise comparison of gradient and proximal point steps, although limited to this application. This involves exploring the little studied influence of computational time, as well as the importance of mini-batching (or variance reduction).

In Section 5.2, we introduce the notations of our setting of stochastic convex quadratic optimization problems, as well as the associated optimization algorithms.

In Section 5.3, we focus on the one-dimensional case. This simple example guarantees the existence of an “optimal” step schedule for both proximal point and gradient steps. Such a step schedule has the interesting property that the expected error at each step k is minimal across all step schedules. Using this step schedule allows us to compare the non-asymptotic behavior of the algorithms in more details than the traditional literature, independently from the choice of step schedule. We compute the expected error of this schedule at each step using recurrence relations, allowing us to precisely compare them in an extensive theoretical study. In particular, we discuss how mini-batching (or variance reduction) both makes proximal point steps more efficient and hurts gradient step.

Section 5.4 presents an experimental study in higher dimensions, using the example of ordinary least square. Although it becomes harder to define and study an optimal step schedule, we experimentally confirm the insights obtained in dimension 1. In particular mini-batching still plays an important role.

Finally, Section 5.5 explores the little-studied tractability tradeoff of proximal point steps. In small dimensions, state-of-the-art linear algebra algorithms and parallelization help reduce the complexity disadvantage of proximal point steps. In higher dimensions, we use an approximately solve the proximal optimization problems with conjugate gradient to significantly the computational burden of proximal point steps. This approach allows us to get the best of both worlds and outperform both gradient and proximal steps. We match the asymptotic computational efficiency of gradient steps, as well as the stability, the easy choice of step schedule, and the mini-batching and parallelization benefits of proximal steps.

5.2 Setting

We introduce the notations and algorithms of our optimization setting.

5.2.1 Stochastic Convex Quadratic Optimization

We consider the unconstrained stochastic optimization problem (5.1), in the special case where $f(.,S)$ is a convex quadratic for all S. That is, we have a semi-definite random matrix $A_S \in S_+^d$, a random vector $b_w \in \mathbb{R}^d$ and a random scalar $\gamma_w \in \mathbb{R}$ such that:

$$f(x,S) = \frac{1}{2}x' A_S x + b_S' x + \gamma_S. \tag{5.5}$$

In particular, if we use the notations $\bar{A} := \mathbb{E}_S[A_S]$, $\bar{b} := \mathbb{E}_S[b_S]$, $\bar{\gamma} := \mathbb{E}_S[\gamma_S]$, we have:

$$F(x) = \frac{1}{2}x'\bar{A}x + \bar{b}'x + \bar{\gamma}. \tag{5.6}$$

We additionally impose $\bar{A} \in S^d_{++}$, that is, $\bar{A}$ is positive definite, and (5.6) has a unique minimizer, $x^* = -\bar{A}^{-1}\bar{b}$.

In the case of mini-batching, we have:

$$\begin{aligned} f_n(x, [S_n]) &= \frac{1}{n}\sum_{i=1}^{n} f(x, S_i) \\ &= \frac{1}{2}x'\left(\frac{1}{n}\sum_{i=1}^{n} A_{S_i}\right)x + \left(\frac{1}{n}\sum_{i=1}^{n} b_{S_i}\right)' x + \left(\frac{1}{n}\sum_{i=1}^{n} \gamma_{S_i}\right) \\ &= \frac{1}{2}x' A_n x + b_n' x + \gamma_n, \end{aligned} \tag{5.7}$$

where A_n, b_n and γ_n are the empirical expectations of A_S, b_S and γ_S from n independent random draws $[S_n]$ of S (we leave their dependence on $[S_n]$ implicit). Note this mini-batch setting is equivalent to a problem with $n = 1$ where we redefined the random variables A_S, b_S and γ_S.

5.2.2 Algorithms

We introduce here the stochastic gradient and proximal point steps om this setting. To build intuition, these steps are represented on Figure 5-1.

Stochastic Gradient Step Stochastic gradient steps (referred to as "gradient steps" for simplicity in the rest of the chapter), with a step-size schedule $[\mu_k] = (\mu_1, \mu_2, \cdots) \in \mathbb{R}_+^{\mathbb{N}}$, correspond to the following updates:

$$\begin{aligned} \forall k \geq 1, \quad x_k &= \mathrm{grad}(x_{k-1}, \mu_k) \\ &:= x_{k-1} - \mu_k \nabla f(x_{k-1}, S^k) \\ &= x_{k-1} - \mu_k \left(A_{S^k} x_{k-1} + b_{S^k}\right), \end{aligned} \tag{5.8}$$

where the super-script k of the random variables corresponds to the independent samples at each step k.

For a general mini-batch size n, we have:

$$\mathrm{grad}_n(x_{k-1}, \mu_k) := x_{k-1} - \mu_k \left(A_n^k x_{k-1} + b_n^k\right). \tag{5.9}$$

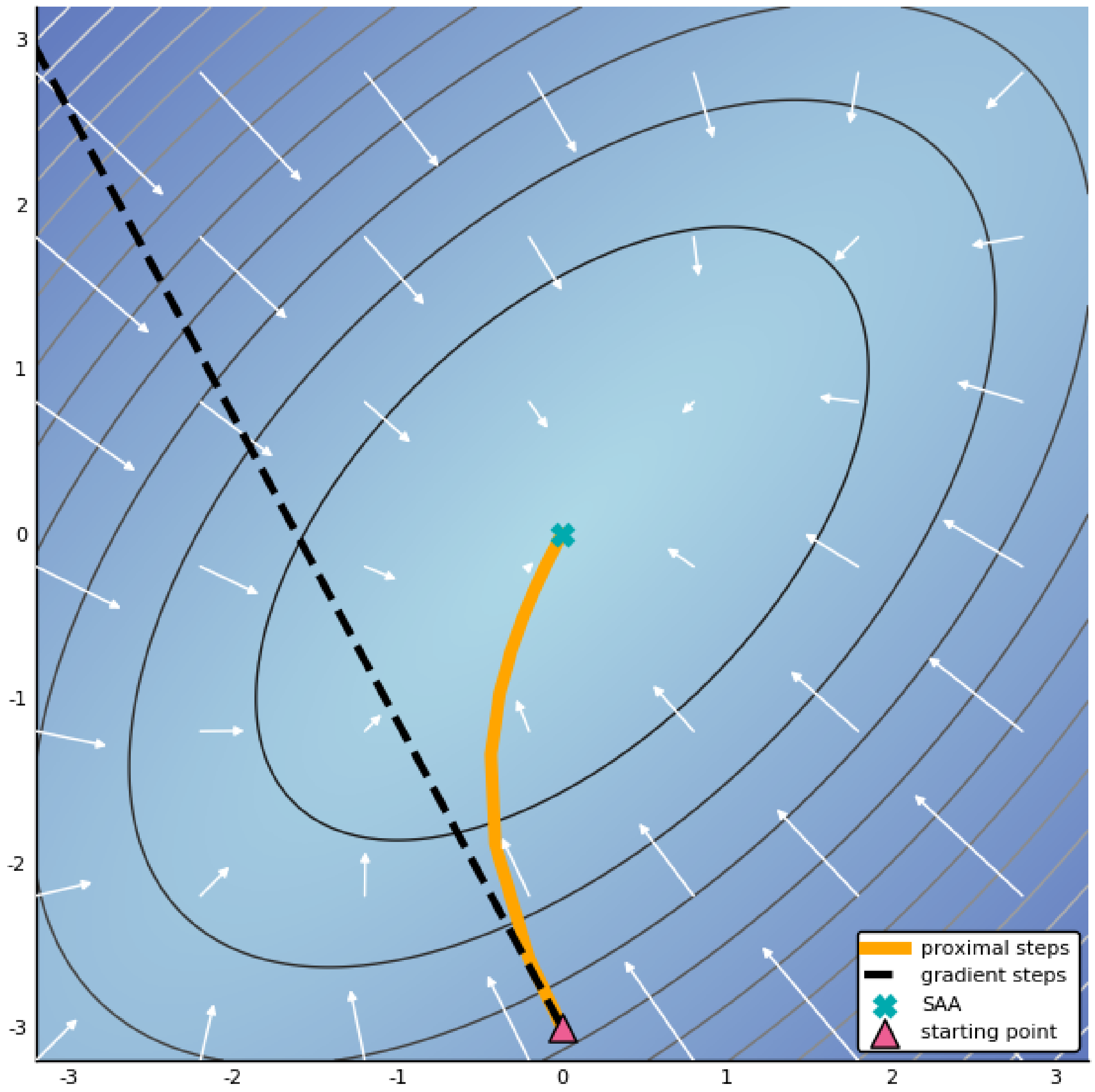


Figure 5-1: Visualizing proximal steps (orange line), gradient steps (black dashed line) and sample average approximation (blue cross) on a sampled convex quadratic function $f(.,S)$ in $d = 2$. The convex quadratic function is represented in blue, and its gradient with white arrows. We represent the set of possible steps when varying the step size μ from 0 to $+\infty$, starting from $(0, -3)$ (pink triangle). Note that gradient steps diverge, proximal steps converge to the sample average approximation (SAA) solution (the minimizer of $f(.,S)$) when $\mu \to +\infty$ and behaves similarly to a gradient step when $\mu \to 0$.

Stochastic Proximal Point Step A stochastic proximal point step (shortened to "proximal step" in the rest of the chapter) corresponds to:

$$\begin{aligned}\forall k \geq 1, \quad x_k &= \operatorname{prox}(x_{k-1}, \mu_k) \\ &:= \operatorname{argmin}_{x \in \mathbb{R}^d} f(x, S^k) + \frac{1}{2\mu_k}\|x - x_{k-1}\|^2.\end{aligned} \tag{5.10}$$

And, for general mini-batch size n,

$$\operatorname{prox}_n(x_{k-1}, \mu_k) := \operatorname{argmin}_{x \in \mathbb{R}^d} f_n(x, [S_n^k]) + \frac{1}{2\mu_k}\|x - x_{k-1}\|^2. \tag{5.11}$$

Computing a proximal step requires to solve an optimization problem, which is the minimization of a convex quadratic with definite positive Hessian (because of the regularization term), and therefore has a unique solution:

$$\begin{aligned} & \nabla_{x_k}\left(\frac{1}{2}x_k' A_n^k x_k + (b_n^k)' x_k + \gamma_n^k + \frac{1}{2\mu_k}\|x - x_{k-1}\|^2\right) = 0 && (5.12)\\ \Longrightarrow \quad & A_n^k x_k + b_n^k + \frac{1}{\mu_k}(x_k - x_{k-1}) = 0 && (5.13)\\ \Longrightarrow \quad & \operatorname{prox}(x_{k-1}, \mu_k)_n = \left(A_n^k + \frac{1}{\mu_k}\mathrm{I}\right)^{-1}\left(-b_n^k + \frac{1}{\mu_k}x_{k-1}\right) && (5.14)\end{aligned}$$

Note that the computation of the proximal step requires solving a linear system, whereas ta matrix-vector multiplication is sufficient for gradient steps.

Sample Average Approximation (SAA) SAA is not an iterative algorithm, but we will use it as a baseline. SAA does not require a starting point, and just computes the unique minimizer of $f_n(., [S_n])$ for some mini-batch size n. In the case where the minimizer is not unique, SAA is not defined.

$$\mathrm{SAA}(n) = \operatorname{argmin}_{x \in \mathbb{R}^d} f_n(x, [S_n]) \tag{5.15}$$

For SAA(n) to be defined, the matrix A_n needs to be almost surely invertible (positive definite), and we have:

$$\mathrm{SAA}(n) = -A_n^{-1} b_n \tag{5.16}$$

When it is defined, SAA(n) corresponds to a proximal step in the limit $\mu \to +\infty$ (see Figure 5-1). It is useful to compare k proximal or gradient steps with mini-batches n to SAA(nk), as the three algorithms need the same number of samples. Also note that grad(x_{k-1}, μ_k), prox(x_{k-1}, μ_k) and SAA(n) are all random functions as they depend on S.

5.3 Benefits of Proximal: One Dimension

In this section, we focus on the case $d = 1$. This simple case will allow us to obtain closed forms of the non-asymptotic behavior of gradient and proximal steps under optimal step schedule, which enables a precise comparison of the two algorithms.

5.3.1 Setting and Simplifications

Simplications In this setting, we have $f(x, S) = \frac{1}{2} a_S x^2 + b_S x + \gamma_S$, and $F(x) = \frac{1}{2} \bar{a} x^2 + \bar{b} x + \bar{\gamma}$. We additionally assume (to simplify the analysis) that a_S, b_S and γ_S are independent random variables. For better readability, we will use $a := a_S$, $b := b_S$ and $\gamma := \gamma_S$. As seen in the previous section, mini-batching is equivalent to replacing a and b by the random variables a_n and b_n.

Without loss of generality, we can assume $\gamma = 0$ as the constant terms do not affect the three optimization algorithm (see (5.8), (5.14), (5.16)). Multiplying the cost functions by a positive constant does not affect SAA, and can be offset for proximal and gradient steps by dividing the step sizes by the same constant (see equations (5.8) and (5.10)). We can therefore divide the cost by $\bar{a} > 0$, and focus without loss of generality on the case $\bar{a} = 1$. Finally, as the three algorithms are isometry-invariant, we change the variable space using the translation $x \to x + \bar{A}^{-1} \bar{b}$ which is equivalent to setting $\bar{b} = 0$.

Putting everything together, we only need to consider the stochastic optimization problem (5.1) with:

$$f(x, S) = \frac{a}{2}x^2 + bx, \tag{5.17}$$

with a and b independent random variables such that $\bar{a} = \mathbb{E}[a] = 1$ and $\bar{b} = \mathbb{E}[b] = 0$. In expectation, we have

$$F(x) = \frac{1}{2}x^2, \tag{5.18}$$

which implies $x^* = 0$. Figure 5-2 illustrates this optimization problem, for various choices of distributions for a and b. In this setting, b represents the gradient error at the optimal point $\frac{\partial f}{\partial x}(x^*, S) = b$, while a is the random curvature of the convex quadratic, as we have $\frac{\partial^2 f}{\partial x^2}(x, S) = a$.

Algorithms The gradient step of Equation (5.8) simplifies to

$$\begin{aligned} \mathrm{grad}(x, \mu) &= (1 - \mu a)x - \mu b \\ \mathrm{grad}_n(x, \mu) &= (1 - \mu a_n)x - \mu b_n, \end{aligned} \tag{5.19}$$

the proximal step of Equation (5.14) becomes

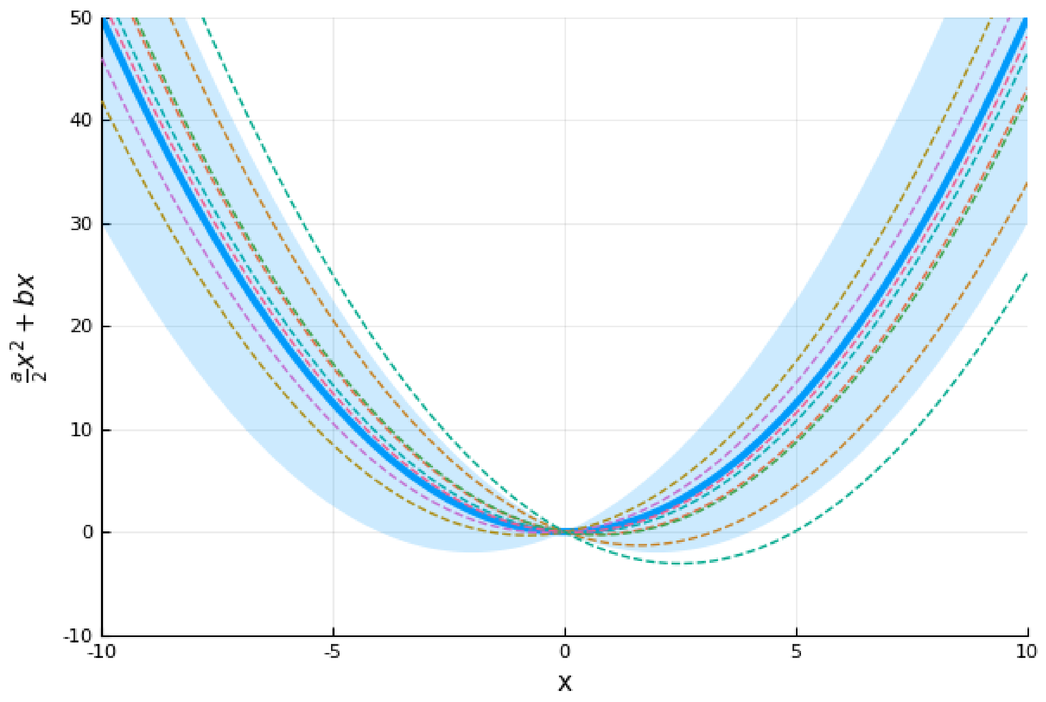
$$\begin{aligned} \mathrm{prox}(x, \mu) &= \frac{x - \mu b}{1 + \mu a} \\ \mathrm{prox}_n(x, \mu) &= \frac{x - \mu b_n}{1 + \mu a_n}, \end{aligned} \tag{5.20}$$

and the SAA (5.16) is

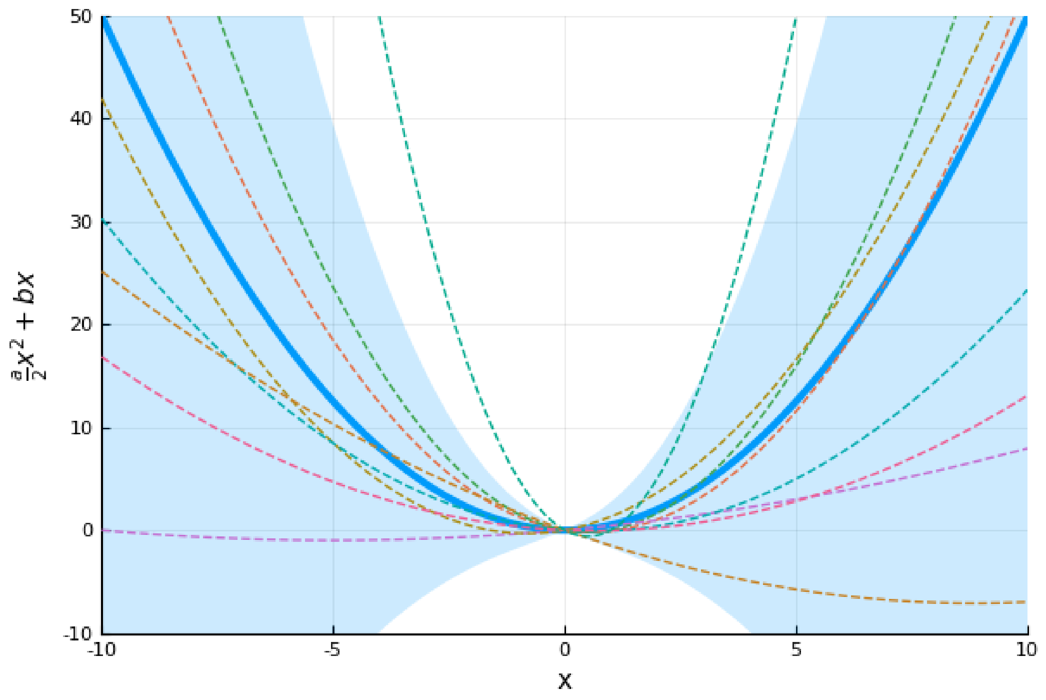
$$\mathrm{SAA}(n) = -\frac{b_n}{a_n}, \tag{5.21}$$

and is only defined when $a_n > 0$.

As it is typical in the literature [98, 112], we measure the error of a solution x as the squared distance from the optimal solution: $(x - x^*)^2 = x^2$. We want to compare the behavior of proximal and gradient steps with respect to this error. Specifically, given the respective step schedules $[\mu_k^{\text{prox}}]$ and $[\mu_k^{\text{grad}}]$, using proximal and gradient steps respectively give us the sequences of (random) solutions $[x_k^{\text{prox}}]$ and $[x_k^{\text{grad}}]$, and we



(a) $a = 1$ and $b \sim \text{Normal}(0, 1)$



(b) $a \sim \text{Gamma}(1, 1)$, $b \sim \text{Normal}(0, 1)$

Figure 5-2: Sample curves of $f(., S)$ (dashed) with expectation F represented as a blue line, and the standard deviation of $f(x, S)$ for each x shown presented with blue shade.

want to compare the sequences of expected errors $\left[\mathbb{E}\left[(x_k^{\text{grad}})^2\right]\right]$ and $[\mathbb{E}\left[(x_k^{\text{prox}})^2\right]]$. To remove the dependence on the choice of step schedule we first introduce an "optimal" step schedule.

5.3.2 Optimal Step Schedule

We know that proximal steps are more stable with respect to the choice of step schedule [3]. But we also want to know what is the "best possible" behavior of gradient and proximal steps across all possible step schedules. Given a random starting point $x \in \mathbb{R}^d$, we first show that the error after a gradient or proximal step only depends on the error of x (i.e., $\mathbb{E}\left[x^2\right]$) and the step size μ:

$$
\begin{aligned}
\mathbb{E}\left[\operatorname{grad}(x,\mu)^2\right] &= \mathbb{E}\left[((1-\mu a)x - \mu b)^2\right] && (5.22)\\
&= \mathbb{E}\left[x^2\right]\left(\mu^2\left(1+\sigma_a^2\right) - 2\mu + 1\right) + \mu^2\sigma_b^2 && (5.23)\\
\mathbb{E}\left[\operatorname{prox}(x,\mu)^2\right] &= \mathbb{E}\left[\left(\frac{x-\mu b}{1+\mu a}\right)^2\right] && (5.24)\\
&= \left(\mathbb{E}\left[x^2\right] + \mu^2\sigma_b^2\right)\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right], && (5.25)
\end{aligned}
$$

where σ_a^2 and σ_b^2 are the variances of a and b. (5.23) and (5.25) are obtained using the independence of a and b, as well as the fact that $\bar{a} = 1$ and $\bar{b} = 0$. The gradient error only depends on the two first moments of a and b, whereas the proximal error has a more complicated relationship with a.

As the expected error after the step is a function of the expected error before the step, we can study the evolution of this expected error independently from x. We therefore update our notation for simplicity, and consider the proximal and gradient steps in the space of expected error $e := \mathbb{E}\left[x^2\right]$:

$$
\begin{aligned}
\operatorname{grad}(e,\mu) &:= e\left(\mu^2\left(1+\sigma_a^2\right) - 2\mu + 1\right) + \mu^2\sigma_b^2\\
\operatorname{prox}(e,\mu) &:= \left(e + \mu^2\sigma_b^2\right)\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right].
\end{aligned}
\qquad (5.26)
$$

We can now look at the lowest expected error we can obtain after one step, given the error of the starting point:

$$\begin{aligned}\operatorname{grad}(e) &:= \min_{\mu} \operatorname{grad}(e,\mu)\\ \operatorname{prox}(e) &:= \min_{\mu} \operatorname{prox}(e,\mu).\end{aligned} \tag{5.27}$$

Starting from x_0 and iteratively applying (5.27), we obtain a step schedule and its associated sequence of errors.

Definition 2 (Optimal step schedule)**.** For gradient and proximal steps, we introduce the *optimal step schedules* $(\mu_k^{*\text{grad}})_{k\geq 1}$ and $(\mu_k^{*\text{prox}})_{k\geq 1}$. We also introduce the corresponding *optimal errors* $(e_k^{*\text{grad}})_{k\geq 1}$ and $(e_k^{*\text{prox}})_{k\geq 1}$. These schedule and errors are computed the following way:

$$\begin{aligned}
e_0^{*\text{grad}} &= e_0^{*\text{prox}} = e_0 := \mathbb{E}\left[(x_0)^2\right] & \\
e_k^{*\text{grad}} &:= \operatorname{grad}(e_{k-1}^{*\text{grad}}) & \forall k \geq 1,\\
e_k^{*\text{prox}} &:= \operatorname{prox}(e_{k-1}^{*\text{prox}}) & \forall k \geq 1,\\
\mu_k^{*\text{grad}} &:= \operatorname{argmin}_\mu \operatorname{grad}(e_{k-1}^{*\text{grad}},\mu) & \forall k \geq 1,\\
\mu_k^{*\text{prox}} &:= \operatorname{argmin}_\mu \operatorname{prox}(e_{k-1}^{*\text{prox}},\mu) & \forall k \geq 1.
\end{aligned} \tag{5.28}$$

These schedule may be constructed in a greedy way, but they are actually "optimal" in our setting ($d = 1$). That is, $e_k^{*\text{grad}}$ and $e_k^{*\text{prox}}$ are the best possible errors that can be reached in k steps given any choice of step schedule. We show this in the following proposition.

Proposition 1. *The optimal schedules have the following property:*

$$\begin{aligned}
e_k^{*\text{grad}} &= \min_{\mu_1,\cdots,\mu_k} \operatorname{grad}(\operatorname{grad}(\cdots\operatorname{grad}(e_0,\mu_1)\cdots,\mu_{k-1}),\mu_k) & \forall k \geq 1,\\
e_k^{*\text{prox}} &= \min_{\mu_1,\cdots,\mu_k} \operatorname{prox}(\operatorname{prox}(\cdots\operatorname{prox}(e_0,\mu_1)\cdots,\mu_{k-1}),\mu_k) & \forall k \geq 1.
\end{aligned} \tag{5.29}$$

Proof. (5.29) is true for $k = 1$, by definition of the optimal schedule. Suppose $\exists k \geq 1$ such that (5.29) is true. We will prove that it is also true for $k+1$, and Proposition 1 will follow by induction. $e \to \operatorname{grad}(e,\mu)$ is an increasing function of e for any μ. We can verify this using (5.26), as this statement is equivalent to

$$\mu^2\left(1+\sigma_a^2\right)-2\mu+1\geq 0 \iff (\mu-1)^2+\mu^2\sigma_a^2\geq 0.$$

$e \to \mathrm{prox}(e,\mu)$ is also increasing function of e as $\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right]\geq 0$. Therefore we have:

$$\min_{\mu_1,\cdots,\mu_{k+1}} \mathrm{grad}(\mathrm{grad}(\cdots\mathrm{grad}(e_0,\mu_1)\cdots,\mu_{k-1}),\mu_k) \tag{5.30}$$
$$= \min_{\mu_{k+1}}\min_{\mu_1,\cdots,\mu_k} \mathrm{grad}(\mathrm{grad}(\cdots\mathrm{grad}(e_0,\mu_1)\cdots,\mu_k),\mu_{k+1}) \tag{5.31}$$
$$= \min_{\mu_{k+1}}\mathrm{grad}(\min_{\mu_1,\cdots,\mu_k} \mathrm{grad}(\cdots\mathrm{grad}(e_0,\mu_1)\cdots,\mu_k),\mu_{k+1}) \tag{5.32}$$
$$= \min_{\mu_{k+1}}\mathrm{grad}(e_k^{*\mathrm{grad}},\mu_{k+1}) \tag{5.33}$$
$$= e_{k+1}^{*\mathrm{grad}} \tag{5.34}$$

where (5.32) uses the fact that grad(., μ) is an increasing function, (5.33) uses the induction hypothesis and (5.34) is the definition of the optimal schedule. And the exact same reasoning applies to proximal steps. □

Computing the optimal schedule If we can evaluate grad(e) and prox(e), Definition 2 provides a recursive way of computing the optimal step schedule:

$$\begin{aligned} e_k^{*\mathrm{grad}} &= \mathrm{grad}^{(k)}(e_0) \\ e_k^{*\mathrm{prox}} &= \mathrm{prox}^{(k)}(e_0) \end{aligned} \tag{5.35}$$

We can find a closed form for grad(e):

$$\begin{aligned} \mu^{*\mathrm{grad}} &:= \mathrm{argmin}_\mu\, \mathrm{grad}(e,\mu) \\ &= \mathrm{argmin}_\mu\, e\left(\mu^2\left(1+\sigma_a^2\right)-2\mu+1\right)+\mu^2\sigma_b^2 \\ \iff\quad & e\left(2\mu^{*\mathrm{grad}}\left(1+\sigma_a^2\right)-2\right)+2\mu^{*\mathrm{grad}}\sigma_b^2=0 \\ \iff\quad & \mu^{*\mathrm{grad}} = \frac{e}{\sigma_b^2+(\sigma_a^2+1)e} \end{aligned} \tag{5.36}$$

$$\begin{aligned}\operatorname{grad}(e) &= \operatorname{grad}(e, \mu^{*\text{grad}}) \\ &= e\frac{\sigma_a^2 e + \sigma_b^2}{(1+\sigma_a^2)e + \sigma_b^2}\end{aligned} \tag{5.37}$$

For proximal steps, we cannot find a general closed form because of the dependence on the distribution of a:

$$\begin{aligned}\mu^{*\text{prox}} &:= \operatorname{argmin}_\mu \operatorname{prox}(e, \mu) \\ &= \operatorname{argmin}_\mu \left(e + \mu^2\sigma_b^2\right)\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right]\end{aligned} \tag{5.38}$$

$$\begin{aligned}\operatorname{prox}(e) &= \operatorname{prox}(e, \mu^{*\text{prox}}) \\ &= \min_\mu \left(e + \mu^2\sigma_b^2\right)\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right]\end{aligned} \tag{5.39}$$

In the case of mini-batches, we also define $\operatorname{grad}_n(e)$ and $\operatorname{prox}_n(e)$ to be the equivalent of $\operatorname{grad}(e)$ and $\operatorname{prox}(e)$ when the mini-batch size is n.

We note that this optimal schedule enables a fair comparison between the two algorithms. It is chosen without the knowledge of the samples $[S^k]$, only their distribution. We can also use this framework to represent the uncertainty in the optimal solution x^*. Indeed, because of the space translation used in Section 5.3.1, there is an equivalence between uncertainty in the optimal solution x^* and the uncertainty in the starting point x_0.

For comparison, we can also compute the error of SAA with n samples (when it is defined):

$$e_n^{\text{SAA}} := \mathbb{E}\left[(\operatorname{SAA}(n))^2\right] = \mathbb{E}\left[\left(-\frac{b_n}{a_n}\right)^2\right] = \frac{\sigma_b^2}{n}\mathbb{E}\left[\frac{1}{a_n^2}\right] \tag{5.40}$$

In a general case, we do not have a closed form for the solution of the recurrence relations (5.35). Nonetheless, we can look at some special cases to build intuition and better understand the general case.

5.3.3 Deterministic Curvature, Random Gradient

We first consider the case $a = 1$, which corresponds to a deterministic curvature (see Figure 5-2a), and the only uncertainty comes from b. In this case, the recurrence relations defining the optimal step schedule simplify and we can get closed forms of the optimal schedules and errors.

Gradient Steps (5.36) and (5.37) simplify to:

$$\begin{aligned} \mu^{*\text{grad}}(e) &= \frac{e}{e+\sigma_b^2} \\ \text{grad}(e) &= \frac{e\sigma_b^2}{e+\sigma_b^2} \end{aligned} \tag{5.41}$$

and it can easily be verified by induction that the closed form solution of the optimal schedule is:

$$e_k^{*\text{grad}} = \text{grad}^{(k)}(e_0) \tag{5.42}$$

$$= \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \tag{5.43}$$

$$\mu_k^{*\text{grad}} = \frac{e_{k-1}^{*\text{grad}}}{e_{k-1}^{*\text{grad}} + \sigma_b^2} \tag{5.44}$$

$$= \frac{1}{\frac{\sigma_b^2}{e_0} + k} \tag{5.45}$$

Proximal Steps (5.38) and (5.39) simplify to:

$$\mu^{*\text{prox}}(e) = \text{argmin}_\mu \frac{e + \mu^2\sigma_b^2}{(1+\mu)^2} \tag{5.46}$$

$$\iff \quad 2\sigma_b^2\mu^{*\text{prox}}(e)(1+\mu^{*\text{prox}}(e))^2 - (e + \mu^{*\text{prox}}(e)^2\sigma_b^2)2(1+\mu^{*\text{prox}}(e)) = 0 \tag{5.47}$$

$$\iff \quad \sigma_b^2\mu^{*\text{prox}}(e)(1+\mu^{*\text{prox}}(e)) = e + \mu^{*\text{prox}}(e)^2\sigma_b^2 = 0 \tag{5.48}$$

$$\iff \quad \mu^{*\text{prox}}(e) = \frac{e}{\sigma_b^2} \tag{5.49}$$

$$\begin{aligned}
\operatorname{prox}(e) &= \operatorname{prox}(e, \mu^{*\mathrm{prox}}(e)) && (5.50)\\
&= \frac{e + (\frac{e}{\sigma_b^2})^2 \sigma_b^2}{\left(1 + \frac{e}{\sigma_b^2}\right)^2} && (5.51)\\
&= \frac{e\sigma_b^2}{e + \sigma_b^2} && (5.52)\\
&= \operatorname{grad}(e). && (5.53)
\end{aligned}$$

where (5.47) is just obtained by computing the derivative and setting it to 0. From (5.53), we get that optimal schedule of gradient and proximal steps have the same optimal errors in the case of deterministic curvature. Nonetheless, the choice of step sizes is not the same:

$$\begin{aligned}
e_k^{*\mathrm{prox}} &= e_k^{*\mathrm{grad}} && (5.54)\\
&= \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} && (5.55)\\
\mu_k^{*\mathrm{prox}} &= \frac{e_{k-1}^{*\mathrm{prox}}}{\sigma_b^2} && (5.56)\\
&= \frac{1}{\frac{\sigma_b^2}{e_0} + (k-1)} && (5.57)
\end{aligned}$$

We note the sub-linear convergence of the optimal step schedule, as well as the following bound:

$$e_k^{*\mathrm{grad}} = e_k^{*\mathrm{prox}} \leq \sigma_b^2 \quad \forall k \geq 1 \tag{5.58}$$

This bound is independent of the starting error e_0. Therefore, the optimal errors converge in one step to a fixed region whose size depends on σ_b^2.

Mini-batching In this setting, using $n > 1$ does not affect the convergence. Formally, given $N, k, n \geq 1$, doing Nk steps with a batch size n is equivalent to doing k steps with batch size nN: only the total number of data-points seen (Nkn) matters.

Indeed, in both case we get the final error:

$$\frac{1}{\frac{1}{e_0} + Nkn\frac{1}{\sigma_b^2}}$$

5.3.4 Random Curvature, Deterministic Gradient

In this setting, we set $b = 0$ but a random. This case is the "noiseless" case (or "easy" in [3]), as $x^* = 0$ is an optimal solution for any choice of a, and we expect linear convergence of the algorithms.

Gradient Steps (5.36) and (5.37) simplifies to:

$$\begin{aligned} \mu^{*\text{grad}} &= \frac{1}{1+\sigma_a^2} \\ \text{grad}(e) &= e\frac{\sigma_a^2}{1+\sigma_a^2} \end{aligned} \tag{5.59}$$

Therefore the convergence is linear and we have:

$$e_k^{*\text{grad}} = e_0\left(\frac{\sigma_a^2}{1+\sigma_a^2}\right)^k \tag{5.60}$$

$$\mu_k^{*\text{grad}} = \frac{1}{1+\sigma_a^2} \tag{5.61}$$

In this case, the step size is constant and the geometric convergence depends on σ_a^2.

Remark 2. In this setting, mini-batching "hurts" the gradient steps: Nk steps with a batch size n leads to a lower error than N steps with batch size kn:

$$e_0\left(\frac{\frac{\sigma_a^2}{n}}{1+\frac{\sigma_a^2}{n}}\right)^{Nk} \leq e_0\left(\frac{\frac{\sigma_a^2}{nk}}{1+\frac{\sigma_a^2}{nk}}\right)^{N} \tag{5.62}$$

Proof. Taking the logarithm, (5.62) becomes:

$$Nk\log\left(\frac{\frac{\sigma_a^2}{n}}{1+\frac{\sigma_a^2}{n}}\right) \leq N\log\left(\frac{\frac{\sigma_a^2}{nk}}{1+\frac{\sigma_a^2}{nk}}\right) \tag{5.63}$$

$$\iff \quad f(x) \leq \frac{1}{k}f\left(\frac{x}{k}\right) \tag{5.64}$$

with $x = \frac{\sigma_a^2}{n}$ and $f(x) = \log(\frac{x}{1+x})$. We then have:

$$\frac{d}{dx}_{x \geq 0} \left(\frac{1}{k} f\left(\frac{x}{k}\right) - f(x)\right) = \frac{\frac{1}{k}}{x(\frac{1}{k}x+1)} - \frac{1}{x(x+1)} = \frac{\frac{1}{k}-1}{x(\frac{1}{k}x+1)(x+1)} \leq 0 \quad (5.65)$$

and

$$\lim_{x \to \infty} \frac{1}{k} f\left(\frac{x}{k}\right) - f(x) = 0 \quad (5.66)$$

Therefore

$$\frac{1}{k} f\left(\frac{x}{k}\right) - f(x) \geq 0 \quad (5.67)$$

which proves our remark. □

Proximal Steps (5.38) and (5.39) simplify to:

$$\begin{aligned} \mu^{*\text{prox}} &= \text{argmin}_{\mu} e \, \mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] \\ &= +\infty \\ \text{prox}(e) &= \mathbb{E}_a \begin{cases} e & \text{if } a = 0 \\ e & \text{otherwise.} \end{cases} \\ &= e\mathbb{P}(a=0) \end{aligned} \quad (5.68)$$

that is, proximal steps converge to $x^* = 0$ for the first non-zero sampled value of a. In terms of error, we have geometric convergence:

$$e_k^{*\text{prox}} = e_0 \left(\mathbb{P}(a=0)\right)^k \quad (5.69)$$

$$\mu_k^{*\text{prox}} = +\infty \quad (5.70)$$

For any value of a, a proximal step reduces the error more than a gradient step, therefore the convergence of the optimal schedule is faster for proximal steps. Furthermore the two algorithms behave extremely differently: If $\mathbb{P}(a = 0) = 0$, proximal is not affected by the curvature noise and converges in one step, whereas gradient steps converge linearly. Furthermore, unlike gradient steps, proximal steps are not affected negatively by mini-batching:

Remark 3. In this setting, mini-batching does not affect the proximal steps: kN steps with a batch size n, or N steps with batch size kn both lead to the error:

$$e_0 \mathbb{P}(a = 0)^{knN} \tag{5.71}$$

Proof. kN steps with batch size n have the error:

$$e_0 \mathbb{P}(a_n = 0)^{kN} = e_0 (\mathbb{P}(a = 0)^n)^{kN} = e_0 \mathbb{P}(a = 0)^{knN} \tag{5.72}$$

N steps with batch size kn have the same error:

$$e_0 \mathbb{P}(a_{kn} = 0)^{N} = e_0 (\mathbb{P}(a = 0)^{kn})^{N} = e_0 \mathbb{P}(a = 0)^{knN} \tag{5.73}$$

□

5.3.5 General Case

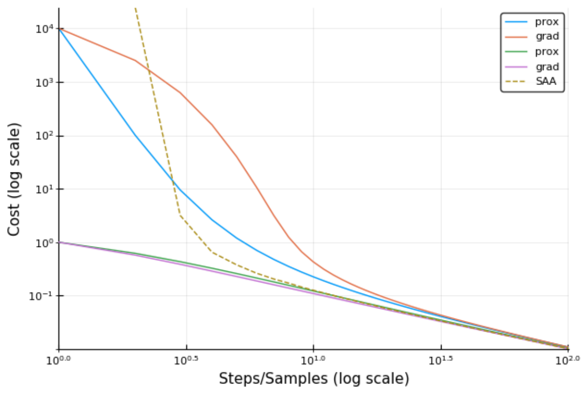
When both a and b are random, we do not have a closed form of the optimal schedule for gradient or proximal steps. But the optimal schedule and the algorithm behavior can still be obtained numerically, as the optimal schedule can be computed iteratively (see Definition 2).

Two regimes of convergence Figure 5-3 shows 4 simulations for various distributions for a. The three algorithms have the same asymptotic convergence under an optimal step schedule. We formalize this finding in the following proposition:

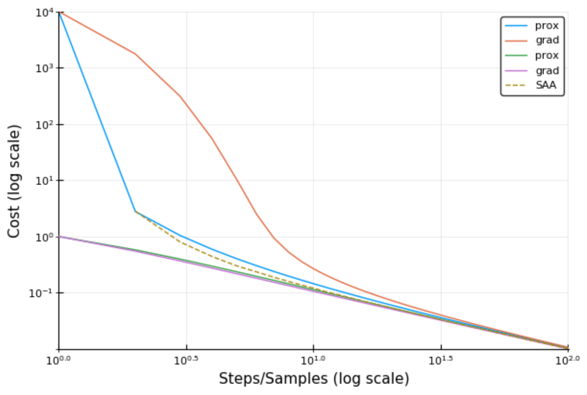
Proposition 2. *We have the following asymptotic behaviors:*

(a) Asymptotic sublinear convergence for gradient steps:

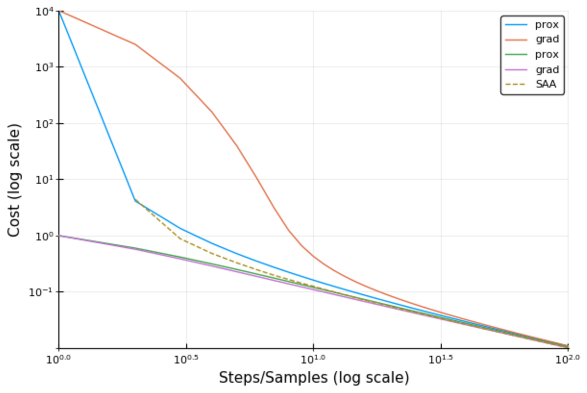
$$e_k^{*\text{grad}} \sim_{k \to +\infty} \frac{\sigma_b^2}{k} \tag{5.74}$$



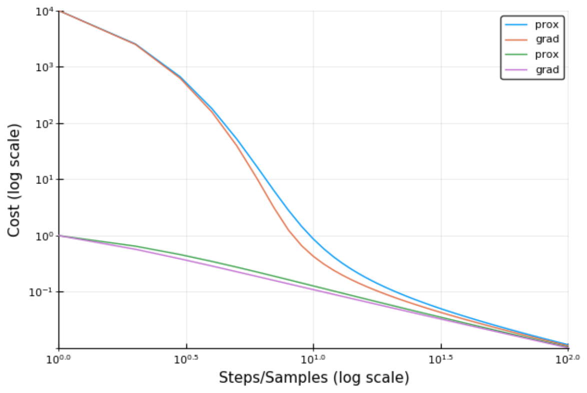
(a) $a \sim \text{Uniform}(0, 2)$ ($\sigma_a^2 = 1/3$)



(b) $a \sim \text{Uniform}(0.2, 1.8)$



(c) $a \sim$ Gamma, ($\sigma_a^2 = 1/3$)



(d) $a \in A \times \text{Bernouilli}(p)$, ($\sigma_a^2 = 1/3$)

Figure 5-3: Errors of the optimal schedule for proximal steps $[e_k^{*\text{prox}}]$ (blue and green curves) and gradient steps $[e_k^{*\text{grad}}]$ (orange and purple) when starting from $e_0 = 10^4$ and $e_0 = 1$. The x-axis represent the number of steps of the iterative algorithms, which also corresponds to the number of samples seen. We compute SAA (dashed yellow) for various values of n. We choose b such that $\sigma_b^2 = 1$ (with $b \sim \text{Normal}(0, 1)$) and we show various distributions for a, always respecting $\bar{a} = 1$. Additionally, (a), ((c) and (d) all have $\sigma_a^2 = 1/3$. In (d), the parameters p and A are chosen to have the desired mean and variance. The figure uses the logarithmic scale on its axis.

(b) *Same asymptotic sublinear convergence for proximal steps*

$$e_k^{*\text{prox}} \sim_{k\to+\infty} \frac{\sigma_b^2}{k} \tag{5.75}$$

(c) *Additionally, if* $\lim_{n\to+\infty} \mathbb{E}[\frac{1}{a_n^2}] = 1$*, which is true for example when a is bounded away from 0, we also have the same asymptotic convergence for SAA:*

$$e_k^{*\text{SAA}} \sim_{k\to+\infty} \frac{\sigma_b^2}{k} \tag{5.76}$$

(d) *Lower bound for gradient steps (with same asymptotic behavior)*

$$e_k^{*\text{grad}} \geq \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \qquad \forall k \geq 0 \tag{5.77}$$

(e) *Same lower bound for proximal steps*

$$e_k^{*\text{prox}} \geq \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \qquad \forall k \geq 0 \tag{5.78}$$

Proof of (a). We want to prove:

$$e_k^{*\text{grad}} \sim_{k\to+\infty} \frac{\sigma_b^2}{k} \tag{5.79}$$

We have:

$$e_{k+1}^{*\text{grad}} = \text{grad}(e_k^{*\text{grad}}) = e_k^{*\text{grad}} \frac{\sigma_a^2 e_k^{*\text{grad}} + \sigma_b^2}{(1+\sigma_a^2)e_k^{*\text{grad}} + \sigma_b^2}. \tag{5.80}$$

In other words, given the sequence $[u_n]$ such that:

$$u_{n+1} = F(u_n) := u_n \frac{\sigma_a^2 u_n + \sigma_b^2}{(1+\sigma_a^2)u_n + \sigma_b^2}, \tag{5.81}$$

and $u_0 = e_0$, we want to prove the asymptotic behavior $u_n \sim_{n\to+\infty} \frac{\sigma_b^2}{n}$, without having a closed form for u_n. First, by trivial induction, as $x \geq 0 \implies F(x) \geq 0$, we have

$$u_n \geq 0 \qquad\qquad \forall n \geq 0. \tag{5.82}$$

We also have:

$$u_{n+1} \leq u_n \qquad\qquad \forall n \geq 0, \tag{5.83}$$

and this follows from a simple inequality:

$$F(u_n) = u_n \frac{1}{1 + \frac{u_n}{\sigma_a^2 e_k^{*\text{grad}} + \sigma_b^2}} \leq u_n \qquad\qquad \forall n \geq 0. \tag{5.84}$$

Therefore $[u_n]$ is a non-negative decreasing sequence, and hence converges to a limit $l \geq 0$. To get the value of l, we take the limit on both sides of (5.81):

$$\lim_{n\to\infty} F(u_n) = \lim_{n\to\infty} u_n \tag{5.85}$$

$$\implies \qquad \lim_{n\to\infty} u_n \frac{\sigma_a^2 u_n + \sigma_b^2}{(1+\sigma_a^2)u_n + \sigma_b^2} = l \tag{5.86}$$

$$\implies \qquad l \frac{\sigma_a^2 l + \sigma_b^2}{(1+\sigma_a^2)l + \sigma_b^2} = l \tag{5.87}$$

$$\implies \qquad l \frac{-l}{(1+\sigma_a^2)l + \sigma_b^2} = 0 \tag{5.88}$$

$$\implies \qquad l = 0 \tag{5.89}$$

Which yields:

$$\lim_{n\to+\infty} u_n = 0 \iff u_n = o(1). \tag{5.90}$$

Let $v_n = \phi(u_n) = \frac{\sigma_b^2}{u_n}$. Using a Taylor approximation of ϕ in u_n, we have, for all $n \geq 0$:

$$v_{n+1} - v_n = (u_{n+1} - u_n)\phi'(u_n) + o(u_{n+1} - u_n) \tag{5.91}$$

$$= -\frac{(u_{n+1} - u_n)\sigma_b^2}{u_n^2} + o(1) \tag{5.92}$$

as $u_{n+1} - u_n = o(1) - o(1) = o(1)$. Using Taylor approximations once more, we also have:

$$u_{n+1} = u_n \frac{\sigma_a^2 u_n + \sigma_b^2}{(1 + \sigma_a^2)u_n + \sigma_b^2} \tag{5.93}$$

$$= u_n \frac{\frac{\sigma_a^2}{\sigma_b^2} u_n + 1}{\frac{1+\sigma_a^2}{\sigma_b^2} u_n + 1} \tag{5.94}$$

$$= u_n(\frac{\sigma_a^2}{\sigma_b^2} u_n + 1)(1 - \frac{1 + \sigma_a^2}{\sigma_b^2} u_n + o(u_n)) \tag{5.95}$$

$$= u_n(1 - \frac{1}{\sigma_b^2} u_n + o(u_n)) \tag{5.96}$$

$$= u_n - \frac{1}{\sigma_b^2} u_n^2 + o(u_n^2). \tag{5.97}$$

We insert this expression in (5.92) and obtain:

$$v_{n+1} - v_n = -\frac{(-\frac{1}{\sigma_b^2} u_n^2 + o(u_n^2))\sigma_b^2}{u_n^2} + o(1) \tag{5.98}$$

$$= 1 + o(1). \tag{5.99}$$

The series $\sum_n v_{n+1} - v_n$ diverges, and equivalence of the terms implies equivalence of the partial sums:

$$\sum_{i=1}^{n-1} v_{i+1} - v_i = n + o(n) \tag{5.100}$$

$$\implies \qquad v_n - v_0 = n + o(n) \tag{5.101}$$

$$\implies \qquad \frac{\sigma_b^2}{u_n} = n + o(n) \tag{5.102}$$

$$\implies \qquad \frac{\sigma_b^2}{n u_n} = 1 + o(1) \tag{5.103}$$

$$\implies \qquad u_n \sim_{n\to+\infty} \frac{\sigma_b^2}{n} \tag{5.104}$$

$$\implies \qquad e_k^{*\text{grad}} \sim_{k\to+\infty} \frac{\sigma_b^2}{k} \tag{5.105}$$

□

Proof of (b) and (e). We want to prove:

$$e_k^{*\text{prox}} \sim_{k\to+\infty} \frac{\sigma_b^2}{k} \tag{5.106}$$

$$e_k^{*\text{prox}} \geq \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \qquad \forall k \geq 0. \tag{5.107}$$

We do not have a closed form for $e_k^{*\text{prox}}$ but we have:

$$e_{k+1}^{*\text{prox}} = \min_\mu \operatorname{prox}(e, \mu) = \min_\mu \left(e_k^{*\text{prox}} + \mu^2\sigma_b^2\right) \mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] \qquad \forall k \geq 0. \tag{5.108}$$

First, we prove the bound:

$$\frac{1}{(1+\mu)^2} \leq \mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] \leq 1 - 2\mu + 3(1+\sigma_a^2)\mu^2. \tag{5.109}$$

The left-hand term is just Jensen inequality on the convex function $a \to \frac{1}{(1+\mu a)^2}$ which gives

$$\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] \geq \frac{1}{(1+\mu\mathbb{E}[a])^2} = \frac{1}{(1+\mu)^2}. \tag{5.110}$$

To prove the right-hand inequality of 5.109, let $f(\mu) = \frac{1}{(1+\mu a)^2}$ with $a \geq 0, \mu \geq 0$. We have $f'(\mu) = \frac{-2a}{(1+\mu a)^3} \leq 0$, $f''(\mu) = \frac{6a^2}{(1+\mu a)^4} \geq 0$ and $f'''(\mu) = \frac{-24a^3}{(1+\mu a)^5} \leq 0$. Therefore, given the signs of the third derivative, a Taylor expansion in $\mu = 0$ for $\mu \geq 0$ gives the bounds:

$$
\begin{aligned}
& & f(\mu) &\leq f(0) + \mu f'(0) + \frac{\mu^2}{2} f''(0) & (5.111)\\
&\implies & f(\mu) &\leq 1 - 2a\mu + 3a^2\mu^2 & (5.112)\\
&\implies & \mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] &\leq 1 - 2\mathbb{E}\left[a\right]\mu + 3\mathbb{E}\left[a^2\right]\mu^2 & (5.113)\\
&\implies & \mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] &\leq 1 - 2\mu + 3(1+\sigma_a^2)\mu^2. & (5.114)
\end{aligned}
$$

Inserting (5.109) into (5.108), we get:

$$
\min_{\mu}\left(e_k^{*\text{prox}} + \mu^2\sigma_b^2\right)\frac{1}{(1+\mu)^2} \leq e_{k+1}^{*\text{prox}} \leq \min_{\mu}\left(e_k^{*\text{prox}} + \mu^2\sigma_b^2\right)\left(1 - 2\mu + 3(1+\sigma_a^2)\mu^2\right) \tag{5.115}
$$

The left side corresponds to the case with $a = 1$ studied in Section 5.3.3, which had the closed form $\frac{1}{\frac{1}{e_0}+k\frac{1}{\sigma_b^2}}$, and with a trivial induction using this lower bound:

$$
\frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \leq e_k^{*\text{prox}} \qquad \forall k \geq 0, \tag{5.116}
$$

which proves part (e) of proposition 2.

We then develop the right side of (5.115):

$$
\begin{aligned}
&\left(e_k^{*\text{prox}} + \mu^2\sigma_b^2\right)\left(1 - 2\mu + 3(1+\sigma_a^2)\mu^2\right)\\
&= e_k^{*\text{prox}} - (2e_k^{*\text{prox}})\mu + (\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}})\mu^2 + (3(1+\sigma_a^2)\sigma_b^2\mu - 2\sigma_b^2)\mu^3\\
&\leq e_k^{*\text{prox}} - (2e_k^{*\text{prox}})\mu + (\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}})\mu^2 \text{ when } \mu \leq \frac{2}{3(1+\sigma_a^2)}. \qquad (5.117)
\end{aligned}
$$

We also have:

$$
\text{argmin}_{\mu} e_k^{*\text{prox}} - (2e_k^{*\text{prox}})\mu + (\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}})\mu^2 = \frac{e_k^{*\text{prox}}}{\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}}}, \tag{5.118}
$$

and this solution satisfies the condition from (5.117):

$$\frac{e_k^{*\text{prox}}}{\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}}} = \frac{1}{\frac{\sigma_b^2}{e_k^{*\text{prox}}} + 3(1+\sigma_a^2)} \leq \frac{2}{3(1+\sigma_a^2)}. \tag{5.119}$$

Therefore we get:

$$\begin{aligned}
&\min_\mu \left(e_k^{*\text{prox}} + \mu^2\sigma_b^2\right)\left(1 - 2\mu + 3(1+\sigma_a^2)\mu^2\right) && (5.120)\\
&\quad \leq \min_\mu e_k^{*\text{prox}} - (2e_k^{*\text{prox}})\mu + (\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}})\mu^2 && (5.121)\\
&\quad = e_k^{*\text{prox}} - \frac{(e_k^{*\text{prox}})^2}{\sigma_b^2 + 3(1+\sigma_a^2)e_k^{*\text{prox}}} && (5.122)\\
&\quad = e_k^{*\text{prox}} \frac{(2+3\sigma_a^2)e_k^{*\text{prox}} + \sigma_b^2}{(1+((2+3\sigma_a^2))e_k^{*\text{prox}} + \sigma_b^2}. && (5.123)
\end{aligned}$$

Note that the last term corresponds to the equation (5.81) of the optimal gradient steps, replacing σ_a^2 by $2+3\sigma_a^2$. Therefore our analysis of the gradient steps applies to this upper bound, and the sequence $(u_n)_n$ such that

$$u_{n+1} = u_n \frac{(2+3\sigma_a^2)u_n + \sigma_b^2}{(1+((2+3\sigma_a^2))u_n + \sigma_b^2}, \tag{5.124}$$

has the asymptotic behavior $u_n \sim_{n\to+\infty} \frac{\sigma_b^2}{n}$. Because of (5.123), we have by immediate induction that $e_k^{*\text{prox}} \leq u_k$, and therefore, also using the lower bound

$$\begin{aligned}
& && \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \leq e_k^{*\text{prox}} \leq u_k && \forall k \geq 0 && (5.125)\\
&\Longrightarrow && \frac{\sigma_b^2}{k} + o(\frac{1}{k}) \leq e_k^{*\text{prox}} \leq \frac{\sigma_b^2}{k} + o(\frac{1}{k}) && \forall k \geq 0 && (5.126)\\
&\Longrightarrow && e_k^{*\text{prox}} \sim_{n\to+\infty} \frac{\sigma_b^2}{n}, && && (5.127)
\end{aligned}$$

which proves part (b) of proposition 2. □

Proof of (c). We want to prove:

$$e_k^{*\text{SAA}} \sim_{k\to+\infty} \frac{\sigma_b^2}{k}, \tag{5.128}$$

when $\lim_{n\to+\infty}\mathbb{E}\left[\frac{1}{a_n^2}\right]=1$. Equation (5.40) gives us

$$e_k^{\mathrm{SAA}}=\frac{\sigma_b^2}{k}\mathbb{E}\left[\frac{1}{a_n^2}\right]. \tag{5.129}$$

As $\lim_{n\to+\infty}\mathbb{E}\left[\frac{1}{a_n^2}\right]=1$, we have $e_k^{\mathrm{SAA}} =\sim_{k\to+\infty} \frac{\sigma_b^2}{k}$. We then prove that $\lim_{n\to+\infty}\mathbb{E}\left[\frac{1}{a_n^2}\right]=$ 1 when a is bounded away from 0, i.e.:

$$(\exists\epsilon>0,\, a\geq\epsilon)\implies \lim_{n\to+\infty}\mathbb{E}\left[\frac{1}{a_n^2}\right]=1. \tag{5.130}$$

By strong law of large numbers, a_n converges to $\bar{a}=1$ almost surely. The continuity of $x\to\frac{1}{x^2}$ implies that $\frac{1}{a_n^2}$ also converges to 1 almost surely. Additionally, we have the bound $a_n\geq\epsilon$ and therefore $\frac{1}{a_n^2}\leq\frac{1}{\epsilon^2}$. Using the theorem of dominated convergence on the sequence of random variables $[\frac{1}{a_n^2}]$, we get:

$$\lim_{n\to+\infty}\mathbb{E}\left[\frac{1}{a_n^2}\right]=\mathbb{E}\left[1\right]=1 \tag{5.131}$$

□

Proof of (d). We want to prove:

$$e_k^{*\mathrm{grad}}\geq\frac{1}{\frac{1}{e_0}+k\frac{1}{\sigma_b^2}} \qquad \forall k\geq 0. \tag{5.132}$$

We have:

$$e_{k+1}^{*\mathrm{grad}}=e_k^{*\mathrm{grad}}\frac{\sigma_a^2 e_k^{*\mathrm{grad}}+\sigma_b^2}{(1+\sigma_a^2)e_k^{*\mathrm{grad}}+\sigma_b^2} \tag{5.133}$$

$$\geq e_k^{*\mathrm{grad}}\frac{\sigma_b^2}{e_k^{*\mathrm{grad}}+\sigma_b^2}. \tag{5.134}$$

Indeed, if $f(\sigma_a^2)=\frac{\sigma_a^2 e_k^{*\mathrm{grad}}+\sigma_b^2}{(1+\sigma_a^2)e_k^{*\mathrm{grad}}+\sigma_b^2}$ then

$$f'(\sigma_a^2)=\left(\frac{e_k^{*\mathrm{grad}}}{(1+\sigma_a^2)e_k^{*\mathrm{grad}}+\sigma_b^2}\right)^2\geq 0. \tag{5.135}$$

Therefore $f(\sigma_a^2) \geq f(0)$ which implies (5.134). We note that (5.134) corresponds to the case of deterministic curvature ($a = 1$) from Section 5.3.3 for which we have a closed form. By immediate induction, we have:

$$e_k^{*\text{grad}} \geq \frac{1}{\frac{1}{e_0} + k\frac{1}{\sigma_b^2}} \qquad \forall k \geq 0, \tag{5.136}$$

which proves part (d) of proposition 2. □

On the other hand, the behavior of the algorithms is not equivalent in the transient phase, and proximal steps outperform gradient steps by orders of magnitude in Figures 5-3a, 5-3b and 5-3c. We formalize this in the following proposition:

Proposition 3. *(a) For gradient steps, we have:*

$$e_k^{*\text{grad}} \geq e_0 \left(\frac{\sigma_a^2}{1+\sigma_a^2}\right)^k \qquad \forall k \geq 1,\ e_0 \geq 0 \tag{5.137}$$

$$e_k^{*\text{grad}} \sim_{e_0 \to +\infty} e_0 \left(\frac{\sigma_a^2}{1+\sigma_a^2}\right)^k \qquad \forall k \geq 1 \tag{5.138}$$

(b) If $\mathbb{E}[\frac{1}{a^2}] < +\infty$, proximal steps are bounded independently of e_0:

$$e_k^{*\text{prox}} \leq \sigma_b^2 \mathbb{E}[\frac{1}{a^2}] \qquad \forall k \geq 0 \tag{5.139}$$

Proof. **Proving (a)** we want to prove

$$e_k^{*\text{grad}} \geq e_0 \left(\frac{\sigma_a^2}{1+\sigma_a^2}\right)^k \qquad \forall k \geq 1,\ e_0 \geq 0 \tag{5.140}$$

We have:

$$e_{k+1}^{*\text{grad}} = e_k^{*\text{grad}} \frac{\sigma_a^2 e_k^{*\text{grad}} + \sigma_b^2}{(1+\sigma_a^2)e_k^{*\text{grad}} + \sigma_b^2} \tag{5.141}$$

$$\geq e_k^{*\text{grad}} \frac{\sigma_a^2}{1+\sigma_a^2} \tag{5.142}$$

indeed, if $f(\sigma_b^2) = \frac{\sigma_a^2 e_k^{*\text{grad}} + \sigma_b^2}{(1+\sigma_a^2)e_k^{*\text{grad}} + \sigma_b^2}$ then

$$f'(\sigma_b^2) = \frac{e_k^{*\text{grad}}}{\Big((1+\sigma_a^2)e_k^{*\text{grad}} + \sigma_b^2\Big)^2} \geq 0 \tag{5.143}$$

therefore $f(\sigma_b^2) \geq f(0)$ which implies (5.142). By immediate induction, we have:

$$e_k^{*\text{grad}} \geq e_0 \left(\frac{\sigma_a^2}{1+\sigma_a^2}\right)^k \qquad \forall k \geq 1,\ e_0 \geq 0 \tag{5.144}$$

Furthermore, we also have

$$e_{k+1}^{*\text{grad}} \sim_{e_0 \to +\infty} e_k^{*\text{grad}} \frac{\sigma_a^2}{1+\sigma_a^2} \qquad \forall k \geq 1,\ e_0 \geq 0 \tag{5.145}$$

again, by immediate induction on k, we have:

$$e_k^{*\text{grad}} \sim_{e_0 \to +\infty} e_0 \left(\frac{\sigma_a^2}{1+\sigma_a^2}\right)^k \qquad \forall k \geq 1 \tag{5.146}$$

which finishes the proof of part (a) of proposition 3.

Proving (b) we want to prove:

$$e_k^{*\text{prox}} \leq \sigma_b^2 \mathbb{E}[\frac{1}{a^2}] \qquad \forall k \geq 0 \tag{5.147}$$

when $\mathbb{E}[\frac{1}{a^2}] < +\infty$.

$$\lim_{\mu\to+\infty} \operatorname{prox}(e_0,\mu) = \lim_{\mu\to+\infty} \left(e_0+\mu^2\sigma_b^2\right)\mathbb{E}\left[\frac{1}{(1+\mu a)^2}\right] \tag{5.148}$$

$$= \lim_{\mu\to+\infty} \mathbb{E}\left[\frac{e_0+\mu^2\sigma_b^2}{(1+\mu a)^2}\right] \tag{5.149}$$

$$= \mathbb{E}\left[\lim_{\mu\to+\infty} \frac{e_0+\mu^2\sigma_b^2}{(1+\mu a)^2}\right] \tag{5.150}$$

$$= \mathbb{E}\left[\frac{\sigma_b^2}{a^2}\right] \tag{5.151}$$

where (5.150) is a consequence of the dominated convergence theorem. To choose the bound needed to apply this theorem, let $f(\mu) = \frac{e_0+\mu^2\sigma_b^2}{(1+\mu a)^2}$. We have:

$$f'(\mu) = \frac{\sigma_b^2 - 2ae_0 - \mu a\sigma_b^2}{(1+\mu a)^2} \tag{5.152}$$

Therefore f is increasing when $\mu \leq \frac{1}{a} - \frac{2e_0}{\sigma_b^2}$, and then decreasing. If $\frac{1}{a} - \frac{2e_0}{\sigma_b^2} \geq 0$, i.e., $\sigma_b^2 \geq 2ae_0$, then we have:

$$f(\mu) \leq f(\frac{1}{a} - \frac{2e_0}{\sigma_b^2}) = \frac{(\sigma_b^2)^2}{4a(b-ae_0)} \leq \frac{(\sigma_b^2)^2}{4a^2e_0} \tag{5.153}$$

as $\sigma_b^2 \geq 2ae_0 \implies b - ae_0 \geq ae_0$,

If $\sigma_b^2 < 2ae_0$, then

$$f(\mu) \leq f(0) = \sigma_b^2 - 2ae_0 \leq \sigma_b^2 \tag{5.154}$$

Therefore we have

$$f(\mu) \leq \sigma_b^2 + \frac{(\sigma_b^2)^2}{4a^2e_0} \tag{5.155}$$

and

$$\mathbb{E}\left[\sigma_b^2 + \frac{(\sigma_b^2)^2}{4a^2e_0}\right] = \sigma_b^2 + \frac{(\sigma_b^2)^2}{4e_0}\mathbb{E}\left[\frac{1}{a^2}\right] < +\infty \tag{5.156}$$

that is, we can do (5.150) by dominated convergence.

We can conclude with $e_1^{*\text{prox}} = \min_\mu \text{prox}(e_0, \mu) \leq \lim_{\mu \to +\infty} \text{prox}(e_0, \mu) = \sigma_b^2 \mathbb{E}[\frac{1}{a^2}]$

We already showed in the proof of proposition 2 that $[e_k^{*\text{prox}}]$ is a decreasing sequence, therefore $e_k^{*\text{prox}} \leq e_1^{*\text{prox}}$ which finishes the proof of part (b) of proposition 3. $\square$

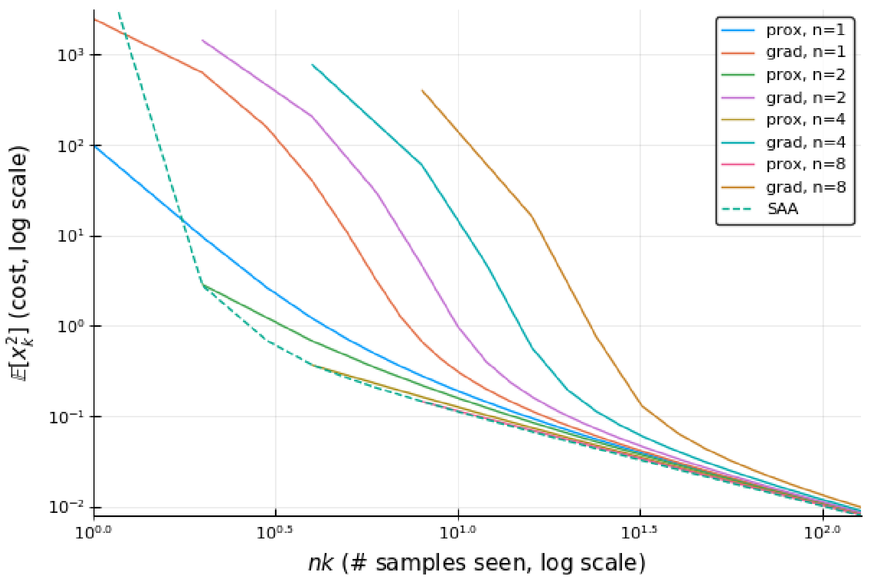
Putting all of this together, there are two regimes of convergence. In the transient phase, when and if $e >> \sigma_b^2$, proximal and gradient steps behave as if b was deterministic (studied in Section 5.3.4). After this phase, they behave asymptotically as if a was deterministic, i.e. the case of Section 5.3.3.

The only caveat is that proximal steps are not as good when a has a high probability to take values arbitrarily close to 0, in particular when $\mathbb{E}[\frac{1}{a^2}]$ is infinite or large. The comparison of figures 5-3a and 5-3b illustrates this difference. As an extreme case, figure 5-3d uses a Bernouilli distribution for a, which implies $\mathbb{P}(a = 0) > 0$ and affects the proximal steps negatively. Note that the gradient steps are not affected by these changes of distribution as long as $\bar{a}$ and σ_a^2 are not changed.

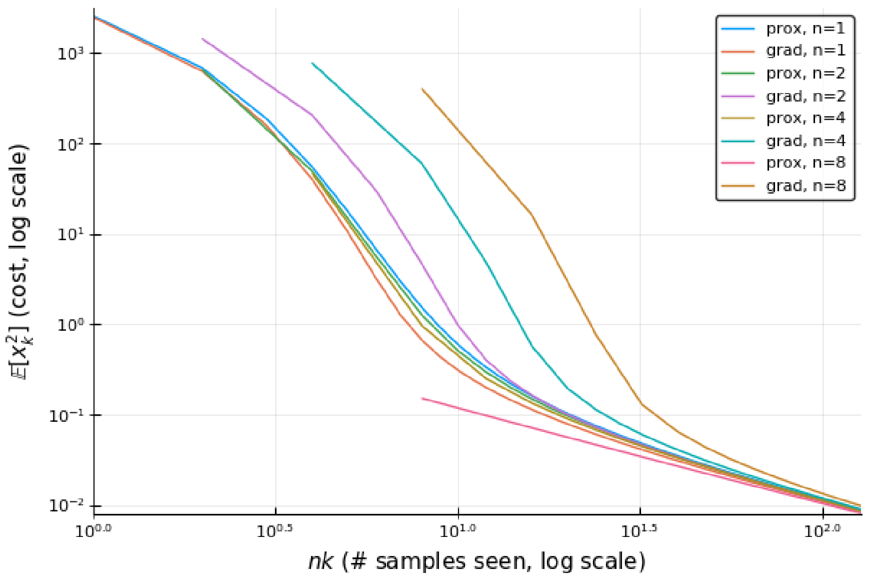
The errors of SAA for various n, when they are well defined, are also represented in Figure 5-3 (dashed yellow). As showed in Proposition 3. the first proximal step with optimal step size must be at least as good as SAA with $n = 1$ and the quality of the proximal transient phase of convergence approximation seems to be related to how much SAA outperforms the gradient transient phase. Note that in Figure 5-3d, we have $e_n^{\text{SAA}} = +\infty$ for any n.

Mini-batching Let $e_{n,k}^{*\text{grad}}$ and $e_{n,k}^{*\text{prox}}$ represent the optimal errors of k gradient and proximal steps with batch size n. In particular we have $e_{1,k}^{*\text{grad}} = e_k^{*\text{grad}}$. The fact that SAA with a batch of n points is sometimes better (Figure 5-3a) than $k = n$ proximal steps with $n = 1$ point means that using a larger batch for proximal steps is beneficial. Indeed, have:

$$e_{n,1}^{*\text{prox}} \leq e_n^{\text{SAA}}, \quad \forall n \geq 1, \forall e_0 \geq 0 \tag{5.157}$$



(a) Same setting as Figure 5-3a



(b) Same setting as Figure 5-3d. Note the strong improvement corresponding to batch size $n = 8$ for proximal.

Figure 5-4: Errors of the optimal schedule for proximal steps $[e_k^{*\text{prox}}]$ and gradient steps $[e_k^{*\text{grad}}]$, and errors of SAA (dashed curve). We start with $e_0 = 10^4$. We use $n \in \{1, 2, 4, 8\}$. We compare the different errors on the scale of nk, which corresponds to the total number of samples seen by the algorithms, i.e., the number of steps k times the number of samples per steps n ($k = 1$ for SAA). On this scale, gradient steps reach lower errors with smaller batches whereas proximal steps need bigger batches. The figure uses the logarithmic scale on its axis.

this inequality comes from proposition 3 (b), and is due to the fact that a proximal step with infinite step size is similar to SAA.

This means that doing one proximal step with a mini-batch n can lead to a better error than n proximal steps with mini-batch 1, unlike the two special cases we saw earlier.

This insight is confirmed and generalized in the simulations of Figure 5-4, where the proximal steps (for fixed total number of samples nk) reach lower errors when the batch size n is bigger, whereas gradient steps behave the opposite way.

We show formalize this result for gradient steps:

Theorem 2. *For a fixed total number of samples, gradient steps reach smaller errors with smaller batch sizes. Formally, for all $N \geq 1$, $n \geq 1$, $k \geq 1$ and $e_0 > 0$ we have:*

$$e^{*\text{grad}}_{n,kN} \leq e^{*\text{grad}}_{nN,k} \tag{5.158}$$

Lemma 1. *For all $n \geq 1$ and $e > 0$, we have:*

$$\text{grad}_n(\text{grad}(e)) \leq \text{grad}_{n+1}(e) \tag{5.159}$$

In other words, the optimal error for one step with a batch size of n after a step with a batch size of 1 is preferable to doing only one step with the $n+1$ samples together.

Proof of Lemma 1. From Eq.(5.37) we have:

$$\text{grad}(e) = e\frac{\sigma_a^2 e + \sigma_b^2}{(1+\sigma_a^2)e + \sigma_b^2} \tag{5.160}$$

$$\text{grad}_n(e) = e\frac{\frac{\sigma_a^2}{n} e + \frac{\sigma_b^2}{n}}{(1+\frac{\sigma_a^2}{n})e + \frac{\sigma_b^2}{n}} \tag{5.161}$$

$$= e\frac{\sigma_a^2 e + \sigma_b^2}{(n+\sigma_a^2)e + \sigma_b^2} \tag{5.162}$$

If $\sigma_b^2 = 0$, we are back to the study of Section 5.3.4 and the result is already proven (see Remark 2). Otherwise, the following sequence of equivalences proves the lemma:

$$\operatorname{grad}_{n+1}(e) \geq \operatorname{grad}_n(\operatorname{grad}(e)) \tag{5.163}$$

$$\iff e\frac{\sigma_a^2 e + \sigma_b^2}{(n+1+\sigma_a^2)e + \sigma_b^2} \geq e\frac{\sigma_a^2 e + \sigma_b^2}{(1+\sigma_a^2)e + \sigma_b^2} \frac{\sigma_a^2 e\frac{\sigma_a^2 e+\sigma_b^2}{(1+\sigma_a^2)e+\sigma_b^2} + \sigma_b^2}{(n+\sigma_a^2)e\frac{\sigma_a^2 e+\sigma_b^2}{(1+\sigma_a^2)e+\sigma_b^2} + \sigma_b^2} \tag{5.164}$$

$$\iff \frac{(1+\sigma_a^2)e + \sigma_b^2}{(n+1+\sigma_a^2)e + \sigma_b^2} \geq \frac{\sigma_a^2 e\frac{\sigma_a^2 e+\sigma_b^2}{(1+\sigma_a^2)e+\sigma_b^2} + \sigma_b^2}{(n+\sigma_a^2)e\frac{\sigma_a^2 e+\sigma_b^2}{(1+\sigma_a^2)e+\sigma_b^2} + \sigma_b^2} \tag{5.165}$$

$$\iff \frac{(1+\sigma_a^2)e + \sigma_b^2}{(n+1+\sigma_a^2)e + \sigma_b^2} \geq \frac{\sigma_a^2 e\frac{\sigma_a^2 e+\sigma_b^2}{(1+\sigma_a^2)e+\sigma_b^2} + \sigma_b^2}{(n+\sigma_a^2)e\frac{\sigma_a^2 e+\sigma_b^2}{(1+\sigma_a^2)e+\sigma_b^2} + \sigma_b^2} \tag{5.166}$$

$$\iff \frac{(1+\sigma_a^2)e + \sigma_b^2}{(n+1+\sigma_a^2)e + \sigma_b^2} \geq \frac{\sigma_a^2 e(\sigma_a^2 e + \sigma_b^2) + \sigma_b^2((1+\sigma_a^2)e + \sigma_b^2)}{(n+1)e(\sigma_a^2 e + \sigma_b^2) + \sigma_b^2((1+\sigma_a^2)e + \sigma_b^2)} \tag{5.167}$$

$$\iff ((1+\sigma_a^2)e + \sigma_b^2)((n+1)e(\sigma_a^2 e + \sigma_b^2) + \sigma_b^2((1+\sigma_a^2)e + \sigma_b^2)) \tag{5.168}$$

$$\geq ((n+1+\sigma_a^2)e + \sigma_b^2)(\sigma_a^2 e(\sigma_a^2 e + \sigma_b^2) + \sigma_b^2((1+\sigma_a^2)e + \sigma_b^2)) \tag{5.169}$$

$$\iff ((1+\sigma_a^2)e + \sigma_b^2)ne(\sigma_a^2 e + \sigma_b^2) \geq ne(\sigma_a^2 e(\sigma_a^2 e + \sigma_b^2) + \sigma_b^2((1+\sigma_a^2)e + \sigma_b^2)) \tag{5.170}$$

$$\iff (\sigma_a^2 e + \sigma_b^2)^2 + e(\sigma_a^2 e + \sigma_b^2) \geq (\sigma_a^2 e + \sigma_b^2)^2 + \sigma_b^2 e \tag{5.171}$$

$$\iff \sigma_a^2 e^2 \geq 0 \tag{5.172}$$

□

Proof of Theorem 2. We prove the theorem by showing that it can be reduced to Lemma 1.

If we define $a' = a_n$, then

$$a_{nN} = \frac{1}{nN}\sum_{i=1}^{nN} a_{w_i} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{n}\sum_{j=1}^{n} a_{w_{j+(i-1)N}} = \frac{1}{N}\sum_{i=1}^{N} a'_{w'_i} = a'_N \tag{5.173}$$

that is, a'_n, the average of n independent samples of a' has the same distribution as a_{nN}, the average of nN independent samples of a. Therefore, if we prove Theorem 2 with $n = 1$, i.e.,

$$e^{*\text{grad}}_{1,Nk} \leq e^{*\text{grad}}_{N,k} \qquad \forall k, N \geq 1. \tag{5.174}$$

Then, by using $a' = a_n$ we get

$$e^{*\text{grad}}_{n,Nk} \leq e^{*\text{grad}}_{nN,k} \qquad \forall k, N, n \geq 1. \tag{5.175}$$

. We will now use induction on k to show that:

$$\begin{aligned} & e^{*\text{grad}}_{1,N} \leq e^{*\text{grad}}_{N,1} \quad \forall N \geq 1, \, \forall e_0 \geq 0 \\ \Longrightarrow \quad & e^{*\text{grad}}_{1,Nk} \leq e^{*\text{grad}}_{N,k} \quad \forall k \geq 1, \, \forall N \geq 1, \, \forall e_0 \geq 0. \end{aligned} \tag{5.176}$$

This is trivial for $k = 1$, suppose now that it is true for some $k \geq 1$

$$\begin{aligned} & e^{*\text{grad}}_{1,N} \leq e^{*\text{grad}}_{N,1} && \forall N \geq 1, \, e_0 \geq 0 && (5.177) \\ \Longrightarrow \quad & (\text{grad})^{(N)}(e_0) \leq \text{grad}_N(e_0) && \forall N \geq 1, \, e_0 \geq 0 && (5.178) \\ \Longrightarrow \quad & (\text{grad})^{(N)}((\text{grad})^{(Nk)}(e_0)) \leq \text{grad}_N((\text{grad})^{(Nk)}(e_0)) && \forall N \geq 1, \, e_0 \geq 0 && (5.179) \\ \Longrightarrow \quad & (\text{grad})^{(N)}((\text{grad})^{(Nk)}(e_0)) \leq \text{grad}_N((\text{grad}_N)^{(k)}(e_0)) && \forall N \geq 1, \, e_0 \geq 0 && (5.180) \\ \Longrightarrow \quad & (\text{grad})^{((k+1)N)}(e_0)) \leq (\text{grad}_N)^{(k+1)}(e_0)) && \forall N \geq 1, \, e_0 \geq 0 && (5.181) \end{aligned}$$

where (5.179) is obtained by using the particular choice of $e_0 = (\text{grad})^{(Nk)}(e_0)$. And (5.180) follows from the induction hypothesis and the fact that $e \to \text{grad}_N(e)$ is an increasing function. By induction, this proves (5.176). We then prove by induction the following implication:

$$\begin{aligned} & \text{grad}_N(\text{grad}(e)) \leq \text{grad}_{N+1}(e) \quad \forall N \geq 1, \, e_0 \geq 0 \\ \Longrightarrow \quad & e^{*\text{grad}}_{1,N} \leq e^{*\text{grad}}_{N,1} \qquad \forall N \geq 1, \, e_0 \geq 0. \end{aligned} \tag{5.182}$$

This is trivial for $N = 1$, suppose that (5.182) is true for some $N \geq 1$. We get:

$$
\begin{aligned}
& & e^{*\mathrm{grad}}_{1,N} &\leq e^{*\mathrm{grad}}_{N,1} & \forall e_0 \geq 0 \qquad (5.183)\\
&\implies & (\mathrm{grad})^{(N)}(e_0) &\leq \mathrm{grad}_N(e_0) & \forall e_0 \geq 0 \qquad (5.184)\\
&\implies & (\mathrm{grad})^{(N)}(\mathrm{grad}(e_0)) &\leq \mathrm{grad}_N(\mathrm{grad}(e_0)) & \forall e_0 \geq 0 \qquad (5.185)\\
&\implies & (\mathrm{grad})^{(N+1)}(e_0) &\leq \mathrm{grad}_{N+1}(e_0) & \forall e_0 \geq 0 \qquad (5.186)\\
&\implies & e^{*\mathrm{grad}}_{1,N+1} &\leq e^{*\mathrm{grad}}_{N+1,1} & \forall e_0 \geq 0, \qquad (5.187)
\end{aligned}
$$

where (5.186) is obtained by applying $\mathrm{grad}_N(\mathrm{grad}(e)) \leq \mathrm{grad}_{N+1}(e)$. Therefore, Lemma 1, together with (5.176) and (5.182) proves Theorem 2. □

For proximal step, formulate the following conjecture, verified numerically but not proven formally:

Conjecture 1. *For an equivalent total number of samples, proximal steps reach smaller errors with larger batch sizes. Formally, for all $N \geq 1$, $n \geq 1$, $k \geq 1$ and $e_0 > 0$ we have:*

$$e^{*\mathrm{prox}}_{nN,k} \leq e^{*\mathrm{prox}}_{n,kN} \tag{5.188}$$

Following the same reasoning as the proof of Theorem 2, proving Conjecture 1 can be reduced to proving:

$$\mathrm{prox}_{n+1}(e) \leq \mathrm{prox}_n(\mathrm{prox}(e)) \qquad \forall e \geq 0,\ n \geq 1 \tag{5.189}$$

The experiments on Figure 5-4 show that mini-batching has a significant impact on the transient phase of the algorithms. Nonetheless, Proposition 2 states that asymptotically the choice of mini-batch size does not matter:

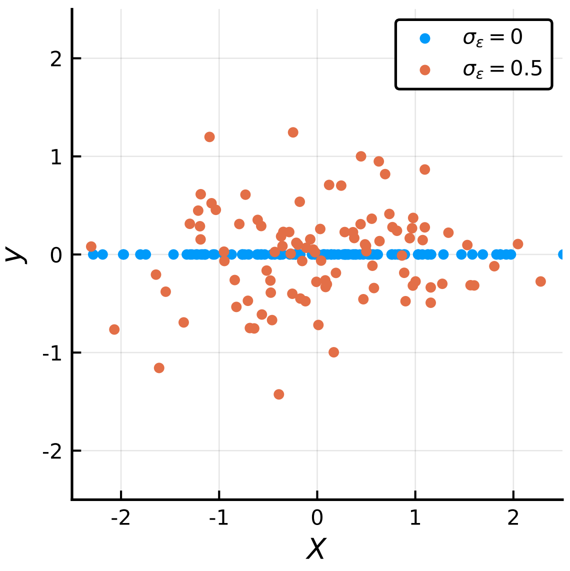
$$e^{*\mathrm{grad}}_{n,k} \sim_{k\to+\infty} e^{*\mathrm{prox}}_{n,k} \sim_{k\to+\infty} \frac{\sigma_b^2}{nk} \qquad \forall n \geq 1 \tag{5.190}$$

Through the use of an optimal step schedule, we studied the differences between proximal and gradient steps, showing that proximal steps can have a significant edge in the transient phase, especially when mini-batching is used.

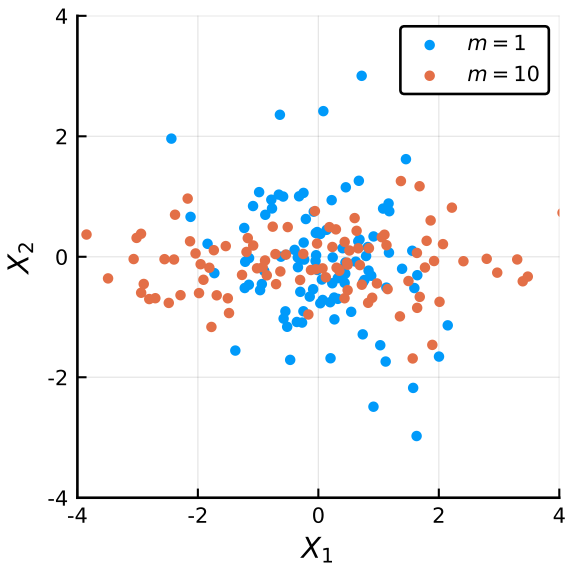
5.4 Benefits of Proximal: Ordinary Least Squares

We now compare the behavior of proximal and gradient steps on higher dimensions convex quadratic optimization problems: ordinary least squares (OLS). This study is empirical, and shows that our findings in 1D generalize to multiple dimensions. This section will compare the algorithms with a similar setting of “optimal schedules”, and the next section will focus on computational time.

5.4.1 Setting



(a) With $d = 1$, impact of noise on y given X (note that $\beta^* = 0$).



(b) With $d = 2$, visualizing the effect of the condition number on the feature distribution X.

Figure 5-5: Examples of linear data distributions in our OLS setting. We vary the distribution parameters σ_ϵ^2 (noise variance) and m (condition number), and represent 100 random samples for each.

We introduce the OLS setting of our experiments. $X_S \in \mathbb{R}^d$ represents the data features and $y_S \in \mathbb{R}$ its labels. To simplify the notations, the dependence on S will be implicit and we will use $X := X_S$ and $y = y_S$

Given some linear coefficients $\beta \in \mathbb{R}^d$, we consider the mean squared error:

$$F(\beta) = \frac{1}{2}\mathbb{E}\left[(X'\beta - y)^2\right] \tag{5.191}$$

The distributions of X and y follow traditional OLS assumptions: independent gaussian features and independent gaussian error. More precisely, we have:

- Each feature $1 \leq i \leq d$ has distribution $X_i \sim \mathcal{N}(0, \sigma_i^2)$, and each feature is independent.
- The distribution of the labels is a linear combination of the features with gaussian noise: $y = X\beta^* + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is independent from X.

Putting everything together, we have:

$$f(\beta, S) = \frac{1}{2}(\beta - \beta^*)'XX'(\beta - \beta^*) - (\epsilon X')(\beta - \beta^*) + \frac{1}{2}\epsilon^2 \tag{5.192}$$
$$F(\beta) = \frac{1}{2}(\beta - \beta^*)'\mathbb{E}[XX'](\beta - \beta^*) + \frac{\sigma_\epsilon^2}{2} \tag{5.193}$$

Our optimization algorithms are invariant by isometry of the decision space, so we can translate the decision space by β^* and set $\beta^* = 0$. We obtain:

$$f(\beta, S) = \frac{1}{2}\beta'XX'\beta - (\epsilon X')\beta + \frac{1}{2}\epsilon^2 \tag{5.194}$$

Which corresponds to the general convex optimization setting of Section 5.2 with $A_S = XX'$, $b_S = \epsilon X$ and $\gamma_S = \frac{1}{2}\epsilon^2$.

In particular, when we have a mini-batch of size n, we have $A_n = \frac{1}{n}X_n'X_n$ where $X_n \in \mathbb{R}^{n\times d}$ is the feature matrix of the mini-batch. That is, each row of X_n is a random independent draw of X. $b_n = \frac{1}{n}\epsilon_n'X_n$ where ϵ_n is the vector of n independent samples of ϵ, and $\gamma_n = \frac{1}{2n}\|\epsilon_n\|^2$.

$\bar{A} = \mathbb{E}[XX']$ is a positive semi-definite symmetric matrix, and is the Hessian of F. As the features are centered and independent, $\bar{A}$ is diagonal with coefficients $[\sigma_i^2]$. Therefore the condition number of the optimization problem is $m = \frac{\max_i \sigma_i^2}{\min_i \sigma_i^2}$.

This setting generalizes to correlated features, as this can be reduced to the case of independent features through isometries of the decision space.

In our experiment, to limit the number of parameters, we study the specific case where the values σ_i^2 are chosen to be regularly spread in the log scale around the value 1: $\sigma_i^2 = m^{\frac{d-i}{d-1}-0.5}$. That is, X is uniquely defined by the value m. The fact that the values σ_i^2 are centered around 1 is without loss of generality, as we have seen in Section 5.3.1 that we can multiply the cost function by a constant without affecting our study.

Also similarly to the 1D case, we start with a random starting point in $\beta_0 \in \mathbb{R}^d$ such that each dimension is independently sampled from a normal distribution: $(\beta_0)_i \sim \mathcal{N}(0, \sigma_0^2)$.

Therefore, our simulations only depend on four parameters: the problem dimension d, the condition number m, the noise variance σ_ϵ^2 and the standard deviation of the starting point σ_0^2. When compared with the $d = 1$ case, σ_ϵ^2 will have a similar role to σ_b^2, and σ_0^2 to e_0.

5.4.2 Algorithms

In this OLS setting, we will also study proximal and gradient steps as well as SAA.

Step computation We apply the results of Section 5.2.2 to get the closed form of our steps:

$$\operatorname{grad}_n(\beta, \mu) = \left(I_d - \mu \frac{1}{n} X_n' X_n\right) \beta + \mu \frac{1}{n} \epsilon_n' X_n \tag{5.195}$$

$$\operatorname{prox}_n(\beta, \mu) = \left(I_d + \mu \frac{1}{n} X_n' X_n\right)^{-1} \left(\beta + \mu \left(\frac{1}{n} \epsilon_n' X_n\right)\right) \tag{5.196}$$

$$\operatorname{SAA}(n) = \left(\frac{1}{n} X_n' X_n\right)^{-1} \left(\frac{1}{n} \epsilon_n' X_n\right) \tag{5.197}$$

Step Schedule Unfortunately, the optimal schedule setting of $d = 1$ (see proposition 1) does not hold in higher dimension. In order to still be able to mimic this situation in a computationally tractable way, we choose the step schedules as follows:

$$\mu_k = \frac{A}{\frac{A}{B} + (k-1)}, \tag{5.198}$$

with $A > 0$ and $B > 0$ constants to be optimized. In particular, $\mu_1 = B$ and $\mu_k \sim_{k\to\infty} \frac{A}{k}$. We choose these schedules as they match our theoretical results in $d = 1$ and work well in practice [24]. In order to provide a fair comparison between gradient and proximal steps, we will optimize A and B to find the "best" schedule.

We choose a (large) total number of steps K. For any choice of A and B we will estimate the expected error of each step $\mathbb{E}\left[\|\beta_k\|2\right]_{1\leq k\leq K}$ (similar to $e_k^{*\text{grad}}$ and $e_k^{*\text{prox}}$ from the $d = 1$ case). To estimate these errors, we simulate a large number of optimization sample paths up to step K (100 in this work), with random starting point β_0 and random mini-batches as described in Section 5.4.1. We then take the empirical expectation of the error of each step to approximate $\mathbb{E}\left[\|\beta_k\|^2\right]_{1\leq k\leq K}$. To choose the value A and B, we want to minimize the error of the last step: $\min_{A,B} \mathbb{E}\left[\|\beta_K\|^2\right]$. We do this using grid-search over the parameters A and B.

A limitation of this approach (as explored later in Section 5.4.3) is that the "optimal" schedules depend on the choice of K. We compensate by choosing values as high as we can: $K \geq 10,000$. The different algorithms will also be compared sharing the same "randomness", i.e. the same samples for the mini-batches and starting points.

5.4.3 Experiment

For a particular choice of parameters of our OLS setting, we study proximal and gradient steps and discuss the impact and limitations of the choice of optimal step schedule.

Standard Setting In order to study the impact of our four parameters d, m, σ_0 and σ_ϵ, we introduce a "standard" simulation setting with fixed values for each parameter: $d = 16$, $m = 10^2$, $\sigma_0^2 = 1$, and $\sigma_\epsilon^2 = 1$. $m = 100$ is a reasonable choice, it means that their is a typical factor 10 between smallest feature and the largest one. $\sigma_0^2 = 100$

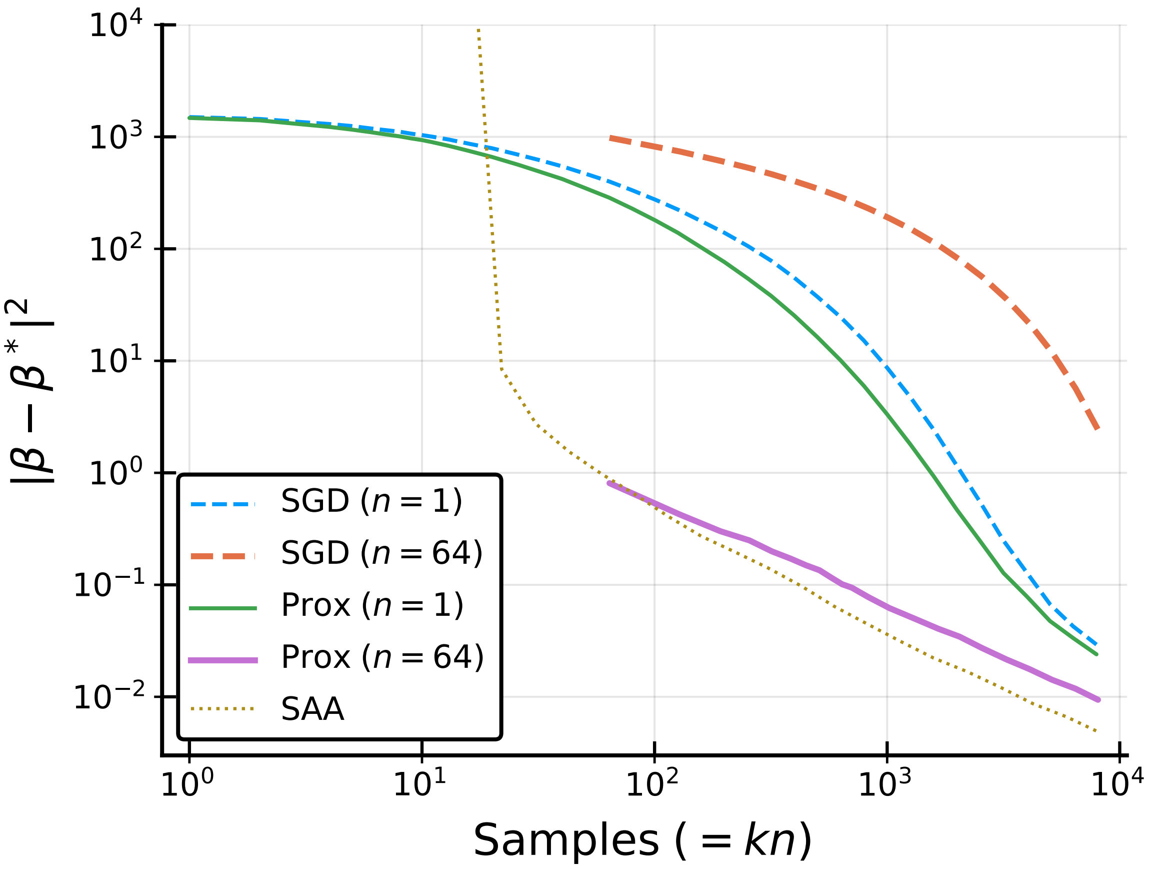


Figure 5-6: Error of the solution path for SAA, stochastic gradient and proximal steps, with batch sizes $n = 1$ and $n = 64$, in the setting introduced in Section 5.4.1. With $d = 16$, $m = 100$, $\sigma_\epsilon^2 = 1$, $\sigma_0^2 = 100$. The x-axis compares the methods by the number of samples processed, which is the number of steps k times the batch size n.

means that starting point are closer to the optimal than in our $d = 1$ experiments (we had $\sigma_0^2 = 10^4$). In section 5.4.4, we will vary each parameter individually from this setting to understand the sensitivity of our results to the parameters.

This standard case is shown in Figure 5-6. We show the error $\mathbb{E}\left[\|\beta_k\|^2\right]$ for each step k, i.e., the distance from the optimal solution. As before we use kn for the x-axis for a sample-by-sample comparison. Note that both axis are on a log scale. Therefore a line with -1 slope indicates sub-linear convergence of the form $\mathbb{E}\left[\|\beta_k\|^2\right] \sim_{k\to+\infty} \frac{A}{k}$ with A a positive constant.

SAA is the dotted yellow curve. It is only defined for $n \geq d$. It has typically high error when n is close to d, and is outperformed by the other algorithms that can use their starting point to reduce the error. But for larger sample size it quickly outperforms the mini-batch algorithms. In our simulations, it always asymptotically converges in $O(\frac{1}{n})$.

Proximal steps are shown with batch sizes of $n = 1$ (thin green line) and $n = 64$ ($= 4d$) (thick purple line). The proximal path with batch n always starts under SAA(n), which can help picture the equivalent paths for other batch sizes. Batch size always accelerates the convergence of proximal steps in this setting.

Gradient steps are shown in dashed lines, for $n = 1$ (thin blue) and $n = 64$ (thick orange). Gradient and proximal steps have the same asymptotic convergence, but the intermediate convergence is in favor of proximal. For example, one need more than 2000 steps of gradient descent with $n = 1$ for one step of proximal with $n = 64$.

We recover our remark from the $d = 1$ case that mini-batch size helps proximal steps but hurts gradient steps.

Optimal Schedule Limitations It is important to also note that the "shape" of the convergence curve of the stochastic methods do not always generalize for a higher number of steps. Indeed, our step schedule is optimized for a precise number of steps K and the whole path would look different for a higher number of steps. For example, it may seem that the convergence of the proximal and gradient steps is not always $O(1/k)$ (whereas it should be), or that the proximal purple line and the green lines in Figure 5-6 will intersect each-other for $k > 10^4$, but this intuition is not correct here.

Indeed, the curves can be very different if we change the parameter K of the optimal schedule. Figure 5-7 shows this with the example of gradient paths. This explains why we cannot generalize the "behavior" of the curves for higher K: the green curve of the figure looks very good until we pass $K = 2048$. All the simulations in this study should be interpreted with this effect in mind. This is also the reason why we are not able to define a truly optimal step schedule independently of K.

Step schedule sensitivity A notable difference between proximal and gradient steps is their stability relative to the choice of step schedule. Figure 5-8 explores the sensitivity of the step schedule. It is known that choosing step sizes that are too high

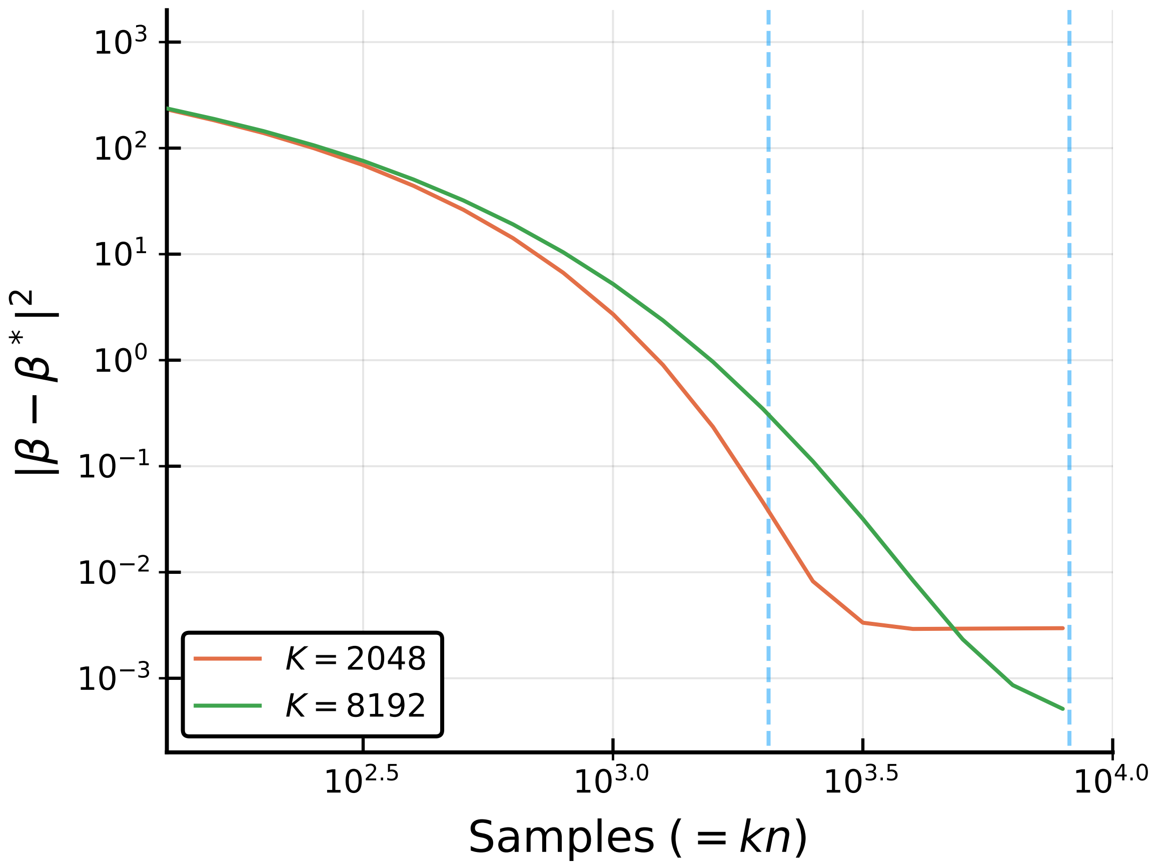


Figure 5-7: Error of optimal schedule for gradient steps ($n = 1$), when choosing $K = 2048$ (orange curve, K corresponds to the first dashed blue line) and $K = 8192$ (green, curve, second dashed blue line). Each schedule minimizes the error for a specific number of steps K, and it can be seen that the paths differ significantly. At $K = 2048$ steps, the orange line is optimal, whereas the green line is optimal at $K = 8192$ steps. I.e., there is not one path that is optimal at each iteration, but the path significantly depends on the value K,

create a transient exponential divergence of gradient steps, whereas it just slightly slows down proximal steps. Choosing step sizes that are too small actually affects the asymptotic convergence rate, which becomes $O(\frac{1}{k^\gamma})$ with $\gamma < 1$ instead of $O(\frac{1}{k})$.

Perhaps more importantly, the tuning process of the step schedule is much easier for proximal steps: when the batch n is not too small, we typically only need to consider step schedules of the form $\frac{A}{K}$, therefore with one less parameter. This is illustrated with the green line on Figure 5-8. Therefore larger batch sizes, on top of making the optimization more efficient, effectively allow us to simplify the step schedules and makes it easier to tune the proximal steps. This is not the case for

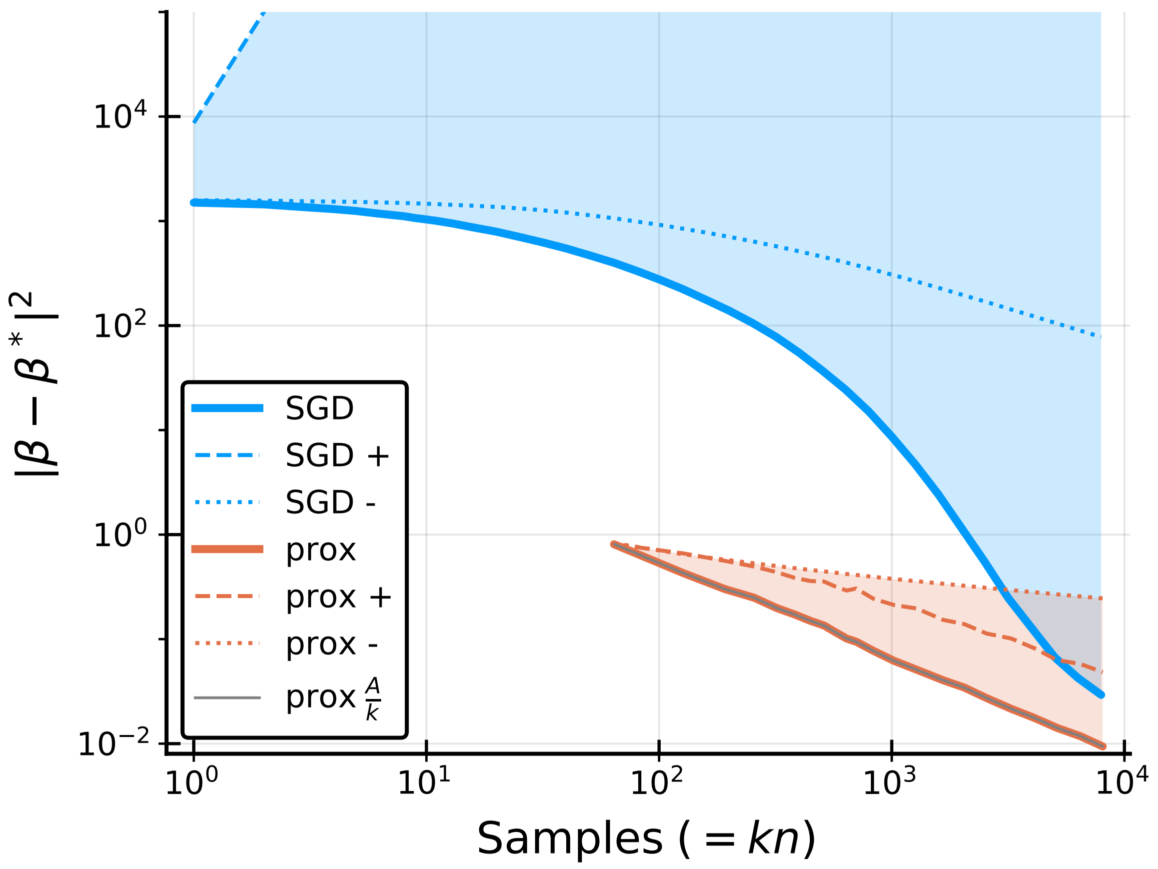
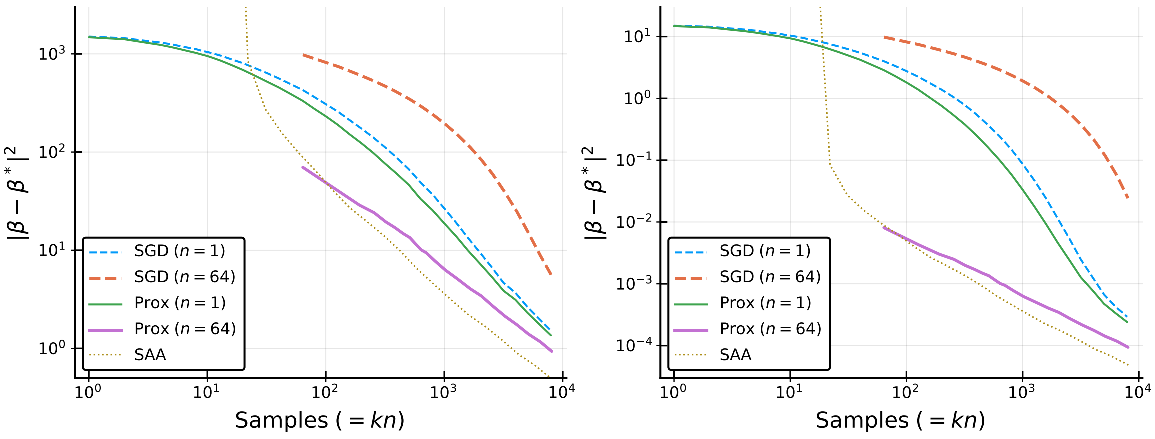


Figure 5-8: We vary the step schedule of the standard setting of Figure 5-6. We modify the step schedules of the best gradient ($n = 1$, blue curves) and proximal ($n = 64$, orange curves) paths to test their sensitivity. We display the original paths (bold), the paths with step sizes multiplied by 10 (dashed) and the path with step sizes divided by 10 (dotted). The thin grey path represents setting $B = +\infty$ in the proximal step schedule and follows the optimal proximal path exactly. This shows that we only need to find one parameter (A) for the proximal step schedule, the parameter B is not required when n is big enough. Proximal steps are much less sensitive to the choice of step schedule than gradient steps.

gradient steps.

5.4.4 Parameter Analysis

Starting from the standard simulation introduced in 5.4.3, we vary the four parameters σ_e^2, m, σ_0^2 and d.



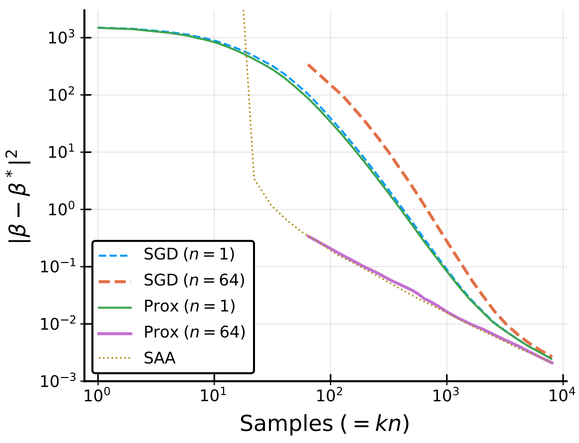
(a) $\sigma_\epsilon^2 = 100$ (b) $\sigma_\epsilon^2 = 0.01$

Figure 5-9: Varying the noise parameter σ_ϵ^2 with respect to the standard setting of Figure 5-6 ($\sigma_\epsilon = 1$).

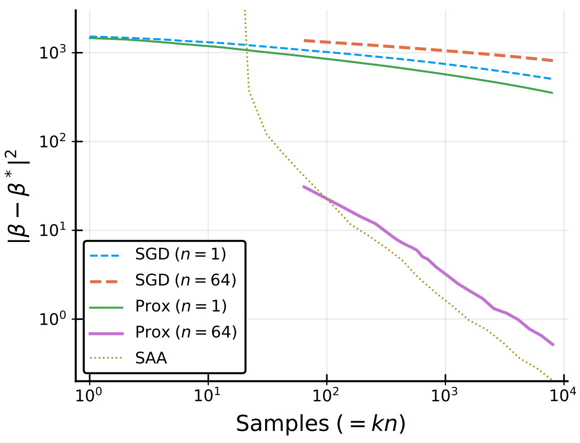
Influence of the noise In Figure 5-9, we vary the noise parameter σ_ϵ^2. High noise (Figure 5-9a) makes all the algorithms more similar, as in the $d = 1$ case when a was deterministic (see Section 5.3.3. In the low noise setting (Figure 5-9b), proximal steps have a clear edge on gradient steps.

Influence of the condition number Figure 5-10 varies the condition number m of the problem. Figure 5-10b corresponds to the limit case of a problem that is perfectly conditioned in expectation. Proximal steps nonetheless still outperform gradient steps, mostly because the sampled cost are not perfectly conditioned even if they are in expectation. Interestingly, as in $d = 1$, all algorithms are equivalent in the asymptotic regime.

In Figure 5-10a, we have a relatively ill-conditioned problem (the smallest features are typically 1000 times smaller that the biggest ones). And similarly to the high



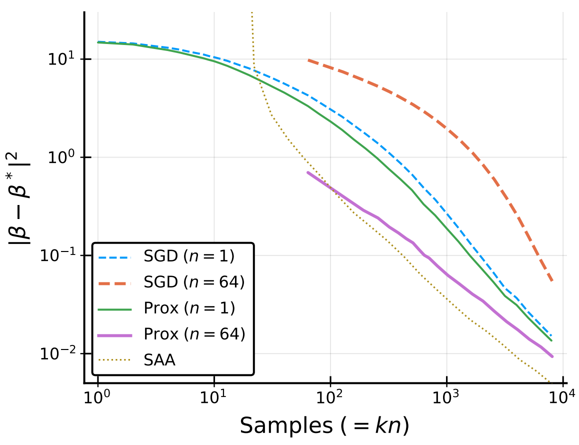
(a) $m = 1$



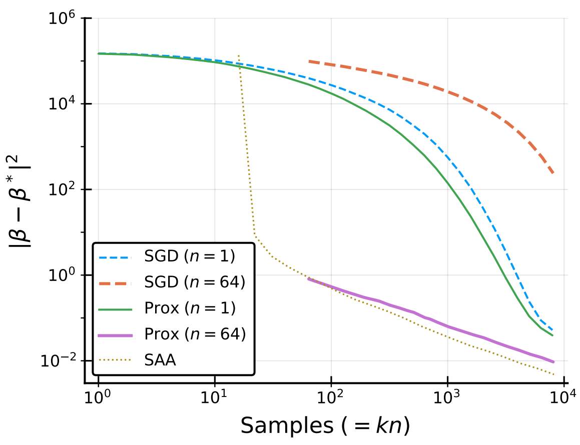
(b) $m = 10^6$

Figure 5-10: Varying the condition number m with respect to the standard setting of Figure 5-6 $(m = 10^2)$.

noise case, it seem that the stochastic algorithms have trouble learning on some of the dimensions, with an extremely slow convergence. Here, batching is the only thing that can make convergence work as we need second-order information to help with the conditioning, and makes the proximal steps extremely competitive.



(a) $\sigma_0^2 = 1$

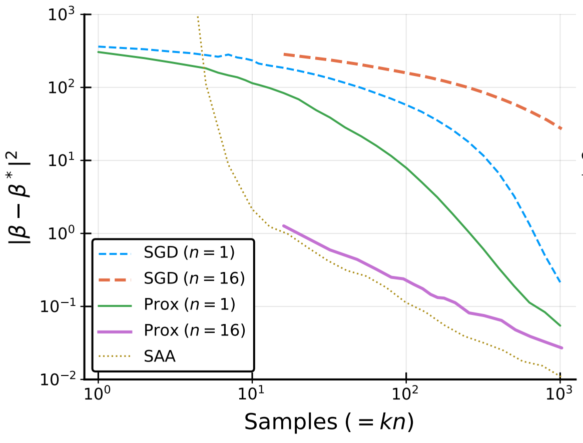


(b) $\sigma_0^2 = 10^4$

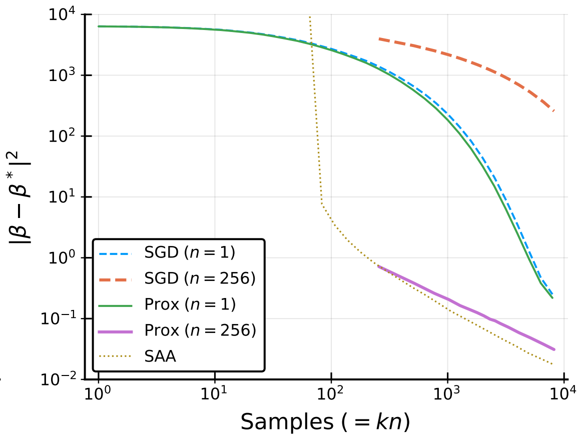
Figure 5-11: Varying the distribution of the starting point $\beta_0 \sim \mathcal{N}(0, \sigma_0^2)$ with respect to the base setting of Figure 5-6 $(\sigma_0^2 = 10^2)$.

Influence of the starting point Figure 5-11 changes the distribution of the starting point: $\beta_0 \sim \mathcal{N}(0, \sigma_0^2)$. Figure 5-11a shows a case where we start closer to the optimal solution. Proximal still outperform gradient steps, but the difference is reduced.

Figure 5-10a has a further-away starting point. And in this situation, proximal methods with a batch of the order of the problem's dimension are not sensitive to the distance of the starting point (it behaves similarly to Figure 5-6), and therefore the error of the first proximal step with batch $n = 64$ is close to the previous case. On the other hand, gradient steps (and proximal with $n = 1$) have a first convergence phase that depends on the distance of the starting point and can be arbitrarily bad. This is related to our study in 1D in Section 5.3.



(a) $d = 4$



(b) $d = 64$

Figure 5-12: Varying the dimension d with respect to the base setting of Figure 5-6 ($d = 16$). Batch size n is also scaled.

Influence of the dimension Figure 5-12 shows that most of the previous comments apply regardless of the dimension. The only caveat is that for proximal to outperform gradient significantly, the mini-batch size should scale with the dimension. Here we choose $n = 4d$. In other words, when the dimension increases, proximal step with $n = 1$ lose their edge over gradient steps, but mini-batching helps recover the benefits of proximal steps.

5.5 Proximal Tractability

So far we compared the algorithms optimization efficiency, i.e. the relationship between the number of samples given to them and the reduction of the expected solution error. This was useful to analyse their behavior in a way that was agnostic of their implementation. But in terms of practical use, the computational time is extremely important, indeed, a fast algorithm that is not the most efficient in terms of optimization, can still process significantly more samples within a given time and therefore lead to lower errors [24].

We will first look at the small dimensions, where computing proximal steps remains surprisingly computationally competitive. In higher dimensions, we introduce an approximate conjugate gradient algorithm to solve the proximal problem and get a hybrid method that has the asymptotic speed of gradient steps, and the robustness and mini-batching behavior of proximal steps.

5.5.1 Low Dimensions

Complexity From Section 5.4.2, we see that computing gradient and proximal steps requires to compute:

$$\mathrm{grad}_n(\beta_{k-1}, \mu_k) = \beta_{k-1} - \left(\frac{\mu_k}{n}\left(X_n'\left(X_n\beta_{k-1} - y_n\right)\right)\right) \tag{5.199}$$

$$\mathrm{prox}_n(\beta_{k-1}, \mu_k) = \beta_{k-1} = A^{-1}b \text{ with } \begin{cases} A & = \frac{1}{\mu_k} I_D + \frac{1}{n} X_n' X_n \\ b & = \frac{1}{\mu_k}\beta_{k-1} + \frac{1}{n} X_n' y_n \end{cases} \tag{5.200}$$

Therefore the gradient step computation is dominated by two matrix-vector computations: $X_n\beta_{k-1}$ and $X_n'\left(X_n\beta_{k-1} - y_n\right)$. These require $O(nd)$ operations.

Proximal steps boil down to solving a linear system. Forming the matrix A requires $O(nd^2)$ for the matrix-matrix multiplication and solving the linear system using LU decomposition is $O(d^3)$, which gives a total complexity of $O((d+n)d^2)$, much higher than the gradient steps.

Implementation Details of implementation are crucial to compare the algorithms with respect to time and make sure that the comparison is fair.

We coded high performance implementations of the algorithms in the computing language *Julia* [20]. Julia uses *OpenBLAS* [135], an optimized library for linear algebra, that can solve linear algebra problems (e.g., matrix multiplication, systems of linear equations . . .) with state-of-the-art algorithms that are specialized for each hardware. We used the fastest version of each algorithm that we could find, and we compared them on a laptop computer, following the steps of (5.199) and (5.200).

When using OpenBLAS, two effects limit the negative effect of proximal steps having a high computational complexity. First, OpenBLAS uses parallelization when advantageous, which favors the use of big mini-batches, which in turns favors proximal steps (see Section 5.4.3). Another effect is that in small dimensions, the asymptotics of the complexity analysis does not hold thanks to many software and hardware optimization, which also limits the computational time of proximal steps.

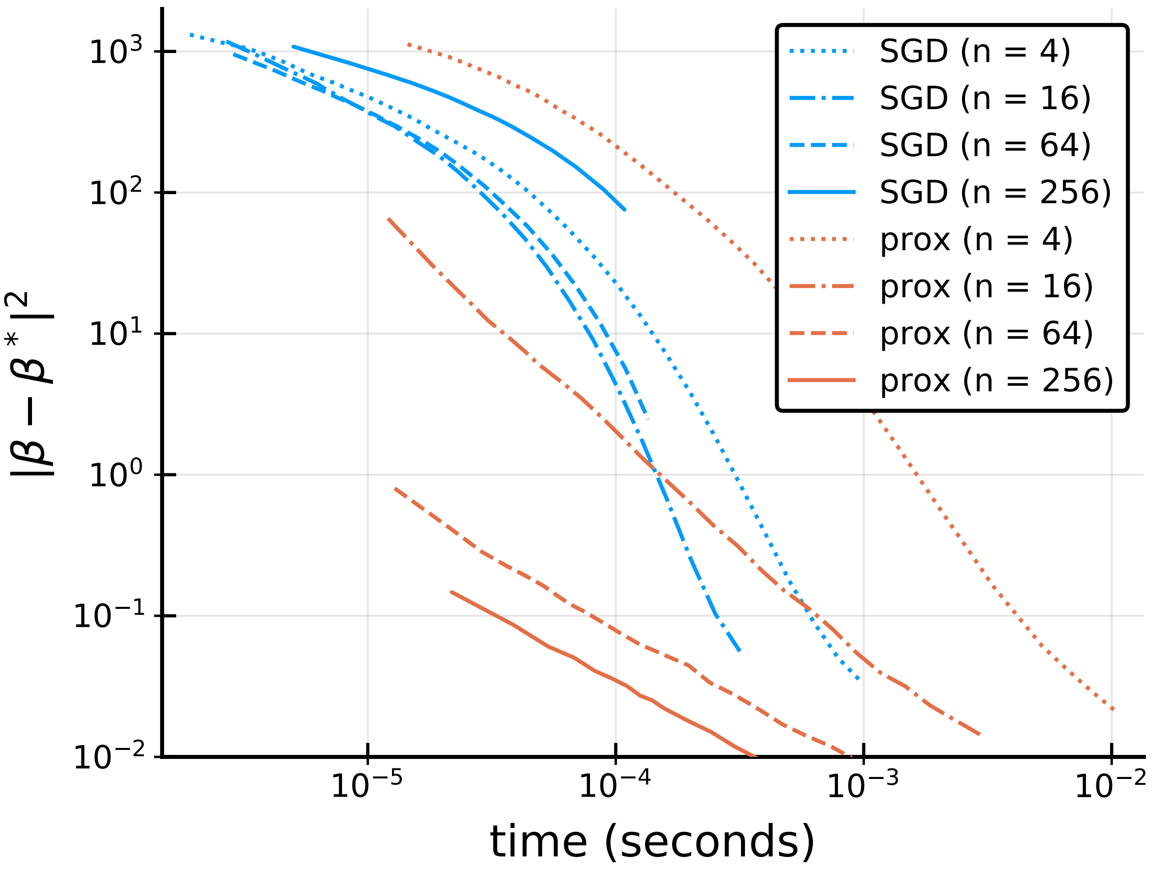


Figure 5-13: Time comparison of the convergence of gradient and proximal steps. We use the standard setting of Figure 5-6 (in particular, $d = 16$). We represent the optimal errors of the gradient (blue) and proximal (orange) algorithms, varying the batch size n. Proximal steps become significantly more efficient when the batch size is increased whereas the tradeoff of gradient steps is more subtle.

Results Using the standard setting from section 5.4.3, we compare gradient steps with respect to computational time, studying the impact of the batch size n. Results are presented in Figure 5-13.

The first thing we notice is that gradient steps computational time is 3 times faster than the proximal steps, which makes them more competitive than the sample-by-sample comparison. Surprisingly, increasing n barely changes the computational time of a step. This is mostly due to parallelization. This makes larger batch size much more attractive than in our previous sample-based comparison (Figure 5-6. But the optimization limitations of large batches compete with the gains: the optimal batch size in this case is $n = 16$.

Nonetheless, perhaps surprisingly, proximal steps computation is efficient enough to be competitive, and their computational time does not seem to be too influenced by the size of n either. For small batch sizes, gradients steps have a clear edge, but proximal steps become competitive with larger batch sizes. And the choice of n is easy: higher is better. Nonetheless, even if it does not appear clearly on this figure, proximal steps will always be asymptotically slower. Indeed, when the step size goes to zero, the proximal steps become equivalent to gradient step, and therefore the computational overhead is not justified. We will see in the next section how we can use this fact to our advantage in higher dimensions.

5.5.2 High Dimensions, the Ap-prox Algorithm

When increasing the dimension, the optimized linear algebra algorithms are less efficient and memory requirements can prevent the use of large batch size n. On the other hand, the conditioning of the problem can worsen, which helps proximal steps. In Figure 5-14, we can compare the gradient (blue) and proximal (orange) errors. These behave similarly to the lower dimension case. But ideally, we would like an algorithm that has the advantages of proximal steps (robustness to the step schedule, easy choice of the parameter n, good transient phase of convergence), while also having the asymptotic speed of gradient steps and their performance with small n.

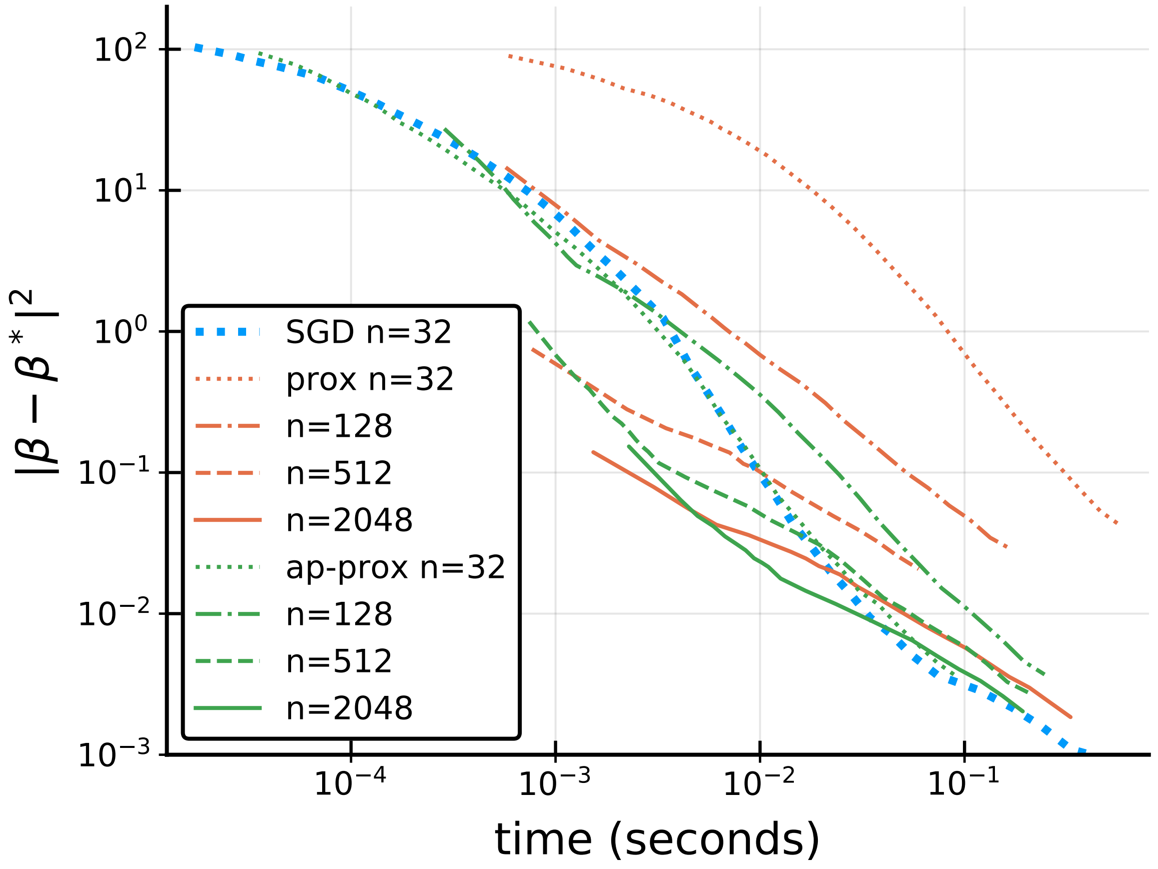


Figure 5-14: Time comparison of the convergence of gradient step and proximal step. We use the standard setting of Figure 5-6, but we increase the problem dimension to $d = 128$. We represent the optimal error of the gradient, with the batch size that gave the best results (blue). We also add the proximal steps (orange) as well as the steps of the ap-prox algorithm for various batch size n. The ap-prox algorithm shares the good characteristics of both gradient steps (speed when n is small, and asymptotic gradient behavior and speed) and proximal steps (strong improvement in the transient phase when n is large).

Proximal Approximation When μ_k becomes small, the condition number of the optimization problem (5.200) improves. In fact, proximal steps will become equivalent to gradient steps as $\mu_k \to 0$. Ideally, we would like the computational time (5.200) to decrease as the step size diminishes, and ultimately to be no slower than the time of a gradient step. This way we could get a good tradeoff between the performance of early proximal steps with large batches, with the speed of gradient steps in the asymptotic regime. But the computational time of (5.200) to optimality typically does not decrease as much with the step-size when using linear algebra openBLAS solvers. We need an algorithm that has this property.

The conjugate gradient algorithm can be used to solve (5.200), and it is guaranteed to converge in d steps (not taking account numerical errors). It also provides the value of the residual at each iteration, and therefore can be stopped early if the residual is small enough. Interestingly, the first conjugate iteration is a line minimization of the proximal cost in the direction of the gradient. Therefore, if we perform only one step, the solving time is of the order of a gradient computation.

The `ap-prox` algorithm We solve approximately the proximal step using the conjugate gradient algorithm with a fixed tolerance $\eta > 0$, and we call this algorithm `ap-prox`. We detail our full implementation in Algorithm 5. The algorithm always does at least one conjugate step, and then keeps doing conjugate steps until the residual is small enough.

Choosing the tolerance η How to choose η? In practice the choice is quite easy and does not depend on the step k. We confirm this theoretically.

Using (5.200), the residual is of a solution β_{k+1} for the proximal step $k+1$ is:

$$R(\beta_{k+1}) = \|r(\beta_{k+1})\|^2 = \|(\frac{n}{\mu_k} I_d + X_n' X_n)\beta_{k+1} - (\frac{n}{\mu_k}\beta_k + X_n' y_n)\|^2 \tag{5.201}$$

where β_{k+1} is the approximate solution to the proximal optimization problem :

$$\min_{\beta_{k+1}} \frac{1}{2}\beta_{k+1} A \beta_{k+1} - \beta_{k+1} b \tag{5.202}$$

Algorithm 5 Conjugate - proximal iteration of the `ap-prox` algorithm. This is an early-stopped conjugate gradient descent to solve $\min_{\beta_{k+1}} \frac{1}{2}\beta_{k+1}A\beta_{k+1} - \beta_{k+1}b$ where $A = \frac{n}{\mu_k}I_d + X_n'X_n$ and $b = \frac{n}{\mu_k}\beta_k + X_n'y_n$. Which corresponds to the formulation of the proximal step of size μ_k starting from β_k. This is equivalent to solving the system $A\beta_{k+1} = b$.

```
Require:
    A data mini-batch $X_n$ and $y_n$.
    A step size $\mu_k$.
    A previous solution $\beta_k$.
    A tolerance value $\eta$.
Ensure:
    Yields $\beta_{k+1}$ such that $R = \|r\|^2 = \|A\beta_{k+1} - b\|^2 \leq \eta$.

    We will update a solution $\beta_{k+1} \in \mathbb{R}^d$ using conjugate gradient steps.
    $r \in \mathbb{R}^d$ will be maintained to be the residual: $r = A\beta_{k+1} - b$
    $p \in \mathbb{R}^d$ will represent the conjugate gradient direction.

 1: $\beta_{k+1} \leftarrow \beta_k$.                    ▷ initialize with the previous solution.
 2: $r \leftarrow X_n'(y_n - X_n\beta_{k+1})$            ▷ value of $r$ for $\beta_{k+1} = \beta_k$
 3: $p \leftarrow r$.                                    ▷ $p$ starts as the gradient value, which corresponds to $r$
 4: $Ap \leftarrow \frac{n}{\mu_k}p + (X_n'(X_np))$      ▷ precompute the multiplication $Ap$
 5: $R \leftarrow \|r\|^2$.
 6: $\alpha = \frac{R}{p'(Ap)}$                          ▷ $\alpha$ is the step size in the conjugate direction
 7: $\beta_{k+1} = \beta_{k+1} + \alpha p$               ▷ first conjugate step
 8: $r = r - \alpha Ap$                                  ▷ update residual
 9: $R_{\text{old}} = R$                                 ▷ save previous value of $R$ for later use
10: $R = \|r\|^2$
11: while $R > \eta$ do                                  ▷ keep doing steps if residual is too big
12:     $p = r + \frac{R}{R_{\text{old}}}p$.             ▷ update the conjugate direction
13:     $Ap = \frac{n}{\mu_k}p + (X_n'(X_np))$
14:     $\alpha = \frac{R}{p'(Ap)}$
15:     $\beta_{k+1} = \beta_{k+1} + \alpha p$
16:     $r = r - \alpha Ap$
17:     $R_{\text{old}} = R$
18:     $R = \|r\|^2$
    return $\beta_{k+1}$
```

with $A = \frac{n}{\mu_k} I_d + X_n' X_n$ and $b = \frac{n}{\mu_k}\beta_k + X_n' y_n$. Let β_{k+1}^* be the optimal solution to this problem, i.e., we have $R(\beta_{k+1}^*) = 0$ from .

Therefore, subtracting $r(\beta_{k+1}^*) = 0$ from (5.201), we obtain:

$$R(\beta_{k+1}) = \|(\frac{n}{\mu_k} I_d + X_n' X_n)(\beta_{k+1} - \beta_{k+1}^*)\|^2 \tag{5.203}$$

We also compute the residual of the starting point $R(\beta_k)$:

$$R(\beta_k) = \|(\frac{n}{\mu_k} I_d + X_n' X_n)\beta_k - (\frac{n}{\mu_k}\beta_k + X_n' y_n)\|^2 \tag{5.204}$$

$$= \|X_n'(X_n\beta_k - y_n)\|^2 \tag{5.205}$$

As we want each step to make progress, we should have $R(\beta_{k+1}) \leq \eta$ small compared to $R(\beta_k)$. Therefore we want to choose η

$$\eta < \|X_n' \left(X_n \beta_k - y_n\right)\|^2 \tag{5.206}$$

$$= \|X_n' \left(X_n \left(\beta_k - \beta^*\right) - X_n' \epsilon_n\right)\|^2 \tag{5.207}$$

$$\leq \|X_n'(X_n \left(\beta_k - \beta^*\right)\|^2 + \|X_n' \epsilon_n\|^2 \tag{5.208}$$

Therefore, η should be chosen to be small in comparison with $\mathbb{E}\left[\|X_n'\epsilon_n\|^2\right]$, and can be reasonably chosen independently from k.

`ap-prox` results Figure 5-14 presents the iterations of the `ap-prox` algorithm (in green). It can be seen that these steps first behave like the proximal steps, and smoothly transition to the behavior of gradient steps, as the step size diminishes. The algorithm performs comparatively or better to both gradient and proximal steps no matter what n is used. It also has the asymptotic behavior of gradient steps.

Figure 5-15 shows the number of conjugate steps that were performed for the iterations of `ap-prox` presented in Figure 5-14. They behave as intended: a large number of conjugate step per batch at first, with more steps when the batch is larger, and then we converge to one step asymptotically.

A limit of the `ap-prox` algorithm is the efficiency of the first step. Indeed, when

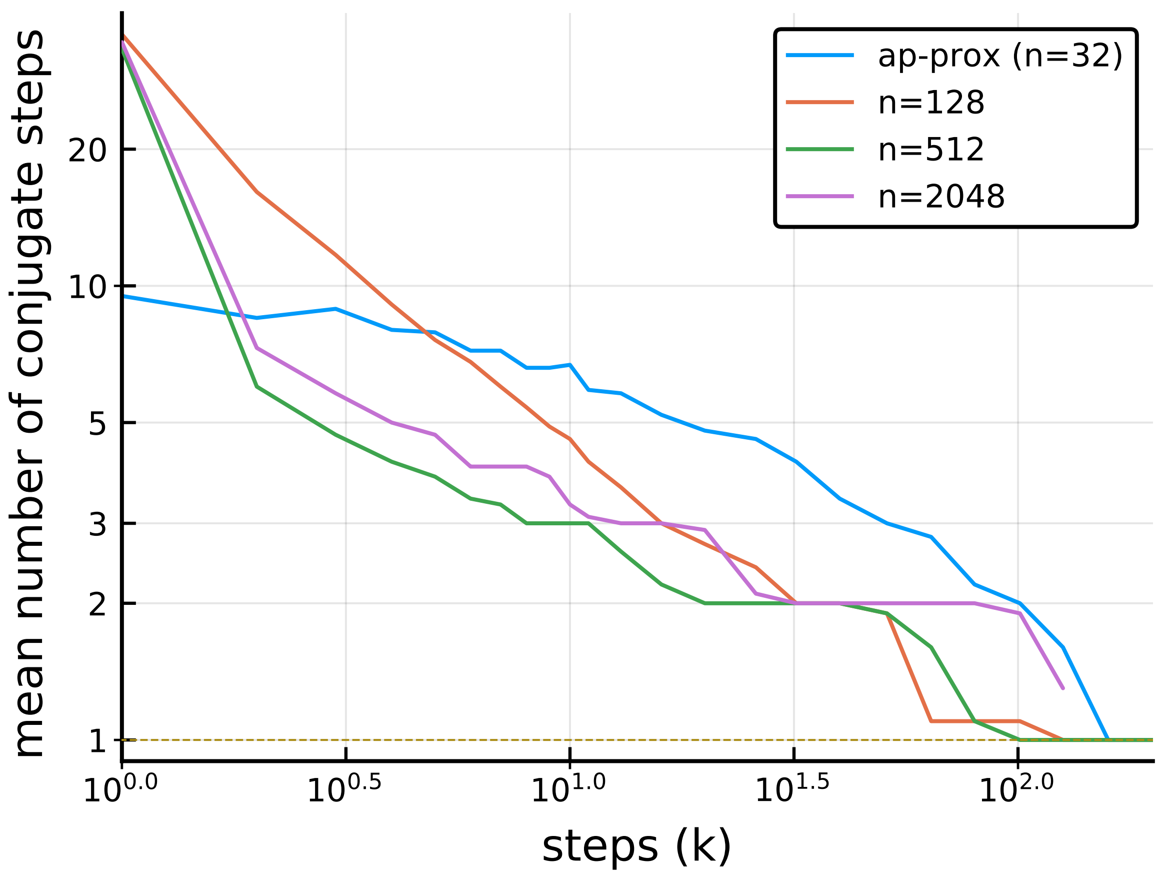


Figure 5-15: We show the average number of conjugate steps that were performed by the `ap-prox` algorithm for each iteration in the experiments of Figure 5-14

the data batch is large enough and the starting point is far from the optimal solution, a large number of conjugate steps can be needed for the first step. This is not efficient as a full proximal step is faster in this case (see Figure 5-14): there are better algorithm than vanilla conjugate gradient descent to solve a linear system. Therefore, we recommend using a full proximal step for the first step when the mini-batch $n > d$. In terms of complexity, using Algorithm 5, computing k conjugate steps takes roughly the same time as $k + 1$ gradient step computations (the first step is more expensive due to the line minimization).

Chapter 6

The Price of Interpretability

6.1 Introduction

Today, predictive models are used in an increasingly high-stakes set of applications, from bail decisions in the criminal justice system [11, 82] to treatment recommendations in personalized medicine [17]. As the stakes have risen, so has the negative impact of incorrect predictions, which could be due to a poorly trained model or to undetected confounding patterns within the data itself [99].

Furthermore, as people start to feel the influence of algorithms in their daily life, many want to understand the reasons for the decisions that affect them the most, from cancer diagnoses to parole decisions to loan applications. Many governments now recognize a "right to explanation" for significant decisions, for instance as part of the European Union's General Data Protection Regulation [64]. However, many state-of-the-art machine learning methods, including random forests and neural networks [29, 60], are black boxes: their complex structure makes it difficult for humans, including domain experts, to understand their predictive behavior.

6.1.1 Interpretable Machine Learning

According to Breiman, machine learning has two objectives: prediction, i.e., determining the value of the target variable for new inputs, and information, i.e., understanding the natural relationship between the input features and the target variable [30]. Studies have shown that many decision makers exhibit an inherent distrust of automated predictive models, even if they are proven to be more accurate than human forecasters [52]. One way to overcome "algorithm aversion" is to give decision makers agency to modify the model's predictions [53]. Another is to provide them with understanding.

Thus, model interpretability, and its tradeoff with predictive accuracy, are of significant interest to the machine learning community [59]. However, a major challenge in this line of research is that the very concept of interpretability is hard to define and even harder to quantify [89]. Many definitions of interpretability have a "know it when you see it" aspect which makes quantitative analysis difficult, though several recent works [54, 63] have introduced new paradigms that could help overcome the ad hoc nature of existing approaches.

Though interpretability remains a loosely-defined concept, there has been extensive research on forgoing complex black box models in favor of more interpretable models. Decision trees [28, 15] are considered interpretable for their discrete structure and graphical visualization, as are close relatives including rule lists [86, 144], decision sets [84], and case-based reasoning [80]. Other approaches include generalized additive models [90], i.e., linear combinations of single-feature models, and score-based methods [133], where integer point values for each feature can be summed up into a final "score".

In the case of linear models, interpretability often comes down to sparsity (small number of nonzero coefficients), a topic of extensive study over the past twenty years [73]. Sparse regression models can be trained using heuristics such as LASSO [131], stagewise regression [130] or least-angle regression [57], or scalable mixed-integer approaches [18, 19].

Many practitioners are hesitant to give up the high accuracy of black box models in the name of interpretability, and prefer to construct *ex post* explanations for a model's predictions. Some approaches create a separate explanation for each prediction in the data-set, e.g. by approximating the nonlinear decision boundary of a neural network with a hyperplane [118]. Others define metrics of feature importance to quantify the effect of each feature in the overall model [60, 46].

Finally, some approaches seek to approximate a large, complex model such as a neural network or a random forest with a simpler one – a decision tree [5], two-level rule list [85], or smaller neural network [31]. Such *global* explanations can help human experts detect systemic biases or confounding variables. However, even if these approximations are almost as accurate as the original model, they may have very different behavior on some inputs and can thus provide a misleading assessment of the model's behavior [63].

6.1.2 Contributions

The contributions of this chapter can be summarized as follows:

- We introduce the notion of an *interpretable path*, which models the sequential reading of a machine learning model by a user as a sequence of models of increasing complexity.

- Motivated by existing approaches, we introduce conditions that *coherent* interpretability metrics (or *losses*) must satisfy, and verify the coherence of typical metrics.

- We show that there exists a natural parametric family of coherent interpretability metrics on the space of interpretable paths that can be extended to the space of models. We demonstrate that the proposed interpretability metric generalizes a number of proxies for interpretability from the literature, such as sparsity in linear models and number of splits for decision trees, and also encompasses other desirable characteristics of interpretability.

- We provide a general optimization formulation to compute the tradeoff between interpretability and predictive accuracy (price of interpretability) and apply it to several different models.

- We explore the implications of our framework in a variety of examples, for which we propose exact mixed-integer formulations and scalable local improvement heuristics to study the price of interpretability on real and synthetic data-sets.

6.2 A Sequential View of Model Construction

6.2.1 Selecting a Model

Most machine learning problems can be viewed through the lens of optimization. Given a set of models $\mathcal{M}$, each model $m \in \mathcal{M}$ is associated with a cost $c(m) \geq 0$, typically derived from data, representing the performance of the model on the task at hand (and potentially including a regularization term). Training a machine learning model means choosing the appropriate m from $\mathcal{M}$ (for example the one that minimizes $c(m)$). To make this perspective more concrete, we will use the following examples throughout the chapter.

Linear models. Given the feature matrix $X \in \mathbb{R}^{n\times d}$ of a data-set of size n with feature space in $\mathbb{R}^d$ and the corresponding vector of labels $y \in \mathbb{R}^n$, a linear model corresponds to a set of linear coefficients $\beta \in \mathbb{R}^d$. In this example, $\mathcal{M} = \mathbb{R}^d$, and the cost $c(\cdot)$ depends on the application: for ordinary least squares (OLS), $c(\beta) = (1/n)\|X\beta - y\|^2$ (mean squared error).

Classification trees (CART). In this case, each model corresponds to a binary decision tree structure [28], so $\mathcal{M}$ is the set of all possible tree structures of any size. Given a tree $t \in \mathcal{M}$ and an input $x \in \mathbb{R}^d$, let $t(x)$ designate the tree's estimate of the corresponding label. Then a typical performance metric $c(t)$ is the number of misclassified points. If we have a data-set with n points $(x_1, \cdots, x_n) \in (\mathbb{R}^d)^n$ associated with classification labels $(y_1, \cdots, y_n) \in \{0, 1\}^n$ then we have $c(t) = \sum_{i=1}^n \mathbf{1}(t(x_i) \neq y_i)$.

Hierarchical clustering. We consider the standard hierarchical clustering problem for a data-set $\mathcal{D}$ of n points in dimension d. Our model space $\mathcal{M}$ is the set of all partitions of the data-set, formally $\cup_{K=1}^{n}\{(A_1,\ldots,A_K) : i \neq j \Rightarrow A_i \cap A_j = \emptyset, \cup_{i=1}^{K} A_i = \mathcal{D}\}$. To evaluate a partition, we can use the within-cluster sum of squares $c(A_1,\ldots,A_K) = \sum_{k=1}^{K} \sum_{x_i \in A_k} \|x_i - \mu_k\|^2$, where $\mu_k = \sum_{x_i \in A_k} x_i/|A_k|$ is the centroid of cluster A_k.

6.2.2 Interpretable Steps

While the performance or accuracy of a model is well-defined, its interpretability is more difficult to grasp and quantify. For our guiding examples, typical proxies include sparsity in linear models [18], a small number of nodes in a classification tree [15], or a small number of partitions.

As we try to rationalize why these models are considered more interpretable, one possible approach is to consider how humans "read" these models. For example, a linear model is typically introduced coefficient by coefficient, a tree is typically read node by node from the root to the leaves, and clusters are typically examined one by one. During this process, we build a model that is more and more complex. In other words, the human process of understanding a model can be viewed as its decomposition into simple building blocks.

We introduce the notion of an *interpretable step* to formalize this sequential process. For every model $m \in \mathcal{M}$, we define a step neighborhood function $\mathcal{S}$ that associates each model m to the set of models $\mathcal{S}(m) \subseteq \mathcal{M}$ that are one step away from m. In other words, m' is one interpretable step away from m if and only if $m' \in \mathcal{S}(m)$. Interpretable steps represent simple model updates that can be chained to build more complex models.

For linear models, one possible interpretable step is adding a feature. That is, given some linear coefficients β, an interpretable step can only change at most one coefficient β_i that was previously set to zero ($\beta_i = 0$). For CART, an interpretable step could be adding a split to an existing tree, i.e. $t' \in \mathcal{S}(t)$ if t' has the same structure and splits as t, except that we can add one new split on one of the leaves of t. For clustering, an interpretable step could be splitting one cluster into two, therefore adding one additional cluster. These three examples are illustrated in Figure 6-1.

Choosing the step neighborhood function $\mathcal{S}$ is a modeling choice and for the examples considered, there may be many other ways to define it. To simplify the analysis, we impose that $\mathcal{S}(m) \neq \emptyset$ for all m (there must always be a feasible next step from any model), which can trivially be satisfied by ensuring $m \in \mathcal{S}(m)$ (an interpretable step can involve no changes to the model).

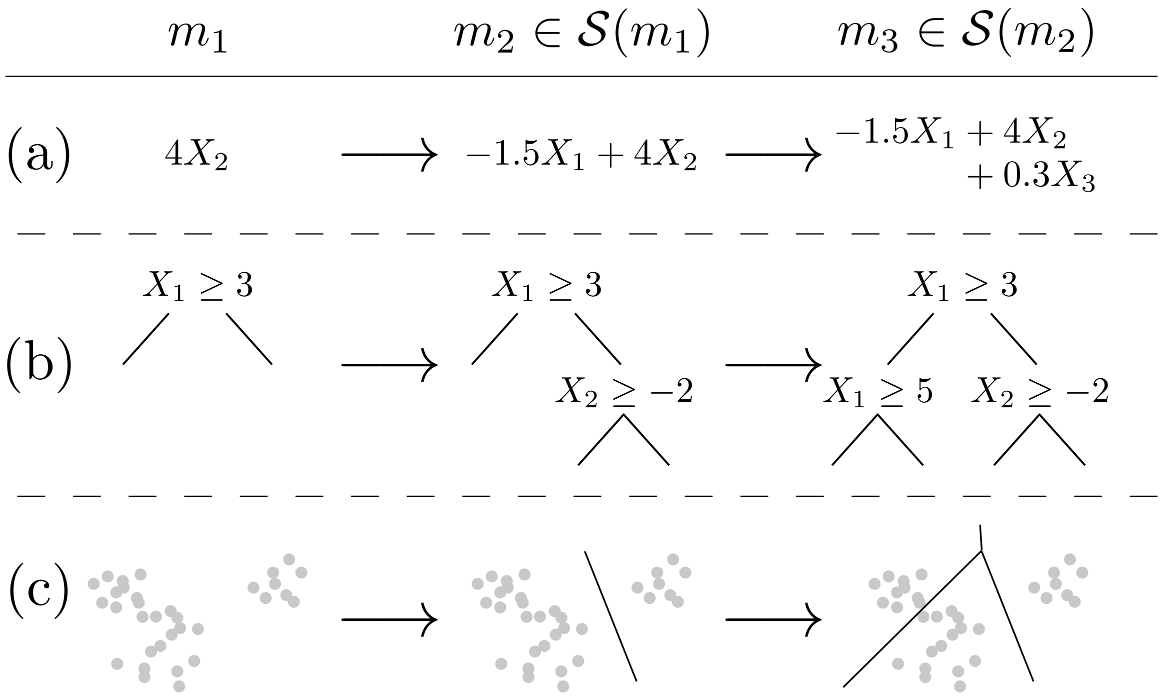

m_1
$m_2 \in \mathcal{S}(m_1)$
$m_3 \in \mathcal{S}(m_2)$
(a)
$4X_2$
$-1.5X_1 + 4X_2$
$-1.5X_1 + 4X_2 + 0.3X_3$
(b)
$X_1 \geq 3$
$X_2 \geq -2$
$X_1 \geq 5$
(c)

Figure 6-1: Illustration of the interpretable path framework with the three examples introduced in Section 6.2.1: (a) is our linear model setting; (b) corresponds to the classification trees (CART); (c) to the clustering setting (in 2 dimensions). For each space of model, we illustrate an example of interpretable path following the choice of steps introduced in Section 6.2.2. Each path has 3 steps: m_1, m_2 and m_3.

Given the choice of an interpretable step $\mathcal{S}$, we can define an *interpretable path* of length K as a sequence of K models $\boldsymbol{m} = (m_1, \cdots, m_K)$ such that $m_k \in \mathcal{S}(m_{k-1})$ for all $1 \leq k \leq K$, i.e., a sequence of interpretable steps starting from a base model m_0. The choice of m_0, the "simplest" model, is usually obvious: in our examples, m_0 could be a linear model with $\beta = \mathbf{0}$, an empty classification tree, or a cluster containing all data points. Given the model space $\mathcal{M}$, we call $\mathcal{P}_K$ the set of all interpretable paths of length K and $\mathcal{P} = \cup_{K=0}^{\infty} \mathcal{P}_K$ the set of all interpretable paths of any length.

Let us consider an example to build intuition about interpretable paths. The `iris` data-set is a small data-set often used to study classification trees. It records the petal length and width and sepal length and width of various iris flowers, along with their species (setosa, versicolor and virginica). For simplicity, we only consider two of the four features (petal length and width) and subsample 50 total points from two of the three classes (versicolor and virginica).

We define an interpretable step as splitting one leaf node into two. Given the `iris` data-set, we consider two classification trees t_{good} and t_{bad}. Both trees have a depth of 2, exactly 3 splits, and a misclassification cost of 2. However, when we consider interpretable paths leading to these two trees, we notice some differences. An interpretable path $\boldsymbol{t}_{\text{bad}}$ leading to t_{bad} is shown in Figure 6-2, and an interpretable path $\boldsymbol{t}_{\text{good}}$ leading to t_{good} is detailed in Figure 6-3.

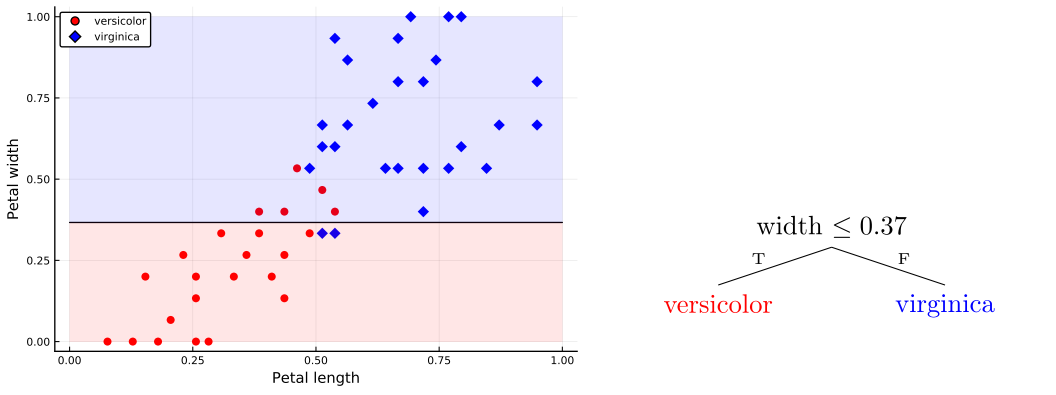
For t_{bad}, the first split results in an intermediate tree with a high classification error: the first split is less “intuitive”. In contrast, the first split of t_{good} gives a fairly accurate intermediate tree when considering only the first split. We will introduce a way to formally pose and answer the problem of which of these two paths is more interpretable.

6.3 The Tradeoffs of Interpretability

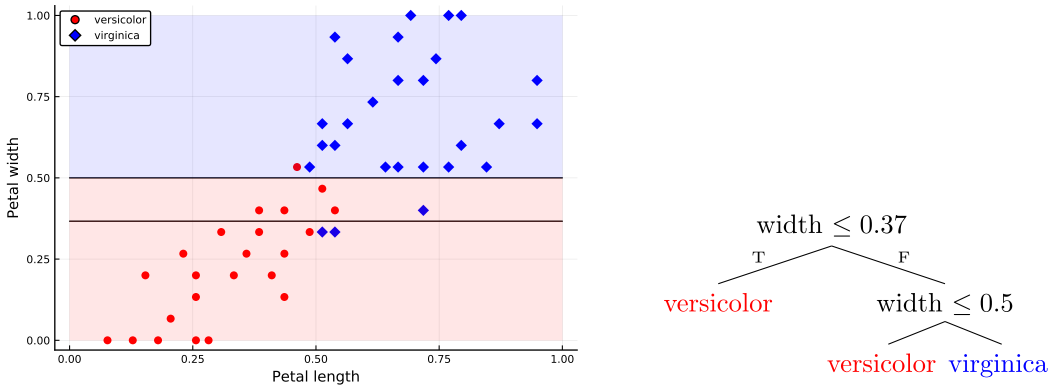
The aim of this section is to define an interpretability loss $\mathcal{L}(\boldsymbol{m})$ for all interpretable paths $\boldsymbol{m} \in \mathcal{P}$ such that a model $\boldsymbol{m}$ is considered “more interpretable” than a model $\boldsymbol{m'}$ if and only if $\mathcal{L}(\boldsymbol{m}) < \mathcal{L}(\boldsymbol{m'})$.

6.3.1 From paths to models

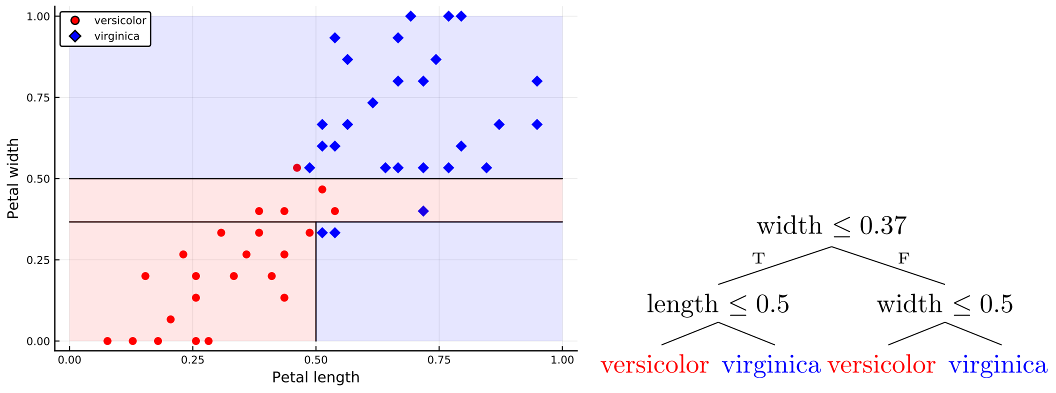
Defining a loss function for the interpretability of a path is important because it can naturally lead to an interpretability loss function on the space of models. More precisely, given a path interpretability loss function $\mathcal{L}$, we can assume that more



(a) 7 misclassified points

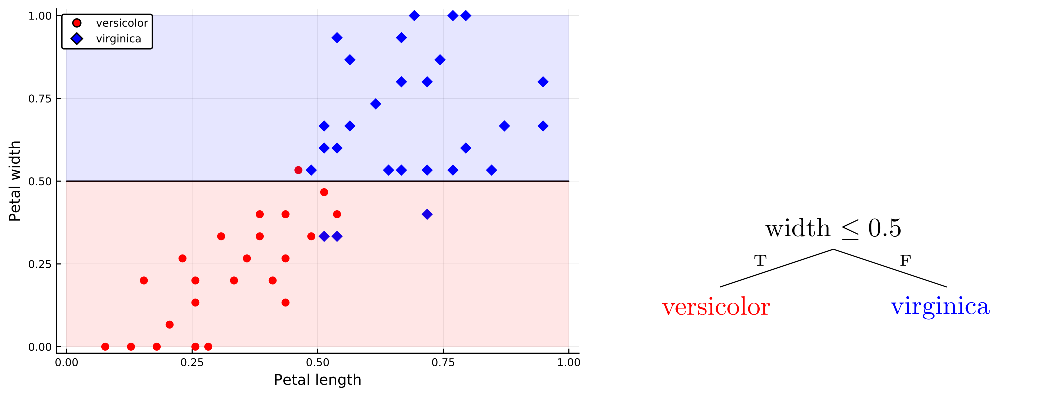


(b) 4 misclassified points

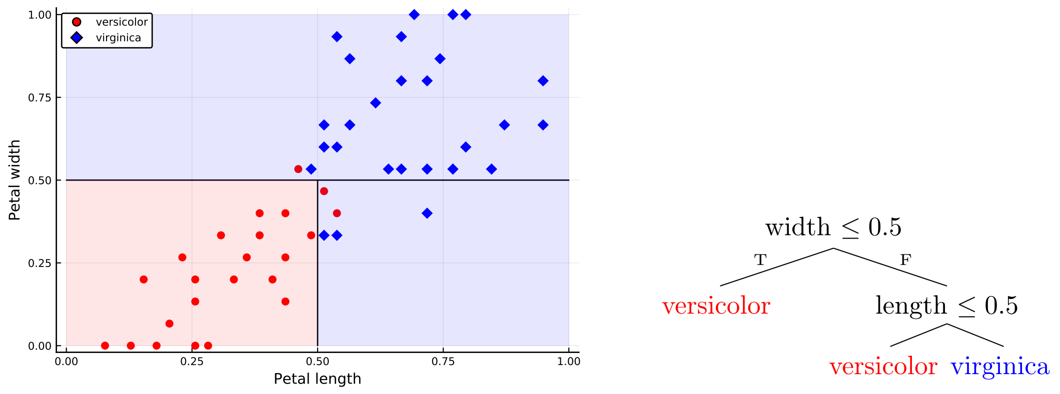


(c) 2 misclassified points

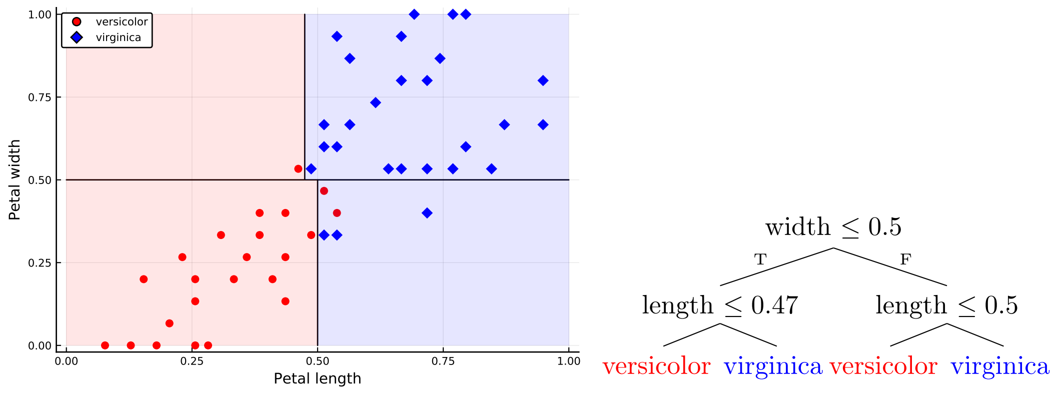
Figure 6-2: Visualization of an interpretable path leading to t_{bad} (shown on the right).



(a) 4 misclassified points



(b) 3 misclassified points



(c) 2 misclassified points

Figure 6-3: Visualization of an interpretable path leading to t_{good} (shown on the right).

interpretable paths lead to more interpretable models, and obtain

$$\mathcal{L}(m) = \begin{cases} \infty, & \text{if } \mathcal{P}(m) = \emptyset, \\ \min_{\boldsymbol{m} \in \mathcal{P}(m)} \mathcal{L}(\boldsymbol{m}), & \text{otherwise}, \end{cases} \tag{6.1}$$

where $\mathcal{P}_K(m) = \{\boldsymbol{m} \in \mathcal{P}_K, m_K = m\}$ designates the set of interpretable paths of length K leading to m, and $\mathcal{P}(m) = \cup_{K=0}^{\infty} \mathcal{P}_K(m)$ designates the set of all interpretable paths leading to m. In other words, the interpretability loss of a model m is the interpretability loss of the most interpretable path among all the paths that lead to m.

As an example, consider the following path interpretability loss function, which we call path complexity and define as $\mathcal{L}_{\text{complexity}}(\boldsymbol{m}) = |\boldsymbol{m}|$ (number of steps in the path). From (6.1) we can then define the interpretability loss of a given model m as

$$\mathcal{L}_{\text{complexity}}(m) = \min_{\boldsymbol{m} \in \mathcal{P}(m)} |\boldsymbol{m}|,$$

which corresponds to the minimal number of interpretable steps required to reach m.

In the context of the examples from Section 6.2, the function $\mathcal{L}_{\text{complexity}}$ recovers typical interpretability proxies. For linear models, $\mathcal{L}_{\text{complexity}}(\beta) = \|\beta\|_0$ is the sparsity of the model (number of non-zero coefficients). For a classification tree t, $\mathcal{L}_{\text{complexity}}(t)$ is the number of splits. In a clustering context, $\mathcal{L}_{\text{complexity}}(A_1, \cdots, A_K) = K$ is just the number of clusters. It is therefore reasonable to refer to this candidate loss function as the model complexity.

A fundamental problem of interpretable machine learning is finding the highest-performing model at a given level of interpretability [30]. Defining an interpretability loss function $\mathcal{L}$ on the space of models $\mathcal{M}$ is important because it allows us to formulate this problem generally as follows:

$$\min_{m \in \mathcal{M}} c(m) \quad \text{s.t.} \quad \mathcal{L}(m) \leq \ell, \tag{6.2}$$

where ℓ is the desired level of interpretability. Problem (6.2) produces models on the Pareto front of accuracy and interpretability. If we compute this Pareto front by solving problem (6.2) for any ℓ, then we can mathematically characterize the *price* of interpretability of a particular class of models on a particular data-set, making the choice of a final model easier.

In the case of model complexity $\mathcal{L}_{\text{complexity}}$, for $\ell = K$ problem (6.2) can be written as

$$\min_{\boldsymbol{m} \in \mathcal{P}_K} \quad c(m_K). \tag{6.3}$$

Problem (6.3) generalizes existing problems in interpretable machine learning: best subset selection (L_0-constrained sparse regression) for linear models [18], finding the best classification tree of a given size [15], or finding the K best possible clusters in a hierarchical clustering setting.

Thus, the framework of interpretable paths naturally gives rise to a general definition of model complexity via the loss function $\mathcal{L}_{\text{complexity}}$, and our model generalizes many existing approaches. By some counts, however, model complexity remains an incomplete interpretability loss. For instance, it does not differentiate between the trees t_{good} and t_{bad}: both models have a complexity of 3 because they can be reached in three steps. More generally, $\mathcal{L}_{\text{complexity}}$ does not differentiate between paths of the same length, or between models that can be reached by paths of the same length.

6.3.2 Incrementality

In the decision tree example from Figures 6-2 and 6-3, we observed that the intermediate trees leading to t_{good} were more accurate than the intermediate trees leading to t_{bad}. Evaluating the costs $c(m_k)$ of intermediate models along a path $\boldsymbol{m}$ may provide clues as to the interpretability of the final model.

Consider the following toy example, where the goal is to estimate the age of a child given their height and weight. We create a synthetic data-set with normalized features X_{Height} and X_{Weight} and centered labels y_{Age}. The features X_{Height} and X_{Weight} have correlation $\rho = 0.9$ and are both positively correlated with the objective. Solving the OLS problem yields:

$$y_{\text{Age}} = 2.12 \cdot X_{\text{Height}} - 0.94 \cdot X_{\text{Weight}} + \varepsilon, \tag{6.4}$$

with ε the error term. The mean squared error (MSE) of this optimal model β^* is $c(\beta^*) = \mathbb{E}\left[\varepsilon^2\right] = 0.25$.

As in Section 6.2, we define an interpretable step to be the addition of a feature to the regression model, keeping all other coefficients constant. In this case, two different interpretable paths $\boldsymbol{m}, \boldsymbol{m}' \in \mathcal{P}(\beta^*)$ lead to model (6.4), depending on which feature is added first:

$$\begin{cases} m_1 : \; 2.12 \cdot X_{\text{Height}} \\ m_2 : \; 2.12 \cdot X_{\text{Height}} - 0.94 \cdot X_{\text{Weight}}, \end{cases} \quad \text{or} \quad \begin{cases} m_1' : \; -0.94 \cdot X_{\text{Weight}} \\ m_2' : \; 2.12 \cdot X_{\text{Height}} - 0.94 \cdot X_{\text{Weight}}. \end{cases} \tag{6.5}$$

Which of $\boldsymbol{m}$ or $\boldsymbol{m}'$ is more interpretable? Both verify $c(m_2) = c(m_2') = 0.25$, but $c(m_1) = 1.13 < c(m_1') = 4.76$. Indeed, m_1' is a particularly inaccurate model, as weight is actually positively correlated with age. However, when using the complexity loss $\mathcal{L}_{\text{complexity}}$, both of these paths are considered equally interpretable.

As discussed in Section 6.2, an interpretable path $\boldsymbol{m}$ leading to model m can be viewed as modeling how we can build the final model in a sequence of easily understandable steps. The costs of intermediate models should play a role in quantifying the interpretability loss of a path; higher costs should be penalized.

One way to ensure that every step of an interpretable path adds value is a greedy approach, where the next model at each step is chosen by minimizing the cost $c(\cdot)$:

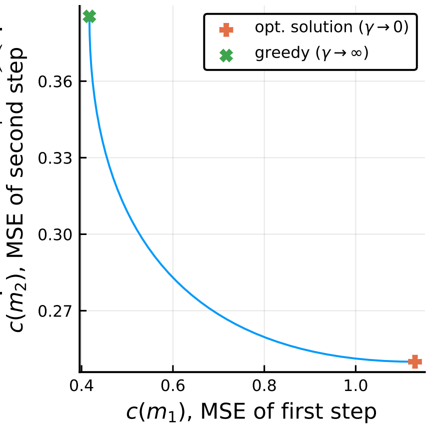
$$m_{k+1}^{\text{greedy}} \in \arg\min \left\{ c(m),\, m \in \mathcal{S}(m_k^{\text{greedy}}) \right\} \quad \forall k \geq 1. \tag{6.6}$$

In our toy example with two steps, this means selecting the best possible m_1^{greedy}, and then the best possible m_2^{greedy} given m_1^{greedy}, as in stagewise regression [130]. This will not yield the best possible model achievable in two steps as in (6.3), but the first step is guaranteed to be the best one possible.

Notice that $c(m_1^{\text{greedy}}) = 0.42 < 1.13 = c(m_1)$, but $c(m_2^{\text{greedy}}) = 0.39 > 0.25 = c(m_2)$. In other words, the first greedy model is much better, but the improvement comes at the expense of the second step. Deciding which of the two paths $\boldsymbol{m}$ and $\boldsymbol{m}^{\text{greedy}}$ is more interpretable is a hard question. It highlights the tradeoff between the desirable incrementality of the greedy approach and the cost of the final model.

	m_1 (first step)			m_2 (
	Coef 1	Coef 2	$c(m_1)$	Coef 1
Opt., coef 1 first	2.12	0	1.13	2.12
Opt., coef 2 first	0	-0.94	4.76	2.12
Greedy	1.28	0	**0.42**	1.28
Tradeoff	1.83	0	0.72	1.83

(a) Interpretable path examples. Rows 1 and 2 present paths $\boldsymbol{m}$ and $\boldsymbol{m'}$, where we add the coefficients of the optimal solution in two possible orders. Row 3 presents the greedy approach $\boldsymbol{m}^{\text{greedy}}$ and row 4 a solution somewhere in between. Optimal MSEs for each step are indicated in bold.



(b) Tradeoff between the cost of the first and second models of the interpretable path.

Figure 6-4: Interpretability tradeoffs for toy problem (6.4).

In Figure 6-4a we present a comparison of the interpretable paths proposed thus far for our toy problem. In addition, the last row presents another interpretable path that is neither greedy nor optimal, accepting a slight sub-optimality in $c(m_2)$ for a significant improvement in $c(m_1)$. This last model is just one among a continuum of paths between $\boldsymbol{m}$ and $\boldsymbol{m}^{\text{greedy}}$. Figure 6-4b shows the entire Pareto front between $c(m_1)$ and $c(m_2)$. The convexity of the Pareto front means we can significantly improve the cost of the first step at small cost to the second (or vice versa).

6.4 A Coherent Interpretability Loss

In the previous section, we developed intuition regarding the interpretability of different paths. We now formalize this intuition to define a suitable interpretability loss.

6.4.1 Coherent Path Interpretability Losses

According to the loss $\mathcal{L}_{\text{complexity}}$ defined in Section 6.3.1, which generalizes many notions of interpretability from the literature, a path is more interpretable if it is shorter. In Section 6.3.2, we saw that the cost of individual models along the path matters as well.

Sometimes, comparing the costs of intermediate models between two paths is easy because the cost of each step along one path is at least as good as the cost of the corresponding step in the other path. For example, it is reasonable to consider $\boldsymbol{m}$ more interpretable than $\boldsymbol{m'}$ because $c(m_1) < c(m'_1)$ and $c(m_2) = c(m'_2)$. In contrast, comparing the interpretability of $\boldsymbol{m}$ and $\boldsymbol{m}^{\text{greedy}}$ is more difficult and user-specific, because $c(m_1) > c(m_1^{\text{greedy}})$, but $c(m_2) < c(m_2^{\text{greedy}})$.

We now formalize this intuition into desirable properties of interpretability loss functions. We first introduce the notion of a cost sequence, which provides a concise way to refer to the costs of all the steps in an interpretable path. We then define axioms for *coherent* interpretability losses.

Definition 3 (Cost sequence)**.** Given an interpretable path of length K, denoted as $\boldsymbol{m} \in \mathcal{P}_K$, the cost sequence $\boldsymbol{c}(\boldsymbol{m}) \in \mathbb{R}^{\mathbb{N}}$ is the infinite sequence $(c_1, c_2, \cdots)$ such that:

$$c_k = \begin{cases} c(m_k), & \text{if } k \leq K, \\ 0, & \text{otherwise.} \end{cases}$$

Definition 4 (Coherent Interpretability Loss)**.** A path interpretability loss $\mathcal{L}$ is *coherent* if the following conditions hold for any two interpretable paths $\boldsymbol{m}, \boldsymbol{m'} \in \mathcal{P}$ with respective cost sequences $\boldsymbol{c}$ and $\boldsymbol{c'}$.

(a) If $\boldsymbol{c} = \boldsymbol{c'}$, then $\mathcal{L}(\boldsymbol{m}) = \mathcal{L}(\boldsymbol{m'})$.

(b) (Weak Pareto dominance) If $c_k \leq c'_k\ \forall k$, then $\mathcal{L}(\boldsymbol{m}) \leq \mathcal{L}(\boldsymbol{m'})$.

Condition (a) means that the interpretability of a path depends only on the sequence of costs along that path. Condition (b) formalizes the intuition described before, that paths with fewer steps or better steps are more interpretable. For instance, if we improve the cost of one step of a path while leaving all other steps unchanged, we can only make the path more interpretable. Under any coherent interpretability loss $\mathcal{L}$ in toy example 6.4, $\boldsymbol{m}$ is more interpretable than $\boldsymbol{m'}$, but $\boldsymbol{m}$ may be more or less interpretable than $\boldsymbol{m}^{\text{greedy}}$ depending on the specific choice of coherent interpretability loss.

In addition, consider a path $\boldsymbol{m} \in \mathcal{P}_K$ and remove its last step to obtain a new path $\boldsymbol{m'} \in \mathcal{P}_{K-1}$. This is equivalent to setting the K-th element of the cost sequence $\boldsymbol{c}(\boldsymbol{m})$ to zero. Since $c(\cdot) \geq 0$, we have that $\boldsymbol{c}(\boldsymbol{m}) \leq \boldsymbol{c}(\boldsymbol{m})$, which implies $\mathcal{L}(\boldsymbol{m'}) \leq \mathcal{L}(\boldsymbol{m})$. In other words, under a coherent interpretability loss, removing a step from an interpretable path can only make the path more interpretable.

Remark 4. The path complexity $\mathcal{L}_{\text{complexity}}(\boldsymbol{m}) = |\boldsymbol{m}|$ is a coherent path interpretability loss.

Proof. If $\boldsymbol{m}$ and $\boldsymbol{m'}$ verify $\boldsymbol{c}(\boldsymbol{m}) = \boldsymbol{c}(\boldsymbol{m'})$, then trivially the two cost sequences become zero after the same number of steps, so $\mathcal{L}_{\text{complexity}}(\boldsymbol{m}) = \mathcal{L}_{\text{complexity}}(\boldsymbol{m'})$. If $\boldsymbol{c}(\boldsymbol{m}) \leq \boldsymbol{c}(\boldsymbol{m'})$ and $\boldsymbol{c}(\boldsymbol{m'})$ becomes zero after exactly K steps, then $\boldsymbol{c}(\boldsymbol{m})$ must become zero after at most K steps, so $\mathcal{L}_{\text{complexity}}(\boldsymbol{m}) \leq \mathcal{L}_{\text{complexity}}(\boldsymbol{m'})$. □

6.4.2 A Coherent Model Interpretability Loss

Axiom (b) of Definition 4 states that a path that dominates another path in terms of the costs of each step must be at least as interpretable. This notion of weak Pareto dominance suggests a natural path interpretability loss:

$$\mathcal{L}_\alpha(\boldsymbol{m}) = \sum_{k=1}^{|\boldsymbol{m}|} \alpha_k c(m_k).$$

In other words, the interpretability loss $\mathcal{L}_\gamma$ of a path $\boldsymbol{m}$ is the weighted sum of the costs of all steps in the path. This loss function is trivially coherent and extremely general. It is more accurate to specified by the infinite sequence of parameters $\boldsymbol{\alpha}$, which specify the relative importance of the accuracy of each step in the model for the particular application at hand.

Defining a family of interpretability losses with infinitely many parameters allows for significant modeling flexibility, but it is also cumbersome and overly general. We therefore propose to select $\alpha_k = \gamma^k$ for all k, replacing the infinite sequence of parameters $(\alpha_1, \alpha_2, \ldots)$ with a single parameter $\gamma > 0$. In this case, following 6.1, we propose the following coherent interpretability loss function on the space of models.

Definition 5 (Model interpretability)**.** Given a model $m \in \mathcal{M}$, its interpretability loss $\mathcal{L}_\gamma(m)$ is given by

$$\mathcal{L}_\gamma(m) = \begin{cases} \infty, & \text{if } \mathcal{P}(m) = \emptyset, \\ \min_{\boldsymbol{m} \in \mathcal{P}(m)} \mathcal{L}_\gamma(\boldsymbol{m}) = \sum_{k=1}^{|\boldsymbol{m}|} \gamma^k c(m_k), & \text{otherwise.} \end{cases} \tag{6.7}$$

By definition, $\mathcal{L}_\gamma$ is a coherent interpretability loss, which favors more incremental models or models with a low complexity. The parameter γ captures the tradeoff between these two aspects of interpretability. Theorem 3 shows that with a particular choice of $\boldsymbol{\gamma}$ one can recover the notion of model complexity introduced in Section 6.3.1, or models that can be built in a greedy way.

Theorem 3 (Consistency of interpretability measure)**.** *Consider the specific weights $\boldsymbol{\gamma} = (\gamma^k)_{k \geq 1}$ for a scalar $0 < \gamma < \infty$. Additionally assume that the cost $c(\cdot)$ is bounded and nonnegative. Let $\mathcal{L}_\gamma$ be the interpretability loss associated with the weights $\boldsymbol{\gamma}$.*

(a) Let $m^+,\ m^- \in \mathcal{M}$ with $\mathcal{L}_{\text{complexity}}(m^+) < \mathcal{L}_{\text{complexity}}(m^-)$ (i.e., m^+ requires less interpretable steps than m^-), or $\mathcal{L}_{\text{complexity}}(m^+) = \mathcal{L}_{\text{complexity}}(m^-)$ and $c(m^+) < c(m^-)$.

$$\lim_{\gamma \to \infty} \mathcal{L}_{\boldsymbol{\gamma}}(m^-) - \mathcal{L}_{\boldsymbol{\gamma}}(m^+) = +\infty. \tag{6.8}$$

.

(b) Given $\boldsymbol{m}^+, \boldsymbol{m}^- \in \mathcal{P}$, if $\boldsymbol{c}(\boldsymbol{m}^+) \preceq \boldsymbol{c}(\boldsymbol{m}^-)$, where $\preceq$ represents the lexicographic order on $\mathbb{R}^{\mathbb{N}}$, then

$$\lim_{\gamma \to 0} \mathcal{L}_{\boldsymbol{\gamma}}(\boldsymbol{m}^-) - \mathcal{L}_{\boldsymbol{\gamma}}(\boldsymbol{m}^+) \geq 0. \tag{6.9}$$

Consequently, given models $m^+, m^- \in \mathcal{M}$, if there is $\boldsymbol{m}^+ \in \mathcal{P}(m^+)$ such that $\boldsymbol{c}(\boldsymbol{m}^+) \preceq \boldsymbol{c}(\boldsymbol{m}^-)$ for all $\boldsymbol{m}^- \in \mathcal{P}(m^-)$, then

$$\lim_{\gamma \to 0} \mathcal{L}_{\gamma}(m^-) - \mathcal{L}_{\gamma}(m^+) \geq 0. \tag{6.10}$$

Intuitively, in the limit $\gamma \to +\infty$, (a) states that the most interpretable models are the ones with minimal complexity, or minimal costs if their complexity is the same. (b) states that in the limit $\gamma \to 0$ the most interpretable models are the ones that can be constructed with greedy steps. Definition 5 therefore generalizes existing approaches and provides a good framework to model the tradeoffs of interpretability.

6.5 Interpretability Losses in Practice

Defining an interpretability loss brings a new perspective to the literature on interpretability in machine learning. In this section, we discuss how this perspective can be useful in practice. For the sake of generality, in the early part of this section we work with the more general interpretability loss $\mathcal{L}_{\alpha}(\cdot)$.

6.5.1 The Price of Interpretability

Given the metric of interpretability defined above, we can quantitatively discuss the price of interpretability, i.e., the tradeoff between a model's interpretability loss $\mathcal{L}(m)$ and its cost $c(m)$. In other words, we want to compute models that are Pareto optimal with respect to $c(\cdot)$ and $\mathcal{L}_{\alpha}(\cdot)$, as in (6.2).

Computing these Pareto-optimal solutions can be challenging, as our definition of model interpretability requires to optimize over paths of any length. Fortunately, the only optimization problem we need to be able to solve is to find the most interpretable path of a fixed length K, i.e.,

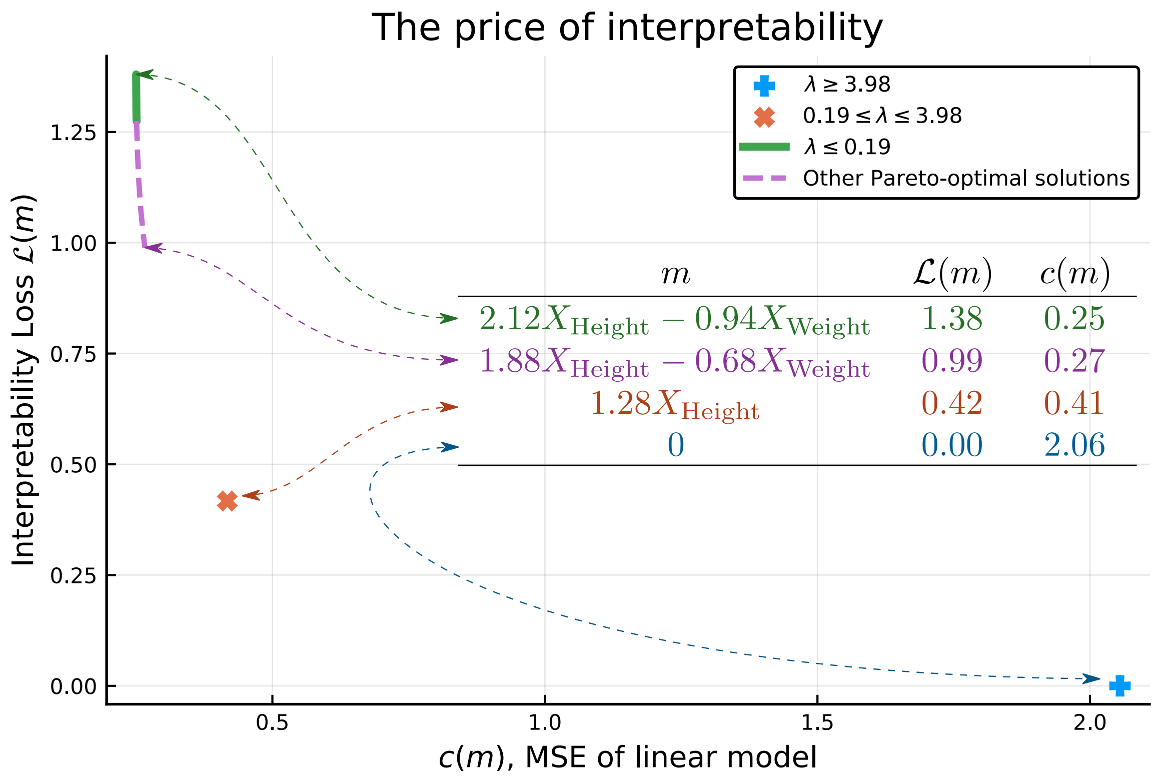
$$\min_{\boldsymbol{m} \in \mathcal{P}_K} \mathcal{L}_{\alpha}(\boldsymbol{m}) = \sum_{k=1}^{K} \alpha_k c(m_k) \tag{6.11}$$

Indeed, the following proposition shows that we can compute Pareto-optimal solutions by solving a sequence of optimization problems (6.11) for various K and $\boldsymbol{\alpha}$.

Proposition 4 (Price of interpretability)**.** *Pareto-optimal models that minimize the interpretability loss $\mathcal{L}_\alpha$ and the cost $c(\cdot)$ can be computed by solving the following optimization problem:*

$$\min_{K\geq 0}\left(\min_{\mathbf{m}\in\mathcal{P}_\mathbf{K}} c(m_K)+\lambda\sum_{k=1}^{K}\alpha_k c(m_k)\right), \tag{6.12}$$

where $\lambda\in\mathcal{R}$ is a tradeoff parameter between cost and interpretability.



m	$\mathcal{L}(m)$	$c(m)$
$2.12X_{\text{Height}} - 0.94X_{\text{Weight}}$	1.38	0.25
$1.88X_{\text{Height}} - 0.68X_{\text{Weight}}$	0.99	0.27
$1.28X_{\text{Height}}$	0.42	0.41
0	0.00	2.06

Figure 6-5: Pareto front between interpretability loss $\mathcal{L}(m) = \mathcal{L}_\gamma(m)$ (with $\gamma = 1$) and cost $c(m)$ on the toy OLS problem (6.4), computed by varying λ in (6.12). The dashed line represents Pareto-optimal solutions that cannot be computed by this weighted-sum method. Note that the front is discontinuous, and that there is an infinite number of Pareto-optimal models with two steps, but only one respectively with one and zero steps. The inset table describes several interesting Pareto-optimal models.

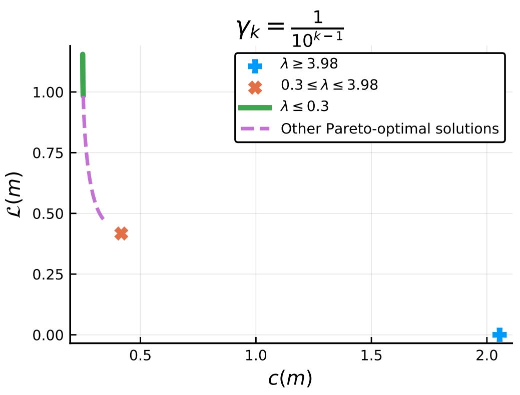
The proof of the proposition is provided in the appendix. Notice that the inner minimization problem in (6.12) is simply problem (6.11) with appropriate modifications of the coefficients $(\alpha_1, \ldots, \alpha_K)$.

In the case of the toy OLS problem introduced in Section 6.3.2, the interpretable step definition (adding a feature) and the number of features (two) mean that we only need to consider $K \leq 2$ when solving (6.12). Indeed, any path of more than two steps must repeat a model and will therefore be suboptimal. We can therefore use this decomposition to compute the price of interpretability in this toy problem, with the interpretability loss $\mathcal{L}_\gamma$ chosen such that $\gamma = 1$. Figure 6-5 shows all Pareto-optimal models with respect to performance cost and interpretability.

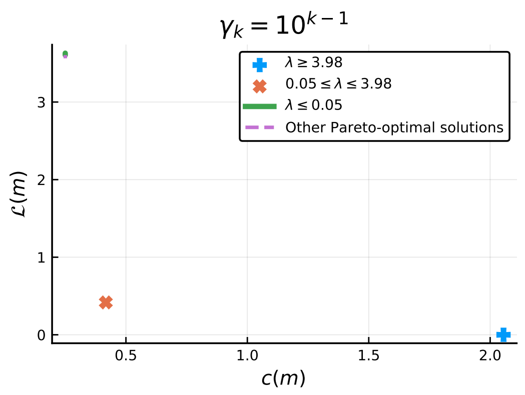
This curve is a useful result of the interpretable path framework. The ad hoc nature of most discussions of interpretability typically precludes formal analysis of the tradeoff between accuracy and interpretability. By defining the general framework of interpretable paths and a natural family of interpretability loss functions, we can understand exactly how much we gain or lose in terms of accuracy when we choose a more or less interpretable model with respect to the selected loss. This is a central question of the growing literature on interpretability in machine learning, and our framework provides a principled way to answer it.

We note that this curve has many interesting features. In the discrete structure of the Pareto front we can identify the combinatorial nature of our interpretability definition, which refines the notion of model complexity (sparsity). In the continuous parts of the front we recognize the tradeoffs of incrementality within a level of sparsity. Readers will notice that the weighted sum of the objectives optimized in Theorem 4 does not necessarily recover the entire Pareto front, which is a well-known result in multi-objective optimization [81].

Using Proposition 4, we can compute the price of interpretability for a range of models and interpretability losses. For instance, we represent in Figure 6-6 the interpretability-accuracy Pareto front on the toy OLS model with the interpretability loss $\mathcal{L}_\gamma$ for different values of γ. We notice that as in Theorem 3, when γ grows large our notion of interpretability reduces to sparsity (discrete Pareto curve), whereas when γ grows small our notion of interpretability becomes closer to incrementality



(a) Here we choose $\gamma = 0.1$, therefore the first steps are much more important than the last steps, and we favor incremental/greedy models (see second part of Theorem 3).



(b) Here we choose $\gamma = 10$, therefore the cost of the intermediate steps much less important than the final models, and we favor sparse models (see first part of Theorem 3). Note that we almost only have 3 points in the Pareto front, corresponding to the 3 possible levels of sparsity in this example.

Figure 6-6: Pareto fronts between model interpretability and cost in the same setting as Figure 6-5, except that we change the definition of interpretability by changing the values α_k.

(continuous Pareto curve).

6.5.2 Computational Considerations

In problem (6.4), the definition of an interpretable step led to a natural bound on the maximum number of steps of any Pareto-optimal interpretable path. To solve (6.12) we could therefore consider a finite number of problems of type (6.11). In other settings or for other interpretable step definitions (such as one we introduce in Section 6.6), there may be no such natural bound, and we may need to consider paths of arbitrary lengths when solving (6.12). Proposition 5 provides a bound for the number of problems of type (6.11) we need to consider in the general case.

Proposition 5. *Assume there exist $c_{\min}$ and $c_{\max}$ such that $0 < c_{\min} \leq c(m) \leq c_{\max}$ for all $m \in \mathcal{M}$ (positive and bounded cost function), and consider the interpretability* *loss* $\mathcal{L}_\gamma$. *If* $\gamma \geq 1$, *then*

$$K_{opt} := \arg\min_{K \geq 0} \left(\min_{\mathbf{m} \in \mathcal{P}_\mathbf{K}} c(m_K) + \lambda \sum_{k=1}^{K} \gamma^k c(m_k) \right) \leq K_{\max}, \tag{6.13}$$

where

$$K_{\max} = \begin{cases} \frac{c_{\max}}{\lambda c_{\min}} & \text{if } \gamma = 1, \\ \frac{\log\left(1 + \frac{(\gamma-1)c_{\max}}{\lambda \gamma c_{\min}}\right)}{\log \gamma} & \text{if } \gamma > 1. \end{cases} \tag{6.14}$$

In other words, under the interpretability loss $\mathcal{L}_\gamma$ with $\gamma \geq 1$, we can find the optimal solution of (6.12) by solving at most $K_{\max}$ problems of type (6.11). The proof of Proposition 5 is provided in the appendix.

A corollary of Proposition 5 is that we can write an optimization formulation of problem (6.12) with a finite number of decision variables. For instance, we can formulate the inner minimization problem with finitely many decision variables for each K and then solve finitely many such problems. The tractability of this optimization problem is application-dependent.

For example, by adapting the mixed-integer optimization formulation from Bertsimas and Dunn [15], we can compute the price of interpretability for decision trees of bounded depth by writing the following mixed-integer formulation of the inner minimization problem in (6.12):

$$\min \quad \sum_{k=1}^{K} \gamma^k f(d_t^k, a_t^k, b_t^k) \tag{6.15a}$$

$$\text{s.t.} \quad (d_t^k, a_t^k, b_t^k) \in \mathcal{T} \qquad \forall k \in [k] \tag{6.15b}$$

$$\sum_{t \in \mathcal{T}_L} d_t^k = k \qquad \forall k \in [K] \tag{6.15c}$$

$$a_t^k \leq a_t^{k+1} \qquad \forall t \in \mathcal{T}_B, k \in [K-1] \tag{6.15d}$$

$$d_t^k \leq d_t^{k+1} \qquad \forall t \in \mathcal{T}_B, k \in [K-1] \tag{6.15e}$$

$$b_t^k - (1 - d_t^k) \leq b_t^{k+1} \leq b_t^k + (1 - d_t^k) \qquad \forall t \in \mathcal{T}_B, k \in [K-1], \tag{6.15f}$$

where the variables d_t^k, a_t^k and b_t^k define K trees of depth at most D, and con-

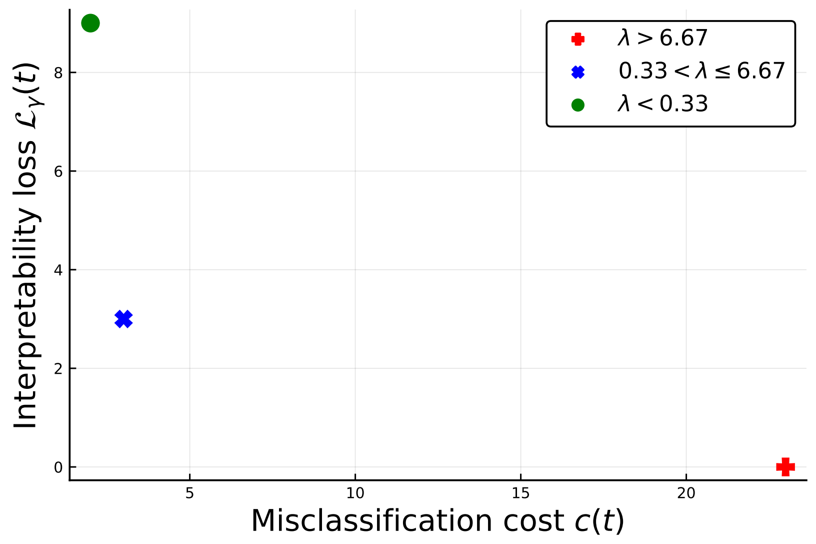


Figure 6-7: Price of interpretability for decision trees of depth at most 2 on the simplified `iris` data-set.

straints (6.15c)-(6.15f) impose an interpretable path structure on the K trees. The set $\mathcal{T}_B$ indicates the set of branching nodes of the trees, the variable d_t^k indicates whether branching node t in tree k is active, a_t^k selects the variable along which to perform the split at branching node t in tree k, and b_t^k is the split value at branching node t in tree k. The function f is the objective value of the tree defined by these split variables, and the set $\mathcal{T}$ designates all the constraints to impose the tree structure for each k (constraint (6.15b) is equivalent to (24) from [15]). Constraint (6.15c) imposes that tree k must have exactly k active splits, Constraint (6.15e) forces tree $k+1$ to keep all the branching nodes of tree k, and constraints (6.15d) and (6.15f) force the splits at these common branching nodes to be the same.

This formulation allows us to compute the price of interpretability on the simplified `iris` data-set from Section 6.2. The resulting Pareto curve is shown in Figure 6-7. It turns out the most interpretable tree with a misclassification error of 2 is t_{good}, with $\mathcal{L}_\gamma(t_{\text{good}}) = 9$ (for $\gamma = 1$).

In general, mixed-integer optimization formulations such as (6.15) may not scale. However, in many cases a provably optimal solution is not necessary and scalable heuristics such as local improvement may be employed. We provide such an example in Section 6.6.

6.5.3 Interpretable Paths and Human-in-the-Loop Analytics

Motivated by the idea that humans read models sequentially, we have used the framework of interpretable paths to evaluate the interpretability of individual models. Viewing an interpretable path as a nested sequence of models of increasing complexity can be useful in the context of human-in-the-loop analytics.

Consider the problem of customer segmentation via clustering. Choosing the number of customer types (k) is not always obvious in practice and has to be selected by a decision-maker. Solving the clustering problem with k clusters and with $k+1$ clusters may lead to very different clusters. Alternatively, using interpretable steps, we can force the solution with $k+1$ clusters to result from the splitting of one of the clusters of the solution with k clusters, for all k. The change between k clusters and $k+1$ clusters becomes more simple and may facilitate the choice of k.

If we assume each k can be chosen with equal probability for $k \leq 10$, the problem of finding the sequence that minimizes the expected cost is:

$$\min_{\boldsymbol{m} \in \mathcal{P}_{10}} \quad \frac{1}{10} \sum_{k=1}^{10} c(m_k), \tag{6.16}$$

which is exactly the decision problem (6.11) with the weights $\alpha_k = 0.1$ for $k \leq 10$, and $\alpha_k = 0$ otherwise. This problem is related to studies in incremental approximation algorithms [88] and prioritization [83], which are typically motivated by a notion of interpretability which simplifies implementation for practitioners.

More generally, we can use interpretable paths to facilitate human-in-the-loop model selection. Given a discrete distribution on the choice of K :

$$p_k = \mathbb{P}(\{k \text{ will be chosen by the decision maker}\}).$$

We can choose $\alpha_k = p_k$ and solve (6.11) to find paths $\boldsymbol{m}$ that minimize the expected cost $\mathbb{E}_k[c(m_k)]$.

6.6 Application: Linear Regression

6.6.1 Setting

In many applications, users of a predictive model are confronted with new data which requires them to update their model. One way to do this is to throw out the old model and train a new model using the new data. In the interest of continuity it may be of interest to update the old model in a more interpretable way.

Using the framework presented in the previous section, it is easy to formulate this problem mathematically. Given a model m_0 (which designates our old model, not 0 as previously), we would like to select a new model m with lower cost, and an interpretable path from m_0 to m.

In the example of linear regression, the models are the weights β, with the following interpretable steps:

$$\mathcal{S}(\beta) = \{\beta' \in \mathbb{R}^d : \|\beta - \beta'\|_0 \leq 1\}, \tag{6.17}$$

which implies that a successor of β differs from β in at most one coordinate. This step neighborhood function is similar to the one presented in the previous section. The only difference is that in Section 2, we could not modify the value of a coefficient once it had been set once, whereas in this case we can modify coefficients as often as we like (but no more than one coefficient in each step). We select the interpretability loss $\mathcal{L}_\gamma$ with $\gamma = 1$ (meaning the costs of all steps matter equally). Given the step function $\mathcal{S}$ defined in (6.17), the convex quadratic cost function $c(\cdot)$, and the initial regression coefficients β_0, our goal is to be able to solve the optimization problem of finding the optimal interpretable path of length K (6.11), as we saw in Section 6.5 that it was enough to compute interpretability tradeoffs.

6.6.2 Algorithms

Optimal. It can be written as a convex integer optimization problem using special ordered sets of type 1 (SOS-1 constraints).

$$\min_{\boldsymbol{\beta_k}} \quad \sum_{k=1}^{K} c(\beta_k) \tag{6.18a}$$

$$\text{s.t.} \quad \text{SOS-1}(\beta_{k+1} - \beta_k) \qquad 0 \leq k < K. \tag{6.18b}$$

The SOS-1 constraint are just a generic way to describe our choice of step (6.17). For reasonable problem sizes ($d \leq 10$, $K \leq 10$ and any choice of n), this problem can be solved exactly using a constrained convex solver such as Gurobi or CPLEX.

Local improvement. In higher-dimensional settings, or when K grows large, the formulation above may no longer scale. Thus it is of interest to develop a fast heuristic for such instances.

A feasible solution $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_K)$ to problem (6.18) can be written as a vector of indices $\boldsymbol{i} = (i_1, \cdots, i_K) \in \{1, \ldots, d\}^K$ and a vector of values $\boldsymbol{\delta} = (\delta_1, \cdots, \delta_K) \in \mathbb{R}^K$, such that for $0 \leq k < K$,

$$(\beta_{k+1})_i = \begin{cases} (\beta_k)_i + \delta_k, & \text{if } i = i_k \\ (\beta_k)_i, & \text{if } i \neq i_k. \end{cases}$$

The vector of indices $\boldsymbol{i}$ encodes which regression coefficients are modified at each step in the interpretable path, while the sequence of values $\boldsymbol{\delta}$ encodes the value of each modified regression coefficient. Thus problem (6.18) can be rewritten as

$$\min_{\boldsymbol{i}} \min_{\boldsymbol{\delta}} C(\boldsymbol{i}, \boldsymbol{\delta}) := \sum_{k=1}^{K} c\left(\beta_0 + \sum_{j=1}^{k} \delta_j e_{i_j}\right), \tag{6.19}$$

where e_i designates the i-th unit vector. Notice that the inner minimization problem is an "easy" convex quadratic optimization problem, while the outer minimization problem is a "hard" combinatorial optimization problem. We propose the following local improvement heuristic for the outer problem: given a first sequence of indices $\boldsymbol{i} = \boldsymbol{i}^0$, we randomly sample one step κ in the interpretable path. Keeping all i_k constant for $k \neq \kappa$, we iterate through all d possible values of i_κ and obtain d candidate vectors $\hat{\boldsymbol{i}}$. For each candidate, we solve the inner minimization problem and keep the one with lowest cost. The method is described in full detail as Algorithm 6, in the more general case where we sample not one but q steps from the interpretable path.

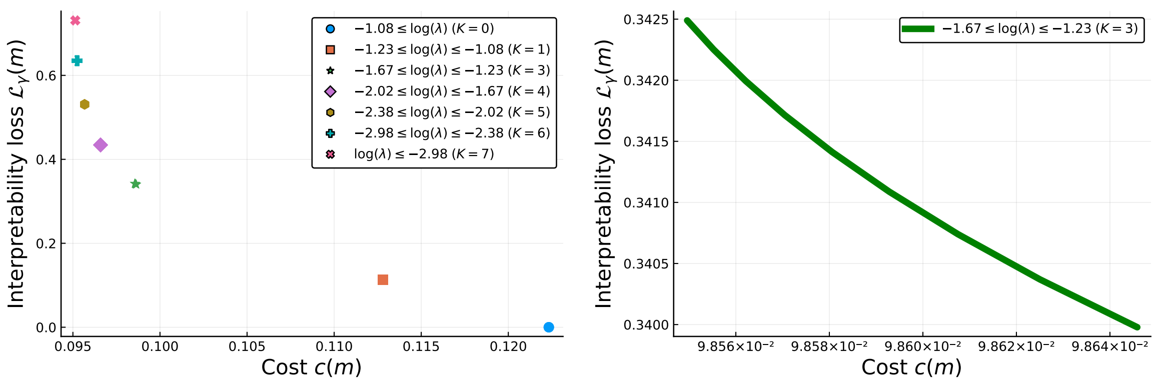
Algorithm 6 Local improvement heuristic. Inputs: regression cost function $c(\cdot)$; starting vector of indices i^0. Parameters: $q \in \mathbb{N}$ controls the size of the neighborhood, $T \in \mathbb{N}$ controls the number of iterations.

```
1: function LocalImprovement($c(\cdot)$, $\boldsymbol{i}^0$, $q$, $T$)
2:     for $1 \leq t \leq T$ do
3:         $\boldsymbol{i}^* \leftarrow \boldsymbol{i}^0$
4:         $\boldsymbol{\delta}^* \leftarrow \arg\min_{\boldsymbol{\delta}} C(\boldsymbol{i}^0, \boldsymbol{\delta})$
5:         $C^* \leftarrow C(\boldsymbol{i}^0, \boldsymbol{\delta}^*)$
6:         Randomly select $\mathcal{K} = \{\kappa_1, \ldots, \kappa_q\} \subset \{1, \ldots, K\}$   ▷ subset of cardinality $q$
7:         $\hat{\boldsymbol{i}} \leftarrow \boldsymbol{i}^*$
8:         $\hat{\boldsymbol{\delta}} \leftarrow \boldsymbol{\delta}^*$
9:         for $(f_1, \ldots, f_q) \in \{1, \ldots, d\}^q$ do
10:            for $1 \leq p \leq q$ do
11:                $\hat{i}_{\kappa_p} = f_p$
12:            $\hat{\boldsymbol{\delta}} \leftarrow \arg\min_{\boldsymbol{\delta}} C\left(\hat{\boldsymbol{i}}, \boldsymbol{\delta}\right)$
13:            if $C\left(\hat{\boldsymbol{i}}, \hat{\boldsymbol{\delta}}\right) < C^*$ then
14:                $C^* \leftarrow C\left(\hat{\boldsymbol{i}}, \hat{\boldsymbol{\delta}}\right)$
15:                $\boldsymbol{i}^* \leftarrow \hat{\boldsymbol{i}}$
16:                $\boldsymbol{\delta}^* \leftarrow \hat{\delta}$
17:    return $\boldsymbol{i}^*, \boldsymbol{\delta}^*$
```

Each iteration of the local improvement heuristic above requires d^q iterations of the main loop (for most problems, this means only $q = 1$ and sometimes $q = 2$ are realistic options).

Results

We now explore the results of the presented approach on a data-set from the 1998-1999 California test score data-set. Each data point represents a school, and the variable of interest is the average standardized test score of students from that school. All



(a) Full price curve (b) Zoom on small portion of price curve

Figure 6-8: Pareto-efficient models from the perspective of interpretability and cost. Notice that the price curve is both discrete and continuous.

features are continuous and a full list is presented in Table 6-9a. Both the features and the target variables are centered and rescaled to have unit variance.

In our example, we assume that we already have a regression model available to predict the number of trips: it was trained using only the percentage of students qualifying for a reduced-price lunch. This model has an MSE of 0.122 (compared to an optimal MSE of 0.095). We would like to update this model in an interpretable way given the availability of all features in the data-set.

The first thing we can do is explore the price of interpretability in this setting. We can use the method presented in Section 6.5.1 to compute find Pareto efficient interpretable models. The resulting price curve is shown in Figure 6-8.

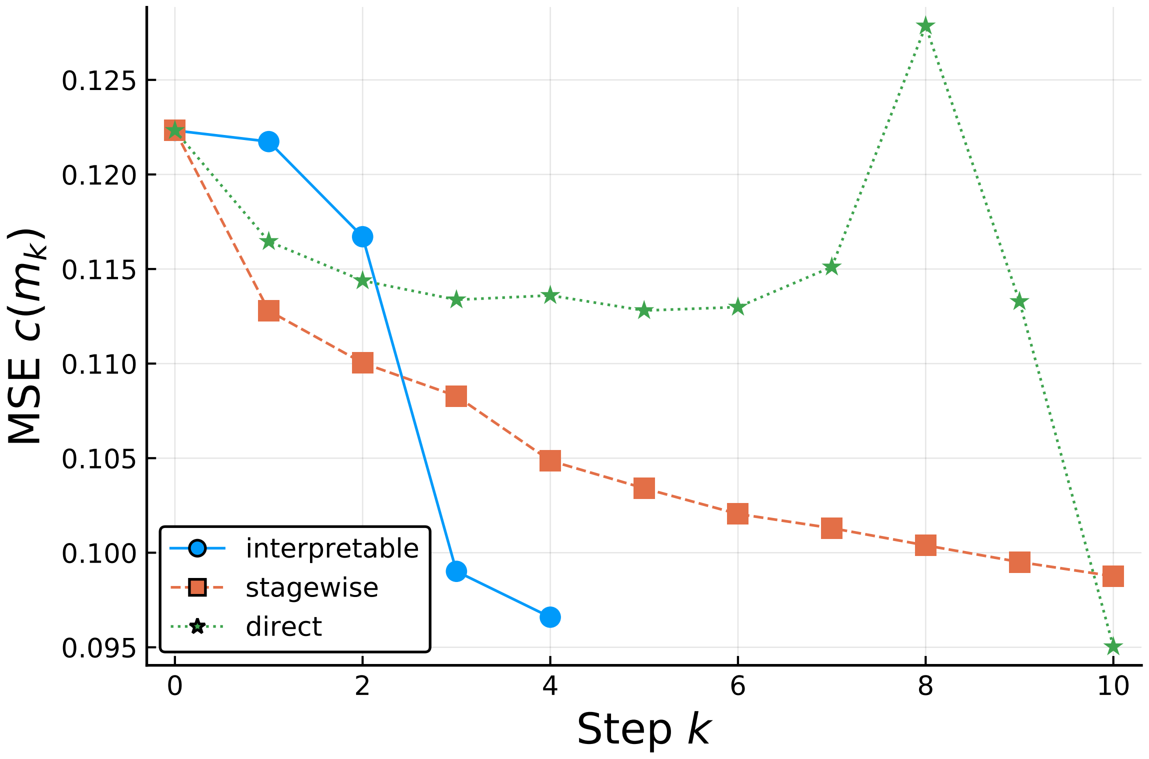
Given this price curve, a good cost-interpretability tradeoff seems to be for $\log(\lambda) \approx -1.65$. This yields the new model (and associated interpretable path) shown in Figure 6-9b. This new model can be obtained from the old in just four steps. First we add the district average income with a positive coefficient, then we correct the coefficient for reduced-price lunch students to account for this new feature, and finally we add the percentage of English learners and the school's per-student spending. The final model has an MSE of 0.097 which is near-optimal. When we compare this path to other methods (see Figure 6-9c) we see that our interpretable formulation allows us to find a good tradeoff between a greedy, "every step must improve" formulation, and a formulation that just sets the coefficients to their final values one by one.

Feature name	Description
Enrollment	Total enrollment
Teachers	Number of teachers
CalWPct	% receiving state aid
MealPct	% with subsidized lunch
Computers	Number of computers
CompStu	Computers per student
ExpnStu	Expenditure per student
StuTeach	Student-teacher ratio
AvgInc	Average income (district)
ELPct	% English Learners

(a) Features of the test score dataset.

	Feature				
Step	MealPct	AvgInc	ELPct	ExpnStu	MSE
0	-0.87	-	-	-	0.122
1	-0.87	**0.23**	-	-	0.122
2	$\mathbf{-0.59}$	0.23	-	-	0.117
3	-0.59	0.23	$\mathbf{-0.18}$	-	0.099
4	-0.59	0.23	-0.18	**0.07**	0.097

(b) Path from old model to new model.



(c) Comparison between interpretable path and other approaches

Figure 6-9: Example of a Pareto-efficient interpretable path. On the left we see the benefits of each coefficient modification. On the right we compare the interpretable path with two other possible paths. The first is the forward stagewise path which greedily selects the best m_{k+1} given m_k. The second is the "direct" path, which adds the optimal least squares coefficients one by one (in the most interpretable order). The direct method is only good when all the coefficients have been added, whereas the greedy approach is good at first but then does not converge. The interpretable path is willing to make some steps that do not improve the cost too much in preparation for very cost-improving steps.

6.7 Conclusions

In this chapter, we have presented a simple optimization-based framework to model the interpretability of machine learning models. Our framework provides a new way to think about what interpretability means to users in different applications and quantify how this meaning affects the tradeoff with predictive accuracy.

6.8 Appendix

6.8.1 Proof of Theorem 3

Proof of part (a). As $c(\cdot)$ is bounded, we have $c_{\max} \in \mathbb{R}$ such that $0 < c(\cdot) \leq c_{\max}$.

Let $\boldsymbol{m}^+ \in \mathcal{P}(m^+)$ be a path of optimal length to the model m^+, i.e., $|\boldsymbol{m}^+| = \mathcal{L}_{\text{complexity}}(m^+)$. . Let $\boldsymbol{m}^- \in \mathcal{P}(m^-)$ be any path leading to m^- (not necessarily of optimal length). By assumption, we have $|\boldsymbol{m}^-| \geq |\boldsymbol{m}^+|$, and by definition of model interpretability, we have $\mathcal{L}_\gamma(m^+) \leq \mathcal{L}_\gamma(\boldsymbol{m}^+)$. Therefore we obtain:

$$\mathcal{L}_\gamma(\boldsymbol{m}^-) - \mathcal{L}_\gamma(m^+) \geq \mathcal{L}_\gamma(\boldsymbol{m}^-) - \mathcal{L}_\gamma(\boldsymbol{m}^+) \tag{6.20}$$

$$= \sum_{k=1}^{|\boldsymbol{m}^-|} \gamma^k\, c(m_k^-) - \sum_{k=1}^{|\boldsymbol{m}^+|} \gamma^k\, c(m_k^+) \tag{6.21}$$

$$\begin{aligned} &= \gamma^{|\boldsymbol{m}^+|} \left(\sum_{k=1}^{|\boldsymbol{m}^+|-1} \frac{1}{\gamma^{|\boldsymbol{m}^+|-k}} \left(c(m_k^-) - c(m_k^+)\right) + \left(c(m_{|\boldsymbol{m}^+|}^-) - c(m_{|\boldsymbol{m}^+|}^+)\right) \right) \\ &\qquad + \gamma^{|\boldsymbol{m}^+|} \left(\sum_{k=|\boldsymbol{m}^+|+1}^{|\boldsymbol{m}^-|} \gamma^{k-|\boldsymbol{m}^+|}\, c(m_k^-) \right) \end{aligned} \tag{6.22}$$

$$\geq \gamma^{|\boldsymbol{m}^+|} \left(-c_{\max} \sum_{k=1}^{|\boldsymbol{m}^+|-1} \frac{1}{\gamma^{|\boldsymbol{m}^+|-k}} + \left(c(m_{|\boldsymbol{m}^+|}^-) - c(m^+)\right) + \sum_{k=|\boldsymbol{m}^+|+1}^{|\boldsymbol{m}^-|} \gamma^{k-|\boldsymbol{m}^+|}\, c(m_k^-) \right) \tag{6.23}$$

where (6.21) follows from the definition of model interpretability, (6.22) is just a development of the previous equation, and (6.23) just bounds the first sum and uses $m_{|\boldsymbol{m}^+|}^+ = m^+$ for the middle term.

If $\mathcal{L}_{\text{complexity}}(m^+) < \mathcal{L}_{\text{complexity}}(m^-)$, we have $|\boldsymbol{m}^+| < |\boldsymbol{m}^-|$, and therefore the last sum in (6.23) is not empty and for $\gamma \geq 1$ we can bound it:

$$\sum_{k=|\boldsymbol{m}^+|+1}^{|\boldsymbol{m}^-|} \gamma^{k-|\boldsymbol{m}^+|}\, c(m_k^-) \geq \gamma^{|\boldsymbol{m}^-|-|\boldsymbol{m}^+|}\, c(m_{|\boldsymbol{m}^-|}^-) \geq \gamma\, c(m^-). \tag{6.24}$$

Therefore, for $\gamma \geq 1$ we have:

$$\mathcal{L}_\gamma(\boldsymbol{m}^-) - \mathcal{L}_\gamma(m^+) \geq \gamma^{|\boldsymbol{m}^+|} \left(\gamma\, c(m^-) - c(m^+) - c_{max} \sum_{k=1}^{|\boldsymbol{m}^+|-1} \frac{1}{\gamma^{|\boldsymbol{m}^+|-k}} \right). \tag{6.25}$$

This bound is valid for all the path $\boldsymbol{m}^-$ leading to m^-, in particular the one with optimal interpretability loss, therefore we have (for $\gamma \geq 1$):

$$\mathcal{L}_\gamma(m^-) - \mathcal{L}_\gamma(m^+) \geq \gamma^{|\boldsymbol{m}^+|} \left(\gamma\, c(m^-) - c(m^+) - c_{max} \sum_{k=1}^{|\boldsymbol{m}^+|-1} \frac{1}{\gamma^{|\boldsymbol{m}^+|-k}} \right). \tag{6.26}$$

which implies (as $c(m^-) > 0$):

$$\lim_{\gamma \to +\infty} \mathcal{L}_\gamma(m^-) - \mathcal{L}_\gamma(m^+) = +\infty \tag{6.27}$$

We now look at the case $\mathcal{L}_{\text{complexity}}(m^+) = \mathcal{L}_{\text{complexity}}(m^-)$ and $c(m^+) < c(m^-)$. For $\gamma \geq 1$, we can easily bound parts of equation (6.23):

$$c(m_{|\boldsymbol{m}^+|}^-) + \sum_{k=|\boldsymbol{m}^+|+1}^{|\boldsymbol{m}^-|} \gamma^{k-|\boldsymbol{m}^+|}\, c(m_k^-) \geq \gamma^{|\boldsymbol{m}^-|-|\boldsymbol{m}^+|} c(m_{|\boldsymbol{m}^-|}^-) \geq c(m^-). \tag{6.28}$$

Putting it back into (6.23), we obtain (for $\gamma \geq 1$)

$$\mathcal{L}_\gamma(\boldsymbol{m}^-) - \mathcal{L}_\gamma(m^+) \geq \gamma^{|\boldsymbol{m}^+|} \left(\left(c(m^-) - c(m^+)\right) - c_{max} \sum_{k=1}^{|\boldsymbol{m}^+|-1} \frac{1}{\gamma^{|\boldsymbol{m}^+|-k}} \right). \tag{6.29}$$

This bound is independent of the path $\boldsymbol{m}^-$ leading to m^-, therefore we have

$$\mathcal{L}_\gamma(m^-) - \mathcal{L}_\gamma(m^+) \geq \gamma^{|\boldsymbol{m}^+|} \left(\big(c(m^-) - c(m^+)\big) - c_{max} \sum_{k=1}^{|\boldsymbol{m}^+|-1} \frac{1}{\gamma^{|\boldsymbol{m}^+|-k}} \right) \rightarrow_{\gamma\rightarrow+\infty} +\infty, \tag{6.30}$$

which ends the proof of part (a) of Theorem 3. □

Proof of part (b). Consider two paths $\boldsymbol{m}^+, \boldsymbol{m}^- \in \mathcal{P}$, such that $\boldsymbol{c}(\boldsymbol{m}^+) \preceq \boldsymbol{c}(\boldsymbol{m}^-)$. By definition of the lexicographic order, either the two paths are the same (in that case the theorem is trivial), or there exist $K \geq 1$ such that:

$$\begin{cases} \boldsymbol{c}(\boldsymbol{m}^+)_k = \boldsymbol{c}(\boldsymbol{m}^-)_k & \forall k < K \\ \boldsymbol{c}(\boldsymbol{m}^+)_K < \boldsymbol{c}(\boldsymbol{m}^-)_K. \end{cases}$$

We have:

$$\mathcal{L}_\gamma(\boldsymbol{m}^-) - \mathcal{L}_\gamma(\boldsymbol{m}^+) = \sum_{k=1}^{|\boldsymbol{m}^-|} \gamma^k\, c(m_k^-) - \sum_{k=1}^{|\boldsymbol{m}^+|} \gamma^k\, c(m_k^+) \tag{6.31}$$

$$= \sum_{k=1}^{\infty} \gamma^k\, \big(\boldsymbol{c}(\boldsymbol{m}^-)_k - \boldsymbol{c}(\boldsymbol{m}^+)_k\big) \tag{6.32}$$

$$= \sum_{k=1}^{K-1} \gamma^k\, \big(\boldsymbol{c}(\boldsymbol{m}^-)_k - \boldsymbol{c}(\boldsymbol{m}^+)_k\big) + \gamma^K\, \big(\boldsymbol{c}(\boldsymbol{m}^-)_K - \boldsymbol{c}(\boldsymbol{m}^+)_K\big) + \sum_{k=K+1}^{\infty} \gamma^k\, \big(\boldsymbol{c}(\boldsymbol{m}^-)_k - \boldsymbol{c}(\boldsymbol{m}^+)_k\big) \tag{6.33}$$

$$= \gamma^K \left(\boldsymbol{c}(\boldsymbol{m}^-)_K - \boldsymbol{c}(\boldsymbol{m}^+)_K + \sum_{k=K+1}^{\infty} \gamma^{k-K}\, \big(\boldsymbol{c}(\boldsymbol{m}^-)_k - \boldsymbol{c}(\boldsymbol{m}^+)_k\big) \right) \tag{6.34}$$

where (6.32) just applies the definition of the sequence $\boldsymbol{c}$, and (6.34) uses $\boldsymbol{c}(\boldsymbol{m}^+)_k = \boldsymbol{c}(\boldsymbol{m}^-)_k \quad \forall k < K$.

The term inside the parenthesis in (6.34) converges to $\boldsymbol{c}(\boldsymbol{m}^-)_K - \boldsymbol{c}(\boldsymbol{m}^+)_K > 0$

when $\gamma \to 0$, as the paths are finite. Therefore

$$\lim_{\gamma \to 0} \mathcal{L}_\gamma(\boldsymbol{m}^-) - \mathcal{L}_\gamma(\boldsymbol{m}^+) \geq 0, \tag{6.35}$$

which proves (6.9). The very end of the theorem is an immediate consequence. $\square$

6.8.2 Proof of Proposition 4

Proof. First, a solution of

$$\min_{m \in \mathcal{M}} (c(m) + \lambda \mathcal{L}(m))$$

is Pareto optimal between the cost $c(\cdot)$ and the interpretability $\mathcal{L}_\alpha(\cdot)$ as it corresponds to the minimization of a weighted sum of the objectives. Furthermore, we can write

$$\begin{aligned}
\min_{m \in \mathcal{M}} (c(m) + \lambda \mathcal{L}_\alpha(m)) &= \min_{m \in \mathcal{M}} \left(c(m) + \lambda \min_{\boldsymbol{m} \in \mathcal{P}(m)} \mathcal{L}_\alpha(\boldsymbol{m}) \right) \\
&= \min_{m \in \mathcal{M},\, \boldsymbol{m} \in \mathcal{P}(m)} (c(m) + \lambda \mathcal{L}_\alpha(\boldsymbol{m})) \\
&= \min_{m \in \mathcal{M},\, K \geq 0,\, \boldsymbol{m} \in \mathcal{P}_K(m)} \left(c(m_K) + \lambda \sum_{k=1}^{K} \alpha_k c(m_k) \right) \\
&= \min_{K \geq 0,\, \boldsymbol{m} \in \mathcal{P}_K} \left(c(m_K) + \lambda \sum_{k=1}^{K} \alpha_k c(m_k) \right).
\end{aligned}$$

$\square$

6.8.3 Proof of Proposition 5

Proof. For any $K \geq 0$ and $\lambda > 0$, define the optimal objective

$$z_\lambda(K) = \min_{\boldsymbol{m} \in \mathcal{P}_K} c(m_K) + \lambda \sum_{k=1}^{K} \gamma^k c(m_k).$$

Because $c(\cdot)$ is bounded below by $c_{\min}$, we can write

$$z_\lambda(K) \geq c_{\min} + \lambda \sum_{k=1}^{K} \gamma^k c_{\min} \geq \lambda c_{\min} \sum_{k=1}^{K} \gamma^k. \tag{6.36}$$

By definition $z_\lambda(0)$ is the cost of the empty model, so by the boundedness of $c(\cdot)$, we have $z_\lambda(0) \leq c_{\max}$. Consider first the case when $\gamma = 1$. Then (6.36) simplifies to

$$z_\lambda(K) \geq \lambda K c_{\min}.$$

Setting $K \geq K_{\max} := c_{\max}/(\lambda c_{\min})$ yields $z_\lambda(K) \geq c_{\max} \geq Z_\lambda(0)$ and so the interpretable path of length 0 has a better objective than any path of length at least $K_{\max}$. Now consider $\gamma > 1$. In this case, (6.36) simplifies to

$$z_\lambda(K) \geq \lambda c_{\min} \gamma \frac{1 - \gamma^K}{1 - \gamma}.$$

Defining $K_{\max}$ as in (6.14), we again see that the interpretable path of length 0 has a better objective than any path of length at least $K_{\max}$, which completes the proof. $\square$

Chapter 7

Conclusions

We conclude this thesis by summarizing our main contributions and outlining potential directions of future research.

New transportation applications A large fraction of this work was related to transportation applications: ride-sharing in Chapter 2, travel time estimation in Chapter 3 and school transportation in Chapter 4. The latest technology developments, such as the use of smartphones with location tracking capability and the rise of on-demand transportation, have created many opportunities for research in transportation optimization. Almost every year brings new major transportation systems: bike sharing, vehicle pooling, on-demand electric scooters, autonomous vehicles are just a few examples ; and all of these systems present their own research challenges. Indeed, as these systems become more centralized and data more available, there is a significant optimization opportunity when managing the associated vehicle fleets. And tractable optimization algorithms can directly lead to impact in this fast-moving field.

A direction that we find particularly promising and interfaces well with this thesis is the special case of hybrid static and dynamic transportation systems. That is, the combination of static high-capacity transportation system such as buses and trains with dynamic ones like ride-sharing in a multi-modal transportation network. These problems are interestingly at the intersection of the design of static bus systems such as the school bus routes presented in Chapter 4, and online decision making such as the ride-sharing example presented in Chapter 2.

Opportunities in large-scale optimization In this thesis, we present many application-specific large-scale optimization algorithms, often relying integer optimization as a sub-routine. Chapter 2 present a `backbone` and `local-backbone` algorithms for scheduling with time-windows. Chapter 3 finds solutions to the inverse shortest path length problem by solving a sequence of second order cone problems. Chapter 4 introduces the bi-objective routing decomposition algorithm for school bus routing and also uses block coordinate descent to solve a generalized quadratic assignment problem. Chapter 5 studies the stochastic proximal algorithm. Chapter 6 uses integer optimization to create interpretable linear models.

A challenge of large-scale optimization is to be able to design scalable optimization algorithms that are not application-specific, and can be easily implemented. Mixed-integer programming formulation provides this flexibility, when it is tractable and associated with state-of-the-art solvers. We saw in this thesis that there are actually many ways, through problem decompositions and optimization-based heuristics, to retain the flexibility of this framework in a large-scale setting. For example, our contribution in school bus routing interfaces well with all the variants of school transportation settings, and this flexibility enables easier implementation. And we believe there are significant research opportunities in this space. As a specific research direction, we believe that the `backbone` and `local-backbone` algorithms presented in Chapter 1 could be extended beyond the scheduling problem with time-windows. These algorithms worked surprisingly well in the example of ride-sharing, and it could also be interesting to theoretically understand the main reasons for this success.

People and algorithms After modeling a real-world problem using an optimization framework, and after finding tractable optimization algorithms to solve this problem, implementation can still be a challenge. For example, Chapter 4 explores the difficulties of using optimization algorithms for the operations of public schools districts. One way to make this transition easier is to encourage communication between optimization researchers and stakeholders. In this work, we chose to get involved in the public policy aspect of choosing schools start times. This experience made us realize the many opportunities of positive impact in public school district operations and other public policy spheres for researchers that are willing to bridge the gap with stakeholders.

But this process can be made easier if we can explain in simple terms our choices of algorithms and models. This is why we introduced a general framework for creating and optimizing model interpretability metrics in Chapter 6. This general framework can be applied to a variety of machine learning and other optimization models, and yield both exciting new applications and interesting optimization problems. We hope that it will generate further research.

> The final test of a theory is its capacity to solve the problems which originated it.
>
> ([45] George Dantzig — *Linear programming and extensions (1963)*)

Bibliography

[1] Atila Abdulkadiroğlu, Parag A Pathak, Alvin E Roth, and Tayfun Sönmez. The boston public school match. *American Economic Review*, 95(2):368–371, 2005.

[2] MOSEK ApS. *Mosek Solver Reference Manual*, 2019.

[3] Hilal Asi and John C. Duchi. Stochastic (Approximate) Proximal Point Methods: Convergence, Optimality, and Adaptivity. *arXiv Optimization and Control*, 2018.

[4] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

[5] Hamsa Bastani, Osbert Bastani, and Carolyn Kim. Interpreting Predictive Models for Human-in-the-Loop Analytics. *arXiv preprint arXiv:1705.08504*, pages 1–45, 2018.

[6] Amir Beck and Marc Teboulle. Gradient-based algorithms with applications to signal-recovery problems. In Daniel P. Palomar and Yonina C. Eldar, editors, *Convex Optimization in Signal Processing and Communications*, chapter 1, pages 42–88. Cambridge University Press, Cambridge, 2010.

[7] Russell Bent and Pascal Van Hentenryck. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. *Operations Research*, 52(6):977–987, 2004.

[8] Russell Bent and Pascal Van Hentenryck. Waiting and relocation strategies in online stochastic vehicle routing. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1816–1821, 2007.

[9] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202:8–15, 2010.

[10] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem. *Journal on Computing*, 24(3):343–355, 2012.

[11] Richard Berk. An impact assessment of machine learning risk forecasts on parole board decisions and recidivism. *Journal of Experimental Criminology*, 13(2):193–216, 2017.

[12] Dimitri P. Bertsekas. Incremental proximal methods for large-scale convex optimization. *Mathematical Programming*, 129(2):163–195, oct 2011.

[13] Dimitris Bertsimas, Arthur Delarue, Patrick Jaillet, and Sébastien Martin. Travel time estimation in the age of big data. *Operations Research*, 2019.

[14] Dimitris Bertsimas, Arthur Delarue, and Sebastien Martin. Optimizing schools' start time and bus routes. *Proceedings of the National Academy of Sciences*, 116(13):201811462, 2019.

[15] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[16] Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019.

[17] Dimitris Bertsimas, Nathan Kallus, Alexander M. Weinstein, and Ying Daisy Zhuo. Personalized diabetes management using electronic medical records. *Diabetes Care*, 40(2):210–217, 2017.

[18] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *Annals of Statistics*, 44(2):813–852, 2016.

[19] Dimitris Bertsimas and Bart Van Parys. Sparse High-Dimensional Regression: Exact Scalable Algorithms and Phase Transitions. *arXiv preprint arXiv:1709.10029*, pages 1–22, 2017.

[20] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, jan 2017.

[21] Pascal Bianchi. Ergodic Convergence of a Stochastic Proximal Point Algorithm. *SIAM Journal on Optimization*, 26(4):2235–2260, 2016.

[22] Lawrence D Bodin and Lon Berman. Routing and scheduling of school buses by computer. *Transportation Science*, 13(2):113–129, 1979.

[23] Michael Bögl, Karl F Doerner, and Sophie N Parragh. The school bus routing and scheduling problem with transfers. *Networks*, 65(2):180–203, 2015.

[24] Leon Léon Bottou and Olivier Bousquet. The tradeoffs of large-scale learning. *Advances in neural information processing systems*, 20:161–168, 2008.

[25] Jeffrey Braca, Julien Bramel, Bruce Posner, and David Simchi-Levi. A computerized approach to the new york cityschool bus routing problem. *IIE transactions*, 29(8):693–702, 1997.

[26] Julien Bramel and David Simchi-Levi. A location based heuristic for general routing problems. *Operations research*, 43(4):649–660, 1995.

[27] Olli Bräysy and Michel Gendreau. Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science*, 39(1):104–118, 2005.

[28] Leo Breiman. *Classification and regression trees.* New York: Routledge, 1984.

[29] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[30] Leo Breiman. Statistical modeling: The two cultures. *Statistical science*, 16(3):199–231, 2001.

[31] Cristian Bucilă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, page 535, New York, New York, USA, 2006. ACM, ACM Press.

[32] S.E. Carrell, T. Maghakian, and J.E. West. A's from ZZZZ's? The causal effect of school start time on academic achievement of adolescents. *American Economic Journal*, 3(3):62–71, 2011.

[33] Bi Yu Chen, Hui Yuan, Qingquan Li, William H. K. Lam, Shih-Lung Shaw, and Ke Yan. Map-matching algorithm for large-scale low-frequency floating car data. *Int. J. Geogr. Inf. Sci.*, 28(1):22–38, January 2014.

[34] Xiaoli Chen, May A Beydoun, and Youfa Wang. Is sleep duration associated with childhood obesity? a systematic review and meta-analysis. *Obesity*, 16(2):265–274, 2008.

[35] Xiaopan Chen, Yunfeng Kong, Lanxue Dang, Yane Hou, and Xinyue Ye. Exact and metaheuristic approaches for a bi-objective school bus scheduling problem. *PloS one*, 10(7):e0132600, 2015.

[36] Zhi-Long Chen and Hang Xu. Dynamic Column Generation for Dynamic Vehicle Routing with Time Windows. *Transportation Science*, 40(1):74–88, 2006.

[37] Geoffrey L Cohen, Julio Garcia, Nancy Apfel, and Allison Master. Reducing the racial achievement gap: A social-psychological intervention. *science*, 313(5791):1307–1310, 2006.

[38] Benjamin Coifman. Estimating travel times and vehicle trajectories on freeways using dual loop detectors. *Transportation Research Part A: Policy and Practice*, 36(4):351 – 364, 2002.

[39] Gérard Cornuéjols, George L Nemhauser, and Laurence A Wolsey. The uncapacitated facility location problem. Technical report, Carnegie-Mellon University, Management Sciences research group, 1983.

[40] Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better Mini-Batch Algorithms via Accelerated Gradient Methods. In J Shawe-Taylor, R S Zemel, P L Bartlett, F Pereira, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1647–1655. Curran Associates, Inc., 2011.

[41] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[42] Stephanie J Crowley, Christine Acebo, and Mary A Carskadon. Sleep, circadian rhythms, and delayed phase in adolescence. *Sleep medicine*, 8(6):602–612, 2007.

[43] Giuseppe Curcio, Michele Ferrara, and Luigi De Gennaro. Sleep loss, learning capacity and academic performance. *Sleep medicine reviews*, 10(5):323–337, 2006.

[44] Fred Danner and Barbara Phillips. Adolescent sleep, school start times, and teen motor vehicle crashes. *J Clin Sleep Med.*, 4(6):533, 2008.

[45] George Dantzig. *Linear programming and extensions.* Princeton university press, 2016.

[46] Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic Transparency via Quantitative Input Influence :. In *2016 IEEE Symposium on Security and Privacy*, 2016.

[47] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal Distributed Online Prediction using Mini-Batches. *arXiv preprint*, dec 2010.

[48] Arthur Delarue and Sebastien Martin. SchoolBusRouting repository. https://github.com/mitschoolbus/SchoolBusRouting, 2018.

[49] Guy Desaulniers, Fausto Errico, Stefan Irnich, and Michael Schneider. Exact Algorithms for Electric Vehicle-Routing Problems with Time Windows. *Operations Research*, 64(6):1388–1405, 2016.

[50] Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

[51] Robert Barkley Dial. A probabilistic multipath traffic assignment model which obviates path enumeration. *Transportation Research*, 5(2):83–111, 1971.

[52] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Algorithm aversion: People erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology: General*, 144(1):114, 2015.

[53] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Overcoming algorithm aversion: People will use imperfect algorithms if they can (even slightly) modify them. *Management Science*, 64(3):1155–1170, nov 2016.

[54] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*, (Ml):1–13, 2017.

[55] Jack W. Dunn. *Optimal Trees for Prediction and Prescription.* PhD thesis, MIT, 5 2018.

[56] Finley Edwards. Early to rise? the effect of daily start times on academic performance. *Economics of Education Review*, 31(6):970–983, 2012.

[57] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least Angle Regression. *Annals of Statistics*, 32(2):407–499, apr 2004.

[58] Katia Fredriksen, Jean Rhodes, Ranjini Reddy, and Niobe Way. Sleepless in Chicago: tracking the effects of adolescent sleep loss during the middle school years. *Child development*, 75(1):84–95, 2004.

[59] Alex A. Freitas. Comprehensible classification models. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10, 2014.

[60] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* Springer series in statistics New York, NY, USA:, 2001.

[61] Armin Fügenschuh. Solving a school bus scheduling problem with integer programming. *European Journal of Operational Research*, 193(3):867–884, 2009.

[62] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[63] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining Explanations : An Approach to Evaluating Interpretability of Machine Learning. *arXiv preprint arXiv:1806.00069*, 2018.

[64] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *arXiv preprint*, pages 1–9, 2016.

[65] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019.

[66] Gabriel Gutiérrez-Jarpa, Guy Desaulniers, Gilbert Laporte, and Vladimir Marianov. A branch-and-price algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows. *European Journal of Operational Research*, 206(2):341–349, 2010.

[67] Marco Hafner, Martin Stepanek, and Wendy M. Troxel. Later school start times in the U.S.: An economic analysis. Technical report, RAND Corporation, 2017.

[68] Peter M Hahn, Bum-Jin Kim, Monique Guignard, J MacGregor Smith, and Yi-Rong Zhu. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications*, 40(3):351, 2008.

[69] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, October 2008.

[70] Flurin Hänseler, Nicholas Molyneaux, and Michel Bierlaire. Estimation of pedestrian origin-destination demand in train stations. To appear in *Transportation Science*, 2017.

[71] Hideki Hashimoto, Toshihide Ibaraki, Shinji Imahori, and Mutsunori Yagiura. The vehicle routing problem with flexible time windows and traveling times. *Discrete Applied Mathematics*, 154(16):2271–2290, 2006.

[72] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning Data Mining, Inference, and Prediction (12th printing)*. Springer, 2009.

[73] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[74] A. Hofleitner, R. Herring, P. Abbeel, and A. Bayen. Learning the dynamics of arterial traffic from probe data using a dynamic bayesian network. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1679–1693, Dec 2012.

[75] Mark Horn. Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C: Emerging Technologies*, 10(1):35–63, 2002.

[76] Cheng-Huang Hung. *On the Inverse Shortest Path Length Problem*. PhD thesis, Georgia Tech ISyE, 2003.

[77] Patrick Jaillet, Jin Qi, and Melvyn Sim. Routing optimization under uncertainty. *Operations Research*, 64(1):186–200, 2016.

[78] Patrick Jaillet and Michael Wagner. Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses. *Operations Research*, 56(3):745–757, 2008.

[79] Erik Jenelius and Haris N. Koutsopoulos. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological*, 53:64 – 81, 2013.

[80] Been Kim, Cynthia Rudin, and Julie Shah. The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification. In *Neural Information Processing Systems (NIPS) 2014*, 2014.

[81] I. Y. Kim and O. L. De Weck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and Multidisciplinary Optimization*, 29(2):149–158, 2005.

[82] Jon Kleinberg, Himabindu Lakkaraju, Jure Leskovec, Jens Ludwig, and Sendhil Mullainathan. Human decisions and machine predictions. *The quarterly journal of economics*, 133(1):237–293, 2017.

[83] Ali Koç and David P. Morton. Prioritization via Stochastic Optimization. *Management Science*, 61(3):586–603, 2014.

[84] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. Interpretable decision sets: a joint framework for description and prediction. *KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1:1675–1684, 2016.

[85] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. Interpretable & Explorable Approximations of Black Box Models. *FAT/ML*, jul 2017.

[86] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[87] Ruimin Li and Geoffrey Rose. Incorporating uncertainty into short-term travel time predictions. *Transportation Research Part C: Emerging Technologies*, 19(6):1006 – 1018, 2011.

[88] Guolong Lin and David Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal Computing*, 39(8):3633–3669, 2010.

[89] Zachary C. Lipton. The Mythos of Model Interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[90] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible Models for Classification and Regression. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–158. ACM, 2012.

[91] Miles Lubin and Iain Dunning. Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

[92] Shuo Ma, Yu Zheng, and Ouri Wolfson. Real-Time City-Scale Taxi Ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2015.

[93] Susan Kohl Malone, Terra Ziporyn, and Alison M Buttenheim. Applying behavioral insights to delay school start times. *Sleep Health: Journal of the National Sleep Foundation*, 3(6):483–485, 2017.

[94] Sebastien Martin. TaxiSimulation Julia Package. https://github.com/sebmart/TaxiSimulation, 2017. [Online; accessed 23-January-2018].

[95] Jo Craven McGinty. How do you fix school bus routes? call mit. The Wall Street Journal, 8 2017.

[96] Fei Miao, Shuo Han, Shan Lin, John A Stankovic, Desheng Zhang, Sirajum Munir, Hua Huang, Tian He, and George Pappas. Taxi Dispatch With Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach. *IEEE Transactions on Automation Science and Engineering*, 13(2), 2016.

[97] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[98] Eric Moulines and Francis R. Bach. Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning. In J Shawe-Taylor, R S Zemel, P L Bartlett, F Pereira, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 451–459. Curran Associates, Inc., 2011.

[99] Sendhil Mullainathan and Ziad Obermeyer. Does machine learning automate moral hazard and error? *American Economic Review*, 107(5):476–480, 2017.

[100] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust Stochastic Approximation Approach to Stochastic Programming. *SIAM Journal on Optimization*, 19(4):1574–1609, jan 2009.

[101] E. Nikolova and N. E. Stier-Moses. A mean-risk model for the traffic assignment problem with stochastic travel times. *Operations Research*, 62(2):366–382, 2014.

[102] NYC. New york city taxi & limousine commission - trip record data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml, 2017. Accessed: 2017-02-12.

[103] Dan O'Brien. MIT's "Quantum Team" wins first-ever BPS transportation challenge with revolutionary computer model. https://www.bostonpublicschools.org/site/default.aspx?PageType=3&DomainID=4&ModuleInstanceID=14&ViewID=6446EE88-σ_b^2D30C-$\sigma_b^2$497E-$\sigma_b^2$9316-$\sigma_b^2$3F8874B3E108&FlexDataID=12431, July 2017.

[104] Masayo Ota, Huy Vo, Cí Audio Silva, and Juliana Freire. STaRS: Simulating Taxi Ride Sharing at Scale. *IEEE Transactions on Big Data*, pages 1–1, 2016.

[105] Masayo Ota, Huy Vo, Claudio Silva, and Juliana Freire. A scalable approach for data-driven taxi ride-sharing simulation. In *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pages 888–897, dec 2015.

[106] Judith Owens, Darrel Drobnich, Allison Baylor, and Daniel Lewin. School start time change: An in-depth examination of school districts in the united states. *Mind, Brain, and Education*, 8(4):182–213, 2014.

[107] Judith A Owens, Katherine Belon, and Patricia Moss. Impact of delaying school start time on adolescent sleep, mood, and behavior. *Archives of pediatrics & adolescent medicine*, 164(7):608–614, 2010.

[108] Neal Parikh and Stephen Boyd. Proximal Algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[109] Junhyuk Park and Byung-In Kim. The school bus routing problem: A review. *European Journal of operational research*, 202(2):311–319, 2010.

[110] Junhyuk Park, Hyunchul Tae, and Byung-In Kim. A post-improvement procedure for the mixed load school bus routing problem. *European Journal of Operational Research*, 217(1):204–213, 2012.

[111] Parag A Pathak and Peng Shi. Simulating alternative school choice options in Boston - Technical appendix. Technical report, MIT School Effectiveness and Inequality Initiative, 2013.

[112] Andrei Patrascu and Ion Necoara. Nonasymptotic convergence of stochastic proximal point methods for constrained convex optimization. *Journal of Machine Learning Research*, 18(198):1–42, 2018.

[113] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés Medaglia. A Review of Dynamic Vehicle Routing Problems. *Cirrelt-2011-62*, pages 0–28, 2011.

[114] Michal Pióro, Yoann Fouquet, Dritan Nace, and Michael Poss. Optimizing flow thinning protection in multicommodity networks with variable link capacity. *Operations Research*, 64(2):273–289, 2016.

[115] B. T. Polyak and A. B. Juditsky. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, jul 1992.

[116] Jean-Yves Potvin and Jean-Marc Rousseau. An Exchange Heuristic for Routeing Problems with Time Windows. *The Journal of the Operational Research Society*, 46(12):1433–1446, 1995.

[117] Mohammed Quddus and Simon Washington. Shortest path and vehicle trajectory aided map-matching for low frequency GPS data. *Transportation Research Part C: Emerging Technologies*, 55:328 – 339, 2015.

[118] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[119] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, sep 1951.

[120] R. Tyrrell Rockafellar. Monotone Operators and the Proximal Point Algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, aug 1976.

[121] Ernest K Ryu and Stephen Boyd. Stochastic proximal iteration: a non-asymptotic improvement upon stochastic gradient descent. 2014.

[122] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–4, 2014.

[123] Patrick Schittekat, Marc Sevaux, and Kenneth Sorensen. A mathematical formulation for a school bus routing problem. In *Intl Conf on Serv Syst and Serv Mgmt*, volume 2, pages 1552–1557. IEEE, 2006.

[124] Johannes Schneider, Christine Froschhammer, Ingo Morgenstern, Thomas Husslein, and Johannes Maria Singer. Searching for backbones -an efficient parallel algorithm for the traveling salesman problem. *Computer Physics Communications*, 96:173–188, 1996.

[125] David Sharfenberg. Computers can solve your problems. you may not like the answer. The Boston Globe (data analysis by Sebastien Martin and Arthur Delarue), 9 2018.

[126] Rebecca Shuster. 2018-19 school bell times equity impact. https://www.bostonpublicschools.org/cms/lib/MA01906464/Centricity/Domain/2389/Equity%20Analysis.pdf, December 2017.

[127] Barbara M Smith and Anthony Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General*, 22(2):97–108, 1988.

[128] Michela Spada, Michel Bierlaire, and Th M Liebling. Decision-aiding methodology for the school bus routing and scheduling problem. *Transportation Science*, 39(4):477–490, 2005.

[129] Elizabeth A. Sullivan. Order on school start time realignment. https://www.bostonpublicschools.org/cms/lib/MA01906464/Centricity/Domain/162/starttimesmotion.pdf, December 2017.

[130] Jonathan Taylor and Robert J. Tibshirani. Statistical learning and selective inference. *Proceedings of the National Academy of Sciences*, 112(25):7629–7634, jun 2015.

[131] Robert J. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[132] Panos Toulis, Dustin Tran, and Edoardo M. Airoldi. Towards Stability and Optimality in Stochastic Gradient Descent. In Arthur Gretton and Christian C Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1290–1298, Cadiz, Spain, may 2016. PMLR.

[133] Berk Ustun and Cynthia Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3):349–391, 2016.

[134] Hongjian Wang, Zhenhui Li, Yu-Hsuan Kuo, and Dan Kifer. A simple baseline for travel time estimation using large-scale trip data. *CoRR*, abs/1512.08580, 2015.

[135] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 25:1—-25:12, 2013.

[136] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 25–34. KDD 2014, August 2014.

[137] Yinhai Wang and Nancy L Nihan. Freeway Traffic Speed Estimation Using Single Loop Outputs. *Transportation Research Record: Journal of the Transportation Research Board*, 1727(1):9, 2000.

[138] Colin Wenzel. Optimale schulanfangszeiten zur entlastung des nahverkehrs in der stadt nürnberg. *Angewandte Mathematik und Optimierung Schriftenreihe (AMOS)*, 2016.

[139] Anne G. Wheaton, Gabrielle A. Ferro, and Janet B. Croft. School start times for middle school and high school students - united states, 2011–12 school year. Technical Report Vol. 64 No. 30, CDC, August 2015.

[140] Greg Wiggan. Race, school achievement, and educational inequality: Toward a student-based inquiry perspective. *Review of Educational Research*, 77(3):310–333, 2007.

[141] K.I. Wong and Michael Bell. The Optimal Dispatching of Taxis under Congestion : a Rolling Horizon Approach. *Advanced Transportation*, 40(2):203–220, 2005.

[142] Zhihai Xiang, Chengbin Chu, and Haoxun Chen. A fast heuristic for solving a large-scale static dial-a-ride problem under complex constraints. *European Journal of Operational Research*, 174(2):1117–1139, 2006.

[143] Ci Yang. *Data-driven modeling of taxi trip demand and supply in New York City*. PhD thesis, Rutgers University, 2015.

[144] Hongyu Yang, Cynthia Rudin, and Margo Seltzer. Scalable Bayesian Rule Lists. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[145] Jian Yang, Patrick Jaillet, and Hani Mahmassani. Real-Time Multivehicle Truckload Pickup and Delivery Problems. *Transportation Science*, 38:135–148, 2004.

[146] Liwei Zeng, Sunil Chopra, and Karen Smilowitz. The covering path problem on a grid. *arXiv preprint arXiv:1709.07485*, 2017.

[147] Xianyuan Zhan, Samiul Hasan, Satish V. Ukkusuri, and Camille Kamga. Urban link travel time estimation using large-scale taxi data with partial information. *Transportation Research Part C: Emerging Technologies*, 33:37 – 49, 2013.

[148] Rick Zhang, Federico Rossi, and Marco Pavone. Routing Autonomous Vehicles in Congested Transportation Networks: Structural Properties and Coordination Algorithms. *Proceedings of Robotics: Science and Systems*, 2016.

[149] Tong Zhang and Tong. Solving large-scale linear prediction problems using stochastic gradient descent algorithms. In *Twenty-first international conference on Machine learning - ICML '04*, page 116, New York, New York, USA, 2004. ACM Press.