Queues, Planes and Games: Algorithms for Scheduling Passengers, and Decision Making in Stackelberg Games.

Sai Mali Ananthanarayanan

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2023

© 2022

Sai Mali Ananthanarayanan

All Rights Reserved

Abstract

Queues, Planes and Games: Algorithms for Scheduling Passengers, and Decision Making in Stackelberg Games.

Sai Mali Ananthanarayanan

In this dissertation, I present three theoretical results with real-world applications related to scheduling and distributionally-robust games, important fields in discrete optimization, and computer science.

The first chapter provides simple, technology-free interventions to manage elevator queues in high-rise buildings when passenger demand far exceeds the capacity of the elevator system. The problem was motivated by the need to manage passengers safely in light of reduced elevator capacities during the COVID-19 pandemic. We use mathematical modeling, epidemiological expertise, and simulation to design and evaluate our algorithmic solutions. The key idea is to explicitly or implicitly group passengers that are going to the same floor into the same elevator as much as possible, substantiated theoretically using a technique from queuing theory known as stability analysis. This chapter is joint work with Charles Branas, Adam Elmachtoub, Clifford Stein, and Yeqing Zhou, directly in collaboration with the New York City Mayor's Office of the Chief Technology Officer and the Department of Citywide Administrative Services.

The second chapter proposes new algorithms for recomputing passenger itineraries for airlines during major disruptions when carefully planned schedules are thrown into disarray. An airline network is a massive temporal graph, often with tight regulatory and operational constraints. When disruptions propagate through an airline network, the objective is to *recover* within a given time frame from a disruption, meaning we replan schedules affected by the disruption such that the new schedules have to match the originally planned schedules after the time frame. We aim to solve the large-scale airline recovery problem with quick, user-independent, consistent, and near-optimal algorithms. We provide new algorithms to solve the passenger recovery problem, given recovered flight and crew solutions. We build a preprocessing step and construct an Integer Program as well

as a network-based approach based on solving multiple-label shortest path problems. Experiments show the tractability of our proposed algorithms on airline data sets with heavy flight disruptions. This chapter is joint work with Clifford Stein, stemming from an internship and collaboration with the Machine Learning team (Artificial Intelligence organization) of GE Global Research, Niskayuna, New York.

The third chapter is about computing distributionally-robust strategies for a popular game theory model called Stackelberg games, where one player, called the leader, is able to commit to a strategy first, assuming the other player(s), called follower(s) would best respond to the strategy. In many of the real-world applications of Stackelberg games, parameters such as payoffs of the follower(s) are not known with certainty. Distributionally-robust optimization allows a distribution over possible model parameters, where this distribution comes from a set of possible distributions. The goal for the leader is to maximize their expected utility with respect to the worst-case distribution from the set. We initiate the study of distributionally-robust models for Stackelberg games, show that a distributionally-robust Stackelberg equilibrium always exists across a wide array of uncertainty models, and provide tractable algorithms for some general settings with experimental results. This chapter is joint work with Christian Kroer.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I first would like to thank my advisor Clifford Stein. Cliff shared his expert algorithmic knowledge generously and was an engaging mentor. I sincerely appreciate how he always treated me as a collaborator and also encouraged my leadership endeavors in the school. The freedom to choose problems and domains that interested me motivated me to work on a variety of sub-fields across the years.

I am grateful to my thesis committee, who share with me the joy of working on OR problems with real data and applications- Adam Elmachtoub, Christian Kroer, Nouri Sakr, and Augustin Chaintreau. Adam's attention to detail and constructive comments encouraged me a lot during our collaboration. He helped me understand how to think without getting lost in details and has been a great motivator. Christian was very enjoyable to work with, especially in a new field for me, and I am thankful for how our work has evolved over the years. I am grateful for helpful advice about research and otherwise from Nouri that motivated me when I first joined the IEOR department. Augustin has been an inspiration to me, and I got to know him during our efforts on diversity, equity, and inclusion in my time here. His work on algorithms for networks taking fairness into account appealed to me, and I thank him for being a part of my committee.

For Chapter 1, my co-authors and I thank the New York City Department of Citywide Administrative Services and Dr. Neal Parikh, Director of Artificial Intelligence in the Mayor's Office of the Chief Technology Officer, New York City, for data used in this work. The study was supported by an award under Columbia Engineering's "Urban Living Tech Innovations" initiative to develop technology innovations for urban living in the face of COVID-19. I was supported by Prof Cliff Stein's grant NSF CCF-1714818. We also thank Columbia University Data Science Institute and Sharon Sputz for fostering fruitful collaboration with the city government.

For Chapter 2, I thank my manager Dr. Paul Ardis (Research Platform Leader), colleagues Dr. Srinivas Bollapragada (Chief Scientist), Dr. Nitish Umang (Lead Scientist), and other members of GE Global Research, Niskayuna, New York for hosting me as an intern in the Machine

Learning team, and giving me the freedom and responsibility to tackle a large scale problem in airline scheduling that directly impacted a core product. I am grateful to my advisor Cliff Stein for guiding me to expand my contributions in this field with new network-based algorithms. Experiments on this chapter were made possible by IBM CPLEX academic license and datasets from ROADEF (French Operations Research and Decision Support Society) and Prof Lindsey McCarty of Cedarville University.

For Chapter 3, I thank Prof Christian Kroer for his fantastic course *Economics, AI, and Optimization* in Spring 2020. He spurred my interest in many topics, including Stackelberg games for the course project, which blossomed into a collaboration. Experiments on this chapter were made possible by Gurobi academic license and GAMUT, a suite of game generators designated for testing game-theoretic algorithms.

I am fortunate for my enjoyable and diverse experiences, especially through Engineering Graduate Student Council and other leadership forums. I thank Columbia Engineering for a memorable opportunity- I will always cherish having been the 2022 Engineering Class Day Speaker in front of thousands of fellow graduates, family, and friends. Thanks to all the peers, staff, and faculty at Columbia Engineering for their constant support and encouragement throughout my journey.

For all their love and support, I owe a huge gratitude to my family and friends.

Dedication

Seetha Paati, my grandma and first math teacher.

Introduction

In this dissertation, I present three results with real-world applications related to scheduling and distributionally-robust games, important fields in discrete optimization, and computer science. The motivations, background, and a summary of contributions are detailed below.

Broadly, scheduling involves making decisions about allocating goods/resources over time and optimizing some objective(s). Traditional scheduling problems can be summarized with a job and machine setting- there are a fixed number of jobs, and all relevant information about them is available at the beginning. There are resources available to be utilized (e.g., people in a factory, runways in an airport, computing power in a cluster, etc.), which may be subject to availability constraints. The tasks/jobs to be scheduled could have features- e.g., priority levels that reflect their importance, release time (i.e., earliest time a task could be scheduled), deadlines, etc. The goal is to compute from scratch an optimal schedule that minimizes or maximizes a given objective [1]. The objective could reward or penalize the proposed schedule based on a combination of the job features. A discussion on multiple types of objectives is presented in Pinedo [1] for two settings: (i) *deterministic*, where the features of the jobs and objectives are available in advance, and (ii) *stochastic*, where job data may not be known except for their distributions.

In practice, scheduling problems face interesting twists that call for developing new approaches that are domain-specific [1]. Scheduling algorithms are important in many fields, one among which is solving complex transportation problems. We are interested in two problems broadly in scheduling for passengers in real-world transportation, albeit with distinct flavors. The first chapter deals with scheduling in elevator systems, a crucial vertical transportation system for urban and building planners. The second chapter is about scheduling in airline networks, a complex network in time and space, with multiple objectives and tight constraints. We discuss in detail the different challenges posed by the two problems and our contributions below.

1. Scheduling problems in the real-world often have new restrictions that could be machine, job, or time-dependent. Instead of pre-planned schedules, in some settings, the decision makers could have crafted rules or algorithms that are followed as and when the jobs arrive [1]. One example is in the stochastic setting when only distributional information about the jobs is known (e.g., arrival time, job processing time). The rules are made initially based on information on the environment and resources. When a new restriction is introduced, the objective could worsen if we still follow the old rules. Note that because there are no pre-planned schedules, only rules, we want new algorithms that adapt to the changed environment when the jobs are dynamic. Theoretical research into general rules applicable to many different real-world problems with their own idiosyncrasies is scant [1]. We are concerned with previously unanticipated constraint(s) in the setting where the original scheduling rules break down, leading to an unacceptable value of the objective. One example of such a restriction could be the handling capacity or the processing speed of the machines, which might severely affect the machines being available for jobs down the line. Many theoretical models do not take machine availability constraints into account [1]. New algorithms that take break traditional norms of the problem (e.g., the processing order of the jobs) could do better in the new regime.

   In Chapter 1, we are interested in scheduling for elevator systems when the passenger demand for the elevators far exceeds the available capacity of the system. There are two settings where our work could be valuable- an appropriate number of elevators could have been installed and operated based on a prior forecast of passenger traffic in the building, and the standard social norm of first-come first-serve (FCFS) could have worked well. In the first setting, there could be an increased passenger demand in the building, or in the second, there could be a new restriction on the capacity of the elevator system. Either setting could lead to a mismatch between demand and supply, and the default social queuing norm for elevators (FCFS) may not work effectively. We propose new algorithms that break FCFS in order to get passengers to their destination at a faster rate in Ananthanarayanan *et al.* [2].

The work was motivated by the needs of high-rise buildings during the COVID-19 pandemic, when social distancing reduces the capacity of elevators, cutting the number of passengers per elevator by as much as 90% of the normal amount [3, 4, 5]. It is imperative to design interventions for people to stay safe in potentially crowded areas. Reduced elevator capacity can cause large lobby queues and long wait times, resulting in crowding and reduced social distancing [3, 6, 7], thereby posing significant safety risks. Thus, an intervention in the public health problem of safely managing queues for elevator systems amidst a pandemic is needed. One can broadly consider two major forms of interventions based on *(i)* changing passenger behavior and *(ii)* elevator artificial intelligence. A variety of technological innovations from elevator companies and building management have been considered during the pandemic [6]. In many elevator systems, especially in older ones, changing the algorithms and technology of how the elevators navigate through the building is either challenging or infeasible and would require long-term planning and expensive modifications. Thus, in order to safely manage how passengers use and board elevators, we focus on *technology-free* interventions in Ananthanarayanan *et al.* [2] that are more accessible and practical for an overwhelming majority of buildings.

In Chapter 1, we develop a general, open-sourced simulation model that captures many of the practical nuances in designing interventions for elevator systems and allows us to study the impact of various interventions or queuing behavior. We propose two interventions that use elevators more efficiently and analytically show their strong performance using a technique from queuing theory known as *stability analysis*. The data is calibrated from a real 25 floor New York City high-rise building that was scheduled for re-opening. While our work was motivated by social distancing rules during a pandemic, it is generalizable to any setting where the passenger demand exceeds the capacity of the elevator system, necessitating intervention. For instance, during high traffic time windows such as morning rush hour or lunch time, building managers could choose to implement the proposed interventions that get passengers to their destination at a faster aggregate rate (higher throughput).

2. An unexpected disruption affecting underlying resources may require recomputing or modifying pre-planned schedules. Restrictions can come up that introduce new constraints and require the decision-makers to react by undertaking large modifications to pre-planned schedules. One may check which jobs have not yet started and generate from scratch a new optimal schedule in the disrupted environment. However, the new schedule may be very different from the original, causing confusion in practice [1]. Thus it may be advantageous to keep changes in the current schedule at a minimum and only make minor changes to it when disrupted. *Proactive scheduling* calls for taking into account robustness during the planning stage, though anticipating the random events goes against the definition of encountering the *unexpected*. Therefore, we focus on *reactive scheduling*, i.e., replan schedules for jobs still in the queue during disruptions, with the new objective comprising of two terms- the original objective and a measure of closeness between the original and new schedules. We are interested in *recovery* algorithms, a reactive approach to cope with irregularities, in the sense that decisions are made when the original schedules are either unfeasible or carry a high cost in the new regime [8]. Schedule recovery problems typically use deterministic schedules and an unexpected random event as input and try to recover the now infeasible, or high-cost schedules, in order to minimize the difference between the new and original schedules.

In Chapter 2, we study an airline scheduling problem that has originally planned schedules that are disrupted. Minimizing costs for airline scheduling involves three competing schedules (for flights, crew, and passengers). A global optimal solution is difficult to compute owing to the complexity of the problem space [9, 10, 11]. Hence, we wish to compute schedules that are close to optimal but can also be solved within reasonable computational time and thus could be used during disruptions. Airlines make their schedules well in advance

[11], a well-studied problem in the literature. Our interest is in reacting to the disruption by replanning for jobs (passenger schedules in an itinerary) that are not complete. Almost half of all disruptions are reactionary and caused by the primary delay as explained in Papiomytis [12]. Thus the disruptions have a snowball effect on the airline network.

We focus on reactive scheduling for passengers in airline systems during disruptions. Thus, passenger schedules are to be recovered, given solutions to flight and/or crew recovery problems. This problem would fall under the partially integrated recovery model. *Holistic recovery* refers to solving the global optimization problem for flights, crew, and passengers all at once, and is challenging to solve in practice (see discussion in Filar *et al.* [10], Ball *et al.* [13], and Clausen *et al.* [14]). Moreover, airlines often separate decision-making on flights, planes, and crews to different teams, since it is widely believed that the entire holistic recovery scheduling problem is computationally intractable [8]. *Integrated recovery* refers to solving more than one of the three recovery problems (flights, crew, and passengers) together, combing the objectives and constraints. We are solving the global optimization problem sequentially- flights and/or crew first, followed by passengers. Passenger recovery costs are the lowest, and its constraints less complex among the three problems and hence is solved last. Of course, such a sequential decision-making model can lead to inefficiencies in solving the global optimization problem. Passenger recovery is an active field of research and of great interest to airlines at the moment [15]. High passenger delay cost and continuous flight disruptions lead to a potential loss of goodwill and long-term reputation damage [16]; hence in this chapter, we use the flight recovery solution as input and aim for a quick and high-quality solution for passenger recovery.

Passenger recovery can be formulated as follows: given recovered flight schedules, and a set of pre-planned passenger itineraries, the set of itineraries where at least one of its flights is disrupted in some way (delays, cancellations, reduced cabin capacity, etc) is called disrupted itineraries. We wish to replan all disrupted itineraries by making decisions on whether passengers can be moved to different flights and/or cabin classes or outright cancel their itinerary. For each disrupted itinerary, we use the recovered flights (given seat availability) necessary to re-accommodate passengers from their starting position at the time of disruption to their destination while minimizing cost [15, 9].

The passenger recovery costs can include both *operating costs* and *disutility costs*. Operating or regulatory costs are directly incurred (and paid in real money) when a passenger cannot complete their scheduled itinerary (e.g., compensation for delay and cancellation, providing refreshments or hotels, as stipulated by government regulations). Disutility costs are the potential losses of future revenue as a result of passenger inconvenience, possibly causing the passenger to switch to a different airline in the future. The costs are approximations made by the airline, can differ per passenger class or frequent flyer status [15, 11], and are meant to be used for reactive decision-making to disincentivize certain outcomes. For example, the disutility for canceling a passenger's itinerary is set to be higher than delaying their itinerary by the maximum allowed delay. The model would thus rather delay than cancel itineraries whenever possible.

In Chapter 2, we construct an Integer Program (IP) to solve passenger recovery using pre-processed graphs pruned for each itinerary. This work was initiated during my internship at GE Global Research with real airline data sets. The sizes of IP may grow huge with real airline data (e.g., the number of integer variables alone could scale linearly with the number of flights, size of passenger itineraries, etc.), and hence there is a need for alternate techniques. Moreover, certain practical constraints are not easy to handle without introducing non-linearity. There have been efforts to construct network-based algorithms (see Palpant *et al.* [15], Bisaillon *et al.* [11], and Righini and Salani [17]) and we contribute a new algorithm without mathematical programs, based on solving multiple-labels shortest path problems. Our multiple-labels shortest path algorithm prunes labels regularly, is iterative, and solves for the passengers who are more high-value (typically decided by decreasing order of cancellation costs of their itineraries). We show the performance of our algorithms on data calibrated from a publicly available data set from an OR challenge on integrated operations and passenger recovery, modeling light, and heavily disrupted scenarios.

Thus, Chapters 1 and 2 are two flavors of real-world passenger transportation problems requiring new scheduling algorithms. For Chapter 1, dispatching rules and other algorithms on managing elevator systems are well-studied (see Barney and Al-Sharif [18], Fujino *et al.* [19], Barney and Dos Santos [20], Lee *et al.* [21], and Al-Sharif *et al.* [22]), but there is not much work on the impact of reducing capacities while maintaining service quality. To our knowledge, our paper Ananthanarayanan *et al.* [2] is the first of its kind, applicable to general building types with random arrival and destination patterns. For Chapter 2, airlines have decades of research on planning schedules well in advance [23], however, the field of recovery is still quite new, with a multitude of open problems in integrated recovery. Our work on passenger recovery is novel, going beyond mathematical programs by working directly on the temporal graph with network-based algorithms.

In Chapter 3, we introduce new algorithms for computing optimal distributionally-robust strategies in a game theory setting with real-world significance called *Stackelberg games*. In multiagent systems, self-interested (as is the case in most economic settings) agents or players have their payoffs linked to the actions of the other players. Many systems assume that the players choose their strategies simultaneously [24]. However, such a model is not always realistic. In many settings referred to as *leadership, commitment*, or *Stackelberg* models, one player commits to a strategy before the other player, and we assume the second player would best respond to whatever strategy the first player picks. Real-world examples include airport security [25], defense against poaching and illegal fishing [26] etc. where the leader (or defender) strategy refers to making patrolling schedules or placing checkpoints, and the followers can surveil the leader strategy over time and plan a best-response in order to attack the locations.

The feature in the Stackelberg game model is that the players are *asymmetric*, and they act in turns: the leader first plays, then the follower sees the leader's action and then adapts to it [27]. Symmetric solution concepts like Nash equilibrium introduced in Nash [28] do not apply to this setting. Optimal strategies in Stackelberg models are significantly different from the model with simultaneous play [24]. A more natural solution concept for these games is the *Stackelberg equilibrium*: the leader commits to an optimal strategy, assuming the followers play their best response to the leader. Note that in standard Stackelberg games, the leader's optimal commitment is not necessarily a best response to the follower's response [29]. The Stackelberg model has found widespread application, particularly in security problems [30]. In the security setting, the goal is for the leader to compute an optimal strategy to commit to, in order to protect some asset.

In many of the real-world applications, the opponent follower payoffs are not known with certainty. Such uncertainty in parameters could occur due to limited scope in observable data, noise or prediction errors (see Pita *et al.* [25], Tsai *et al.* [31], and Fang *et al.* [26]). In Bayesian Stackelberg games, we assume publicly-known distribution over utility functions, and computing an optimal strategy for the leader even with a finite set of follower types is NP-hard [24]. A robust optimization approach is computing an optimal leader strategy given that the worst-case follower utility will be selected from an uncertainty set, and then the follower best responds based on this utility function [32]. However, robust optimization can often lead to overly conservative solutions, due to considering the worst-case nature over a potentially large uncertainty set.

Distributionally-robust optimization (DRO) addresses this issue by allowing a distribution over possible model parameters, where this distribution comes from a set of possible distributions. The goal is to maximize the expected utility with respect to the worst-case distribution. We initiate the study of distributionally-robust models for computing the optimal strategy to commit to. We consider normal-form games with uncertainty about the follower utility model. Our main theoretical result is to show that a distributionally-robust Stackelberg equilibrium always exists across a wide array of uncertainty models. In Stackelberg models, a follower could have multiple best responses for a given leader strategy. We use a tie-breaking rule that breaks the tie in favor of the leader, referred to as *strong* Stackelberg equilibrium in literature [33]. For the case of a finite set of possible follower utility functions we present two algorithms to compute a distributionally-robust strong Stackelberg equilibrium (DRSSE) using mathematical programs. Next, in the general case where there is an infinite number of possible follower utility functions and the uncertainty is repre-

sented by a Wasserstein ball around a finitely-supported nominal distribution, we give an incremental mixed-integer-programming-based algorithm for computing the optimal distributionally-robust strategy. Experiments substantiate the tractability of our algorithm on a classical Stackelberg game called Inspection game, showing that our approach scales to medium-sized games. We also discuss the special case of finite follower utility functions in the Wasserstein setting, with a mixed integer program and experiments on classical Stackelberg games as well as synthetic data.

Chapter 1: Queuing Efficiently in Elevator Systems

In this chapter, we present our work on passenger scheduling algorithms for elevator systems when passenger demand far exceeds the capacity of the system. This work is based on the article Ananthanarayanan *et al.* [2] written in collaboration with Charles Branas, Adam Elmachtoub, Clifford Stein and Yeqing Zhou, published in the *Production and Operations Management* journal in 2022.

1.1 Background and Motivation

We are interested in scheduling for elevator systems when the passenger demand for the elevators far exceed the available capacity of the system. For building and urban planners, vertical transportation solutions are a crucial problem to consider when designing high rise buildings [18]. In the planning stage, the characteristics of an elevator system such as number of elevators, speed, capacity of each elevator etc. are chosen operated based on a prior forecast of passenger traffic in the building. Thus, elevators installed and operated after careful planning, and many systems only expect passengers to follow the standard social norm of first-come first-serve (FCFS) [34, 20, 18].

In practice, the elevator systems can face new challenges that disrupt the ability to carry passengers to their destinations effectively. Firstly, there could be an increased passenger demand in the building, or secondly, there could be a new restriction on the capacity of the elevator system. Both settings lead to a mismatch between passenger demand and capacity of the system, and make wait times and queue lengths longer for the passengers using the elevators of the building. Below we explain the genesis of our work, motivated by the second scenario where a sudden restriction on elevator capacities are imposed.

The COVID-19 pandemic has made it imperative to design interventions for people to stay safe in potentially crowded areas. For high-rise buildings, social distancing reduces the capacity of elevators, cutting the number of passengers per elevator by two-thirds or as much as over 90% the normal amount [3, 4, 5]. Reduced elevator capacity can cause large lobby queues and long wait times, resulting in crowding and reduced social distancing [3, 6, 7, 35]. With no interventions and reduced capacity on elevators, the increased waiting times and queue lengths in the lobby could pose significant safety risks. Thus, an intervention to the public health problem of safely managing queues for elevator systems amidst a pandemic is needed (our team is composed of operations researchers and an epidemiologist). *In fact, this project was directly in collaboration with the NYC Mayor's Office of the Chief Technology Officer and the Department of Citywide Administrative Services. These offices had continuous input into our work throughout the process and allowed us to conduct several on-site visits with building managers, where we talked to frontline staff and gathered input. We also presented our findings multiple times to a variety of agency staff, developed an instructional video [1], and provided open source code that can be tailored to the needs of different types of buildings[2].*

One can broadly consider two major forms of interventions based on *(i)* changing passenger behavior and *(ii)* elevator artificial intelligence. A variety of technological innovations from elevator companies and building management have been considered during the pandemic [6]. In many elevator systems, especially in older ones, changing the algorithms and technology of how the elevators navigate through the building is challenging or infeasible, and would require long-term planning and expensive modifications. Thus, in order to safely manage how passengers use and board elevators, we focus on *technology-free* interventions which should be more accessible and practical for an overwhelming majority of buildings with elevators [2]. We specifically provide a detailed simulation study in addition to theoretical results of queuing systems to model and assess the efficacy of our various interventions.

Currently, many elevator systems take a hands-off approach to managing the flow of people to elevators, resulting in something that resembles first-come first-serve [19]. Our simulations, using

[1] `https://youtu.be/5KvX7_WNGFw`

[2] `https://github.com/saimali/elevators`

data calibrated from a large New York City government building, show that such a hands-off approach will lead to large and unsafe queues if building occupancy returns to pre-pandemic levels while elevator capacities are still reduced due to social distancing. Thus, it is imperative that we design interventions that use the elevators more efficiently – getting passengers to their destination at a faster aggregate rate (higher throughput) – by more carefully managing who uses which elevator when. For instance, we shall consider interventions where we try to get passengers going to the same or nearby floor to ride an elevator together as well as interventions where passengers are encouraged to walk up or down a floor after riding the elevator.

1.1.1 Contributions

Using mathematical modeling, epidemiological expertise, and simulation, we design and evaluate simple interventions to load passengers in elevators that can drastically reduce the length of lobby queues. The proposed interventions increase efficiency of the elevator system, and are effective beyond the constraints imposed by a pandemic, making them useful even after the pandemic to manage lobby queues. Our interventions could be applied to any building where passenger demand and available elevator capacity have a mismatch. The interventions do not require programming the elevators, and rely on using only signage and/or a queue manager (QM) to guide passengers. They are effective in reducing both the number of stops per trip and the travel distance per trip. Next, we outline our contributions in detail.

1. We develop a general, open-sourced simulation model that captures many of the details of elevator systems and allows us to study the impact of various interventions and queuing behavior. Our simulation model allows us to specify the number of elevators, capacity, elevator speed, and boarding times, and to measure and visualize the queue length and wait time of elevator systems for the various interventions we consider. We primarily focus on a case study calibrated by data from a large government building in New York City that is in need of managing elevator traffic amidst the COVID-19 pandemic.

2. We propose an intervention we call *Cohorting*, in which we attempt to find any and all pas-

sengers going to the same floor as the first person in the queue. Simulations show Cohorting reduces waiting time for passengers and the number of people in the lobby (queue length) significantly. In limited lobby spaces, we recommend the *Cohorting with Pairing* intervention, where we pair passengers going to the same floors. Pairing is practically easier to implement as it only requires matching two people at a time (rather than four, for instance). We also explore the impact of some passengers' willingness to walk up or down one floor from their destination. The queue length can be further reduced if just a small fraction of passengers are willing to walk.

3. We also propose the *Queue Splitting* intervention where we create a different queue for different groups of floors and load the elevators from queues in a round-robin fashion. The travel time of elevators is naturally reduced since passengers are likely to be going to the same or nearby floors. Queue Splitting requires less management efforts comparing to Cohorting, and splitting into just two groups achieves comparable performance to Cohorting in our case study.

4. We analytically investigate the reason behind the strong performance of Cohorting and Queue Splitting using a technique from queuing theory known as stability analysis. Specifically, we characterize the system parameters required for each intervention to ensure that the queues do not increase in length over time, i.e., the queues are stable. Our theoretical analysis reveals that these interventions can effectively reduce the average distance traveled and the number of stops per elevator trip.

1.1.2 Related Literature

Although researchers have studied algorithms for managing elevator systems [20, 18, 21, 19, 36, 22], to the best of our knowledge, there is not much literature on designing elevator systems with pandemic safety considerations.

Our simulation model is based on queuing theory. A discrete event simulation [37] models the operation of a system as a sequence of events in time, thus we utilize a detailed simulation to estimate mean wait times and queue lengths. Previous works that utilize queuing theory in elevator systems [38, 18, 39], and papers that use simulation for elevator traffic studies [40, 41] do not consider the impact of reducing capacities while maintaining service quality. A recent work Swinarski [4] in the context of the COVID-19 pandemic models and predicts elevator traffic in an university classroom building when passengers mainly travel in the pre-determined short time periods between two classes, with known class schedules and traffic patterns. The authors discuss an intervention which directs passengers to sort themselves into pairs of passengers with a shared destination floor which can improve the performance, but is not as effective as moving classes to lower floors or staggering course start times. In this work, we consider general building types with random arrival and destination patterns. In Mulvany and Randhawa [42], the authors consider breaking FCFS rules in exchange for fairness considerations, whereas we break FCFS in exchange for safety reasons and improved performance.

The stability analysis in this work is built upon literature in queuing and stochastic processing networks [43, 44]. The interventions in the work are designed specifically for an elevator system, while they share similarity to some well-studied dispatch policies in multiclass queuing networks. The Cohorting intervention resembles the first-in-first-out dispatch policy [45] if we consider passengers going to the same floor as a class and an empty server always picks a class whose head-of-line job arrived first. The Queue Splitting intervention is essentially a Round Robin dispatch policy in the multiclass queuing literature, which is related to fair queuing policies widely studied in the computer network literature (see, for example Demers *et al.* [46] or Parekh and Gallager [47]).

1.1.3 Organization

We introduce the simulation model of the elevator system in Section 1.2 and present the simulation results in Section 1.3. Results for the Allocation intervention are in Section 1.4. Section 1.5 analyzes the stability condition for each intervention we propose and compares the results to

FCFS. In Section 1.6, we discuss some practical issues and solutions for the Cohorting intervention. Finally, we conclude and discuss ideas for future work in Section 1.7.

1.2 Simulation Model

In this section, we describe our modeling framework. In particular, the model considers moving passengers upwards through a building from a lobby, which presents the biggest challenge for social distancing in a high-rise building. We study low-tech solutions (requiring no programming of elevators and no knowledge of internal elevator algorithms) and describe interventions to manage the queue of passengers in the lobby. We focus on analyzing solutions that work for high volume periods, e.g. morning rush hour, lunchtime, etc. where social distancing is a challenge. These busy periods are referred to as *uppeak* [20] and typically an elevator system working efficiently during the morning uppeak can handle interfloor traffic and downpeaks without any issues [18]. Below we describe the model we use in the simulation. We will simplify some of the assumptions when deriving the analytical results in Section 1.5.

We model a building as having a lobby on floor 1, m destination floors denoted $2\ldots, m+1$, and N elevators denoted $1, \ldots, N$. We assume passengers wanting to go to floor j at time t arrive at the lobby according to a non-stationary Poisson process with arrival rate $\lambda_j(t)$. The Poisson assumption for individual arrivals is considered a good approximation to the arrival process [18, 38]. Each of the N elevators has a capacity of C, the number of people that the elevator can safely transport while ensuring social distancing.

Remark 1 (Safe Capacities)**.** *The capacity C of the elevators should be set based on the physical dimensions of each elevator. Social distancing needs to be taken into account to put floor markers for passengers to stand inside an elevator, e.g., opposite corners of a diagonal for loading two people or all corners for loading four people. In general, there is a fundamental trade-off between setting a lower elevator capacity and increased queues in the lobby.*

In many high-rise buildings, elevators are constrained to certain floors so we let $S(n)$ denote the set of destination floors that elevator n can serve. If there is no restriction on the service range of the elevator n, then $S(n) = \{2, \ldots, m+1\}$. We assume the elevator travel time per floor ν is constant and the (de)boarding times of the elevator are a function of the number of people k that are (de)boarding, denoted by $BoardingTime(k)$. The (de)boarding time $BoardingTime(k)$ is a constant time ω to open and close the elevator door, and additional time depending on the number of passengers k entering (exiting). For our theoretical results in Section 1.5, we assume that the service time only depends on ν and ω. But for the purpose of creating a realistic simulation tool we describe the elevator dynamics in greater detail.

The travel time to start at floor j_1 and stop at floor j_2 is $T(j_1; j_2) = \nu(j_2 - j_1)$. If k (at most C) passengers with destinations $d_1 \leq d_2 \ldots \leq d_k$ board an elevator n at the lobby, we can create a count $\vec{F} = \{F_2, \ldots, F_{m+1}\}$ of the number of passengers deboarding at each floor. Note that $\sum_{j=2}^{m+1} F_j = k$, the number of passengers boarding at the lobby. The floor $H := d_k$ is typically referred to as the highest reversal floor in the literature [21] and is useful in calculating the round trip time. We also need to approximate inter-floor traffic (including down traffic), which we do by using an estimated multiplier β (e.g., $\beta = 1.3$ which means it takes 30% longer down) from the time the elevator takes to drop off the last passenger. Then the ascent time $AscentTime(\vec{F})$ without accounting for stops, is given by $AscentTime(\vec{F}) = T(1; d_k)$, the time spent making stops is $StopTime(\vec{F}) = \sum_{j=2}^{m+1} BoardingTime(F_j)$, and the descent time from when the last passenger has deboarded is $DescentTime(\vec{F}) = \beta \times AscentTime(\vec{F}) = \beta T(1; d_k)$. The time to board at the lobby is $BoardingTime(\vec{F}) = BoardingTime(\sum_{j=2}^{m+1} F_j)$. Thus the total round trip $RoundTripTime(\vec{F})$ time of an elevator is

$$RoundTripTime(\vec{F}) = BoardingTime(\vec{F}) + AscentTime(\vec{F}) + StopTime(\vec{F}) + DescentTime(\vec{F}). \tag{1.1}$$

To measure the performance of different interventions in the simulation, we consider the following metrics: average waiting time of a passenger at the lobby (measured at every time unit), average number of passengers at the lobby (queue length, measured at every time unit), and the average round trip time of elevators. We also explore qualitative considerations like human and material resources needed, ease of understanding for managers and passengers, and perceived inequity [34, 48] (e.g., when an intervention lets some passengers skip ahead of others).

We want to mathematically characterize the performance of the system. Let W_i denote the wait time for passenger i. Let $N(t)$ denote the number of people waiting in the lobby at time t. In a classic service system, the traditional goal is to minimize the total (average) expected wait time, i.e., $\mathbb{E}[\sum_i W_i]$. However, the primary objective in the context of a pandemic is to maximize safety, which corresponds to minimizing the number of people in the lobby that are waiting for an elevator. Metrics of interest are the expected number of passengers in the lobby over a time horizon of T periods, $\frac{1}{T}\int_0^T \mathbb{E}[N(t)]dt$, and the maximum queue length, $\max_t N(t)$.

Finally, we describe our system dynamics in the simulation. Passengers arrive at the (1st floor) lobby according to the Poisson process $\lambda_j(t)$ defined above and queue in a line or multiple lines, depending on the intervention being implemented. When an elevator is available at the lobby (either there is a free elevator already or passengers wait for an elevator to arrive), it is loaded according to the rules of the intervention, up to the capacity limit C of the elevator. The logic of each intervention can be found in Subsection 1.2.1. At constant intervals (Δt seconds), we update the system by loading available elevators in the lobby with passengers already in line(s) using the rules of the intervention. Once loaded, the elevators make stops corresponding to destinations of the passengers, and finally come back to the lobby to be loaded again.

We shall refer to an *instance* as the sequence of random passenger arrival times and destinations generated during one simulated rush hour morning. We record all quantitative metrics listed above. We simulate 100 independent random instances for every set of parameters and report the average performance for each metric. The code for the simulations is publicly available online.

1.2.1 Interventions

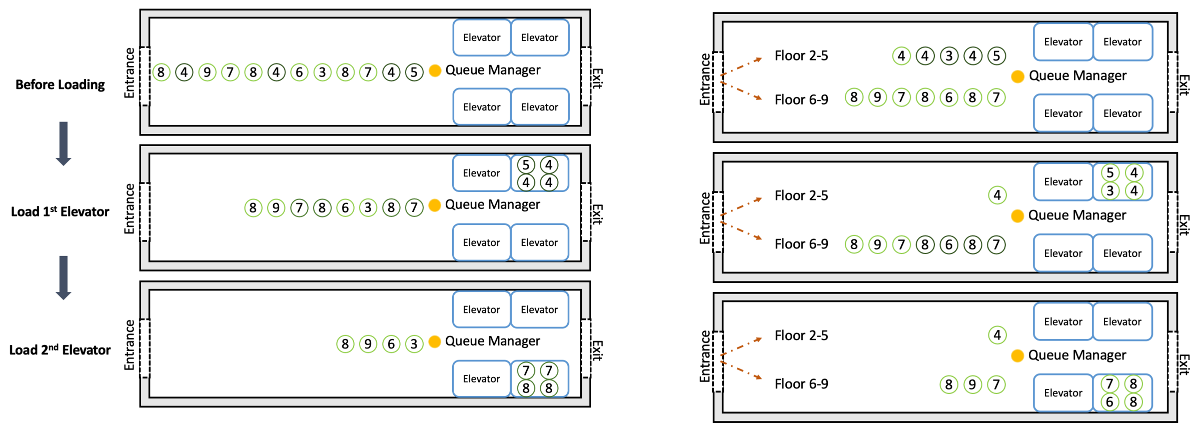
The standard way most elevator systems operate is akin to first-come first-serve (FCFS). However, moving towards safe interventions requires moving away from FCFS, which means that some people may be allowed to "cut in line" in order to decrease queue lengths and waiting times, while serving as many passengers as possible. Our interventions may rely on a queue manager (QM) for implementation, where the QM can be thought of as a personnel or a device with a screen.

First, we discuss the status quo - *FCFS* - where the passengers who arrive at the lobby first will enter an elevator first. FCFS for elevator loading follows the standard social norm of queuing. There are obvious advantages for using the status quo, as it ensures fairness and requires no management of the queue. However, even pre-COVID – especially during rush hours such as morning and lunchtime – the lobby may be crowded with passengers, elevators are fully loaded and they may make many stops during a trip. With a social distancing rule during a pandemic such as COVID-19, the dramatically reduced elevator capacity could cause a severe increase in congestion in the lobby and thus increase the risk of disease spread.

Next, we propose the intervention which we call *Cohorting*, which seeks to group together passengers going to the same floor. In this intervention, passengers line up in a queue in order of arrival. When an elevator arrives, the first passenger boards. Then, the QM asks if anyone in the queue is going to the same floor as the first passenger and then they board as well (according to their arrival order). This creates a cohort going to the same floor (such passengers are allowed to "cut in line"). If there is still capacity in the elevator, then the passenger at the front of the queue enters and the QM again allows passengers going to the same floor to board the elevator. This process is repeated until the elevator is full or the queue is empty. See Algorithm 1 in Section 1.2.2 for detailed simulation pseudocode. Cohorting is the best-performing intervention to improve efficiency (as seen in Section 1.5), but requires a QM to interact effectively with the queue to learn where passengers are going. It may be difficult for the QM to know the destinations of passengers that are far back in the queue. Thus, in Section 1.6 we consider easier to implement variants where we can only communicate with the first few people in the queue and where we only try to cohort in pairs.

The next intervention we propose is *Queue Splitting*, where we form a separate queue for disjoint groups of floors. In Queue Splitting, floors are assigned to different groups, where each group consists of consecutive floors, e.g., $2-8$ and $9-16$. We create a queue corresponding to each floor group. Arriving passengers join a queue corresponding to their floor group, and elevators are boarded from the queues in a round robin fashion (possibly with the help of a QM). For instance, there can be 4 queues, each corresponding to 6 floors. When any elevator arrives, one of the queues sends the first C passengers in line to it. If there are less than C in the queue, then the next queue sends passengers and so on. The queues are chosen in a round-robin fashion (or in a way to dynamically balance the length of each queue). See Algorithm 2 in Section 1.2.2 for detailed simulation pseudocode. By creating queues for every floor group, the travel time of elevators is naturally reduced since passengers are likely to be going to the same or nearby floors, which achieves an effect similar to Cohorting. The number of stops is also reduced compared to FCFS by grouping passengers in a limited floor range. This intervention does not require any programming of the elevator system, only requiring organizing the lobby space. A schematic showing the implementation of Cohorting and Queue Splitting is shown in Figure 1.1.

Figure 1.1: Illustrating the Cohorting and Queue Splitting interventions.



(a) Cohorting **(b)** Queue Splitting

Note. Passengers enter from the left and are guided by the Queue Manager to the elevators. Those exiting the elevator leave the building on the right, to ensure social distancing from entering passengers. In this example, the QM is loading an elevator in the lobby. We indicate the passengers who will board the next elevator using dark green circles. Under the Cohorting intervention in (a), the first elevator will stop at two floors, which are the destinations of the first and second passenger. Under the Queue Splitting intervention in (b), the QM is first loading from the queue for floor 2-5, and thus the elevator will stop at 3 floors.

Finally, we discuss the *Allocation* intervention, where each elevator is assigned to only go to predetermined floors. This intervention can be accomplished by changing the elevator control system, or simply by adding signs on each elevator door. We propose several floor allocation interventions, including partitioning into ranges of floors, or splitting into odd and even floors. For instance, one building in our case study has 14 elevators and 24 destination floors to serve. We can split the 14 elevators into 2 groups of 7, where each group goes to 12 floors. Another possibility is to split into odd and even floors which may encourage people to use one level of stairs to reach their final destination. The key intuition behind the allocation intervention is that each elevator, or each set of elevators is only serving a small range of floors. By doing so, the chances of two people in the same group going to the same or nearby floor increases, compared to FCFS. Thus Allocation has an effect that resembles Cohorting and Queue Splitting, leading to a reduction in travel and deboarding time. It is also perhaps the easiest intervention to implement. In fact, many buildings are using Allocation to create a separation of high floors and low floors in practice, where some of the elevators only serve high floors and the others serve low floors. In this case, Cohorting and Queue Splitting can also be applied in addition to the Allocation intervention (we provide simulation results in Section 1.4.2). We also note that Queue Splitting and Allocation are less likely to be perceived as unfair, as no one visibly cuts the line although it is possible that passengers do not board in FCFS order. In this work, we focus on the performance of Cohorting and Queue Splitting because Allocation does not perform well compared to the other two interventions. We discuss later the performance of the Allocation intervention in Section 1.4.

1.2.2 Algorithms for the proposed interventions

In this subsection, we describe the algorithms to evaluate the proposed interventions. For all algorithms, we simulate a passenger arrival sequence over a time horizon T as an input file, and update the evolution of the system every Δt seconds. Denote $P(t)$ as the list of passengers that arrive before time t. In the Queue Split intervention, we have k queues, and a queue index I indicates from which queue we should load in a round-robin fashion. $I \rightarrow I + 1$ denotes the transition to the next queue, and specifically, the $(k + 1)$-th queue is equivalent to the first queue. The set of destinations of the I-th queue is denoted as D_I.

Algorithm 1: Cohorting

```
1  $t = 0$ // current time
2  $Q = \emptyset$ // current queue
3  $\vec{F} = \vec{0}$ // number of passengers of an elevator deboarding at each
       floor
4  $E = \emptyset$ // set of empty elevators in the lobby
5  while $t < T$ do
6      $t = t + \Delta t$
7      Update the current queue $Q = Q \cup P(t) \backslash P(t - \Delta t) := \{p_1, p_2, \dots, p_l\}$, where $l$ is the length
         of current queue and $p_1$ is the first passenger in the queue
8      Record queue length $N(t) = l$
9      Update the elevators in lobby $E = E \cup \{e : ReturnTime(e) \in [t - \Delta t, t)\}$
10     while there exist elevators in lobby and there are passengers waiting in the lobby do
11         $e$ is the first elevator in $E$
12         while there exist capacity in $e$ and there are passengers waiting in the lobby do
13             Update the current queue, $l$ is the length of current queue and $p_1$ is the first passenger
                 in the queue
14             $leader = p_1$
15             Remove $p_1$ from $Q$ and record wait time $W_1$
16             Update $F$ according to $p_1$'s destination
17             $i = 2$
18             while there exists remaining capacity in the elevator and $i \leq l$ do
19                 if destination of $p_i$ is the same as leader then
20                     $p_i$ enter the current elevator and record wait time $W_i$
21                     Remove $p_i$ from $Q$
22                     Update $\vec{F}$ according to $p_i$'s destination
23                 else
24                     $i \rightarrow i + 1$
25                 end
26             end
27         end
28         Update $ReturnTime(e) = t + RoundTripTime(\vec{F})$
29         Remove elevator $e$ from $E$
30         Update $\vec{F} = \vec{0}$
31     end
32 end
```

Algorithm 2: Queue Splitting (k queues)

```
1  t = 0 // current time
2  $Q_i = \emptyset$ for $i = 1, \ldots, k$
3  $\vec{F} = \vec{0}$ // number of passengers of an elevator deboarding at each
       floor
4  $E = \emptyset$ // set of empty elevators in the lobby
5  $I = 1$ // start from the first queue
6  **while** $t < T$ **do**
7      $t = t + \Delta t$
8      Update the current queues $Q_i = Q_i \cup \{p : p \in P(t) \backslash P(t - \Delta t), p$'s destination $\in D_i\}$ for
         $i = 1, \ldots, k$
9      Record the total queue length $N(t)$;
10     Update the elevators in lobby $E = E \cup \{e : ReturnTime(e) \in [t - \Delta t, t)\}$
11     **while** *there exist elevators in lobby and there are passengers waiting in the lobby* **do**
12         $e$ is the first elevator in $E$
13         $RemainCap = C$
14         **if** *there are at least $C$ passengers in queue $Q_I$* **then**
15             Load the elevator with the first $C$ passengers in $Q_I$, remove from $Q_I$, record wait
                 time, and update $\vec{F}$
16             $I \to I + 1$
17         **else**
18             Load the elevator with all passengers in $Q_I$, remove from $Q_I$, record wait time, and
                 update $\vec{F}$
19             $RemainCap = C - |Q_I|$
20             $I \to I + 1$ // try to load the current elevator from the next
                   queue
21             **while** *there exists remaining capacity in elevator e and there are passengers in queue
                 $Q_I$* **do**
22                 Load the elevator with up to $RemainCap$ passengers in $Q_I$, remove from $Q_I$,
                     record wait time, and update $\vec{F}$
23                 $RemainCap = RemainCap - |Q_I|$
24                 $I \to I + 1$
25             **end**
26         **end**
27         Update $ReturnTime(e) = t + RoundTripTime(\vec{F})$
28         Remove elevator $e$ from $E$
29         Update $\vec{F} = \vec{0}$
30     **end**
31 **end**
```

1.3 Simulation Results

In this section, we describe our findings via simulation from three examples corresponding to a small, medium, and large building. The discussions in this section are primarily centered around the large building, and the small and medium buildings are discussed in Section 1.3.3. The large building is calibrated using data from a large government building in New York City that is planning for re-opening and urgently needs to manage elevator traffic amidst the COVID-19 pandemic. It is a historical building with a legacy elevator system, so only technology-free solutions can be implemented. Moreover, this building is heavily used and had more than 5500 people (staff and visitors) accessing it on a pre-pandemic day during the rush hour. The building has 25 floors and the 28 elevators are split into two elevator banks (North and South). Without loss of generality, we consider the South bank, where 14 elevators serve about 2750 visitors during the morning rush hour from 8 AM to 10 AM ($N = 14, m = 24$). In the two-hour period, we assume the arrival process is a stationary Poisson process with an arrival rate $\frac{2750}{7200}$ passengers per second. Arriving passengers are equally likely to go to any of floors 2 through 25. Based on the physical dimensions of the elevators, the capacity is $C = 4$. Every elevator n serves all floors, i.e., $S(n) = \{2, \ldots, 25\}$. It take 15 seconds for one passenger to (de)board, and an additional 2 seconds per extra passenger. Thus $BoardingTime(k) = 15 + 2(k - 1)$ seconds for k passengers in an elevator. The elevators have a constant travel time per floor of 1.4 seconds/floor, hence the time to travel from floor j_1 to floor j_2 is $T(j_1; j_2) = 1.4(j_2 - j_1)$ seconds. The speed multiplier β to account for inter-floor traffic is $\beta = 1.3$.

We summarize in Table 1.1 the parameters used in our simulations throughout the paper. The parameters can be easily customized to any building in our simulation. The code for the simulation is publicly available online[3].

[3]https://github.com/saimali/elevators

Algorithm 3: Cohorting with Pairing

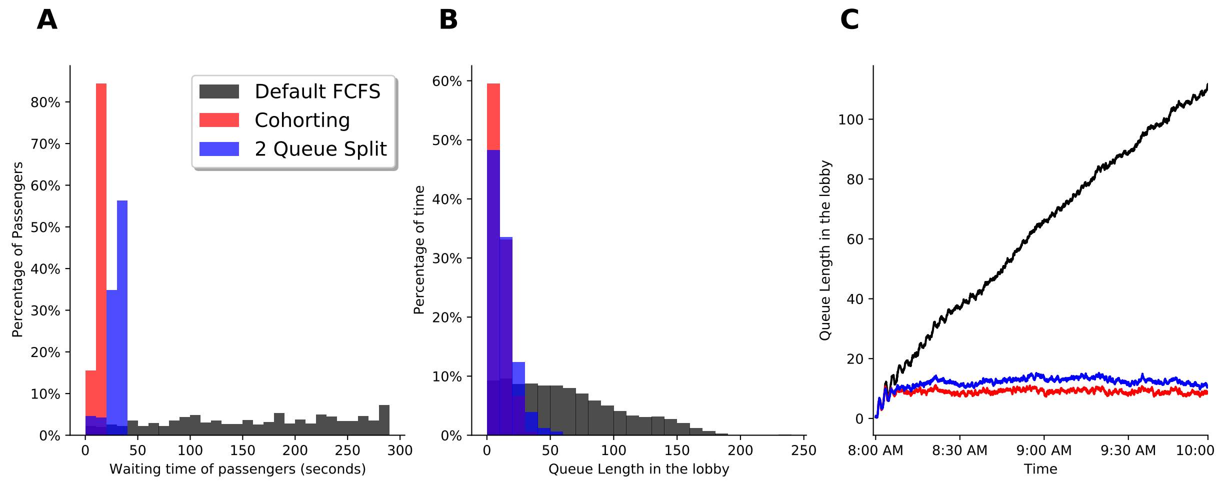
```
$t = 0$
$Q = \emptyset$
$\vec{F} = \vec{0}$ // number of passengers of an elevator deboarding at each
    floor
$E = \emptyset$ // set of empty elevators in the lobby
while $t < T$ do
    $t = t + \Delta t$
    Update the current queue $Q = Q \cup P(t) \backslash P(t - \Delta t) := \{p_1, p_2, \dots, p_l\}$, where $l$ is the length
      of current queue and $p_1$ is the first passenger in the queue
    Record queue length $N(t) = l$
    Update the elevators in lobby $E = E \cup \{e : ReturnTime(e) \in [t - \Delta t, t)\}$
    while there exist elevators in lobby and there are passengers waiting in the lobby do
        $e$ is the first elevator in $E$
        while there exist capacity in e and there are passengers waiting in the lobby do
            Update the current queue, $l$ is the length of current queue and $p_1$ is the first passenger
              in the queue
            $leader = p_1$
            Remove $p_1$ from $Q$ and record wait time $W_1$
            Update $\vec{F}$ according to $p_1$'s destination
            $i = 2$
            // Start finding a passenger that goes to the same
                destination of $p_1$
            if there exists remaining capacity in the elevator then
                while destination of $p_i$ is not the same as leader and $i \leq l$ do
                    $i \rightarrow i + 1$
                end
                if $i \leq l$ then // if $i = l + 1$, there is no passenger to be
                  paired with the leader
                    $p_i$ is paired with the leader and enters the elevator
                    Record wait time $W_i$ of passenger $p_i$
                    Remove $p_i$ from $Q$
                    Update $\vec{F}$ according to $p_i$'s destination
                end
            end
        end
        Update $ReturnTime(e) = t + RoundTripTime(\vec{F})$
        Remove elevator $e$ from $E$
        Update $\vec{F} = \vec{0}$
    end
end
```

Parameter	Large government building in NYC	An example medium sized building	An example small building
Building Configuration			
Number of destination floors (m)	24	16	6
Number of elevators (N)	14	6	2
Capacity of elevators (C)	4	4	2
Elevator configuration			
Travel time per floor of elevators ν	1.4 sec/floor		
Speed multiplier β (coming down)	1.3, extra 30% to approximate down traffic		
Loading time $BoardingTime(.)$	ω = 15 sec to board, additional 2 sec per passenger		
Unloading time $StopTime(.)$	ω = 15 sec to deboard, additional 2 sec per passenger		
Dedication on elevators	None		
System update interval Δt	1 second		
Passenger Profile			
Number of passengers	2750	1500	400
Arrival pattern to the lobby	Poisson process between 8 AM to 10 AM (rush hour)		
Destination	Uniformly at random in 2 to 25	Uniformly at random in 2 to 16	Uniformly at random in 2 to 7
Willingness-To-Walk (WtW)	0%		

Table 1.1: Input Parameters for the simulation models

Note that the queues will all eventually diminish after the end of rush hour because the passenger traffic goes down, but we do not simulate this. In the figures below, we simulate only until the end of rush hour (peak) and hence the queue decline after this time is not shown.

Figure 1.2: Comparison of interventions for our large building case study.



Note. We run 100 independent random instances and report the average performance. **A)** Plot of percentage of passengers experiencing different waiting times in the lobby across interventions. **B)** Plot of percentage of time different queue lengths in the lobby occur (measured every 1 second) across interventions. **C)** Plot of queue length in the lobby from beginning to end of the busy period across interventions.

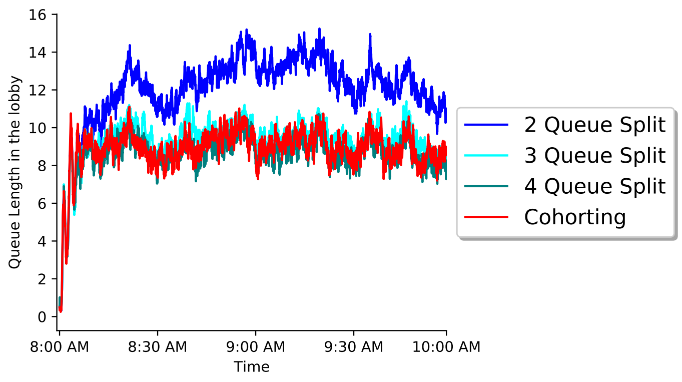
The results for FCFS, Cohorting, and Queue Splitting (2 queues with floor ranges $2-13$ and

$14 - 25$) on the large building are presented in Figure 1.2. One can observe that for FCFS, the number of people in the lobby grows linearly during the rush hour period we simulate. In fact, by the end of the rush hour, there can be up to 100 people in the queue and wait times can reach almost five minutes. Thus, an intervention is absolutely necessary to avoid this unsafe buildup of passengers. We see that Cohorting has a much lower range of queue lengths and waiting times compared to FCFS. Cohorting has a maximum queue length of around 12, which is over a factor of eight times smaller than the maximum queue length of FCFS. In other words, a passenger arriving at any point in the rush hour is likely to experience a queue of at most 12 people with the Cohorting intervention.

In the Queue Splitting intervention, we do not allocate any elevators but rather form a queue for every group of floors. In Figure 1.2, we see that Queue Splitting (into 2 queues) has a much lower range of queue lengths and waiting times compared to FCFS. Similar to Cohorting, the maximum queue length is around 15. In other words, a passenger arriving at any point in the rush hour is likely to experience a queue of less than 15 people in the 2 Queue Split intervention, which is over a factor of five times smaller than the maximum queue length of FCFS. One can see in Figure 1.2 that the maximum wait times and total queue length are relatively stable over time for this intervention, and with average reductions of over 80% compared to the default FCFS. Thus a 2 Queue Split achieves comparable performance to Cohorting, the best intervention.

We also consider the effect of the number of queues used in Queue Splitting, with the results displayed in Figure 1.3 for the large building. The floor ranges for each queue are split evenly: $\{(2 - 13), (14 - 25)\}$ in the 2 Queue Split, $\{(2 - 9), (10 - 17), (18 - 25)\}$ in the 3 Queue Split, and $\{(2 - 7), (8 - 13), (14 - 19), (20 - 25)\}$ in the 4 Queue Split. In this building, 4 queue split works better than 2 and 3 queue splits (higher number of queue splits achieves an effect similar to Cohorting). There is a marked improvement (Figure 1.3) from 2 to 3 queue split and only a marginal return on 4 queue split (which has almost the same queue length performance as Cohorting) instead of 3 queue split. In fact, the more queues we create, the more efficient the system becomes. However, the tradeoff is that more queues requires a more complex operation

Figure 1.3: Impact of Cohorting and Queue Splitting intervention into 2, 3 and 4 queues for our large building case study.



Note. We plot the queue length in the lobby (measured every 1 second) throughout rush hour. We run 100 independent random instances and report the average performance.

and more space in the lobby, especially for horizontal separation between the queues. We find that simply splitting into 2-4 queues already recovers most of the benefit in comparison to the Cohorting intervention.

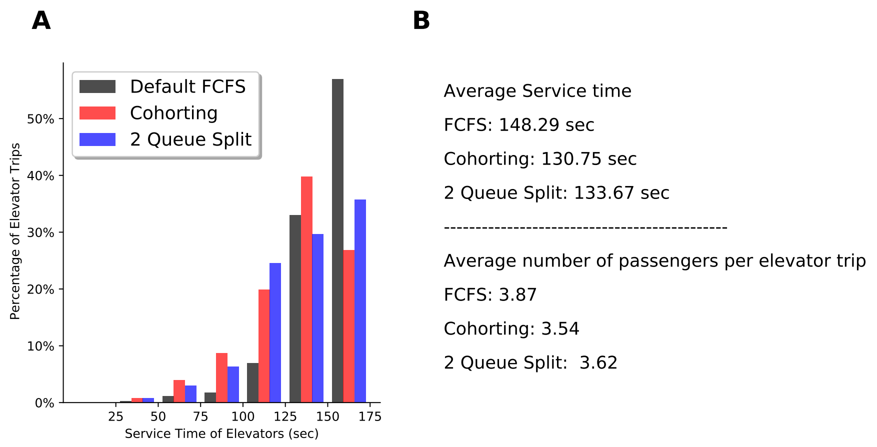
1.3.1 Difference in Round Trip Time

The round trip time of an elevator trip (service time) determines the efficiency of the elevator system. The shorter the round trip time is, the faster the elevator can come back and serve more people. We record the service time profile for FCFS, Cohorting, and 2 Queue Split in the simulation in Figure 1.4.

In Figure 1.4, Cohorting and Queue Splitting have a lower average round trip time (131s and 134s respectively) than FCFS (148s). In terms of number of trips they can complete in the given time period, Cohorting and 2 Queue Split are much better than FCFS, indicating that our proposed interventions indeed make the elevator trips more efficient. Our main indicator of system performance, queue length, is typically inversely related to average service time in classic queuing models [49]), and a seemingly small improvement in the round trip time as in Figure 1.4 has a big impact on system performance.

In Figure 1.4.B, we also report the average number of passengers per elevator trip across in-

Figure 1.4: Comparison of interventions using round trip for our large building case study



Average Service time

FCFS: 148.29 sec

Cohorting: 130.75 sec

2 Queue Split: 133.67 sec

Average number of passengers per elevator trip

FCFS: 3.87

Cohorting: 3.54

2 Queue Split: 3.62

Note. We run 100 independent random instances and report the average performance. **A)** Plot of percentage of elevator trips with different round trip times (service times) across interventions **B)** Reporting average service time and average number of passengers per elevator trip for all interventions.

terventions. With capacity 4, elevators under FCFS carry 3.87 passengers per trip on average, so most trips are at full capacity. Elevators under Cohorting carry only 3.54 passengers per trip on average and similarly under the 2 Queue Split intervention, elevators carry 3.62 passengers per trip on average. Our proposed interventions make the elevators more efficient compared to FCFS, *using less of the elevator capacity on average per trip*.

The round trip time of an elevator, as seen in (1.1) and discussed in detail in Section 1.5 is primarily determined by the number of stops S made and the highest reversal floor H of the trip. Therefore, we analyze the performance of different interventions by comparing the quantities S and H in Figure 1.5.

In Figure 1.5.A, we plot the distribution of S, i.e., the percentage of elevator trips with different number of stops made (or equivalently the number of buttons pressed in each trip by the passengers) across interventions. In FCFS, about 75% of the trips make 4 stops. In Cohorting only about 25% of the trips make 4 stops and about 70% elevator trips make only 2 or 3 stops, which leads to shorter round trip times. 2 Queue Splitting does not perform as well as Cohorting, although it is better than FCFS, with more than 50% of the trips making 4 stops and 40% rides making only 2 or 3 stops.

In Figure 1.5.B, we plot the distribution of H, i.e., the percentage of elevator trips with different highest reversal floors across interventions. In the distribution of H for FCFS, more percentage of elevator trips reverse at the topmost floors than other interventions. For instance, about 50% of FCFS trips reverse in the last four floors ($H \geq 23$), while only 40% of trips under Cohorting do so, suggesting that elevators come back to the lobby faster in Cohorting. 2 Queue Split has similar distributions of H among the destination ranges of the two mini-queues and hence, 40% of the trips have H at most 13, which is much better than Cohorting (less than 20% of trips) and FCFS (less than 10% of trips) for the same range.

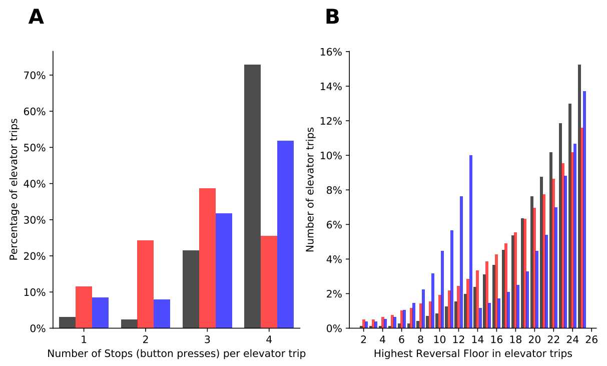
In Figure 1.5, we report the average of the number of stops and highest reversal floors for all interventions, to supplement the distribution plots. In 1.5.D, we also report the standard deviation of the estimated mean of S and H over 100 instances (i.e, we estimate the mean value of S and H for each instance, and show the standard deviation over 100 independent random instances). Elevators under FCFS makes an average of 3.64 ± 0.05 stops, whereas under Cohorting, they make on average only 2.78 ± 0.04 stops, a key driver of lower round trip time. 2 Queue Split does not perform as well as Cohorting, making an average 3.27 ± 0.07 trips which is comparable to FCFS. Similarly, FCFS has the highest average H of 20.3 ± 0.17, whereas Cohorting has an average H of 18.7 ± 0.19, 2 Queue Split has the lowest H at 17.6 ± 0.17, due to the two similar distributions among the mini-queues.

Though 2 Queue Split makes a comparable average number of stops to FCFS, having the lowest highest reversal floor among all interventions explains why it performs closer to Cohorting in service time in Figure 1.4. Ultimately, the impact of S is higher than that of H in service time due to simulation parameters (each stop adds 15s to service time, but each additional floor only adds 1.4s one way).

1.3.2 Sensitivity Analysis

In this subsection, we study how the performance of the interventions may change when parameters change. We investigate the sensitivity of average queue lengths to the two parameters that

Figure 1.5: Comparison of interventions using number of stops and highest reversal floor of elevator trips for our large building case study



C

Average Number of Stops S
FCFS: 3.64
Cohorting: 2.78
2 Queue Split: 3.27

Average Highest Reversal Floor H
FCFS: 20.4
Cohorting: 18.8
2 Queue Split: 17.6

D

Standard Deviation of mean estimate of S
FCFS: 0.05
Cohorting: 0.04
2 Queue Split: 0.07

Standard Deviation of mean estimate of H
FCFS: 0.17
Cohorting: 0.19
2 Queue Split: 0.17

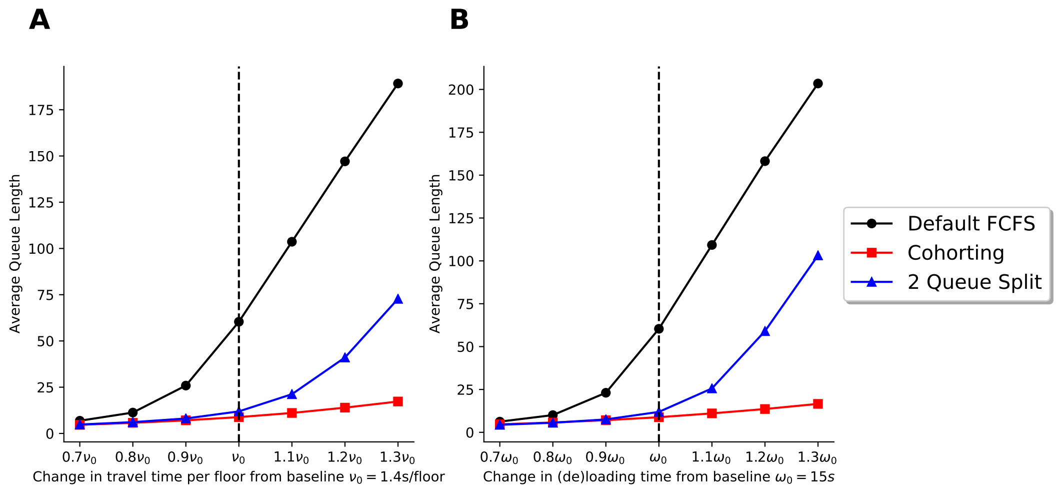
Note. We run 100 independent random instances and report the average performance. **A)** The percentage of elevator trips with different number of stops made (or equivalently the number of buttons pressed in each trip by the passengers) across interventions. **B)** The percentage of elevator trips with different highest reversal floor H across interventions **C)** Reporting average over 100 instances of the number of stops and highest reversal floor for all interventions. **D)** Reporting standard deviation of the estimated mean (over 100 instances) of number of stops S and highest reversal floor H for all interventions.

determine round trip time - travel time per floor ν and (de)boarding time ω.

In Figure 1.6.A, we vary travel time per floor ν with a scaling in the interval $[0.7, 1.3]$ of the baseline $\nu_0 = 1.4$s/floor and observe the average queue lengths for FCFS and our interventions. The average queue length in FCFS is very sensitive to the parameter ν, since FCFS has the largest average highest reversal floor. 2 Queue Split performs comparably to Cohorting until the baseline, with an average queue length of less than 20 but rises to 50 when ν becomes 30% more than the baseline. Cohorting is the least sensitive, maintaining an average queue length of less than 20 throughout.

In Figure 1.6.B, we vary (de)boarding time ω with a scaling in the interval $[0.7, 1.3]$ of the baseline $\omega_0 = 15$s and observe the average queue lengths for FCFS and our interventions. The average queue length in FCFS is very sensitive to the parameter ω, since FCFS has the largest number of stops. 2 Queue Split performs comparably to Cohorting until the baseline, with an average queue length of less than 20 but rises to 100 when ν becomes 30% more than the baseline. The parameter ω has a bigger sensitivity impact on 2 Queue Split than ν, showing the importance of number of stops to the round trip time. Cohorting is the least sensitive to ω, maintaining an average queue length of less than 20 throughout.

Figure 1.6: Sensitivity of average queue lengths to ν and ω for our large building case study.



Note. We run 100 independent random instances and report the average performance. Between 8 to 10 AM, 2750 passengers with destinations ranging from floors 2 to 25 are served by 14 elevators (each with capacity 4). **(A,B)** Plots of average queue length from beginning to end of the busy period across interventions. In **A)** We change the travel time per floor parameter ν. In **B)**We change the (de)boarding time parameter ω.

Other parameters also affect the performance of Cohorting and Queue Splitting. When the elevator capacity C becomes larger, the round trip time will become longer for all the interventions. Queue Splitting can keep the average highest reversal floor at a relatively low level because it separates the trips for high floors from low floors. However, it may be difficult to keep the number of stops low. On the other hand, Cohorting can still keep the number of stops S at a relatively low value. However, it becomes harder to keep the highest reversal floor H low, because we do not have the separation of high floors to low floors, and it is more likely to mix high floor trips with low floor trips when the capacity becomes larger. Also when capacity increases, it takes more time and effort to find a cohort of people going to the same floor. This practical issue may cause the Cohorting intervention to be less attractive, while the management effort for Queue Splitting does not scale up when capacity increases.

To summarize, we showed via simulation that Cohorting and Queue Splitting reduce the round trip time for the elevator trips and serve more passengers in a given time period. These numerical results provide motivation to understand why these two interventions perform similarly well while managing the queue completely differently. We offer theoretical support for our proposed interventions and focus on the distribution of the number of stops and highest reversal floor in Section 1.5.

1.3.3 Results for other building types

We primarily report the results of interventions in the large building setting in the above section. For understanding a more general performance, we also model another two examples- (1) a 7-story small building with 6 destination floors being served by 2 elevators with capacity 2 for 400 passengers arriving during rush hour; (2) a 17-story medium sized building with 16 destination floors being served by 6 elevators with capacity 4 for 1500 passengers arriving during rush hour. Figure 1.7 shows the performance in interventions in these two other examples.

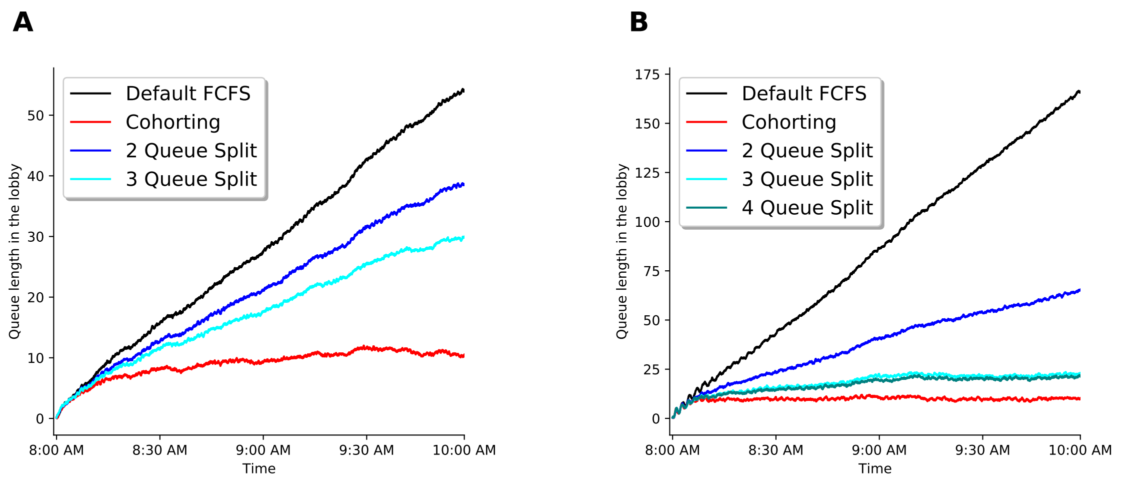
Figure 1.7 A shows that for the small building, the queue length in FCFS builds up reaching a peak of more than 50 people at the end of rush hour, whereas Cohorting has a queue length of no more than 10, which is over a 75% improvement. Queue Splitting is better than FCFS but not as beneficial as Cohorting. Queue lengths in 2 Queue Split build up to 40 people and up to 30 people in 3 Queue Split. Cohorting would be the best overall solution in this example.

Figure 1.7 B shows that for the medium sized building, the queue length in FCFS builds up reaching a peak of up to 175 people at the end of rush hour, whereas Cohorting has a queue length of around 15, which is a huge improvement. Queue Splitting is better than FCFS and the number of queues impact the performance. The queue length in 2 Queue Split steadily builds up to 60 people, whereas 3 and 4 Queue Split are better and perform similarly with only around 25 people. Thus, Cohorting or a 3 Queue Split would be good solutions in this example.

1.4 Performance of Allocation

In this section, we study the performance of the Allocation intervention. We considered several distinct ways to allocate elevators to floors, although not all of them worked properly. For example, we initially tested the performance of allocating half of the elevators to the odd levels and the rest to the even levels. The main motivation for this allocation is to encourage people to walk up or down

Figure 1.7: Comparison of interventions in examples of small and medium sized buildings.

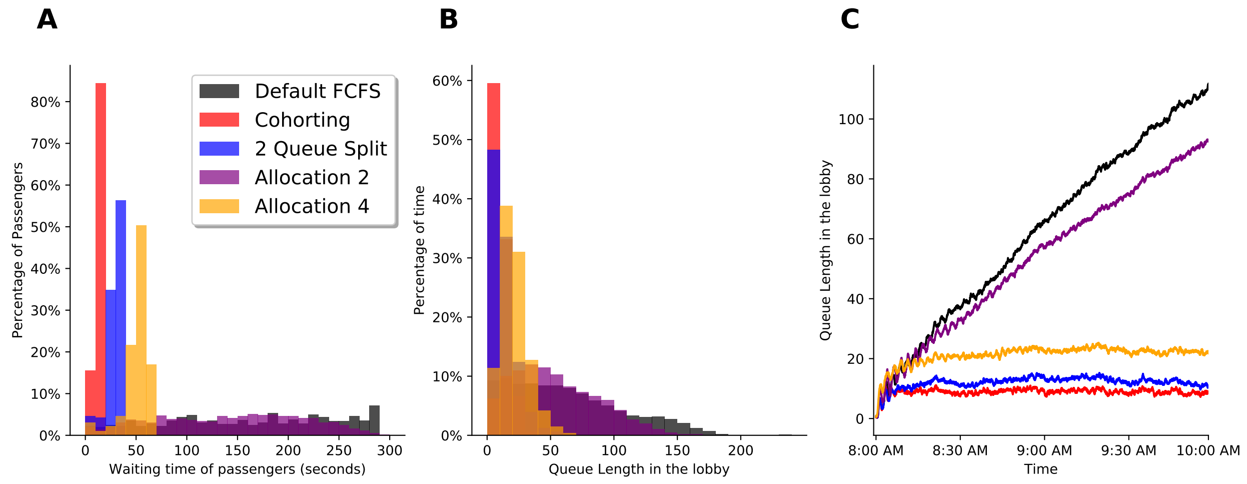


Note. In Queue splitting, we always split the floor ranges (nearly) equally among all queues. **A)** Plot of queue length in the lobby from beginning to end of the busy period across interventions for an example small building on a typical Monday morning rush hour. Between 8 to 10 AM, 400 passengers with destinations ranging from floors 2 to 7 are served by 2 elevators (each with capacity 2). **B)** Plot of queue length in the lobby from beginning to end of the busy period across interventions for an example medium building on a typical Monday morning rush hour. Between 8 to 10 AM, 1500 passengers with destinations ranging from floors 2 to 16 are served by 6 elevators (each with capacity 4).

one level because it is always feasible for a passenger who is willing to walk to take the elevator in the other group of elevators. However, the improvement is rather negligible in comparison to FCFS even with people willing to walk. This is because the distribution of the highest reversal floor is barely changed, and it is almost as likely as FCFS that the elevator trips will end up in very high floors. Due to the poor performance, we do not describe the odd-and-even intervention in detail and do not recommend such strategies that cannot reduce the average highest reversal floors.

Next, we consider the Allocation intervention that dedicates each elevator to a predetermined floor range. It is a common practice in high rise buildings that a dedicated group of elevators serves the higher floors, and other elevators serve low floors. By implementing this allocation intervention, the chances of two random passengers in the same group going to the same floor becomes relatively high, and trips to high floors are grouped together. This results in a natural cohorting phenomenon and travel time reduction. Note that Queue Splitting is theoretically better than the Allocation intervention with the same division of floor ranges, as there is an extra constraint in the usage of elevators in the Allocation intervention. We observe the drastic difference in Figure 1.8.

Figure 1.8: Comparison of interventions, including Allocation intervention for our large building case study.

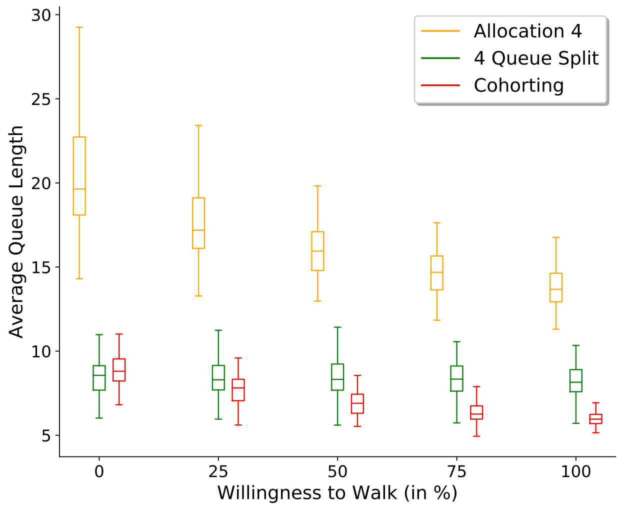


Note. Between 8 to 10 AM, 2750 passengers with destinations ranging from floors 2 to 25 are served by 14 elevators (each with capacity 4). We run 100 independent random instances and report the average performance. **A)** Plot of percentage of passengers experiencing different waiting times in the lobby across interventions. **B)** Plot of percentage of time different queue lengths in the lobby occur (measured every 1 second) across interventions. **C)** Plot of queue length in the lobby from beginning to end of the busy period across interventions.

In the case study of the 25-floor high rise building, we numerically evaluate the performance of the Allocation 4 intervention, where we divide the 14 elevators into groups of $3, 3, 4, 4$, with each group serving 6 floors. The two groups with 4 elevators serve the relatively higher floor ranges. We also try the Allocation 2 intervention in which we divide the elevators into 2 groups of 7 elevators and each group serves 12 floors.

The results are shown in Figure 1.8. The Allocation 2 intervention can delay the build-up of queues slightly comparing with FCFS, but generally still have a large queue length. Using the Allocation 4 intervention, the queue length and waiting time can be reduced quite significantly, and the queue does not keep building up in the lobby. Therefore, with proper allocation of elevators, the safety concerns in elevator management can be controlled. However, in comparison with the performance of Cohorting and 2 Queue Split, the Allocation 4 intervention results in much longer queue length. The key reason why Allocation is not as effective as Queue Splitting is that when one of the elevator groups has stabilized its queue, it cannot help another group which has unstable queue. This reasoning can be seen in the theoretical stability analysis in Section 1.4.1. In addition to the worse performance, there are higher complexity to implement the Allocation intervention as the elevators require extra programming and control with the relevant floor ranges. Generally speaking, using the Allocation intervention can significantly reduce the queue length, though not as much as the Cohorting and Queue Splitting intervention.

Figure 1.9: The impact of WtW in the Allocation 4 Intervention.



Note. Plot showing the average queue lengths in the lobby of both Cohorting and Allocation 4 interventions with the WtW parameter varying between 0% to 100%. As the WtW increases, there is negligible impact on Cohorting whereas the performance of Allocation 4 improves markedly.

Next, we consider the impact of willingness-to-walk to the performance of the Allocation intervention. Suppose a passenger is willing to walk up or down one level to the destination floor x, he or she will first check all the queues that can reach either the destination floor x, $x-1$, or $x+1$, and choose the queue with shortest length to join when arriving to the system. Therefore, when the elevators are assigned to small ranges of floors, only the passengers whose destinations are at the boundary of the floor ranges can make the choice of which queue to join. In Figure 1.9, we show the impact of WtW under the Allocation 4 intervention. Unlike the impact of WtW on Cohorting and Queue Splitting intervention where the improvement is relatively small, the improvement with respect to the increased WtW is much more dramatic for Allocation. This is due to the fact that the service resources in Allocation are not shared between groups, so that the passengers who shift from a long queue to a short queue help the system achieve a better match between its service capacity and demand. Therefore, in a building where the Allocation intervention is physically implemented, the management team may try to ask passengers switching to another queue by walking up/down one level to better utilize the elevator capacities. This is an alternative way to reduce the queue length.

1.4.1 Stability Condition for Allocation

In the Allocation intervention where m floors and N elevators are divided into l groups, the queueing system is essentially l independent queue with FCFS service rule. The standard stability condition (1.2) for Allocation is

$$\frac{\lambda}{Cl}\mathbb{E}[\tau(H,S)|\text{group } i] < \frac{N}{l} \text{ for all } i = 1,\dots,l.$$

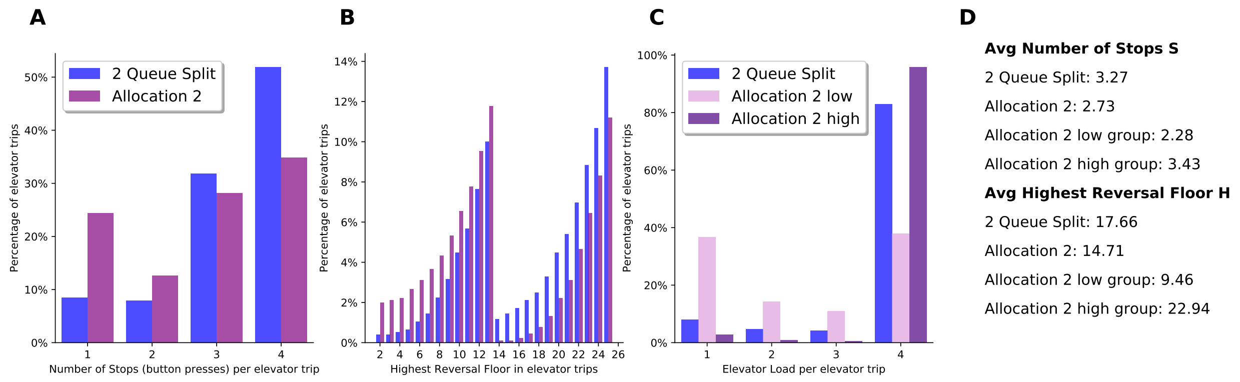
Equivalently, the queuing system is stable if and only if the arrival rate λ is lower than

$$\eta_{Allocation} := \min_{i=1,\dots,l}\left\{\frac{NC}{\mathbb{E}[\tau(H,S)|\text{group } i]}\right\} = \min_{i=1,\dots,l}\left\{\frac{NC}{2\nu(\mathbb{E}[H|\text{group } i]-1)+\omega\mathbb{E}[S|\text{group } i]}.\right\}$$

Comparing with the stability condition of Queue Splitting, we can see that the stability threshold for Allocation is lower then Queue Splitting (assuming the floors are split the same way). Indeed, whether the queuing system is stable or not is determined by the highest floor group (when there is an even split) since the round trip time for the highest floor group has the largest value among all floor groups. One can also consider allocating the floors unevenly to minimize the stability threshold $\eta_{Allocation}$, but this requires solving a difficult optimization problem that is hard to calibrate.

In Figure 1.10, we report the histogram of number of stops, highest reversal floor, and elevator load for 2 Queue Split and Allocation 2. Because we use the same floor groups for both interventions, we can observe a similar behavior in the distribution shape of highest reversal floor. Due to the fact that passengers in different floor groups can be mixed together in Queue Splitting intervention when a queue does not have enough people to fill up the elevator capacity, we observe that Allocation 2 leads to a smaller average highest reversal floor H overall. However, the queue is not stable because the high floor group has a much higher average H value of 22.94. Indeed,

Figure 1.10: Comparing Queue Splitting and Allocation using number of stops and highest reversal floor of elevator trips.



Avg Number of Stops S
2 Queue Split: 3.27
Allocation 2: 2.73
Allocation 2 low group: 2.28
Allocation 2 high group: 3.43
Avg Highest Reversal Floor H
2 Queue Split: 17.66
Allocation 2: 14.71
Allocation 2 low group: 9.46
Allocation 2 high group: 22.94

Note. We run 100 independent random instances and report the average performance. **A)** The percentage of elevator trips with different number of stops made. **B)** The percentage of elevator trips with different highest reversal floor H across interventions. **C)** The percentage of elevator trips with different load. **D)** Reporting average number of stops and average highest reversal floor.

the queue for the low floor group is mostly empty because the elevator load is full only for 40% of the trips. About 50% of the elevator trips only utilize half or less than half of the capacity. It indicates an extremely imbalanced situation in the Allocation 2 intervention: there are too many elevators for the low floors, but not enough for the high floors. When Allocation is implemented in a building, the elevators are programmed to only serve a certain floor range. Therefore, empty elevators for the low floor group cannot help the long queue for the high floor group, which is a dramatic drawback of restricting service resources to a dedicated job type.

Of course, the performance for Allocation can be improved if the number of elevators for each floor group is optimized. We can also make the floor range for high floors smaller. However, when facing demand fluctuation and imbalanced arrival patterns, the Allocation intervention lacks the flexibility to fully utilize the service capacity. Therefore, we recommend choosing Queue Splitting over Allocation.

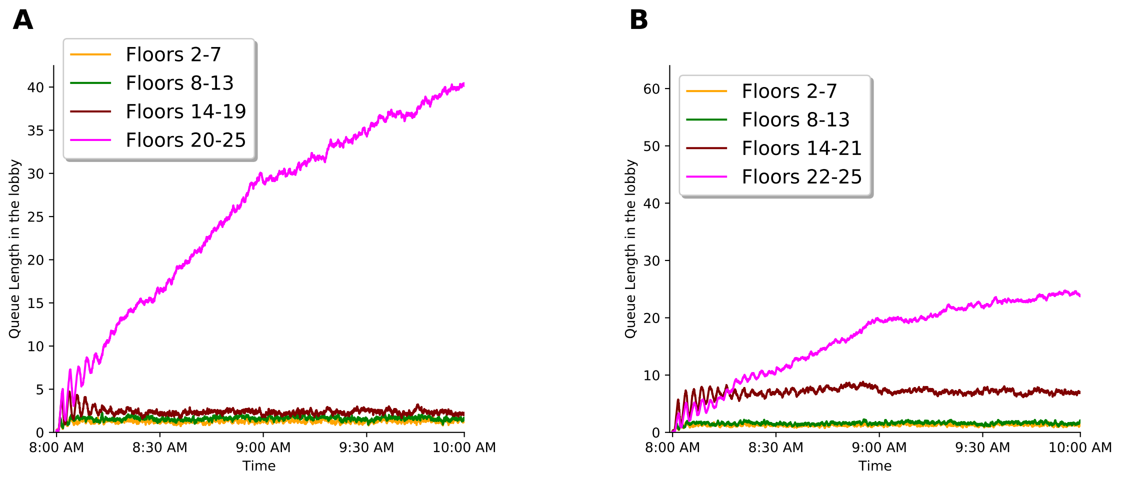
1.4.2 Hard Constraints on Elevators

In reality, many buildings (including the large NYC building we studied) have banks of elevators with pre-determined allocation of floor ranges for different elevators. Consider the allocation where half the elevators (7 elevators) serve half the floors $2-13$ and the other half (7 elevators) serve the floors $13-25$. These *hard constraints* on elevator systems are essentially the Allocation intervention we just discussed, and we can further apply the Cohorting or Queue Splitting intervention to the system with hard constraints.

In this case, the Cohorting and Queue Splitting interventions still perform much better than FCFS, as expected. Irrespective of interventions, passengers whose destinations are in higher floors make a big impact on performance since the round trip time of these elevators are bigger. At the very least, careful management of higher floor passengers should be considered, e.g., in queue splitting intervention, one can shorten the ranges for higher floor passengers to encourage more intrinsic cohorting.

In Figure 1.11, we show the simulation results under the large building setting with 7 elevators dedicated to floor $2-13$ and the other 7 elevators serving floor $14-25$. We consider the performance of the 4 queue split intervention by plotting the length of each of the 4 queues in the lobby over the entire busy period.

Figure 1.11: Impact of Queue Splitting (4 queues) intervention for our large building case study.



Note. Between 8 to 10 AM, 2750 passengers with destinations ranging from floors 2 to 25 are served by 14 elevators (each with capacity 4). In this setting, not all elevators serve all floors. Instead, 7 elevators serve floors 2 to 13 and 7 serve floors 14 to 25. **(A,B)** Plots of queue length in the lobby from beginning to end of the busy period. In **A)** we evenly split the floors in the ranges (2 - 13) and (14 - 25) and each queue gets exactly 6 floors. In **B)** we evenly split the floors in the two queues serving (2 - 13) but unevenly split the two queues serving (14 - 25) into one serving (14 - 21) and the other serving (22 - 25).

In Figure 1.11 A, we split the floor ranges $2-25$ equally among all the queues so that each queue gets exactly 6 floors. While the three lower queues have good performance (less than 5 people on average), the queue for the highest floors $20-25$ builds up over time to more than 40 people at the end of rush hour. This imbalance arises because floors $13-25$ are only served by 7 elevators, hence the passengers going to the highest floors wait longer for the busy elevators to come back to the lobby.

In Figure 1.11 B, we split the floor ranges $2-13$ equally as before, whereas in the floor range $14-25$, we assign one queue to serve eight floors $14-21$ and assign only four floors $22-25$ for the last queue. The low floor queues for 2 to 13 have good performance as before, but both the high floor queues, i.e., the ones serving floors $14-21$ and $22-25$ absorb the imbalance and their lengths build up over time to at most 25 people at the end of rush hour for floors $22-25$ and at most 10 people for floors $14-21$. The queue serving $22-25$ will also have fewer number of stops since there are only four floors in this range, and the elevators taking these passengers are more likely to come back faster to the lobby. Thus tuning the floor ranges for the queues serving the higher floors leads to better performance compared to Figure 1.11 A where all queues have their destination ranges split equally. Allocation therefore may be a hard constraint, and when applying an intervention like queue splitting, a careful design of floor ranges is *necessary* for buildings with pre-determined floor allocations for elevators.

1.5 Stability Analysis

In this section, we investigate the theory behind the good performance of the proposed Cohorting and Queue Splitting interventions. As observed in Figure 1.2, the queue length does not grow over time under the Cohorting and Queue Splitting intervention, while under FCFS it keeps increasing. In other words, by using the Cohorting and Queue Splitting intervention, we manage to transfer an unstable queuing system into a stable one under the simulation setting in Section 1.3. In this section, we aim to establish stability conditions for each intervention and explain why the proposed interventions work. We find that our proposed interventions can be stable under higher arrival rates than FCFS for two reasons. First, the interventions reduce the number of stops in comparison to FCFS, and second, the interventions reduce the average total distance traveled by the elevators. In particular, the second reason is supported by a stochastic dominance result for the highest reversal floor distribution of the different policies. In Section 1.5.1 we provide a background on queuing stability and in Section 1.5.2 we explain assumptions needed to prove our results. In Section 1.5.3 we calculate the stability condition for a special case with two floors and one elevator and in Section 1.5.4 we extend our analysis to general settings.

1.5.1 Stability Condition for a Queuing Network

In this section, we describe results in the literature on stability of multiclass queuing networks with different operations rules. The model we focus on has I buffers (or arrival classes) and K types of resources to serve the arrivals. Each arrival class is served by a specific resource type. The arrival process for each buffer i is a Poisson process with rate λ_i, and the service for type i requires a random time with mean t_i. Each resource k is a pool of b_k identical servers, where $\vec{b} := (b_1, \ldots, b_K)$. Each arrival class is processed by servers from a single specified pool, and each such service is accomplished by a single server from the pool. The set of buffers that resource k can serve is defined as $\mathcal{I}(k)$. The load vector $\vec{\rho} := (\rho_1, \ldots, \rho_K)$ is defined as $\rho_k = \sum_{i \in \mathcal{I}(k)} \lambda_i t_i$.

In Lemma 1 below, we see that the stability condition, also known as the standard load condition,

$$\vec{\rho} < \vec{b} \tag{1.2}$$

is a sufficient and necessary condition for the stability of the queuing network we study in this work. For some queuing systems, such as $M/M/1$ queue, it is well-known that the system is stable if and only if the load vector ρ is less than 1. However, it is not always true that Eq. (1.2) is a sufficient condition for a general queuing network [45]. In Lemma 1 below, we establish the stability condition for a feedforward queuing network under a non-idling control policy. Fortunately, the queuing models corresponding to the elevator interventions all fall within the category of a *feedforward queuing network*, which is defined as a queuing network in which the resources can be numbered in such a way that the arrival jobs never move from higher numbered servers to lower numbered ones (all passengers/jobs see one elevator/server). Also, the interventions we propose are *non-idling dynamic control policies*, which is defined as a control policy where no server remains idle while there is a job waiting in any of the buffers that are processed by the server.

Lemma 1. *Under any non-idling policy, a feedforward queuing network is stable if and only if the stability condition $\vec{\rho} < \vec{b}$ holds.*

Proof. Proof. The proof is supported by multiple results from [43], which we list in Lemma 2 and 3 below.

Lemma 2 (Proposition 5.1 and Theorem 5.2 in [43])**.** *If a unitary network is stable, then it satisfies the standard load condition $\rho < b$.*

Lemma 3 (Theorem 8.14 in [43])**.** *In a feedforward queuing network, if the standard load condition $\vec{\rho} < \vec{b}$ holds, then the queuing network is stable under any non-idling policy.*

A unitary network is a general type of stochastic processing network. In simple words, it requires a one-to-one relationship between the service activity and the buffers, and there is only one way to process jobs of any given class. We note that the feedforward queuing network, which is how our interventions can be described, is a special case of a unitary network. Thus, by Lemma 2 we can conclude that if a feedforward queuing network is stable, then the standard load condition $\vec{\rho} < \vec{b}$ must hold. Finally, combining with Lemma 3, we can conclude that the condition $\vec{\rho} < \vec{b}$ is indeed a sufficient and necessary condition for the queuing network we are interested in. $\square$

In the following subsections, we will specify the structure of the queuing network for each intervention, and derive the stability condition by using Lemma 1.

1.5.2 Assumptions and Justification

We first simplify the simulation model from Section 1.2 for the sake of analysis. We formulate the elevator system as a queuing network, in which the N elevators are the servers and the round trip time of an elevator trip is the service time. As discussed in Section 1.2, the round trip time is composed of the boarding time, stop time, ascent time, and descent time. We simplify the calculation of round trip time by omitting the boarding time at the lobby, which is the same for all interventions. We let ν be the time it takes an elevator to travel one floor and we simplify the stop time of an elevator for one stop to be ω. We also simplify the ascent time and descent time to be identical. We assume the destination of passengers is uniformly at random across all the floors, and the aggregate arrival rate is λ. Recall that $\vec{F}$ is the number of passengers going to each floor in a particular elevator trip. We let S be the random variable denoting the number of stops in an elevator trip, i.e., $S := \sum_{j=2}^{m+1} \mathbb{1}\{F_j > 0\}$. We let H be the random variable denoting the highest reversal floor in an elevator trip, i.e., $H := \max_j j\mathbb{1}\{F_j > 0\}$.

We use $\tau(H, S)$ to denote the round trip time for an elevator trip, which is a random variable with finite support, and its mean value depends on the random variables H and S. The conditional expected round trip time is defined as

$$\mathbb{E}[\tau(H,S)|H,S] := 2\nu(H-1) + \omega S, \tag{1.3}$$

where $2\nu(H-1)$ is the ascent and descent time, and ωS is the stop time. The distributions of H and S are determined by the random arrival process and the intervention, and are used to determine the expected round trip time. We do not make any further assumption on the distribution of the round trip time.

For the Queue Splitting intervention, we assume that the m floors are divided into l groups, where each group is a separate buffer and consists of k consecutive floors, i.e., $m = lk$ and $l, k \geq 2$ are integers. The j-th floor in the i-th group can be written as $x := (i-1)k + j + 1$, for $i = 1, \ldots, l$; $j = 1, \ldots, k$. For example, floor 2 is the first floor in the first group.

Next, we describe the queuing networks that represent our elevator system under different interventions. Our goal is to derive the stability conditions, under which the queue does not grow over time under, for each intervention. Thus, we only consider such a condition in a system with extremely long (infinite) queues. We assume that from now on, we treat a set of C passengers as one arrival job to the system, which is without loss of generality in this regime. The assumption of treating C passengers as one job allows us to utilize tools from the literature and have a sufficient and necessary condition for stability (Lemma 1). We note that [44] provide stability analysis for batch service systems, and $\vec{\rho} < \vec{b}$ is a sufficient condition for stability for the queue under the operation rule corresponding to the Cohorting and Queue Splitting intervention.

We now connect the interventions to the setup described in Section 1.5.1. In FCFS, there is only one buffer and an arrival job will be C random passengers with independent and uniformly distributed destinations among all floors. In the Queue Splitting intervention, each floor group corresponds to one buffer. An arrival job will be C random passengers who are going to the same group of floors. In Cohorting, each destination floor corresponds to one buffer, and an arrival job is C passengers who are going to the same floor. Note that for FCFS, Cohorting, and Queue Splitting, there is only one resource type, i.e., all elevators can serve all floors. In the Allocation intervention where elevators are assigned to floor groups, the elevators are divided into groups, which represents different types of resources. Each floor group forms a separate FCFS queue that can only be served by one resource type. This distinction makes the Allocation less effective than the other proposed interventions, as we show in Section 1.4. In the following subsections, we focus on the analysis of the stability condition (1.2) for FCFS, Cohorting, and Queue Splitting. The stability analysis for Allocation is provided in Section 1.4.1.

1.5.3 1 Elevator and 2 Destination Floors

We first focus on the simplest setting where the building only has two destination floors 2 and 3 and one elevator with capacity $C = 2$. The passengers arrive at the building according to a Poisson process with rate λ, and go to floor 2 or 3 with equal probability. Equivalently, the arrival process for passengers who go to floor 2 (similarly for floor 3) is a Poisson process with rate $\frac{\lambda}{2}$.

The elevator system under FCFS is an $M/G/1$ queue, where the new jobs arrive according to a Poisson process with rate $\lambda/2$, and the elevator is the server with the round trip time being the service time. In an $M/G/1$ queue, the stability condition Eq. (1.2) is well-known and can be found in the literature and the textbooks (e.g. [49] Example 4.3(a)). It can also be derived from Lemma 1 since an $M/G/1$ queue is a special case of a feedforward queuing network, and FCFS is a non-idling policy.

For the Cohorting and 2 Queue Splitting intervention, we can think of the following multi-class queuing network with two queuing buffers and one server. The passengers who go to floor 2 form a queue in buffer 1, and the passengers who go to floor 3 form a separate queue in buffer 2. The passengers are also assigned into pairs, and a pair of passengers is considered as a job that is waiting to be served. This network with two buffers and one server satisfies the condition of a feedforward queuing network. In the Queue Splitting intervention, the server will choose a buffer to serve in a round-robin fashion once it finishes the previous job. In the Cohorting intervention, the elevator serves the first passenger in the queue and we let the second passenger whose destination is the same as the first passenger board the same elevator. We can represent the dynamic as a multi-class queue where each floor destination forms a buffer, and the server decides which job to take by choosing the buffer whose head of queue arrives the earliest to the system. In the literature, this is called a FCFS control policy [50] since the way a server choosing between buffers is in the first-come first-serve fashion. Both the Queue Splitting and Cohorting intervention are non-idling policies. Therefore, we can again use Lemma 1 to derive the stability condition.

Proposition 1. *(a) The elevator system is stable under FCFS if and only if the arrival rate*

$$\lambda < \frac{4}{7\nu + 3\omega} := \eta_{FCFS}.$$

(b) The elevator system is stable under the Cohorting and 2 Queue Splitting if and only if

$$\lambda < \frac{4}{6\nu + 2\omega} := \eta_{Cohort} = \eta_{QS}.$$

Proof. Proof. (a) The queuing system corresponding to FCFS consists of only 1 buffer with $\lambda/2$ being the arrival rate of each pair of passengers. The service time distribution is a mixture of 3 distributions. With probability 0.25, the highest reversal floor H is 2 and the number of floors and the total number of stops S is 1. Similarly, with probability 0.25, the elevator only stops once and H is 2. With probability 0.5, H is 3 and S is 2. Therefore, the expected service time is

$$\begin{aligned}&\frac{\mathbb{E}[\tau(2,1)] + 2\mathbb{E}[\tau(3,2)] + \mathbb{E}[\tau(3,1)]}{4}\\ &= \frac{2\nu(2-1) + \omega + 2(2\nu(3-1) + 2\omega) + 2\nu(3-1) + \omega}{4} = \frac{7\nu + 3\omega}{2}.\end{aligned}$$

By Lemma 1, the system is stable if and only if $\frac{\lambda}{2} \cdot \frac{7\nu+3\omega}{2} < 1$, which is equivalent to $\lambda < \frac{4}{7\nu+3\omega}$.
(b) The queuing system for Queue Splitting and Cohorting intervention has 2 buffers, each with arrival rate $\frac{\lambda}{4}$ for a pair of passengers. For buffer 1, all passengers are going to floor 2, so the expected service time is $\mathbb{E}[\tau(2,1)] = 2\nu(2-1) + \omega$. Similarly, for buffer 2, all passengers are going to floor 3, so the expected service time is $\mathbb{E}[\tau(3,1)] = 2\nu(3-1)+\omega$. By Lemma 1, the system is stable if and only if $\frac{\lambda}{4} \cdot \mathbb{E}[\tau(2,1)] + \frac{\lambda}{4} \cdot \mathbb{E}[\tau(3,1)] < 1$, which is equivalent to $\lambda < \frac{4}{6\nu+2\omega}$. $\square$

Since the stability thresholds above provides an upper bound on the total arrival rate of passengers, then the higher the threshold is, the better the system can deal with rush hour traffic. In the following proposition, we establish by how much the proposed interventions can improve the stability threshold.

Proposition 2. *Consider a building with 1 elevator, two destination floors, and elevator capacity of two. The Cohorting and Queue Splitting intervention can increase the stability threshold by at least* 16.67%*, and at most* 50%*.*

Proof. Proof. In this proof, we want to bound the ratio $\frac{\eta_{QS}}{\eta_{FCFS}}$. Plugging in the threshold we get from Proposition 1, we have $\frac{\eta_{QS}}{\eta_{FCFS}} = \frac{7\nu+3\omega}{6\nu+2\omega}$. Since ν and ω are positive real numbers, $\frac{7}{6} < \frac{7\nu+3\omega}{6\nu+2\omega} < \frac{3}{2}$, which yields the final result. $\square$

Proposition 2 provides a clean explanation to the phenomenon we observe from the simulation: when facing the same passenger arrival pattern, the stability condition for FCFS is violated while the arrival rate is still below the threshold for Queue Splitting and Cohorting. Thus the key driver for the good performance of the proposed interventions is that the expected service time is much shorter, thanks to the fact that both the highest reversal floor and the number of stops have smaller values. When we use the FCFS intervention, $H = 3$ with probability 0.75, despite the fact that only half of the passengers go to floor 3. Using Cohorting and Queue Splitting intervention, we can make sure that only 50% of the time the elevator will go to the higher floor. For the number of stops per elevator trip, using Cohorting and Queue Splitting intervention, we can ensure that the elevator only makes one stop in each elevator trip, while in FCFS, 50% of the time the elevator will make 2 stops. Our proposed interventions can simultaneously reduce the stop time and the travel time of the elevator, and make the elevator trips more efficient. The analysis for the two-destination system can be extended to multiple floors. In the 2 Queue Split intervention, we have two destination ranges, the high floors and the low floors. When using FCFS, 75% of the trips will end up in the high floor range, while using 2 Queue Split, we can reduce the fraction of trips going to the high floors to 50%. In the next subsection, we provide detailed analysis to a building with multiple elevators and floors.

1.5.4 General Case

In this subsection, we focus on a general building with m destination floors $2, \ldots, m+1$ and N identical elevators that can serve all the floors. Though the expected service time is complicated to compute exactly in the general setting, we can focus on the distribution of H and S and show that the stability threshold for Queue Splitting and Cohorting is much higher than the one for FCFS. In this subsection, we again assume that we group C passengers together and let them enter the lobby as a new job to the queuing system. Lemma 3 below shows that the stability threshold η for each intervention is a simple function of the $\mathbb{E}[H]$ and $\mathbb{E}[S]$.

Proposition 3. *For intervention $\pi \in \{FCFS, Cohorting, QS\}$, the queue is stable if and only if the arrival rate $\lambda < \eta_\pi := \frac{NC}{2\nu(\mathbb{E}[H_\pi]-1)+\omega\mathbb{E}[S_\pi]}$.*

Proof. Proof. We start with FCFS. By Eq. (1.2), the stability condition is $\frac{\lambda}{C}\mathbb{E}[\tau(H_{FCFS}, S_{FCFS})] < N$, which is equivalent to

$$\frac{\lambda}{C}\left(2\nu(\mathbb{E}[H_{FCFS}] - 1) + \omega\mathbb{E}[S_{FCFS}]\right) < N. \tag{1.4}$$

Under the Cohorting intervention, the queuing network consists of m independent buffers for m floors, and an idle server will choose to serve the buffer with the earliest arrival time. By Eq. (1.2), we need to specify the average round trip time for each buffer. The highest reversal floor for buffer i is simply i, and every trip only has one stop so $\mathbb{E}[S_{Cohort}] = 1$. Therefore, the stability condition (1.2) becomes $\sum_{i=1}^{m} \lambda_i \mathbb{E}[\tau(H, 1)|H = i] < N$, where $\lambda_i = \frac{\lambda}{Cm}$. Note that the highest reversal floor H_{Cohort} follows a uniform distribution, we can rewrite the left hand side of the stability condition into

$$\sum_{i=1}^{m} \lambda_i \mathbb{E}[\tau(H,1)|H = i] = \frac{\lambda}{C}\sum_{i=1}^{m}\frac{1}{m}\mathbb{E}[\tau(H,1)|H = i] = \frac{\lambda}{C}\left(2\nu(\mathbb{E}[H_{Cohort}] - 1) + \omega\right). \tag{1.5}$$

Under the Queue Splitting intervention, recall that we assume that the m floors are divided into l groups, where each group is a separate buffer and consists of k consecutive floors. $m = lk$ and $l, k \geq 2$ are integers. The stability condition becomes $\sum_{i=1}^{l} \lambda_i \mathbb{E}[\tau(H, S)|H \in \text{ group } i] < N$, where $\lambda_i = \frac{\lambda}{Cl}$ in this case. Since the jobs are processed in a round robin fashion, the probability for a new job to be in group i is $\frac{1}{l}$. Then we can rewrite the left hand side of the stability condition as

$$\begin{aligned}\sum_{i=1}^{l} \lambda_i \mathbb{E}[\tau(H,S)|H \in \text{group } i] &= \frac{\lambda}{Cl}\sum_{i=1}^{l}\left(2\nu(\mathbb{E}[H_{QS}|H \in \text{group } i] - 1) + \omega\mathbb{E}[S_{QS}|H \in \text{group } i]\right) \\ &= \frac{\lambda}{C}\left(2\nu(\mathbb{E}[H_{QS}] - 1) + \omega\mathbb{E}[S_{QS}]\right).\end{aligned} \tag{1.6}$$

Therefore, following the result in Lemma 1, for FCFS, Cohorting, and Queue Splitting intervention, the queue is stable if and only if $\lambda < \frac{NC}{2\nu(\mathbb{E}[H_\pi]-1)+\omega\mathbb{E}[S_\pi]}$. $\square$

From Lemma 3, we know that the key to compare the stability conditions for different interventions is to compare the expectation of the highest reversal floor and number of stops. The analysis for FCFS can be found in the literature of elevator analytics [18], whereas the Cohorting and Queue Splitting intervention require new analysis. Next, we compute the distribution and expectation of the highest reversal floor (Lemma 4) and number of stops (Lemma 5) and provide a comparison.

Lemma 4. *(a) For $x = 2, \dots, m+1$, the cumulative density function of the highest reversal floors is*

$$\begin{aligned}
\mathbb{P}[H_{FCFS} \leq x] &= \left(\frac{x-1}{m}\right)^C, \\
\mathbb{P}[H_{QS} \leq x] &= \frac{\left\lfloor \frac{x-1}{k} \right\rfloor}{l} + \frac{1}{l}\left(\frac{x-1-k\left\lfloor \frac{x-1}{k}\right\rfloor}{k}\right)^C, \\
\mathbb{P}[H_{Cohort} \leq x] &= \frac{x-1}{m}.
\end{aligned}$$

(b) The expectation of the highest reversal floors is

$$\begin{aligned}
\mathbb{E}[H_{FCFS}] &= m+1-\frac{1}{m^C}\sum_{x=1}^{m-1} x^C, \\
\mathbb{E}[H_{QS}] &= \frac{k(l+1)}{2}+1-\sum_{j=1}^{k-1}\left(\frac{j}{k}\right)^C, \\
\mathbb{E}[H_{Cohort}] &= \frac{m+1}{2}+1.
\end{aligned}$$

(c) H_{FCFS} stochastically dominates H_{QS}, and H_{QS} stochastically dominates H_{Cohort}, i.e., $H_{FCFS} \succeq H_{QS} \succeq H_{Cohort}$. Therefore, $\mathbb{E}[H_{FCFS}] \geq \mathbb{E}[H_{QS}] \geq \mathbb{E}[H_{Cohorting}]$.

Proof. Proof. (a) In the FCFS intervention, the random event that the highest reversal floor to be no larger than x is equivalent to the event that the destination of each passenger is randomly chosen from 2 to x, which directly gives us the result.

For the Queue Splitting intervention, we can rewrite floor x as $(i-1)k+j+1$, so that floor x is the j-th floor in the i-th group. Therefore, $i-1 = \left\lfloor \frac{x-1}{k} \right\rfloor$, and $j = x-1-(i-1)k$. Conditioning on the fact that the current service group is the i-th group, the probability of the highest reversal floor is no larger than $x = (i-1)k + j + 1$ is equal to $\left(\frac{j}{k}\right)^C$, following the same argument for the FCFS intervention. Note that if x is in floor group i, all the trips in group $1, \dots, i-1$ satisfy the condition $H_{QS} \leq x$.

$$\mathbb{P}[H_{QS} \leq x] = \sum_{y=1}^{i-1} \mathbb{P}[H_{QS} \text{ in group } y] + \mathbb{P}[H_{QS} \leq (i-1)k + j + 1 | H_{QS} \text{ in group } i] = \frac{i-1}{l} + \frac{1}{l}\left(\frac{j}{k}\right)^C.$$

The distribution of the highest reversal floor for Cohorting directly follows from the definition of a uniform distribution on value $2, \dots, m+1$.

(b) A straightforward calculation shows $\mathbb{E}[H_{Cohort}] = \frac{m+1}{2} + 1$. We next use the tail formula to derive the expectation for FCFS and Queue Splitting intervention.

$$\begin{aligned}
\mathbb{E}[H_{FCFS}] &= \sum_{x=1}^{\infty} \mathbb{P}[H_{FCFS} \geq x] = 1 + \sum_{x=2}^{m+1} (1 - \mathbb{P}[H_{FCFS} \leq x-1]) = m + 1 - \sum_{i=2}^{m+1} \left(\frac{x-2}{m}\right)^C \\
&= m + 1 - \frac{1}{m^C} \sum_{x=1}^{m-1} x^C. \\
\mathbb{E}[H_{QS}] &= \sum_{x=1}^{\infty} \mathbb{P}[H_{QS} \geq x] = 1 + \sum_{x=2}^{m+1} \mathbb{P}[H_{QS} \geq x] = m + 1 - \sum_{x=2}^{m+1} \mathbb{P}[H_{QS} \leq x-1] \\
&= m + 1 - \sum_{x=2}^{m} \mathbb{P}[H_{QS} \leq x] = m + 1 - \left(\sum_{i=1}^{l}\sum_{j=1}^{k} \mathbb{P}[H_{QS} \leq (i-1)k + j + 1] - 1\right) \\
&= m + 1 - \left(\sum_{i=1}^{l} \frac{i-1}{l} k + \sum_{j=1}^{k} \left(\frac{j}{k}\right)^C - 1\right) = \frac{k(l+1)}{2} + 1 - \sum_{j=1}^{k-1} \left(\frac{j}{k}\right)^C.
\end{aligned}$$

(c) We first prove the that the random variables preserve stochastic dominance, i.e., $H_{Cohort} \preceq H_{QS} \preceq H_{FCFS}$. By definition, we only need to verify that $\mathbb{P}[H_{Cohort} \leq x] \geq \mathbb{P}[H_{QS} \leq x] \geq \mathbb{P}[H_{FCFS} \leq x]$ is true for all x. The first inequality is easy to verify since

$$\mathbb{P}[H_{Cohort} \leq x] = \frac{(i-1)k + j}{kl} = \frac{i-1}{l} + \frac{1}{l}\frac{j}{k} \geq \frac{i-1}{l} + \frac{1}{l}\left(\frac{j}{k}\right)^C = \mathbb{P}[H_{QS} \leq x].$$

Next we verify the second inequality.

$$\begin{aligned}\mathbb{P}[H_{FCFS} \le x] &= \left(\frac{(i-1)k+j}{m}\right)^C = \left(\frac{i-1}{l} + \frac{1}{l}\frac{j}{k}\right)^C = \left(\frac{(l-1)}{l}\frac{i-1}{(l-1)} + \frac{1}{l}\frac{j}{k}\right)^C \\ &\le \frac{(l-1)}{l}\left(\frac{i-1}{l-1}\right)^C + \frac{1}{l}\left(\frac{j}{k}\right)^C = \frac{i-1}{l}\left(\frac{i-1}{l-1}\right)^{C-1} + \frac{1}{l}\left(\frac{j}{k}\right)^C \\ &\le \frac{i-1}{l} + \frac{1}{l}\left(\frac{j}{k}\right)^C = \mathbb{P}[H_{QS} \le x].\end{aligned}$$

The first inequality follows from Jensen's inequality, and the second inequality follows from the fact that $i-1 \le l-1$. Following the stochastic dominance result, we obtain the desired ordering in expectation. $\square$

Since our simulation results in Section 1.3 are for a 25-story building, the highest reversal floor plays an essential role in the performance of the elevator system. Lemma 4 strongly supports the good performance of Cohorting and Queue Splitting in the simulation, as the distribution of the highest reversal floor preserves stochastic dominance across the three interventions we study. Note that the expected ascent time and descent time is equal to $2\nu(\mathbb{E}[H]-1)$. With the formula of $\mathbb{E}[H]$ from Lemma 4, in Propostion 4 we provide guarantees on the potential improvement by deriving the ratio of the expected ascent time and descent time between interventions.

Proposition 4. *(a) The ratio of the expected ascent time and descent time between FCFS and Cohorting is at least* $\frac{2Cm}{(C+1)(m+1)}$.

(b) For the special case of $C = 2$ *and* $m \ge 3$*, the ratio of the expected ascent time and descent time between FCFS and Cohorting is equal to* $\frac{4m-1}{3m}$*, which at least* $\frac{11}{9}$*. The ratio reaches the lower bound when there are only 3 floors and reaches the upper bound when the number of floors grows to infinity.*

(c) For the special case of $C = 2$*, the ratio of the expected ascent time and descent time between FCFS and Queue Splitting is equal to* $\frac{4m^2+3m-1}{3m^2+3m+mk-l} > 1$.

Proof. Proof. We first bound $\mathbb{E}[H_{FCFS}]-1$ and $\mathbb{E}[H_{QS}]-1$ by replacing summation with integral.

$$\mathbb{E}[H_{FCFS}]-1 = m - \frac{1}{m^C}\sum_{i=1}^{m-1} i^C \geq m - \int_{x=0}^{m} x^C dx = m - \frac{m}{C+1},$$
$$\mathbb{E}[H_{QS}]-1 = \frac{k(l+1)}{2} - \sum_{j=1}^{k-1}\left(\frac{j}{k}\right)^C \leq \frac{k(l+1)}{2} - \int_{x=0}^{k-1}\left(\frac{x}{k}\right)^C dx = \frac{k(l+1)}{2} - \frac{(k-1)^{C+1}}{(C+1)k^C}.$$

(a) We compare the expected ascent time and descent time between FCFS and Cohorting by considering the ratio

$$\frac{2\nu(\mathbb{E}[H_{FCFS}]-1)}{2\nu(\mathbb{E}[H_{Cohort}]-1)} \geq \frac{m-m/(C+1)}{(m+1)/2} = \frac{2Cm}{(C+1)(m+1)}.$$

(b) When $C=2$, we can compute $\mathbb{E}[H_{FCFS}]$ explicitly. Therefore, the ratio becomes

$$\frac{2\nu(\mathbb{E}[H_{FCFS}]-1)}{2\nu(\mathbb{E}[H_{Cohort}]-1)} = \frac{m-\frac{1}{m^2}\sum_{i=1}^{m-1} i^2}{(m+1)/2} = \frac{m-\frac{1}{m^2}\frac{(m-1)(m-1+1)(2(m-1)+1)}{6}}{(m+1)/2} = \frac{4m-1}{3m}. \quad (1.7)$$

Note that since Eq. (1.7) is increasing in m, we can plug in $m=3$ and yield the lower bound on the ratio and send m to infinity to get the upper bound.

(c) In the special case of $C=2$, we can also explicitly compute the expectation of H_{QS}. Note that since $m=kl$, we can simplify the ratio and get

$$\frac{2\nu(\mathbb{E}[H_{FCFS}]-1)}{2\nu(\mathbb{E}[H_{QS}]-1)} = \frac{m-\frac{1}{m^2}\frac{(m-1)(m-1+1)(2(m-1)+1)}{6}}{\frac{k(l+1)}{2}-\frac{(k-1)(2k-1)}{6k}} = \frac{4m^2+3m-1}{3m^2+3m+mk-l}.$$

Note that $\frac{4m^2+3m-1}{3m^2+3m+mk-l}$ is always greater than 1 since $l<m$. $\square$

From Proposition 4, we can observe that the value of the ascent and descent time can be shifted to lower values through the Cohorting and Queue Splitting interventions. The more groups it splits into, the greater the reduction can be. Next, we consider the distribution of the number of stops. The distribution and expected value of the number of stops can be found in the book [18]. We summarize the results in the Lemma 5 below.

Lemma 5. *(a) For each intervention, the distribution of the number of stops is as follows*

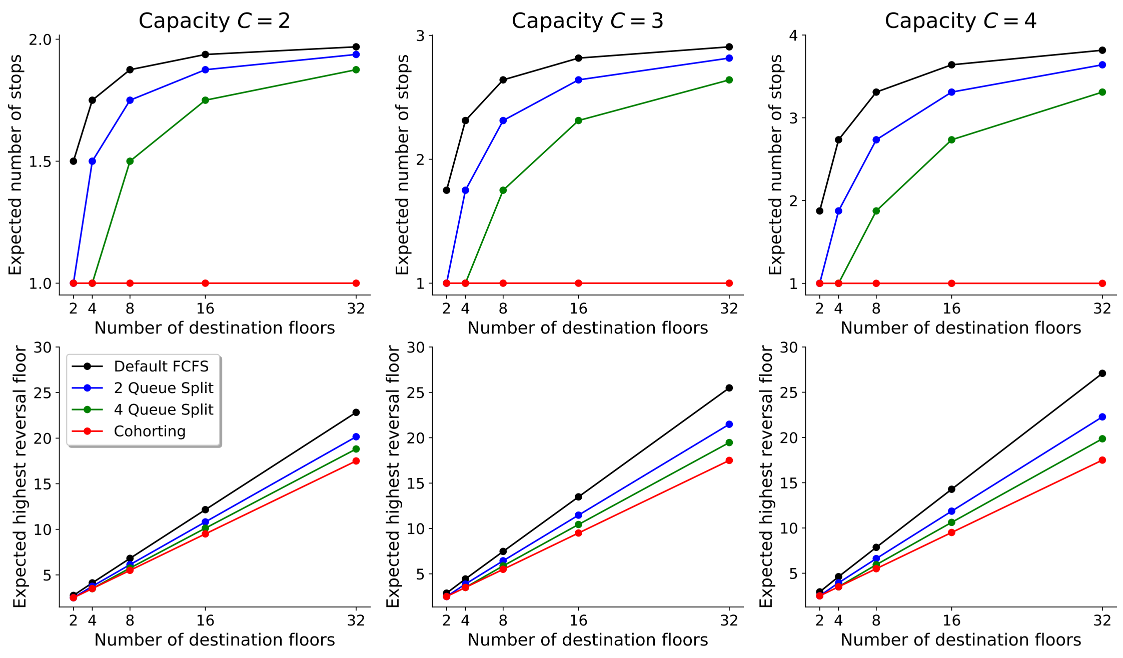
$$S_{Cohort} = 1 \text{ with probability } 1,$$
$$\mathbb{P}[S_{FCFS} = x] = \frac{x!}{m^C}\binom{m}{x}\left\{ {C \atop x} \right\}, x = 1, \dots, \min\{C, m\},$$
$$\mathbb{P}[S_{QS} = x] = \frac{x!}{k^C}\binom{k}{x}\left\{ {C \atop x} \right\}, x = 1, \dots, \min\{C, k\},$$

where $\left\{ {C \atop x} \right\}$ *is the Stirling number of the second kind, the number of ways to partition a set of* C *objects into* x *non-empty subsets.*

(b) The expected number of stops for FCFS and Queue Splitting is

$$\mathbb{E}[S_{FCFS}] = m\left[1 - \left(\frac{m-1}{m}\right)^C\right], \ \mathbb{E}[S_{QS}] = k\left[1 - \left(\frac{k-1}{k}\right)^C\right].$$

Figure 1.12: Expected highest reversal floor and number of stops in Lemma 4 and 5



With the distribution of H and S being derived in in Lemma 4 and 5, we plot the expected values in Figure 1.12 for various parameter settings. When the number of destination floors m becomes large, the expected number of stops will approach the capacity C in both FCFS and

Queue Splitting intervention. However, Queue Splitting is increasing much slower, and the more groups we split into, the lower the value is. Similar behavior can be observed in the expected highest reversal floor graphs.

When the elevator capacity C increases, the reduction in the expected values of S and H becomes more significant for both the Queue Splitting and Cohorting intervention when compared with FCFS. Using a building with 32 floors as an example, 2 Queue Split can reduce the average number of stops by 4.6% when $C = 4$, but only by 1.6% when $C = 2$. For the average highest reversal floor, 2 Queue Split reduces it by 17.8% when $C = 4$, but only by 11.8% when $C = 2$. In general, the higher the capacity C is, and the larger the number of floors m is, the more difference we can observe from Figure 1.12. Therefore, it is indeed more critical for higher buildings with larger elevators to apply the proposed interventions to reduce the round trip time and make the queuing system more efficient.

1.6 Practical Issues in Cohorting

Cohorting as discussed in Section 1.5 is focused on the situation where there is a large queue and we can create a cohort of passengers perfectly. Under this setup, we proved Cohorting is always the best in terms of highest reversal floor and number of stops (Lemma 4 and 5). Of course, in our simulations it may not be always possible to cohort perfectly if the queue length is under control, and we see in the numerical results that Cohorting has inferior performance in highest reversal floor compared to 2 Queue Split in Figure 1.5. Cohorting has a slightly higher average H value (18.8) than 2 Queue Splitting (17.6). However the number of stops in Cohorting (2.78) is lower than 2 Queue Split (3.27). Therefore the overall round trip time, impacted by both number of stops and highest reversal floor is lower in Cohorting in the simulations, since each stop takes 15 seconds whereas travelling one floor takes only 1.4 seconds.

In the remainder of this section, we discuss three practice-related issues about our interventions that may improve or hurt their performance. For concreteness, we focus on Cohorting, but similar modifications can be made to other interventions.

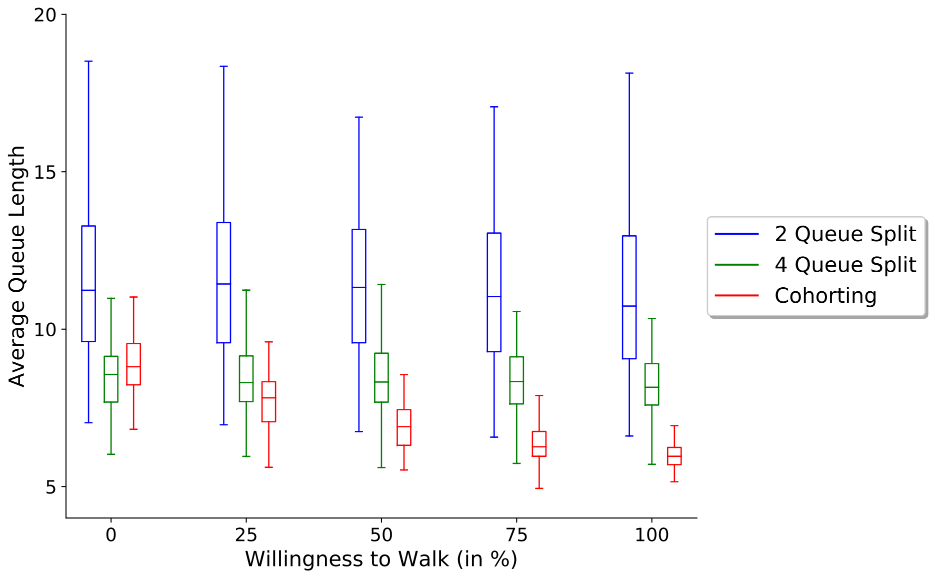
1.6.1 The Impact of Willingness-to-Walk

First, when the queue length is not large, we may not find enough people who go to the same floor. One way to increase the chance of people going to the same floor is by asking them to take the stairs. We model this behavior using the Willingness-to-Walk (WtW) parameter indicating the probability that a given passenger would walk one floor up or down from their intended destination instead of preferring to only go to their destination floor. For example, if $WtW = 20\%$, then 20% of all passengers whose intended destination is floor d would consider the option of taking an elevator to any of the floors $d-1, d$ or $d+1$. Our consideration of WtW is inspired by literature showing that leveraging demand-side flexibility can be effective in managing operations [51, 52].

When some passengers have the willingness of walking one floor up or down to their intended destination, the system may benefit from the potential reduction in the number of stops each elevator trip needs to make. Using simulation, we can see how much value it provides for different levels of WtW on the Cohorting and Queue Splitting interventions. In the Cohorting intervention, passengers line up in a single queue and the Queue Manager asks along the line whether the passengers are going to the target floor, which is the first passenger's destination. If a passenger is going to an adjacent floor of the target floor and is willing to walk, then they would say yes and join the cohort with the first passenger. In the Queue Splitting intervention, we assume that a passenger who is willing to walk will choose the shortest queue to join upon arrival, which can either go to the final destination directly, or stop at one floor lower or higher. Note that this option is only available to passengers whose destination floor is at the boundary of a floor group. We summarize the results in Figure 1.13.

In Figure 1.13, we report the boxplot of the average queue length for 3 interventions with WtW level from 0% to 100%. When $WtW = 0.25$, then Cohorting can be improved an additional $10-20\%$, while if $WtW = 100\%$ (which is idealistic), Cohorting can be improved by up to $30-40\%$. There is barely no change in the queue length when increasing WtW from 0% to 100% for Queue Splitting, though the performance varies slightly due to randomness in the 100 simulated instances, because the distribution of the highest reversal floor and the number of stops

Figure 1.13: Plot of average queue length in the lobby v.s. Willingness-to-Walk.



Note. We run 100 independent random instances and report the average performance. Between 8 to 10 AM, 2750 passengers with destinations ranging from floors 2 to 25 are served by 14 elevators (each with capacity 4). In the 4 queue split intervention, we divide the floor ranges equally among all the queues. The queue length under the Default FCFS intervention is on average 62 passengers across the 100 random instances.

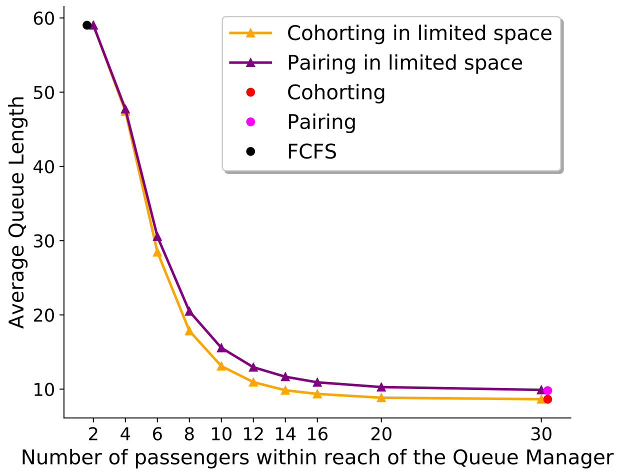
do not change much when we allow passengers whose destination is at the boundary of the queue ranges to switch to another queue. Overall, the willingness of passengers to walk one flight can improve the performance if Cohorting is implemented, but the benefit is marginal (comparing to the queue length decrease one can observe by solely using Cohorting, which is approximately 62 in FCFS to 9 in Cohorting). Furthermore, the effect may be overestimated since passengers may not comply with their willingness to walk a flight of stairs once they board, since they can easily push the floor button they desire, and it may take extra time for communication when walking is included in the operations.

1.6.2 Limited Space and Communication Time

The second practical issue relates to the number of passengers the QM can reach. In a particular building, the QM may not be able to communicate with everyone in the line. The Queue Manager cannot reach out to people beyond a point, perhaps due to a turn in the hallway or the small size of the lobby. We reevaluate the Cohorting intervention with an extra constraint, which is that the

Queue Manager can only consider a certain number of passengers from the front of the queue. In Figure 1.14, we study Cohorting with a limited number of people within reach of the QM for the large building case study.

Figure 1.14: Performance of the Cohorting intervention with practical considerations.



Note. We run 100 independent random instances and report the average performance. Between 8 to 10 AM, 2750 passengers with destinations ranging from floors 2 to 25 are served by 14 elevators (each with capacity 4). We consider two practical issues: (1) limited number of passengers within reach of the QM, and (2) only finding another passenger to be paired with the first passenger. The black, red and pink dot in the graph are performance benchmarks without the limit on the number of passengers the QM can talk to.

The final practical issue is the extra time Cohorting may take due to the communication time it takes for the QM to learn about the passengers' destination. To simplify the Cohorting implementation in this simulation, we propose the Cohorting with Pairing intervention, which only requires the Queue Manager to find one other passenger with the same destination as the first person and create a "pair" to board the same elevator. In the original Cohorting intervention, the QM tries to match up to $C - 1$ people with the first person in line. Loading an elevator of capacity 4 with one pair leads to at most 3 stops, and two pairs leads to 2 stops being made by the elevator. A pseudocode implementation of the Cohorting with Pairing intervention is available in Section 1.2.2.

Figure 1.14 shows the performance of Cohorting and Cohorting with Pairing interventions when the number of people within reach of the QM is limited. As comparison benchmarks, we also plot the performance of the FCFS, Cohorting, and Cohorting with Pairing intervention without the constraint on the number of passengers that can be considered by the QM. The first observation is that Cohorting with Pairing is an effective and easy-to-implement intervention, as it performs almost as good as the Cohorting intervention when the QM can reach the same number of passengers. Moreover, as the QM can approach passengers further in the queue, the average queue length shrinks rapidly. When the Queue Manager can reach out to about 10 people, the queue length is already less than 20 and being able to reach more than 10 passengers adds marginal value. Therefore, it is critical to design a safe queuing plan with physical distancing such that 10 people can hear the QM which is practical and reasonable target. Moreover, the QM can simply implement the Cohorting with Pairing intervention in the limited lobby space. The figures may vary across different buildings and our code implementation can easily be changed to analyze different settings.

1.7 Conclusions and Future Work

This project was done with the guidance of New York City Mayor's Office of the Chief Technology Officer and the Department of Citywide Administrative Services, which had continuous input into our work throughout the process. Through this work, we combine mathematical modeling and epidemiological expertise to design interventions for safely managing elevator systems amidst a pandemic. The social distancing requirement during a pandemic may lead to large buildup of queues in the lobby during busy periods when using FCFS. We propose various interventions with a Queue Manager to help load passengers in the lobby. The fundamental idea behind these interventions is to try to reduce queue buildup by maximizing the number of people in an elevator trip going to the same floor (or nearby floors), which in turn reduces boarding/deboarding times as well as travel times. The interventions we study apply to generic buildings, and we have provided open-source code so other building settings can be studied. The intervention chosen by a building may depend on its particular simulation results, physical layout, personnel, and epidemiological considerations. For example, in the large NYC building case study, the maximum queue length in Cohorting and 2 Queue Split are respectively over a factor of eight and five times smaller than that of FCFS. Cohorting is even effective when the QM can only talk to the first few people in the queue, or when we cohort in pairs only. We also provide customizable open source code and an instructional video explaining our interventions.

A comparison of the two proposed interventions is provided in Table 1.2. Our simulations show that the Cohorting intervention leads to lower waiting time for passengers in the lobby and reduces the number of people in the lobby (queue length) significantly. If the QM cannot talk to many people in the line, we suggest the Cohorting with Pairing intervention in limited space which is easier to implement and provides similar benefits as Cohorting, as long as the QM's announcement can reach a suitable number of people in the line. We also propose the Queue Splitting intervention which implicitly groups similar passengers together to improve efficiency while needing less communication from the QM. Queue Splitting with even a small number of queues achieves comparable performance to Cohorting. The proposed interventions are effective beyond the constraints imposed by a pandemic, and thus are still useful after the pandemic to manage lobby queues.

Table 1.2: Cohorting vs. Queue Splitting

	Cohorting	Queue Splitting
Pros	• Shortest service time in theory • Shorter queue length across buildings	• Less communication needed, QM not necessary • Ease of understanding for users and managers
Cons	• Need a QM for good implementation • Communication may be difficult	• Worse than Cohorting under heavy traffic • Need space for horizontal separation of queues

In this work, we have only considered the problem of moving people upwards in a building from the lobby. Without any elevator AI, it is near-impossible to do any interventions for downward and inter-floor movement. Using sensors, it would be possible to know how many people are in each elevator, where they are going, which floors have a request, and how many people are waiting on each floor. We could then intervene, allowing us to design algorithms that balance efficiency of the system with fair waiting times, while maintaining the safety standards necessary [36]. For instance, due to the reduced elevator capacities, a passenger on a middle floor may have difficulty leaving the building during lunchtime. Every time an elevator arrives, it may be filled with passengers from higher floors. In future work, one can design algorithms that mitigate such a situation, which is likely (and known) to occur.

There are many other considerations to be investigated. Due to perceived inequity in interventions like Cohorting which let passengers jump the queue maybe for the greater good, there could be individual frustrations [34, 48]. One can also only implement an intervention when the queue length exceeds a threshold and otherwise rely on FCFS, which reduces the overall need of a QM. In our study, the passenger arrival patterns and destinations were generally stationary and uniform, and different effects may occur otherwise. However, we note that if some floors are more popular than others, then it may actually be easier to implement Cohorting. Finally, given more data and knowledge of the internal elevator algorithms, our models could simulate inter-floor traffic more accurately.

Chapter 2: Recovering Passenger Schedules in Airline Disruptions

In this chapter, we present our work on passenger scheduling algorithms for the airline recovery problem. This work is in collaboration with Clifford Stein, and based on my internship experience at GE Global Research in 2019 with Dr. Paul Ardis (Research Platform Leader) and Dr. Srinivas Bollapragada (Chief Scientist).

The organization of this chapter is as follows- we provide the background behind airline disruptions and recovery in Section 2.1 and summarize the prior work in the field in Section 2.2. We outline our contributions in 2.3. In Section 2.4, we introduce the basic concepts of the problem space. Section 2.5 details our proposed algorithms for airline recovery, with experiments based on a real-world data set in Section 2.6. Finally, we conclude and discuss ideas for future work in Section 2.7.

2.1 Background on airline disruptions

In many transportation problems (e.g., airline scheduling, container transshipment, traffic control, vehicle routing etc.), operations research tools are needed and used to help the decision makers to prepare schedules [53, 8]. The planned schedules suffer in practice when availability conditions on resources change unexpectedly, and thus the initial plan is cannot be fulfilled as planned. There are two main approaches in literature to handle the uncertainty: (i) to anticipate uncertainty explicitly when constructing the schedules, called *proactive or robust scheduling* (see detailed discussion in Albers [54] and Grötschel *et al.* [55]); (ii) to modify decisions when data is revealed, called *reactive scheduling* [56, 57].

Robust scheduling makes sense when irreversible structural decisions are made over a long planning horizon, or when uncertainties can be quantified in some way (see discussion in Kall *et al.* [56]). Of course, one can think of constructing robust schedules at the planning stage that can adapt to disruptions. However, in the setting of airline scheduling, disruptions are complex, propagate through the network, and are unpredictable [53, 8, 15]. Robust planning is a difficult task particularly due to the unpredictability of airline disruptions and resource constraints. There have been efforts to set up a theoretical framework to evaluate and quantify robustness (see a summary in Clausen *et al.* [14]). In this work, we focus on reactive scheduling to manage disruptions by re-planning schedules within a given computational requirement. We wish to keep the new schedules close to the original pre-planned schedules during the disruption, with the mandate of strictly getting back to the initial schedule after the end of a *recovery window* [11].

The top challenges for airlines in 2018 were network disruptions, unplanned maintenance, and fuel overspend [12]. Airline disruptions occur due to a resource needed to operate a flight unavailable before departure and may affect the departure of one or more flights [58]. Disruptions occur at short notice and cause flight delays and cancellations, ferried flights (i.e., almost empty flights being flown from a different airport just to be used at the destination), crew and passenger misconnects, and have a domino effect on the whole network. According to industry statistics, in 2018, 5.6 million departing flights were delayed for an average of 57 minutes, $436,000$ flights were canceled, and the travel plans of over 655 million passengers were disrupted [12]. Irrespective of any gradual improvements, such statistics reflect poor performance. Moreover, disruptions are now commonplace [12], affecting airline branding, airline costs, and passenger loyalty. Delays cost airlines approximately $\$97$ per minute and cancellations an average of $\$68,000$ per flight, with an overall cost of $\$33.4$ billion in 2018 [12].

Figure 2.1 from Papiomytis [12] shows the breakdown of disruption costs to airlines. The major costs are passenger welfare (reimbursing cancelled itineraries, providing hotels or refreshments, passenger dissatisfaction due to delays, etc.), followed by the costs of changing aircraft and crew schedules which are often tightly regulated.

Figure 2.2 from Papiomytis [12] highlights the reason for airline delays and disruptions. There is a myriad of reasons shown for network disruptions, for example, infrastructure constraints such

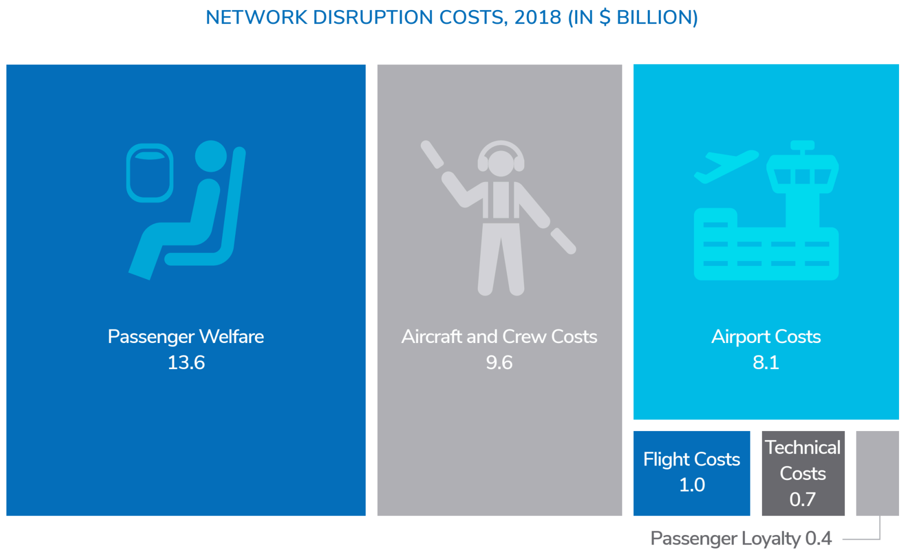


Figure 2.1: Breakdown of costs due to disruptions. Passenger welfare is the leading cost when there are airline disruptions, highlighting the need for airlines to reschedule passengers efficiently and quickly. From Papiomytis [12].

as airport and airspace (congestion). The reality is that two-thirds of all delays and cancellations are within an airline's control. Indeed, almost half of all delays (43%) are reactionary in nature and caused by the primary delay (e.g., bad weather or airport congestion). Thus, the significance of reacting well to the uncertainty of a primary disruption by re-planning schedules is illustrated in Figure 2.2.

Airlines make their schedules well in advance [11, 59]. Figure 2.3 from Kasirzadeh *et al.* [59] shows how airlines think about their decision-making in scheduling. A flight schedule is created using demand predictions and resource constraints six to twelve months in advance [8], followed by a fleet assignment that allocates aircraft types while maximizing revenue. Next, one to six months in advance, aircraft routing determines which aircraft (*tails)* fly which routes, while taking maintenance into account [8]. A month in advance, crew pairings (sequence of flights and duties) are assigned, often solved with a crew bidding algorithm [59]. Passengers book flight tickets subject to the capacity of each flight and cabin class and expect airlines to stick to the posted schedule.

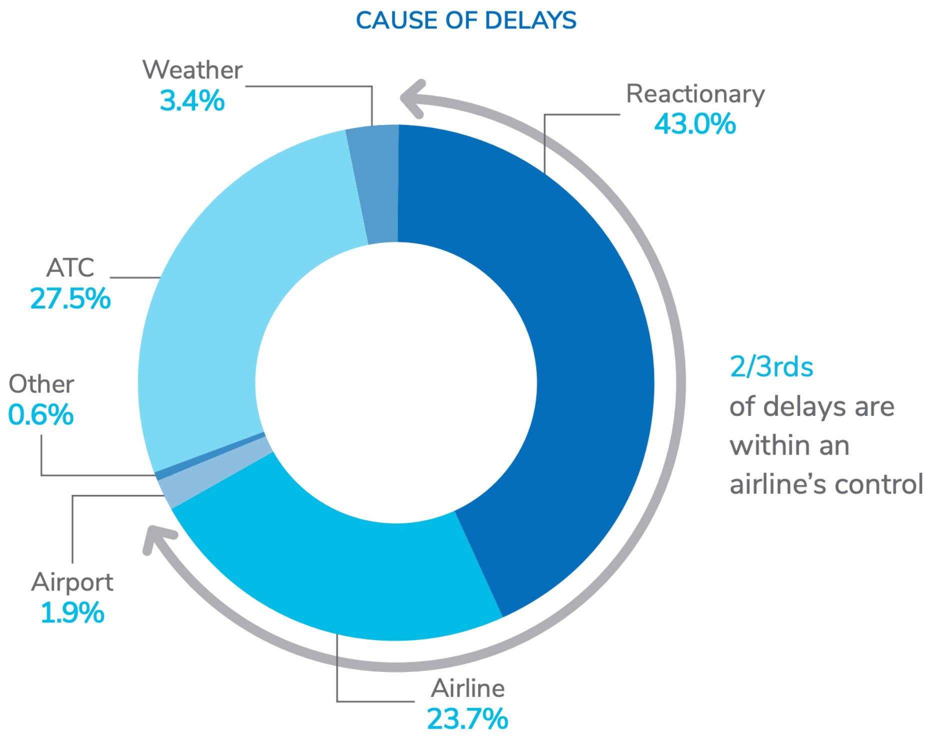


Figure 2.2: Reasons for disruption. Reactionary delays are responsible for 43% of disruptions. From Papiomytis [12].

Thus, a carefully pre-planned schedule for flights, crew, and passengers are all affected by random events that cause disruption, and a domino effect of the primary disruptions propagates throughout the network. Figure 2.3 shows the problem of replanning in the face of disruptions, called *airline recovery procedure* for flights, crew members, and passengers. A host of decisions can be made- adjusting flight plans, aircraft assignments, crew assignments, and passenger itineraries within the period of time called the recovery period [58, 9], which by definition also means that at the end of the time window, the schedules are required to match the original schedules. The length of the recovery period depends on the airline and disruption [60]. Airlines expect to make quick, user-independent, consistent, and near-optimal recovery decisions [12, 15].

There are three major recovery problems- replanning flight schedules (*operations recovery*), crew schedules/pairings (*crew recovery*), and passenger schedules (*passenger recovery*). As explained in Clausen *et al.* [14], recovering schedules is a complex task, since many resources (crew, aircraft, passengers, slots, catering, cargo, etc.) have to be re-planned. Large airlines focus on the ground problems first, necessitating a sequential process of solving flight infeasibilities, followed

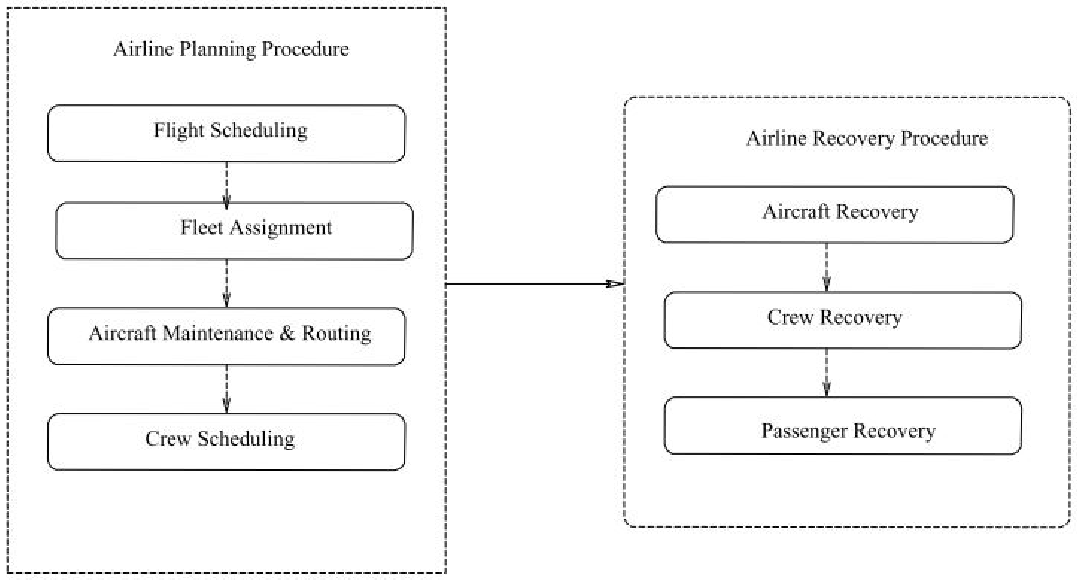


Figure 2.3: Airline decision making. From Kasirzadeh *et al.* [59].

by crew problems. Finally, the impact on passengers is evaluated [10, 14]. The objective function can be composed of several conflicting and sometimes non-quantifiable goals [14]. In most airlines, controllers performing the recovery have only limited decision support to help them construct recovery options or evaluate the quality of the recovery action they are about to implement. Often, controllers are content with only producing one viable recovery plan since there is no time to consider alternatives [14].

Thus, the recovery problems are traditionally solved sequentially: operations, then crew, followed by passenger recovery [9, 15, 61]. Instead of a global optimization approach, decision-making at one level can affect other problems, leading to huge inefficiencies [15]. Integrating different levels of recovery (called *holistic recovery*) is a significant goal to achieve.

Sequential optimization in the presence of stochastic external disturbances presents many challenges. In disruption of networks, an operations recovery solution may be very unfriendly to crew recovery feasibility. The bigger picture here is that we have planned schedules and resource allocations but there could be major drivers of anamoly- unusual events and key performance indicators (e.g. on-time performance). Right now, there is no reliable way to score solutions in the solution space of these problems, giving us no strong measure of the quality of a solution [9]. Therefore, instead of designing robust schedules, we prefer to obtain fast (in order of minutes) recovery solutions. In the next section, we present prior work on airline disruptions, and the different solution approaches proposed in literature.

2.2 Prior Work and Approaches

Airlines have been using OR models to solve complex planning and operational problems [62]for decades, but the expanding size of airlines and their networks have led to increased complexity. Since 2001, there has been a focus on suboptimal and fast solutions that fare well under uncertainty [63, 64, 65, 66] rather than expecting optimal solutions. With advances in processing and clusters, solving large linear or mixed integer programs is becoming easier. Recent work has been focused on the challenge of holistic or integrated recovery [15, 9] with not just mathematical programs, but heuristics like neighborhood search [58], as well as graph-based algorithms [11].

We state the definition of *integrated recovery*- solving for at least two schedules among flight, crew, and passenger recovery together. We list the approach to airline recovery into categories based on prior works (see surveys in Filar *et al.* [10], Bratu and Barnhart [67], Ball *et al.* [13], Clausen *et al.* [14], and Castro *et al.* [16]):

- *Non-integrated recovery*, where only one of the three networks or dimensions is considered.
- *Partially integrated recovery*, where some two of the dimensions (aircraft, crew, and passengers) are considered.
- *Holistic recovery*, where the three dimensions (aircraft, crew, and passengers) are considered simultaneously.

Integrated recovery has been an active research topic in recent years [16, 9]. A summary of important work related to partially integrated recovery and holistic recovery is presented in Figure 2.4 from Vink *et al.* [68].

In our work, we focus on solving only one of the three scheduling problems- the passenger recovery problem. We sequentially optimize, by using the solution to flight recovery as an input.

Thus, our work falls under the partially integrated recovery domain. In the next few subsections, we discuss prior works on partially integrated recovery, especially those that include passenger recovery.

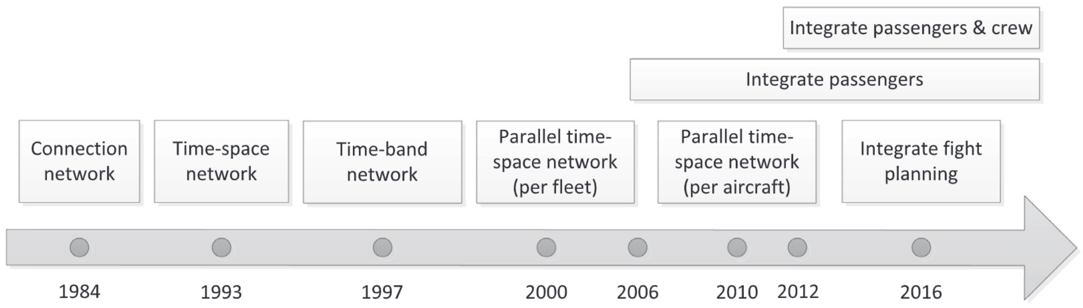


Fig. 1. Timeline of developments in modeling the aircraft recovery problem.

Figure 2.4: Timeline of developments in modeling the aircraft recovery problem, from Vink *et al.* [68]

2.2.1 Passenger Recovery

In airline disruption management, high passenger delay costs and continuous flight disruptions lead to a potential loss of goodwill and long-term reputation damage and hence passenger recovery solutions are sought after by airlines currently (see discussions in [9, 12, 16]. Recall Figure 2.1 where passenger welfare costs were the leading cost ($13.6 billion in 2018) among all airline disruption costs.

Passenger recovery can be formulated as follows: given recovered flight schedules, and a set of pre-planned passenger itineraries, the set of itineraries where at least one of its flights is disrupted in some way (delays, cancellations, reduced cabin capacity, etc) is called disrupted itineraries. We wish to replan all disrupted itineraries, by making decisions on whether passengers can be moved to different flights, cabin classes, or outright cancel their itinerary. For each disrupted itinerary, we use the recovered flights (given seat availability) necessary to reaccommodate passengers from their starting position at the time of disruption to their destination while minimizing cost [15, 9].

The passenger recovery costs can include both *operating costs* and *disutility costs*. Operating or regulatory costs, often referred to as *hard* costs in literature [15] are directly incurred (and paid in real money) when a passenger cannot complete their scheduled itinerary (e.g., compensation for delay and cancellation, providing refreshments or hotels, as stipulated by government regulations). Disutility costs (or *soft* costs) are the potential losses of future revenue as a result of passenger inconvenience, possibly causing the passenger to switch to a different airline in the future. The costs are approximations made by the airline, can differ per passenger class or frequent flyer status [15, 11], and are meant to be used for reactive decision-making to disincentivize certain outcomes. For example, the disutility for cancelling a passenger's itinerary is set to be higher than delaying their itinerary by the maximum allowed delay. The model would thus rather delay than cancel itineraries, whenever possible.

We summarize in detail the prior works on partially integrated recovery that incorporate passenger recovery. In Kohl *et al.* [61], the authors present a sequential modeling with a distinct approach for each of the problem dimensions, aiming to minimize operating costs and passenger costs. In particular, for the passenger dimension, multi-commodity network programming was used. The model considers the cost of delay at the final destination for the relocated passenger, direct costs for rescheduled passengers (hotel, food, etc.), impact on customer perception, and costs related to class upgrade and downgrade (in this case strongly impacting the customers' experience and their willingness to fly again with the company).

Maher [69, 70] present a sequential approach using mathematical programs, and resolution with a column generation method. Passenger recovery occurs after the resolution of aircraft and crew dimensions. The passenger cost depends on the number of passengers relocated due to flight cancellation and also on the delay costs for passengers relocated to other flights.

Jafari and Zegordi [71] use mixed mathematical programming modeling, applying Benders' decomposition method. Passenger costs depend on the number of passengers impacted by cancellations and on delays to passengers related to their final destination. Jozefowiez *et al.* [72] present a large search in the vicinity heuristic, considering, for passengers, the costs of delay, flight cancellation, and cabin downgrades.

Some authors propose exact optimization methods for flight and passenger recovery. For example, the authors in Hu *et al.* [73] presented an integrated integer programming model where the objective is to minimize the total cost associated with the reassignment of aircraft and passengers to flights. One assumption the authors make is that all passenger itineraries are comprised of a single flight leg, which is not reflective of real-world data. There are also mixed-integer non-linear programming models [74, 75] and integer linear programming models [76] with different settings and constraints considered. The authors in Cook *et al.* [77] studied the inconvenience experienced by passengers as a sigmoid function of delay duration. Most works generally use a piece-wise linear relation for delay costs, if they seek to avoid a nonlinear recovery model [15].

A key set of papers involving passenger recovery emerged due to an OR challenge organized by The French Operational Research (OR) and Decision Support Society (ROADEF) in 2009 jointly with Amadeus S.A.S., called “Disruption Management for Commercial Aviation" [15]. Multiple teams of researchers participated with the top finalists publishing their work. The winning paper by Bisaillon *et al.* [11] proposed a large neighborhood search heuristic combining fleet assignment, aircraft routing, and passenger assignment. Sinclair *et al.* [78] improved the above heuristic, and a post-optimization approach was further proposed in Sinclair *et al.* [79]. Jozefowiez *et al.* [72] was another finalist, and in their approach, the passengers are grouped by itinerary and prioritized based on the size of the group. Later work by Zhang *et al.* [60] provided a mixed-integer programming based approach that beat all the finalists. Recently, a multi-objective genetic algorithm approach by Yang and Hu [80] allowed passengers to either accept an itinerary change or demand a refund of the tickets. Jafari and Zegordi [71] and Jafari and Hessameddin Zegordi [81] used aircraft rotations and passenger itineraries instead of flights, but do not reflect operations of a large airline [9]. A MIP formulation made up of multi-commodity flow problems is constructed in [82]. The authors of Eggermont *et al.* [83] decompose the recovery into smaller subproblems, with the output of each becoming the input of the next, and use shortest path algorithms partly.

Since 2009, there has been only one work on the passenger recovery problem stand-alone (see discussion in Hassan *et al.* [9]). In this work, the authors McCarty and Cohn [84] present a two-stage approach to handle rerouting passengers, by re-accommodating passengers as soon as a delay is known and before the length of the delay is realized. In the first stage, once information about a flight being delayed is known, passengers are preemptively assigned to new itineraries in anticipation of the delay's impact. The second stage further modifies itineraries for passengers who miss connections after the delay has been realized [9]. Their experimental case study considered a recovered flight solution that delays only one flight. Thus, the context of our work is to present new algorithms for the passenger recovery problem, taking flight recovery as an input.

2.3 Our contributions

In this chapter, we focus on developing fast and near-optimal algorithms for passenger recovery. Section 2.4 details the concepts of the recovery problem space, and our contributions are explained in brief below.

1. In Section 2.5.1, we construct, for each itinerary, directed acyclic graphs containing data on feasible airports and flights between them, using depth-first search and pruning. The *preprocessing* step is crucial in reducing the problem sizes of our proposed algorithms.

2. We propose an integer program (IP) based Algorithm 4 in Section 2.5.2. We experimentally test the IP algorithm on realistic data sets from the ROADEF challenge [15]. The run time of a recovery algorithm also includes the time to construct the model, and requires a solution in the order of minutes [68, 9, 15]. We illustrate the impact of the preprocessing step- the problem size of the IP was $< 1\%$ using preprocessed graphs than without the preprocessing step.

   In our experiments, we use a commercial IP solver (Gurobi). The IP solutions contain only a small number of nonzero variables, and the run times are in the order of seconds for the largest data sets we use. Publicly available data on passenger itineraries is limited [9, 15]. Thus, it is a challenge to pin down tractability in real-world airline networks, since the number of variables and constraints of the IP has a linear dependence on (i) the number of flights in the network, and (ii) the number of passenger itineraries served.

3. Therefore, in Section 2.5.3, we are interested in developing new algorithms without mathematical programs. Inspired by some existing ideas in literature for solving resource-constrained elementary shortest path problems [17], we develop a multiple label shortest path (MLSP) Algorithm 5 with labels only on the nodes, with frequent pruning, and a breadth-first search to compute new labels. The MLSP algorithm iterates on each disrupted passenger itinerary, solving for high-value itineraries first. Another advantage of a network-based approach is that they can handle itinerary-specific constraints easily. For example, in practice, there is often a restriction on the maximum number of legs a recovered itinerary can have, compared to the original itinerary [15]. For instance, if a given itinerary has j legs, the new itinerary could have at most $j+2$ flight legs. We would need a non-linear constraint in the IP approach to handle such a constraint.

4. Because we solve for one itinerary at a time, we could exhaust the precious remaining flight capacities on an earlier itinerary, and have to potentially make suboptimal choices for subsequent itineraries. Hence, we also investigate the idea of considering "batches" of itineraries together [85, 86] to potentially improve the solution quality of the MLSP algorithm.

5. We perform experiments and compare the performance of our algorithms on a data set calibrated from an OR challenge [15] on integrated operations and passenger recovery. We generate two types of disruption scenarios: (i) one where no flights are cancelled, and a small percentage of flights are delayed. Thus, only a small fraction of the original itineraries are disrupted and require replanning. (ii) one where there are plenty of cancellations and flight delays. Thus, most original itineraries are disrupted and require replanning.

2.4 Problem setup and concepts

There is a lack of publicly available data sets with enough detail for the passenger recovery problem [9] apart from Palpant *et al.* [15], and therefore, our experimental data sets are calibrated from the ROADEF challenge. We use the setup from Palpant *et al.* [15] to describe the basic concepts of the passenger recovery problem.

2.4.1 Time-space network

An *airline network* is a set of flight legs (defined as a nonstop flight from an origin to a destination) offered by an airline [15]. The airline operates an aircraft fleet $\mathcal{Q}$, where each aircraft $q \in \mathcal{Q}$ (e.g., a Boeing 747 with a particular tail number) has a cabin configuration fixed for the aircraft type. A cabin configuration is the number of seats available for each cabin class, comprising of Economy (E), First Class (F), and Business (B). The set of cabin classes $\mathcal{M} := \{F, B, E\}$ have a natural ordering, with $F > B > E$ when modeling downgrading costs, to be defined later.

A *flight schedule* s is a set of all flights operated by an airline in a given time period. We represent the flight network as a graph $G_s = (V_s, A_s)$ at a given time period, where V_s are the set of nodes/airports, and each arc in A_s represents a flight leg. The same aircraft could be used for multiple flights, and disruptions can make aircraft unavailable and thus, removing all flights using that aircraft from the network. Each flight in the schedule is defined by:

- a flight number f along with the duration, e.g., Flight 261 of duration 3 hours and 30 minutes.
- origin airport f_{orig} and destination airport f_{dest}, e.g. JFK to LAX.
- flight type (domestic, continental or intercontinental) which is abbreviated D/C/I.
- departure time and date f_{dep}, arrival time and date f_{arr}, e.g., $1/1/2018\ \ 8$ AM.
- aircraft $q_f \in \mathcal{Q}$ used by the flight that also determines the capacity of the flight in each cabin class $m \in \mathcal{M}$.

As per the ROADEF data set [15], we also allow flights that are just surface public transportation between airports in the same region (e.g., shuttles between airports like JFK and LGA). The surface transport “flights" are assumed to have infinite capacity and no operating cost.

2.4.2 Passengers

Passengers have prior reservations on flights. We group passengers into a set K of *itineraries*, where each itinerary has the following features:

- a unique itinerary number k.
- number of passengers n_k in the itinerary.
- average cost paid by a passenger in this itinerary c_k^{price}.
- nature of itinerary- inbound or outbound (A/R in French)- outbound is a one-way trip or outbound portion of a round trip whereas Inbound is the return portion of a roundtrip whose outbound portion was finished before the planning horizon. Inbound is prioritized over outbound when replanning, as passengers should not be left stranded in an intermediate airport (which might require hotel costs to be paid by the airline).
- description of the itinerary- one or several flight legs, with one cabin class m for each leg.

An example of an itinerary is

```
2 Out $1752.5 27 F243 20/01/08 B F245 21/01/08 B
```

This is itinerary $k = 2$ with $n_2 = 27$ passengers booked from Singapore to London (on flight 43 on $20/01/08$) and London to Paris-Charles de Gaulle (Flight 245 on $21/01/08$), on the *outbound* portion of their trip, travelling in cabin class $m = B$ on both flights, and having paid $c_2^{price} = 1,752.50$ units of money on average.

2.4.3 Disruption and recovery period

The goal of any recovery problem is to resume normal operations as quickly as possible, after disruptions occur, affecting the planned schedule. We model the problem as a specific time at which a disruption occurs, followed by the *recovery period* for which we must determine new schedules, and at the end of the recovery period, the original schedules must resume. The recovery period depends on the needs and philosophy of each airline (e.g., 24 hours or up to 96 hours) [60].

Airline disruptions can reflect in practice in many ways: (i) Air Traffic Control restrictions (e.g., they are told to reduce their in and out flight volume by say 50% between $3 - 8$ PM at JFK airport due to inclement weather), (ii) an aircraft being unavailable due to maintenance/repair, (iii) a flight being delayed or cancelled.

In any case, the disruption occurs at a start time, say 3 PM and the airline can take say 24 hours to recover until the normal schedule resumes at 3 PM the next day. Recall in Figure 3.1, that half of all delays (43%) are reactionary in nature and caused by the primary delay.

2.4.4 Decision-making during disruptions

The first decision to be made is *operations or flight recovery*, where the airline must come up with a new flight schedule using the available aircraft and operational constraints. We denote the solution to operations recovery as F_{recov}, computed by solving a cost minimization problem while making decisions such as intentional flight cancellations and delay of an existing flight f, adding new flights or aircraft changes q_f for flight f. In our work, we assume F_{recov} is available as an input to the problem.

The next decision is making passenger recovery schedules given the set of recovered flights. The costs in passenger recovery are modeled as a combination of (i) *operating or regulatory costs*, i.e., delaying or canceling a flight forces the airline to compensate for food or hotels for the passengers, or refund fares partially or fully; (ii) *passenger inconvenience*, i.e., model costs that penalize changes to aspects of the passenger itinerary that worsen their experience [15]. There may be some itineraries $k \in K$ where all its flight legs are unimpacted, with no change in flight characteristics. The passenger recovery solution K_{recov} to be computed keeps the unimpacted itineraries the same and only replans for passengers who are disrupted in some way, denoted by the set $K_{dis} \subseteq K$. Three key decisions can be made for each itinerary $k \in K_{dis}$:

1. re-accommodate some or all passengers to a new itinerary, using flights and cabin classes

from F_{recov}.

2. downgrade cabin classes (e.g., $F \to E$) on some or all of the flights in the original itinerary.

3. cancel the itinerary for some or all of its passengers. We model cancellations in the recovery solution by assigning the passengers to a dummy flight g from the origin to the destination of the itinerary.

In the passenger recovery literature, we typically let the airlines determine the replanned passenger schedules [15]. In practice, the new itinerary is presented as an option to the passengers, and there could be a probability of the passenger accepting or declining the proposed change. We recognize such a scenario is not part of our model, and future works could incorporate passengers having a choice in the itinerary change.

2.4.5 Operating costs

The model for mandatory delay and cancellation costs for the passengers Palpant *et al.* [15] is inspired by the European Union regulations. For a proposed new itinerary $k \in K_{recov}$, the operating costs are modeled as a function of the delay k_{delay}^{recov} hours (i.e., the difference between the arrival time of the original itinerary and the new itinerary) and the duration $k_{duration}$ hours of the original itinerary.

- The airline must provide drinks and a meal in case of a delay longer than: two hours on a trip with an initially planned duration strictly less than two hours; three hours on a trip with a duration greater than or equal to two hours and strictly less than four and a half hours; four hours on a trip with a duration greater than or equal to four and a half hours. We denote c^{meal} for the above cost (e.g., 15 euros per passenger),

$$c_k^{meal} = 15\Big[\mathbb{I}\{k_{delay}^{recov} \geq 2 \;\&\; k_{duration} < 2\} + \mathbb{I}\{k_{delay}^{recov} \geq 3 \;\&\; 2 \leq k_{duration} < 4.5\} + \mathbb{I}\{k_{delay}^{recov} \geq 4 \;\&\; k_{duration} \geq 4.5\}\Big]$$

- In addition, the airline must provide lodging (if necessary) in the case of a delay longer than five hours. The cost of a hotel night is assumed to be c^{hotel}, e.g., 60 euros per passenger,

$$c_k^{hotel} = 60\,\mathbb{I}\{k_{delay}^{recov} \geq 5\}$$

- In the case of a cancellation, the airline must reimburse the ticket price regardless of the length of the trip, as well as provide financial compensation. The financial compensation $c^{reimburse}$ per passenger is: 250 euros for a trip with an initially planned duration strictly less than two hours; 400 euros for a trip with a duration greater than or equal to two hours and strictly less than four and a half hours; 600 euros for a trip with a duration greater than or equal to four and a half hours.

$$c_k^{reimburse} = c_k^{price} + 250\mathbb{I}\{k_{duration} < 2\} + 400\mathbb{I}\{2 \leq k_{duration} < 4.5\} + 600\mathbb{I}\{k_{duration} \geq 4.5\}.$$

2.4.6 Passenger disutility costs

Apart from operating costs, disutility or an inconvenience perceived by a passenger when there is a delay, downgrade or cancellation of their flights are also part of the model [87]. We penalize the objective of the recovery schedule if any of the following are not fulfilled as much as possible, as explained in Jozefowiez *et al.* [72]:

- Passengers should not be delayed or their whole itinerary cancelled. Cancellations are only a last resort when the maximum possible delay is already considered.

- Maximum delay in the recovered itinerary should not exceed a threshold (e.g., 18 hours for domestic flights or 36 hours for international flights, depending on the airline's philosophy).

- Passengers should not be downgraded from their reference class (e.g., a business class passenger being assigned an economy seat).

- At the end of the recovery period, the passenger schedules should return to the originally planned schedules.

Given an itinerary k, if an itinerary is composed of several legs, the itinerary's reference cabin class m_k is assumed to be the highest of the booking cabin classes on those legs. For example, a passenger who travels in both economy and business class on different flights is assigned m_k to be business, i.e., they are a "business class" passenger. Costs can be calculated based on the reference cabin class. The itinerary type k_{type} is defined as the type of its longest flight leg (with ordering I = intercontinental > C = continental > D = domestic).

Disutility costs could be of three types- downgrading, delay, and cancellation costs. For an itinerary $k \in K_{recov}$, we detail all the costs below.

- **Downgrading cost:** Costs associated with downgrading are applied only in the case of re-accommodation of a passenger, calculated on an individual leg basis, for all legs of the recovered itinerary. For each leg, these costs depend upon the type of the leg and the level of downgrading (the difference between the itinerary's reference cabin class and the cabin in which the passenger actually travels on that leg). For an itinerary k containing a flight f in cabin class m, the passenger downgrading cost is modeled as,

$$c_{fmk}^{down} = \alpha_{fmk},$$

where α_{fmk} is a constant depending on f, m and k. Typical values are provided in the ROADEF data sets [15] in Table 2.1.

	D	C	I
F → B	150	400	750
F → E	200	500	1500
B → E	150	400	750

Table 2.1: Typical downgrading disutility costs α_{fmk} (in Euros) from the ROADEF challenge [15]. Row headings are cabin class downgrades from the original to the proposed new itinerary, and column headings are reference flight type of the original itinerary.

- **Delay cost:** Delay costs for passenger inconvenience are only applied if there is a net delay at the destination between the original and recovered schedule, irrespective of changes in intermediate airports. We model the total delay costs c^{delay} as the sum of two components: (i) inconvenience delay costs, which are linear in the delay length k_{delay}^{recov}, with the slope $\beta_{fm_k k}$ (typical values in Table 2.2) determined by flight type (I/C/D) and reference cabin class m_k of the passenger; (ii) operating delay costs c_k^{meal} and c_k^{hotel} providing the intercept,

$$c_{fk}^{delay} = \beta_{fm_k k} k_{delay}^{recov} + c_k^{meal} + c_k^{hotel}.$$

	D	C	I
F	1.25	1.25	1.25
B	0.8	0.85	0.9
E	0.05	0.15	0.25

Table 2.2: Typical delay disutility costs β_{fk} per minute of delay (in Euros) from the ROADEF challenge [15]. Row headings are cabin classes, and column headings are reference flight types of the original itinerary.

- **Cancellation cost:** Cancellation costs are intuitively set to be much larger than the maximum delay cost for the trip to disincentivize cancellations. In practice, we assign a dummy flight g from origin to destination in the recovery solution, so that any passenger assigned to this dummy flight is meant to have their trip cancelled and refunded. We model the total cancellation costs c^{can} as the sum of two components: (i) inconvenience cancellation costs, which are much larger in case of a return (inbound) portion of a trip, hence we would prefer to cancel an outbound over an inbound portion of a trip; (ii) operating cancellation costs $c_k^{reimburse}$,

$$c_{gk}^{can} = \gamma_k + c_k^{reimburse},$$

where γ_k a constant that depends on whether the itinerary k is inbound or outbound. Tables 2.4 and 2.3 show typical values from the ROADEF challenge [15].

	D	C	I
F	2500	2750	3000
B	1500	1750	2000
E	250	600	1000

Table 2.3: Typical cancellation disutility costs γ_k for outbound itineraries (in Euros) from the ROADEF challenge [15]. Row headings are cabin classes, and column headings are reference flight types of the original itinerary.

	D	C	I
F	7500	8250	9000
B	4500	5250	6000
E	750	1500	3000

Table 2.4: Typical cancellation disutility costs β_{fk} for inbound itineraries (in Euros) from the ROADEF challenge [15]. Row headings are cabin classes, and column headings are reference flight types of the original itinerary.

Thus, the passenger disutilities in the objective of the recovery are modeled as a linear combination of downgrading, delay, and cancellation costs. Typically, the weights between the three are set equally, as is the case of the ROADEF challenge [15].

2.4.7 Inputs

We summarize the inputs to the passenger recovery problem below.

1. **Configurations:** Start and end of recovery window, cost parameters for the delay, downgrading, and cancellation.

2. **Original and Recovered Flights:** List of original flights and solution to operations recovery F_{recov}, i.e. recovered flights after disruption. We can find mathematical programs (typically mixed-integer programs) in literature [66] to compute F_{recov}.

3. **Aircraft information:** For each aircraft type, e.g. A320, we have their info

   ```
   Aircraft Model Seat Capacities
   ```

4. **Itineraries:** Original planned itineraries K, set of disrupted itineraries K_{dis} given the operations recovery solution.

5. Turnaround time between flights of the same aircraft, e.g., 30 minutes.

2.4.8 Objective

The objective in the passenger recovery problem is to minimize the sum of the three costs, c^{delay}, c^{down} and c^{cancel}, by creating a recovered set of itineraries K_{recov} for the disrupted itineraries K_{dis}. Note that undisrupted itineraries from the original set K are by default part of the recovery solution.

2.4.9 Result

We want the recovered itineraries K_{recov} i.e. a solution to the passenger recovery problem given original itineraries, original flight schedules, recovered flight schedules, and other inputs in Section 2.4.7.

2.5 Algorithms for Passenger Recovery

In this section, we present our main contributions- a preprocessing step that can efficiently compute and store graphs for each itinerary, and two approaches for passenger recovery: (i) an Integer Program (IP) to solve for recovered passenger itineraries from scratch, and (ii) a network-based approach based on solving shortest path problems with multiple labels.

2.5.1 Preprocessing

From the original schedules (flight and passenger) and knowledge of disruptions, a preprocessing step is performed to obtain pruned graphs relevant to each itinerary. In practice, preprocessing can be performed at the beginning of the recovery window using the recovered flight solution, and airlines could solve passenger recovery multiple times as new disruptions emerge for passengers. We later show experimentally that upon preprocessing, the integer program model can be constructed and solved with fewer variables and constraints, and hence the passenger recovery schedules can be computed much faster than without preprocessing. Our network-based approach also takes advantage of the preprocessed graphs to store only a subset of the airports (nodes) and flights (arcs) instead of the entire flight network. We summarize the preprocessing below after introducing the concept of feasible flights and reachable airports for an itinerary.

Given an itinerary k, we say a flight $f \in F_{recov}$ is a *feasible flight* for the itinerary if and only if: (i) the departure time f_{dep} is after the start time of the itinerary; (ii) the arrival time f_{arr} is within the end of the recovery window; (iii) there is a path from the source of the itinerary to the origin airport f_{orig} of the flight; (iv)there is a path from the destination airport f_{dest} of the flight to the sink of the itinerary. Thus, if we construct all feasible paths for the itinerary k, the flight f is a part of at least one of the paths. Similarly, we say an airport a is *reachable* for itinerary k if a is either the origin or destination for at least one feasible flight of the itinerary. Note that the source and sink of the itinerary are by default reachable.

The aim of preprocessing is to construct itinerary-specific graphs of the time-space flight network that can be used by passenger recovery algorithms. Given an itinerary $k \in K_{dis}$ and the entire flight network G_s (defined in Section 2.4.1), we create the graph $G_k = (V_k, A_k)$ with source s_k and sink t_k, that represents all the feasible flights in F_{recov} for itinerary k. Let F_k be the set of all feasible flights in the itinerary including dummy flight g from s_k to t_k, to be used for canceling the itinerary. No flight in F_k starts before the start time of k, and all possible flight legs a passenger can take from s_k to t_k are computed. The set V_k only contains airports reachable feasibly for k, including the source and sink. We *prune* G_k so that there are no flights that reach beyond the end of the recovery window, as well as upper bound the number of legs a passenger can take to reach their destination. Furthermore, in preprocessing, let for each $f \in F_{recov}$ not originating from the source, extract

$$\text{Prev}_f = \{f' \in F_{recov} : f_{orig} = f'_{dest}, \;\; f_{dep} \geq f'_{arr} + TT\}.$$

The set Prev_f represents the list of all flights f' which arrive at the same airport that f departs from, and such that f' arrives at least TT minutes before f departs, where TT is the turnaround time needed between flights for passengers in a layover (usually $30 - 50$ minutes, depending on the airline). We do not define Prev_f for those flights with $f_{orig} = s_k$, the source of the itinerary since passengers start from the source airport, and are available at the departure time of f.

More explicitly, we compute two directed acyclic graphs (DAG) for each itinerary k: (i) the $airportDAG$ stores a list of all *reachable airports* j given a starting airport i (a reachable airport is defined as an airport j such that there is at least one feasible flight in the arc set F_k from i to j); and (ii) the $flightDAG$ stores a list of all feasible flights between the arc $(i \rightarrow j)$ for two airports i and j in V_k. In our implementation, we compute the above DAGs using a depth-first search (DFS) approach.

A summary of the preprocessing approach to obtain the DAGs is presented below. For each itinerary k,

1. Consider the entire flight network graph G_s with node set comprising all the airports, and arc set comprising all the flights in the network in the recovery window.

2. Initialize the set of visited nodes Ψ_k to the empty set.

3. Start a depth-first procedure at the root node n as the source airport s_k.

4. Compute the set of next-reachable airports $NR(n)$ from the node n, as all airports j which have a flight f from n to j that is within the disruption window, and after the start time of the itinerary k.

5. In $airportDAG$, for the key n, append each $j \in NR(n)$ to the list of values, and in $flightDAG$ to the key (n, j), append each feasible flight f to the list of key values.

6. For each of the next reachable airports j from n, add j to the set of visited nodes Ψ_k.

7. If $j \notin \Psi_k$, recursively perform the DFS steps $3 - 6$ by setting $n = j$, the current node of the DFS. Else, set n to another airport j in $NR(n)$.

8. Prune the graph to remove:

   (a) those nodes through which there are no paths from source s_k to sink t_k of length at most the maximum number of legs allowed for itinerary k.

   (b) flights in F_{recov} that reach after a maximum delay allowed for itinerary k.

(c) connecting flights that do not obey a turnaround time between flights.

9. Repeat the procedure until all airports of G_s are visited and Ψ_k can no longer be updated.

10. Prune the $airportDAG$ to remove all airports which are not in a path from the source s_k to sink t_k of the itinerary. Remove corresponding flights from $flightDAG$.

Preprocessing is an important step we develop that helps the running time of our algorithms. The overall airline graph G_s consists of all the airports and flights between them in the disruption period. The overall graph is huge and not relevant. Our contribution here is preprocessing the overall graph to construct the graphs G_k for each itinerary k, which reduces the complexity of the problem by only considering smaller graphs in our IP and network-based algorithms.

A preprocessed graph example of an itinerary starting at $8AM$ on a given day, and originally ending at $2PM$ is shown in Figure 2.5. In this itinerary k moving n_k passengers, the original schedule consists of Flight 1 from JFK to BOS and Flight 10 from BOS to LAX. In this graph G_k, $airportDAG = \{$ JFK : [BOS, ORD, LAX], BOS:[ORD, LAX], ORD:[BOS, LAX] $\}$ and $flightDAG = \{(\text{JFK}, \text{BOS}) : [1, 2], (\text{JFK}, \text{ORD}) : [3, 4], (\text{BOS}, \text{ORD}) : [6], (\text{ORD}, \text{BOS}) : [5], (\text{JFK}, \text{LAX}) : [11], (\text{BOS}, \text{LAX}) : [9, 10], (\text{ORD}, \text{LAX}) : [7, 8]\}$.

For the integer program, we show in the experiments that preprocessing reduces the size of the problem- Building the IP using preprocessing has $< 1\%$ of the variables of the IP built without preprocessing. Moreover, the overall run times of the IP with preprocessing are less than 1 second, compared to 90 seconds and even up to 300 seconds when the IP without preprocessing is constructed and solved.

For the network-based multiple labels shortest path approach, the number of labels stored, as well as the time taken to compute labels is dependent on the size of the graph. Preprocessing helps in considering for each itinerary, the graph containing only the airports and flights that can be reached for the parameters of that itinerary. We illustrate the impact with a toy example below, simplified from our experimental data.

Consider an originally planned itinerary k in Figure 2.5 where $n_k = 10$ passengers depart from

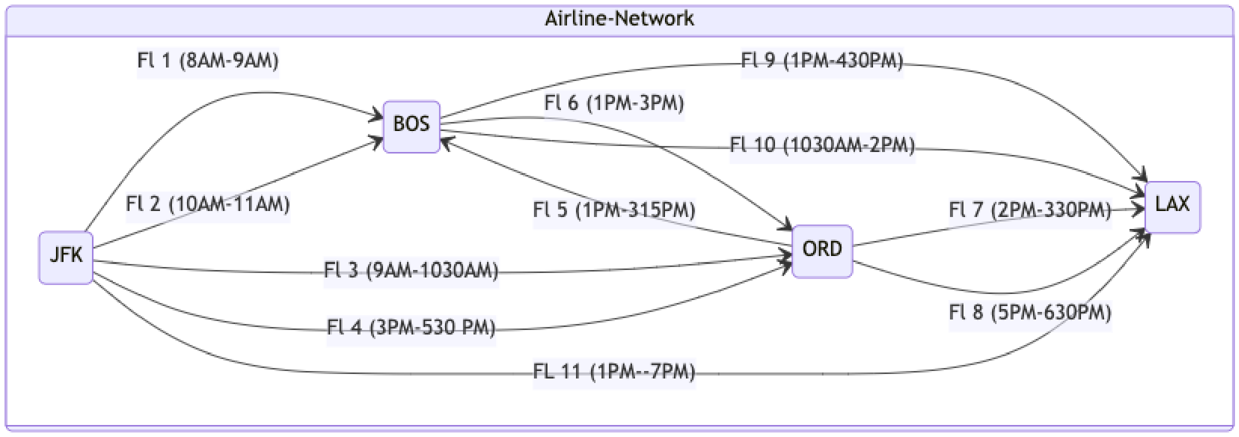


Figure 2.5: An example of a preprocessed graph G_k for an itinerary from origin JFK to destination LAX. The original schedule consists of Flight 1 from JFK to BOS and Flight 10 from BOS to LAX. Thus the original start time of the itinerary was 8AM and the original end time was 2PM.

an airport JFK at 8 AM on $01/01/2022$, take two flights 1 and 10 with a layover at an intermediate airport BOS, and arrive at the airport LAX on the same day at 2 PM. Thus, the itinerary contains 2 legs and is of duration $k_{dur} = 6$ hours. If at least one of the flights 1 or 10 was affected by the disruption (i.e., in the recovered flight solution F_{recov}, they are delayed or cancelled), we consider the itinerary k to be disrupted and wish to replan from scratch. To run a multiple-label shortest path approach, we need the graph of all flights from JFK (source) to LAX (sink) and the airports which could function as layovers. Without preprocessing, we must consider all airports and flights in the network that start after 8 AM and lie within the recovery window. With preprocessing, the directed acyclic graphs $airportDAG$ and $flightDAG$ specific to itinerary k are already pruned- for example, they do not contain (i) any intermediate airports which are not reachable from the source, and from which there are no paths to the sink using flights in F_{recov}, (ii) paths from source to sink that has too many (e.g., if itinerary k had $q = 2$ legs, no path considered in the preprocessed graph has more than $q + 2 = 4$ legs), (iii) flights in F_{recov} that reach after a maximum delay allowed for itinerary k (e.g, flights that reach 12 hours after the original itinerary end time 2 PM), (iv) connecting flights that do not obey a turnaround time of 30 minutes between flights, etc.

Even in one of the smallest data sets from the ROADEF challenge used for our experiments, the graph G_k for an itinerary that could potentially have 35 nodes and ~ 600 arcs is reduced to a graph with 4 nodes and 11 arcs specific to one itinerary. As explained in detail, the graph-based algorithm needs many labels to be placed on each node in a BFS manner, scaling with the number of flights, and hence, the preprocessing is a crucial step to help the run time of the multiple-label shortest path approach.

2.5.2 Integer Program formulation

We propose the following IP formulation to solve passenger recovery using the integer variables y_{fmk}, the number of passengers in the recovery solution who are assigned flight $f \in F_{recov}$ in itinerary $k \in K_{dis}$ and cabin class $m \in \mathcal{M}$. For each itinerary $k \in K_{dis}$, we also add a variable y_{gk} that denotes the number of passengers taking a dummy flight g from source to sink, which represents cancellations for those passengers. The flight g is also added to the arc set A_k of the itinerary and the set of recovered flights F_{recov}.

Furthermore, let y_{ijmk} be the number of passengers moved between airports $i \rightarrow j$ in itinerary k and class m, and $cap(f, m)$ be the remaining capacity (in the flight recovery solution F_{recov}) of flight $f \in F_{recov}$ in cabin class m.

$$\min_{y} \sum_{k \in K_{\text{dis}}} \left[c^{\text{can}}_{gk}\, y_{gk} + \sum_{m \in \mathcal{M}} \sum_{(i,j) \in A_k} \sum_{f \in F_k(i,j)} c^{\text{down}}_{fmk}\, y_{fmk} + \sum_{(i,t_k) \in A_k} \sum_{f \in F_k(i,t_k)} c^{\text{delay}}_{fk} \sum_{m \in \mathcal{M}} y_{fmk} \right] \tag{2.1}$$

$$\text{s.t.} \quad \sum_{i:(i,j) \in A_k} \sum_{m \in \mathcal{M}} y_{ijmk} - \sum_{i:(j,i) \in A_k} \sum_{m \in \mathcal{M}} y_{jimk} = 0, [\forall j \in V_k \setminus \{s_k, t_k\},\ \ \forall\, k \in K_{\text{dis}}] \tag{2.2}$$

$$\sum_{j:(s_k,j) \in A_k} \sum_{m \in \mathcal{M}} y_{s_k jmk} = n_k, [\forall\, k \in K_{\text{dis}}] \tag{2.3}$$

$$y_{ijmk} = \sum_{f \in F_k(i,j)} y_{fmk}, [\forall\, k \in K_{\text{dis}}, (i,j) \in A_k] \tag{2.4}$$

$$\sum_{k \in K_{\text{dis}}} \mathbb{I}\{(i,j) \in A_k, f \in F_{ij}\} y_{fmk} \le \text{cap}(f,m), [\forall\, f : (i \to j) \in F_{recov}\ \ \forall m \in \mathcal{M}] \tag{2.5}$$

$$y_{fmk} \le \text{cap}(f,m), [\forall\, f \in F_{recov}, m \in \mathcal{M}, k \in K_{\text{dis}}] \tag{2.6}$$

$$\sum_{m \in \mathcal{M}} y_{fmk} \le \sum_{f' \in \text{Prev}(f)} \sum_{m \in \mathcal{M}} y_{f'mk}, [\forall\, f : (i \to j) \in F_{recov}, i \ne s_k, k \in K_{\text{dis}}] \tag{2.7}$$

$$y_{fmk} \in \mathbb{Z}_{\ge 0},\ \ [\forall\, f \in F_{recov}, m \in \mathcal{M}, k \in K_{\text{dis}}] \tag{2.8}$$

In the objective function (2.1), $\sum_{(i,j) \in A_k} \sum_{f \in F_k(i,j)} c^{\text{down}}_{fmk} y_{fmk}$ is the downgrading cost for all passengers assigned to flight f in cabin class m in the itinerary k. Similarly $\sum_{(i,t_k) \in A_k} \sum_{f \in F_k(i,t_k)} c^{\text{delay}}_{fk} \sum_{m \in \mathcal{M}}\ y_{fmk}$ are the delay costs, and the cancellation cost $c^{\text{can}}_{gk}\ y_{gk}$ is for those passengers assigned on a dummy flight g which represents a cancelled itinerary k.

The first constraint (2.2) is a flow constraint- for every itinerary and every intermediate airport j which is not the origin s_k or destination t_k of the itinerary k, i.e., the number of passengers of k coming into the airport i is the same as the number of passengers of k going out of i over all cabin classes.

The second constraint (2.3) ensures that for a given itinerary k, the number of passengers leaving the source airport s_k is n_k, the passenger count of the itinerary. Note that the flights considered also include the dummy flight g, representing cancellations.

The third constraint (2.4) splits the decision variables y_{fmk} into new variables y_{ijmk} representing the number of passengers moved between airports $i \rightarrow j$ in cabin class m and itinerary k. For the source airport $i = s_k$ of the itinerary k, the RHS includes the variable y_{gk} representing cancellations.

The fourth constraint (2.5) is about remaining flight capacities. Across all itineraries k, the number of passengers utilizing a flight f in cabin class m should be at most the remaining capacity $cap(f, m)$ available in $f \in F_{recov}$ in cabin class m. This constraint is tricky since there is a sum over all itineraries k in the LHS and thus, hinders parallelization of the IP among itineraries. If not, we could have split the IP (objective and all the constraints) over each itinerary separately.

The fifth constraint (2.6) upper bounds the decision variables with the remaining capacity of the flight and cabin class. The capacity of the dummy flight g is implicitly set to n_k, the passenger count of the itinerary from (2.2), and hence is not included in both (2.5) and (2.6).

The sixth constraint (2.7) ensures for each flight f with origin Orig_f, the number of passengers who can board f in all cabin classes does not exceed the number of passengers who arrive at Orig_f before the departure time of f plus a turnaround time. The set Prev_f stores the list of feasible previous flights for any flight f. Therefore, we can only send into a flight f, at most the number of people who are available at the airport Orig_f having come from a previous flight. We do not consider flights from the itinerary source for this constraint.

The seventh constraint (2.8) ensures the variables of the mathematical program are nonnegative integers.

Problem Size: The IP (2.1) has $|F_{recov}| \times |\mathcal{M}| \times |K_{dis}| + |K_{dis}| = O(|F_{recov}| \times |\mathcal{M}| \times |K_{dis}|)$ integer variables (the second term $|K_{dis}|$ counts the dummy flights y_{gk}). The number of constraints depends on the exact structure of the graphs G_k for each itinerary. A weak upper bound on the size of the constraints can be written by summing up the number of constraints in (2.2) to (2.6) as

$$\begin{aligned}
&O(N|K_{dis}| + |K_{dis}| + |K_{dis}|.|F_{recov}| + |F_{recov}|.|\mathcal{M}| + |F_{recov}| \times |\mathcal{M}| \times |K_{dis}| + |F_{recov}|.|K_{dis}|)\\
=&O(|F_{recov}| \times |\mathcal{M}| \times |K_{dis}| + N|K_{dis}|)
\end{aligned}$$

constraints, where N is the total number of airports in the network.

We summarize Algorithm 4 below to solve for K_{recov}, the set of recovered itineraries. For each $k \in K_{dis}$, we cancel the whole itinerary and replan. For example, even if only a passenger's second leg is disrupted, we cancel the whole itinerary and re-plan for a better solution. The recovery solution K_{recov} starts with all itineraries in $K \setminus K_{dis}$. For each $k \in K_{dis}$, we solve the IP (2.1) to get the recovered passenger itinerary.

A post-processing step constructs the recovered passenger itineraries from the output variables y^*_{fmk}. In literature, finding passenger itineraries is similar to decomposing a flow into a number of paths [88] which does not always have a unique solution. We only need paths not to exceed the maximum number of legs of an itinerary and do not worry about the typical objectives of the flow decomposition problem [88]. We construct recovered itineraries as follows- For an itinerary k where n_k passengers need to be re-accommodated, first check if $y^*_{gk} > 0$, and cancel the itinerary for y_{gk} passengers. So we must replan only for $n_k - y^*_{gk}$ passengers. Then obtain all the non-zero y^*_{fmk} variables in the solution to make the solution graph G^*_k whose arc set contains only the nonzero flight arcs with flow y^*_{fmk}. Find a path $p = \{(f_1, m_1) \to (f_2, m_2) \ldots\}$ from source s_k to sink t_k of the itinerary in G^*_k, and make a new recovered itinerary following the path, with the number of passengers $\tau_p = \min\{y^*_{fmk} : (f, m) \in p\}$ among the arcs in that path. Update G^*_k by reducing the flows on the arcs in the path p by τ_p, and deleting the updated arcs if the new flows on that arc are zero. We now have one new recovered itinerary for τ_p passengers. Repeat the procedure of finding paths, and updating G^*_k until all arcs of G^*_k have zero flow. Now all the $n_k - y^*_{gk}$ passengers would have been re-accommodated into some number of recovered itineraries.

We place the relevance of Algorithm 4 in the passenger recovery literature. There are existing approaches to solving passenger recovery utilize mathematical programs, especially mixed integer programs where the integer variables typically denote the number of passengers (see details and discussions in Bisaillon *et al.* [11], Jozefowiez *et al.* [72], Acuna-Agost *et al.* [82], and McCarty and Cohn [84]). While our mathematical program (2.1) contains $O(|F_{recov}| \times |\mathcal{M}| \times |K_{dis}|)$ integer variables, the structure of the problem in practice as well as using preprocessed graphs to construct

Algorithm 4: Compute recovered passenger itineraries using IP (2.1)

```
1  Input: Configurations, original and recovered flights, airline information, and itineraries. // as given in Section 2.4.7
2  Output: $K_{recov}$ // set of recovered itineraries
3
4  Initialize $K_{recov} = K \setminus K_{dis}$.
5  Solve the IP (2.1) and obtain the outputs $y^*_{fmk}$.
   // for each recovered itinerary, post process.
6  for $k \in K_{dis}$ do
7      If $y^*_{gk} > 0$, cancel the trip for $y_{gk}$ passengers of itinerary $k$, add this as a new itinerary to $K_{recov}$.
8      Construct the graph $G^*_k$ using only the arcs $\{(f,m) : y^*_{fmk} > 0\}$ with flows on the arcs $y^*_{fmk}$.
9      Initialize remaining passengers to be handled, $remPax = n_k - y^*_{gk}$.
10     while $remPax > 0$ do
           // alternatively, until $G^*_k$ is not an empty graph
11         Find a path $p$ from source $s_k$ to sink $t_k$ for the itinerary $k$, where each arc of the path contains a nonzero flow, and whose length is at most maximum number of legs for itinerary $k$.
12         Obtain $\tau_p = \min\{y^*_{fmk} : (f,m) \in p\}$, number of passengers that can be served by the path.
13         Create a new itinerary in $K_{recov}$, where $\tau_p$ passengers take the path.
           // contains both flights and cabin classes to be taken by the passengers.
14         Update $G^*_k$ by reducing the flow on each arc of $p$ by $\tau_p$, deleting arcs which now have zero flow.
15         Update $remPax \leftarrow remPax - \tau_p$.
16     end
17 end
18 Output $K_{recov}$, the set of all recovered itineraries.
```

the IP variables and constraints leads to a reasonably fast solution, with an attractively low number of non-zero variables in the optimizer, as we show in the experiments. Nevertheless, the number of disrupted itineraries $|K_{dis}|$ itself depends on the original flight schedule, the disruption, and the recovered solution F_{recov}, and thus in theory could be as large as the number of original itineraries $|K|$. Similarly, the recovered flights could be as large as the set of original flights when there are no flight cancellations or aircraft unavailability, and hence there could be as much as $|F| \times |K| \times |\mathcal{M}|$ integer variables in the formulation. A similar argument applies to the size of the constraints as well. For example, in one of the smallest data sets in the ROADEF challenge [15], there are roughly 600 flights and 1900 itineraries, with 3 cabin classes, and the IP could have almost 3.4

million variables. Using our preprocessed graphs, one example IP we constructed has $\sim 1,65,000$ variables.

Exact formulations to solve passenger recovery could have tractability issues in commercial solvers when scaling to real-world airline network sizes, and many formulations make simpler assumptions than our problem setup (Hu *et al.* [73], Cook *et al.* [77], Santos *et al.* [76], and Palpant *et al.* [15]). Publicly available data for passenger itineraries are limited [9] and hence it is a challenge to present a full analysis for different sizes of airline networks. In our experiments, we show the performance of our IP approach Algorithm 4 on the ROADEF data sets.

Furthermore, the IP (2.1) cannot handle additional constraints that arise in practice- for example, in the ROADEF scenario [15], for any itinerary k, the maximum number of legs of a recovered itinerary cannot be two more than the original number of legs in k. One way to model such a constraint is,

$$\sum_{f \in F_{recov}, m \in \mathcal{M}} \mathbb{I}\{y_{fmk}\} \leq \text{number of legs}(k) + 2, \forall k \in K_{dis},$$

which is not linear. We would like an approach that can handle itinerary-specific constraints that can be present in real-world data as well.

2.5.3 Multiple Labels Shortest Path formulation

Since airlines expect recovery solutions to be fast, consistent and near-optimal rather than exact [83, 11], we develop an (iterative) heuristic approach that shows promising tractability on larger problem sizes. In this section, we present a graph-based algorithm for passenger recovery, using a multiple-label shortest paths (MLSP) approach. We take advantage of the graphs G_k constructed in our preprocessing steps in Section 2.5.1. In the following subsection, we summarize prior work on graph-based approaches to motivate our formulation.

2.5.4 Motivation

In the literature, network-based approaches have been studied that take advantage of the graph structures, and rely less on mathematical programs. In Righini and Salani [17], the authors solve a resource-constrained elementary shortest path problem (RCESPP), which arises in branch-and-price algorithms for vehicle routing problems with additional constraints. The Decremental State Space Relaxation (DSSR) algorithm proposed by Righini and Salani [17] creates labels (with multiple components) associated with nodes. The labels hold information on feasible partial paths to reach the node. If several labels are active at the same node, one can remove labels that are *dominated*, since we know that they cannot lead to the optimal path. A label l at node i is dominated by another at the same node l', if each component of l is no better than the corresponding component of l', and if at least one of the components of l' is strictly better. For labels at node i not eliminated by domination rules, we can create new labels at node j for every feasible arc from i to j. The lowest cost label(s) at the sink provides the optimal solution. In Bierlaire *et al.* [8], the authors combine the above algorithm from Righini and Salani [17] in each recovery network with a column generation approach to solve the flight recovery problem. In Bisaillon *et al.* [11], the authors use a step where they solve a series of shortest path problems with arc labels. On flight networks, each arc corresponds to a flight and is orders of magnitude higher than the number of nodes (airports).

We present an algorithm based on computing multiple labels for each node, and solving a shortest path problem based on these labels that take into account, (a) pruning labels by domination at each node dynamically similar to what happens in DSSR in Righini and Salani [17], and (b) efficiently storing just enough labels at each node that can handle the total number of passengers in the itinerary required to be moved.

The key constraint in the passenger recovery problem is the remaining capacities of the flights (recall we are only solving for disrupted itineraries, and leave the undisrupted itineraries as is). Among the three major costs in the passenger recovery problem, cancellation costs are the highest in practice (an example is provided in Section 2.4.5). When solving in a mathematical program-based approach as in Algorithm 4, we solve for all the disrupted itineraries together, whereas an iterative graph-based approach requires prioritizing the commodity in low supply, i.e., remaining flight capacities for high-value flights. We, therefore, sort all disrupted itineraries in decreasing order of cancellation costs and aim to compute recovered flight schedules for each itinerary in order. During a real-world disruption, there is typically an end of recovery window shorter than what is needed to replan all itineraries, and a certain fraction of itineraries end up getting cancelled. Hence solving first for highest cancellation cost itineraries is justified.

2.5.5 MLSP Algorithm

The key idea of the multiple-label shortest path algorithm is computing and storing multiple labels for each node. Each label $L_v = (L_v^1, L_v^2, L_v^3, L_v^4)$ for any node v in any V_k for a given itinerary k can be described as,

1. first component L_v^1 that stores the length of the path traversed so far, i.e., the number of nodes visited not including the source. In our problem setup, L_v^1 is upper bounded by the length of the original itinerary k plus a parameter that sets a limit on the number of legs a new itinerary can increase by (e.g., if the original itinerary had p legs, any recovered itinerary could be limited to no more than $p + 2$ legs). Labels exceeding the upper bound in the first component are immediately pruned (removed) at the node.

2. second component L_v^2 stores information about the flights and cabin classes used so far to get to the node v. We use the second component of the labels in the sink node to construct the full recovered itinerary for each passenger.

3. third component L_v^3 stores the total downgrading cost c^{down} for every pair (f, m) of flight and cabin class traversed (recall each arc is a feasible flight f and traversing f on cabin class m yields a potential downgrading cost). Thus,

$$L_v^3 = \sum_{(f,m)\in L_v^2} c_{fmk}^{down}.$$

   If v is the sink node t_k of the itinerary, the third component is also appended with the delay costs c_{fk}^{delay} of the itinerary k, based on the last flight f taken to enter t_k in the second component of the label.

4. fourth component L_v^4 stores the maximum number of passengers that can be moved to node v by the path denoted by L_v^2. Formally,

$$L_v^4 = \min_{(f,m)\in L_v^2}\{\text{remcap}(f,m)\},$$

where remcap(f,m) is the remaining capacity available in flight f and cabin class m on the graph G_k for itinerary k.

We define *domination* of labels similar to Righini and Salani [17]. A label L_v at node v is dominated by $\hat{L}_v$ at the same node, only if:

- $L_v^1 \geq \hat{L}_v^1$
- $L_v^3 \geq \hat{L}_v^3$
- $L_v^4 \geq \hat{L}_v^4$

and at least one of the equalities is strict. In other words, $\hat{L}_v$ has no more legs, no higher downgrading (plus potentially delay) costs, and can carry no fewer passengers in the path describing it than L_v. Moreover, $\hat{L}_v$ is strictly better in at least one of the three aspects.

Using the labels defined above, we now summarize the multiple labels shortest path (MLSP) approach to compute recovered passenger itineraries. A pseudocode is provided in Algorithm 5.

The set of disrupted itineraries are sorted according to decreasing order of their cancellation costs c_{gk}^{can}, reflecting the value of the itineraries according to the rationale explained as follows. The remaining flight capacities $cap(f,m)$ are a precious resource since we would expect to cancel some itineraries when recovering from a disruption. The cancellation costs are typically the largest in order to disincentivize cancelling itineraries (see Tables 2.1-2.4), and therefore we prioritize using remaining flight capacities first for passengers who have the highest cancellation costs. For the remainder of the section, we assume K_{dis} is sorted from highest to lowest cancellation costs.

For each itinerary $k \in K_{dis}$, we parse the graph G_k in a BFS order starting from the source s_k. Initialize all nodes of V_k to the labels $(0, \emptyset, 0, 0)$, and mark them unvisited. We traverse the graph recursively in a BFS manner to compute a set of labels L_k for each node while pruning labels that are dominated. Assume we are currently at a node $u \in V_k$ which has a set of labels $\mathcal{L}^u$. We consider each reachable airport v from u and each feasible flight f between the arc $u \to v$. For every label $L_u \in \mathcal{L}^u$, we create new labels at node v using f as follows. First, duplicate L_u at node v as L_v. For each $m \in \mathcal{M}$,

1. If the flight f was in the original itinerary k, update $L_v^3 \leftarrow L_u^3 + c_{fmk}^{down}$.
2. If the flight f was not in the original itinerary k, use the reference cabin class m_k and update $L_v^3 \leftarrow L_u^3 + c_{fm_k k}^{down}$.
3. Update $L_v^1 \leftarrow L_u^1 + 1$ to capture the length of the path traversed so far by the passenger (or, the number of legs so far for the passenger in the recovered itinerary).
4. Update L_v^2 from L_u^2 by adding the tuple (f, m), thus storing the flight and cabin classes used by the path so far.
5. Update $L_v^4 \leftarrow \min(L_u^4, remcap(f, m))$, the number of passengers that can be carried by the path until v is the smaller among the number of passengers that can be carried by the path until u, and the remaining flight capacity of $f \in F_{recov}$ at cabin class m.

Since we are trying to move a maximum of n_k passengers in the itinerary k, we next prune the set of labels at each node to remove dominated labels and also prune so that the fourth components sum up to no more than $\lambda \times n_k$, where λ is a parameter we choose for pruning. That is, we only store sufficient labels to carry at most a multiple of n_k passengers through the graph. In the BFS order, labels for the node v are computed only from the labels from predecessor nodes (i.e., nodes u from which v is reachable in the graph G_k).

Finally, we consider $\mathcal{L}_{t_k}$, the set of labels at the sink. For each $L \in \mathcal{L}_{t_k}$, we also add the delay costs c_{fk}^{delay} to the components L^2. Apply the pruning step and obtain the top $\lambda \times n_k$ labels sorted by increasing order of the third components L^3, i.e., the sum of delay and downgrading costs. The challenge in the shortest path graph approach is that delay costs cannot be computed until we reach the sink labels.

Once the sorted labels are computed, we allocate the passengers going label by label as follows. We want to replan for n_k passengers in total, and let the variable $remPax$ store the remaining passengers among the n_k passengers who have not been replanned yet for a recovered schedule.

1. For each L, we use the second components L^2 that contain the paths denoted by the label. Say $L^2 = \{(f_1, m_1) \to (f_2, m_2) \ldots \to (f_z, m_z)\}$.

2. Extract L^4 the maximum number of passengers the label can serve. Let $\delta = \min(L^4, remPax)$.

3. Create the new recovered itinerary for δ passengers in the original itinerary k as: they take flight f_1 in cabin class m_1, flight f_2 in cabin class m_2 and so on, till flight f_z in cabin class m_z. Update $remPax \leftarrow remPax - \delta$ for the itinerary.

4. Decrease all the remaining flight capacities $remcap(f, m)$ for each tuple in L_2 by δ, since these seats have been used.

5. The cost of the new recovered itinerary is δL^3.

Perform the allocation procedure until $remPax$ becomes zero or the labels are exhausted. If there are still any remaining passengers, they cannot be rescheduled in the recovery solution, and therefore cancel the itinerary for $remPax > 0$ passengers incurring the cancellation cost $c_{gk}^{cancel} \times remPax$.

Thus we can create a recovered passenger itinerary for itinerary k. Before the next iteration for computing the recovered itinerary for a new k', we update the remaining capacities of the flights and prune $G_{k'}$ to remove infeasible flights and airports. Algorithm 5 shows the pseudocode for our multiple labels shortest path approach.

Algorithm 5: Multiple Label Shortest Paths (MLSP) for solving passenger recovery

```
1  Input: Configurations, original and recovered flights, airline information and itineraries // as
      given in Section 2.4.7
2  Result: $K_{recov}$ // set of recovered itineraries
3  Initialize $K_{recov} = K \setminus K_{dis}$
4  Initialize the total cost of the solution $cost_{MLSP} = 0$.
5  for $k \in K_{dis}$ do
6      Initialize source $s_k$ and all nodes of $G_k$ with the label $(0, \emptyset, 0, 0)$
7      Let $labels(v)$ denote the set of labels for each node $v$.
8      Initialize the set of visited nodes $\Omega$ to $False$ for all nodes in $V_k$.
9      Update labels in a BFS manner as follows,
10     Start with $u \leftarrow s_k$
       // bfs starts at source
11     while $\Omega$ contains $False$ do
           // There are nodes not visited in $G_k$
12         Set node $u$ as $True$ in $\Omega$ // $u$ is visited
13         for each airport $v$ reachable from $u$, and each feasible flight $f$ and cabin class $m$, between
             $u$ and $v$, do
14             Add all labels $labels(u)$ to $labels(v)$
15             for each label $L_v$ in the newly added labels, do
16                 Update $L_v^3 \leftarrow L_u^3 + c_{fmk}^{down}$, using $m = m_k$ if $f$ was not in $k$ originally.
17                 Update $L_v^1 \leftarrow L_u^1 + 1$, the number of flight legs used so far.
18                 Add the tuple $(f, m)$ to $L_v^2$.
19                 Update $L_v^4 \leftarrow \min(L_u^4, remcap(f, m))$
20                 Prune all dominated labels at $v$, maintaining only enough labels to carry a total of
                     $\lambda n_k$ passengers.
21             end
22         end
23         For each label of the sink $t_k$, add delay costs $c_{fk}^{delay}$ to the second component $L^2$.
24         Prune dominated labels at the sink, and sort by increasing order of costs (second
             components) to make the set $\mathcal{L}_{t_k}$
25         Allocate passengers according to the subroutine Algorithm 6.
26         Thus we can create a recovered passenger itinerary for itinerary $k$.
27     end
28 end
```

Algorithm 6: Subroutine that creates recovered itinerary using sink labels of the given itinerary.

```
1  Input: Sink labels $\mathcal{L}_{t_k}$ for one itinerary $k$.
2  Result: Recovered itinerary for the $n_k$ passengers in $k$ (including cancellations) along with costs.
3  Initialize $remPax \leftarrow n_k$. // remaining passengers to be allocated.
4  while $remPax > 0$ do
5      for each $L \in \mathcal{L}_{t_k}$ do
6          Set $\delta = 0$. // number of passengers this label will serve
7          . Let $L^2 = \{(f_1, m_1) \rightarrow (f_2, m_2) \ldots \rightarrow (f_z, m_z)\}$.
8          Update $\delta = \min(L^4, remPax)$.
9          Create a new itinerary in $K_{recov}$ for $\delta$ passengers- they take the flights in $L^2$.
10         Increase $cost_{MLSP}$ by $\delta L^3$.
11         Update remaining flight capacities $remcap(f, m) \leftarrow remcap(f, m) - \delta$
12         Update $remPax \leftarrow remPax - \delta$.
13     end
14     If all labels at the sink are exhausted, cancel the itinerary for the remaining $remPax$
         passengers.
15     Increase $cost_{MLSP}$ by $remPax \times c_{gk}^{cancel}$.
16 end
17 Obtain $K_{recov}$ and the cost $cost_{MLSP}$.
```

2.5.6 MLSP with Batching

Because of the iterative solving for itineraries based on cancellation costs in Algorithm 5, the solution is possibly suboptimal due to exhausting flight capacities in earlier iterations. Unlike the IP approach, we do not consider all the itineraries together.

In the literature of shortest path algorithms, batching has been studied as a way to speed up the iterative process, or to improve the solution (see Hotz *et al.* [85] and Zhang *et al.* [86]). The main idea is to partition the sorted K_{dis} into batches of size γ. For each batch B of itineraries, instead of creating new itineraries according to Algorithm 5, we combine all the top sink labels to create a bigger set of labels,

$$\mathcal{L}_B = \{\cup_{k \in B} \mathcal{L}_{t_k}\}.$$

We sort $\mathcal{L}_B$ in increasing order of third components (costs), and allocate passengers similar to Algorithm 6. In the experiments, we also discuss the results of batched MSLP for various batch sizes.

2.6 Experiments

In this section, we give a comprehensive experimental treatment to show the performance of our proposed algorithms. Results are presented for (i) The IP based Algorithm 4, (ii) network based MLSP Algorithm 5, (iii) batched MLSP of various batch sizes.

The data used for our experiments is calibrated from the publicly available data set of the ROADEF challenge [15]. There are three classes of data from the challenge, labeled “A"", “B" and “X" instances, in order of complexity. All the papers emanating from the ROADEF challenge, e.g., Bisaillon *et al.* [11], Sinclair *et al.* [89, 78], Jozefowiez *et al.* [72], Jafari and Zegordi [71], and Jafari and Hessameddin Zegordi [81], solve for integrated flight and passenger recovery whereas our focus is on developing new algorithms for passenger recovery, of which there has only been one work McCarty and Cohn [84] since 2009. In McCarty and Cohn [84], the data presented considers a disruption that only delays one flight in the flight recovery solution.

We would like to test our algorithms for a wide variety of disruptions, ranging from *light* disruptions where a certain subset of flights are delayed and nothing else to *heavy* disruptions where many flights are outright cancelled, and many are delayed. We are not able to get the variety from ROADEF data, and thus we perform our experiments with modified data i.e., calibrating our data set from Palpant *et al.* [15], and impose our own disruptions. Such an approach to experiments is also common in recovery literature, e.g., McCarty and Cohn [84] manually delaying one flight, and in papers like Kohl *et al.* [61] on airline recovery, the tests were run on data where small irregularities in a database of 4000 events were generated randomly, at most 10% of the flights were delayed from 15 minutes to 2 hours.

2.6.1 Computational Goal

Most airline operations controllers demand operational disruption models to provide good solutions at the fleet level in order of minutes Vink *et al.* [68] and Hassan *et al.* [9]. THE ROADEF challenge had a strict time limit of ten minutes Palpant *et al.* [15] on an AMD Turion64x2 processor with 2GB RAM.

In our work, the recovery algorithms were implemented in Python using Gurobi 9 to solve Integer Programs and Mixed Integer Programs (default internal parameters), on a Macintosh with 2.4 GHz 64 bit Quad-Core intel Core $i5 - 3470$ processor with 8GB RAM. Run times are reported in seconds. We aim to use our preprocessing approach and solve for passenger recovery in the order of minutes. Note that for the IP-based Algorithm 4, the runtimes include the time to build the variables and constraints in the IP model, as well as the runtime of the IP solving itself.

2.6.2 Data set characteristics

We use the calibrated data from the "B" and "X" instances from the ROADEF challenge [15]. We create disruptions of two types by cancelling and delaying (and leaving unchanged) different percentages of the original flights F. let R be the length of the recovery window in hours.

1. **Heavy disruption:** Cancel 50% of all flights and delay 25% of all flights by a number of

	A1	A2	A3	A4	A5
No. of airports	35	35	35	35	35
No. of flights	608	608	608	608	608
Size of K	1659	1659	1659	1659	3395
Size of K_{dis} (heavy)	1382	1350	1416	1323	2832

Table 2.5: Characteristics of A instances used in our experiments under heavy disruptions. Disruptions started on January 7th 2006 at 12 PM, and the recovery window was till January 8th 2006 4 AM, with length $R = 16$ hours.

	B1	B2	B3	B4	B5
No. of airports	45	45	45	45	45
No. of flights	1422	1422	1422	1422	1422
Size of K	11214	11214	11214	11214	11214
Size of K_{dis} (heavy)	9072	9391	9418	9321	9336
Size of K_{dis} (light)	2981	2760	2878	2931	3154

Table 2.6: Characteristics of B instances used in our experiments under heavy and light disruptions. Disruptions started on March 1st 2008 at 4 PM, and the recovery window was till March 3rd 2008 04 AM, with length $R = 36$ hours.

hours uniformly random between 0 and $R/2$ hours, and the number of minutes uniformly random between 0 and 60 minutes.

2. **Light disruption:** Delay 20% of all flights by number of hours uniformly random between 0 and $R/2$ hours, and number of minutes uniformly random between 0 and 60 minutes.

A description of the data sets used, (A1-A5), (B1-B5), (XA1-XA4) and (XB1-XB4) are presented in Tables 2.5, 2.6, 2.7 and 2.8 respectively. The XA and XB were the larger data sets and were the main data used in testing the algorithms of the ROADEF challenge. In both heavy and light disruptions, we first modify the flights by cancelling or delaying some percentage of the flights, and the resulting schedules form the flight recovery F_{recov}. Next, we check each itinerary to see if any of its flights are disrupted in F_{recov} to form the disrupted itineraries K_{dis}.

2.6.3 Impact of Preprocessing

In Section 2.5.1, we discussed how to obtain graphs specific to each itinerary by preprocessing. The IP (2.1) can be implemented without using the preprocessed graphs, in which case the variable set is $\{y_{fmk} : k \in K_{dis}, m \in \mathcal{M}, f \in F_{recov}\}$. If we use preprocessing, the variable set is much

	XA1	XA2	XA3	XA4
No. of airports	35	35	35	35
No. of flights	608	608	608	608
Size of K	1659	3395	1608	3288
Size of K_{dis} (heavy)	1355	2856	1332	2732
Size of K_{dis} (light)	424	921	428	858

Table 2.7: Characteristics of XA instances used in our experiments under heavy and light disruptions. Disruptions started on March 1st 2008 at midnight, and the recovery window was till March 4th 2008 6 AM, with length $R = 77$ hours.

	XB1	XB2	XB3	XB4
No. of airports	45	44	45	44
No. of flights	1423	1423	1423	1423
Size of K	11214	11214	11214	11214
Size of K_{dis} (heavy)	9226	9282	9499	9488
Size of K_{dis} (light)	1421	2951	3233	3280

Table 2.8: Characteristics of XB instances used in our experiments under heavy and light disruptions. Disruptions started on March 1st 2008 at midnight, and the recovery window was till March 4th 2008 6 AM, with length $R = 77$ hours.

smaller, since we build the IP to only those feasible flights and cabin classes with non-empty remaining capacity for each itinerary. In Table 2.9, we present a comparison of implementing Algorithm 4 without and with preprocessing on the A instances, in the heavy disruption scenario where the problem sizes are bigger than the light disruption scenario. For the data sets A1-A5, without preprocessing, the IP built a large problem of between $49 - 101$ million variables and took between $92 - 303$ seconds overall. Preprocessing itself took a maximum of 7.73 seconds in the instances, and using the resulting graphs shrunk the IP problem size to a maximum of $\sim 25,000$ variables ($< 1\%$ of the previous size) and the runtime went down to less than two minutes (again, $\sim 1\%$ of the previous runtime without preprocessing). Therefore, preprocessing is a crucial step in solving the IP, and we are also able to use it for the MLSP algorithm. The runtime of Algorithm 4 without preprocessing was too large (timed out at 1000 s) on the B and X instances to report (whereas the runtime with preprocessing was still in the order of seconds in these instances).

	A1	A2	A3	A4	A5
Preprocessing Time (s)	5.51	4.82	6.58	5.86	7.73
No. of IP vars with preprocessing	25013	23531	23640	20251	23369
No. of IP vars without preprocessing	49,39,268	48,24,900	50,60,784	47,28,402	1,01,21,568
No. of nonzero vars in IP solution	1650	1618	1707	1527	2921
IP Runtime with preprocessing (s)	0.88	0.75	0.83	0.87	0.98
Runtime without preprocessing (s)	92.57	92.82	155.30	163.19	303.14

Table 2.9: Comparing the performance of building the IP and running Algorithm 4 in Gurobi with and without preprocessing. We consider a heavy disruption scenario on all A instances.

2.6.4 Comparison of IP and MLSP

We present a comparison of Algorithms 4 and 5 with two metrics: (i) running time (in seconds), (ii) solution quality, measured as the percentage difference in the objective value of the MLSP compared to the objective value of the IP. Define the solution gap,

$$\epsilon := \frac{[\text{opt-value}(MLSP) - \text{opt-value}(IP)]}{\text{opt-value}(IP)} \times 100.$$

As explained before, the IP may not account for some network-specific constraints (e.g. maximum number of legs of an itinerary), hence the IP is still not the *exact* solution to the passenger recovery problem. Therefore, the solution gap ϵ is an upper bound on the optimality gap between the MLSP (heuristic) solution and the exact solution to the passenger recovery problem.

In Table 2.10, we consider the heavy disruption scenario on both XA and XB instances, which were the final data sets in the ROADEF challenge used to rank participants. On the XA data sets, the preprocessing steps are a maximum of 11.66 seconds, and the IPs constructed for Algorithm 4 only have up to $41,539$ variables (among a potential size of > 5 million, the product of ~ 3000 itineraries, 600 flights and 3 cabin types), and the IP algorithm takes roughly one second to run on a commercial solver. The MLSP approach Algorithm 5 takes slightly only slightly longer (< 2.4 seconds), and ϵ a maximum of $\sim 7\%$, suggesting a near-optimal performance of the network-based approach. Recall the IP approach Algorithm 4 we consider is not quite the exact solution- the network-based approaches can handle the practical constraint of the maximum number of legs in a recovered itinerary.

On the XB data sets in Table 2.10, preprocessing is an order of magnitude longer than the XA data sets, taking up to 78.31 seconds, and the IPs take up to almost 3.5 seconds. The MLSP approach takes almost twice as long as the IP, though the solution gaps ϵ are again reasonable at $< 6\%$. In the ROADEF challenge, many of the finalists could not compute the solution to the integrated flight and recovery problem in less than ten minutes (see discussion in Bisaillon *et al.* [11] and Sinclair *et al.* [78]) on the XB1-XB4 data sets. We are able to compute passenger recovery solutions efficiently and near optimally, suggesting the flight recovery problem is significantly more complex to solve and to integrate with passenger recovery for large data sets.

	XA1	XA2	XA3	XA4	XB1	XB2	XB3	XB4
Preprocessing Time (s)	7.45	11.66	6.23	11.23	78.31	77.84	77.38	75.72
Number of IP variables	30757	23909	35382	41539	90461	84407	52246	42373
Runtime of IP (s)	1.07	1.09	1.08	1.11	3.32	3.51	2.61	2.19
Runtime of MLSP (s)	1.46	1.43	1.66	2.40	5.95	5.16	4.81	4.72
solution gap ϵ in $\%$	2.52	3.21	6.71	1.27	4.42	4.27	3.11	3.91

Table 2.10: Performance of proposed algorithms, (i) IP Algorithm 4 (ii) MLSP Algorithm 5 with heavy disruptions on the XA and XB data sets. Runtimes are reported in seconds.

We now investigate what happens during a *light disruption*- like many of the ROADEF data set disruptions. The recovered flight solution differs from the original flight schedules by a small number- 20% in our setting. As a consequence, the number of undisrupted itineraries is high, and the size of K_{dis} is small. Intuitively, compared to heavy disruptions, Algorithm 4, which has a lesser size of variables and constraints (specifically, $O(|K_{dis}|)$), and Algorithm 5, which has number of iterations $O(|K_{dis}|)$ should both be faster. Let us check ϵ experimentally in Table 2.11, on the B instances in the light disruption scenario. The solution gap is $< 1\%$ in most instances- when most flights and cabin classes remain the same, the total costs due to downgrading and delay are low and cancellations make up most of the cost. Cancellations occur when there is no remaining capacity in the paths available for passengers, and both algorithms would make the same decisions. Thus the gap is quite low between both approaches when disruptions are light. We would next like to compare the performance of batching, which we do so in the heavy disruption setting, in order to allow for bigger gaps.

	B1	B2	B3	B4	B5
Preprocessing Time (s)	36.22	47.38	45.97	40.62	36.07
Number of IP variables	10430	30341	9323	28231	12066
Runtime of IP (s)	0.76	1.74	0.61	0.88	0.93
Runtime of MLSP (s)	1.53	2.36	1.92	1.60	1.21
solution gap ϵ in $\%$	0.07	1.35	0.49	0.41	0.34

Table 2.11: Performance of proposed algorithms, (i) IP Algorithm 4 (ii) MLSP Algorithm 5 with light disruptions on the B data sets. Runtimes are reported in seconds.

2.6.5 Performance of MLSP with batching

We consider various batch sizes $\gamma \in \{5, 25, 100, 250, 500\}$ and show the performance of MLSP Algorithm 5 with batching. We present the comparison to batching on the B instances, since they are one of the largest in the size of itineraries, and the number of flights.

In Table 2.12, we consider the heavy disruption scenario on the B instances, to understand the performance of the proposed passenger recovery algorithms. First, our preprocessing methods work well, with a maximum of 98 seconds in the B instances to build graphs for a total of $11,000$ itineraries. Building and running the IP is faster than our heuristic methods in all the data sets we consider. The IP is built and run in an optimized commercial solver (Gurobi 9.1), with a few tens of thousands of integer variables. Using more efficient data structures and programming optimization techniques, we might get a faster MLSP running time.

Comparing the solution gap ϵ in Table 2.12, our proposed network-based heuristic method performs quite well compared to the IP solution. The value optgap$_{MLSP}$ is between $1.6 - 4.1\%$ for the iterative Algorithm 5. With the batching approach, we do not always notice an improvement in the solution gaps for all batch sizes over the default MLSP (which by definition has batch size γ = 1). Batching can solve issues with respect to edge cases- for example, the top label for the itinerary j in K_{dis} could have lower downgrading and delay cost (given by the third component) than the last label for itinerary $j-1$, i.e., we could end up using precious remaining flight capacity on a higher cost label, and be forced to cancel the best label of itinerary j. Because we group all (pruned) labels of a batch together and sort them, batching can eliminate the edge cases within itineraries in the batch. However, there is a tradeoff- the cancellation costs are not considered between all itineraries of the same batch. For example, say $\gamma = 50$, and there is a flight f with remaining capacity 1 on a cabin class $'F'$ in the recovered flight solution. There could be an itinerary k_1 with one passenger, and a cancellation cost of 9000 (Values used in the experiments are provided in Tables 2.1-2.4) and sink label with downgrading/delay cost 100 in its third component, and an itinerary k_2 with a cancellation cost of 2500, with a sink label with downgrading/delay cost 99 units. Suppose both k_1 and k_2 need the precious first class seat in flight f. If the itineraries end up in the same batch, the label for k_2 is used to allocate passengers first, instead of k_1. The contribution to the objective cost is now $99 + 7500 = 7599$, instead of a potential $100 + 2500 = 2600$.

In Table 2.12, we do nevertheless see large batch sizes helping with the solution gap compared to $\gamma = 1$. We cannot discern a clear pattern, which suggests batch sizes are dependent on the exact structure of the itineraries and the costs involved. The general takeaway is that without much difference in solving time, we can obtain better solutions by attempting higher batch sizes than $\gamma = 1$ of the default MLSP Algorithm 5.

For the sake of completion, we also report the performance of batching on the ROADEF final data sets, the XA and XB instances. In Table 2.13, we consider the *light* disruption scenario on the XA data sets. Recall from Table 2.7, the number of disrupted itineraries is low in the light disruption scenario. One key takeaway from Table 2.13 is that the running times of MLSP are sometimes even lower than the IP (e.g., 0.26s for MLSP vs 0.31s for XA1 instance), while batching typically takes more time, possibly due to storing and running a large number of labels in each batch. For the solution quality, in the XA1 and XA4 instances, batching has worse solution quality, whereas, in the XA2 instance, both MLSP without batching as well as with batching has no solution gap. Overall, changing batch sizes do not seem to have much effect in Table 2.13, possibly because of the low number of disrupted itineraries in XA instances under the light scenario (check Table 2.7.

In Table 2.14, we consider the *heavy* disruption scenario on the XB data sets. Recall from

Table 2.8, the number of disrupted itineraries is high in the heavy disruption scenario. In Table 2.14, MLSP is mostly slow compared to the IP run on a commercial solver, though the differences are smaller (the largest difference is for XB4 where IP takes 2.37s and the MLSP takes 3.84 s). Batching slows down the MLSP in every instance. For the solution quality, solution gaps are between $3-5\%$ in the XB instances. There seem to be noticeable improvements in ϵ for different batch sizes. In the XB1 instance, increasing batch sizes to $\gamma = 250$ or 500 brings down the solution gap to just 0.17%, down from 3.29% without batching. A similar effect is observed for the other instances as well, though a higher γ does not always reduce the solution gap. As explained before, this is because the cancellation costs are not considered between all itineraries of the same batch. Recall among the ROADEF finalists, some algorithms could not solve the XB instances within the required time [78, 11] for the integrated recovery problem. We are able to compute passenger recovery solutions efficiently and near optimally with the batched MLSP, suggesting the flight recovery problem is significantly more complex to solve and to integrate with passenger recovery for large data sets.

	B1	B2	B3	B4	B5
Preprocessing Time (s)	71.28	97.74	93.17	96.30	81.01
Number of IP variables	45322	49071	32273	9483	41985
Runtime of IP (s)	4.77	3.75	4.22	3.15	3.03
Runtime of MLSP (s)	4.99	4.72	5.08	4.33	3.84
Runtime of MLSP -Batching (s)					
$\gamma = 5$	5.73	8.29	4.01	4.32	4.28
$\gamma = 25$	6.05	6.51	3.92	4.01	3.84
$\gamma = 100$	5.72	5.64	4.16	3.53	3.19
$\gamma = 250$	6.34	7.09	5.20	3.35	3.08
$\gamma = 500$	7.97	6.38	4.82	3.22	3.27
solution gap ϵ in $\%$	2.74	2.80	4.07	1.61	1.92
solution gap ϵ of MLSP -Batching in $\%$					
$\gamma = 5$	4.91	5.88	2.48	4.42	4.32
$\gamma = 25$	2.69	4.87	1.56	3.02	1.66
$\gamma = 100$	0.81	4.28	1.33	1.76	0.56
$\gamma = 250$	0.47	3.49	2.39	0.16	0.03
$\gamma = 500$	0.38	2.22	1.41	0.43	0.76

Table 2.12: Performance of proposed algorithms, (i) IP Algorithm 4 (ii) MLSP Algorithm 5 (iii) MLSP Algorithm 5 with batching on B instances with heavy disruptions. Runtimes are reported in seconds.

	XA1	XA2	XA3	XA4
Preprocessing Time (s)	3.76	4.52	2.55	7.58
Number of IP variables	10210	10344	8015	19630
Runtime of IP (s)	0.31	0.34	0.35	0.66
Runtime of MLSP (s)	0.26	0.42	0.25	1.38
Runtime of MLSP -Batching (s)				
$\gamma = 5$	0.55	0.97	0.73	3.24
$\gamma = 25$	0.54	1.04	0.75	2.79
$\gamma = 100$	0.56	0.96	0.77	2.37
$\gamma = 250$	0.57	0.98	0.74	2.43
$\gamma = 500$	0.56	0.98	0.72	2.37
solution gap ϵ in $\%$	2.25	0.00	0.02	0.14
solution gap ϵ of MLSP -Batching in $\%$				
$\gamma = 5$	2.38	0.00	0.68	0.46
$\gamma = 25$	2.37	0.00	0.68	0.48
$\gamma = 100$	2.37	0.00	0.68	0.48
$\gamma = 250$	2.37	0.00	0.68	0.48
$\gamma = 500$	2.37	0.00	0.68	0.48

Table 2.13: Performance of proposed algorithms, (i) IP Algorithm 4 (ii) MLSP Algorithm 5 (iii) MLSP Algorithm 5 with batching on XA instances with light disruptions. Runtimes are reported in seconds.

	XB1	XB2	XB3	XB4
Preprocessing Time (s)	70.35	77.41	76.49	70.43
Number of IP variables	55605	60931	54755	41442
Runtime of IP (s)	3.05	3.80	2.99	2.37
Runtime of MLSP (s)	3.59	3.80	4.73	3.84
Runtime of MLSP -Batching (s)				
$\gamma = 5$	4.15	5.65	5.48	4.17
$\gamma = 25$	4.16	5.05	6.02	4.34
$\gamma = 100$	4.29	5.13	6.21	4.06
$\gamma = 250$	4.65	5.07	6.23	3.98
$\gamma = 500$	5.09	5.38	6.14	4.22
solution gap ϵ in $\%$	3.29	4.33	3.17	2.92
solution gap ϵ of MLSP -Batching in $\%$				
$\gamma = 5$	4.38	6.29	2.50	2.09
$\gamma = 25$	0.68	3.91	2.75	2.30
$\gamma = 100$	0.20	1.90	0.52	1.79
$\gamma = 250$	0.17	0.25	0.67	0.84
$\gamma = 500$	0.17	0.17	1.22	0.92

Table 2.14: Performance of proposed algorithms, (i) IP Algorithm 4 (ii) MLSP Algorithm 5 (iii) MLSP Algorithm 5 with batching on XB instances with heavy disruptions. Runtimes are reported in seconds.

2.7 Conclusions and Future Work

In this chapter, we provide two major solution approaches to solve the passenger recovery problem given the flight recovery solution as an input, thus placing our work in the context of partially integrated recovery. Our first contribution is a preprocessing step, that computes the graphs of reachable airports and feasible flights for each itinerary. Experiments show preprocessing is in the order of minutes on large airline problem sizes. Next, we provide an integer program, which when constructed and solved using the preprocessed graphs, provides an upper bound on the exact solution within a few seconds on a commercial solver. The variables of the IP denote the number of passengers that can be moved through a given flight, on a given cabin class for a given itinerary. The objective is separable by itineraries, and so are almost every set of constraints, except for one which aggregates the variables over all itineraries to be upper bounded by flight capacities. Thus, parallelizing may not be an option for the IP. The IP could also scale linearly in each of the various parameters of the airline network (e.g., number of disrupted itineraries, number of flights, etc.). Moreover, the IP approach may not be able to include practical constraints like a maximum increase in the number of legs of an itinerary, while keeping linear constraints.

Hence, we foresee the need for a network-based approach, that solves a number of multiple-label shortest path (MLSP) problems, with the number of iterations linear in the size of disrupted itineraries. The key idea is to store a pruned list of labels at every node, with each label containing four components- information on downgrading costs, the number of legs used so far, the maximum number of passengers that could be allocated to the path, as well as the path itself. Delay costs for an itinerary can only be added at the sink of the itinerary, hence the sink labels are sorted and an allocation procedure is performed that constructs the recovered passenger itineraries.

Experiments suggest the MLSP approach yields good solution quality, with a solution gap no more than 5% on large airline networks, and the heuristic also runs in a time that is comparable to the IP algorithm. To potentially improve the solution, we also propose a batching extension to the MLSP algorithm that can solve edge cases inherent in the iterative process, at the risk of ignoring the cancellation costs of itineraries within the batch. Experiments suggest batching to be effective at yielding better quality solutions, though there does no way to pick an ideal batch size without knowing more about the problem structure.

There are also post-optimization ideas that help in lowering the itinerary costs even further- e.g. using non-disrupted itineraries (notice our formulation uses only re-planned disrupted itineraries) to intentionally delay non-disrupted passengers to make way for cleverly chosen disrupted passengers, similar to the proposal in Sinclair *et al.* [78]. We can also preprocess by rounding flight departure and arrival times in the flight network with *rounding*. If we wish to construct faster algorithms at the expense of optimality, we can increase the complexity of preprocessing the flight network. One direction for future work could be a rounding idea- split the entire recovery window into buckets of Ψ_k (e.g. 15 minutes), and round all flight takeoff times down and landing times up, to the boundaries of the bucket. Instead of creating one node for each airport, we do the same steps as preprocessing, but instead create a new node for each pair (airport, bucket interval).

In the real world, airlines utilize powerful processors to obtain recovery solutions, and expect to re-plan within a few minutes, and send the information out to their customers. If real-world data on passenger itineraries can be obtained with enough detail, we could put our algorithms to test on computing clusters to obtain practical performance guarantees.

Chapter 3: Computing the optimal distributionally-robust strategy in Stackelberg Games

In this chapter, we present our work on computing distributionally-robust Stackelberg strategies. This chapter is based on the article Ananthanarayanan and Kroer [90] under review, written in collaboration with Christian Kroer. A preprint is available at `https://arxiv.org/abs/2209.07647`.

3.1 Introduction

Stackelberg games are a popular game model for settings where one player is able to commit to a strategy, before the other player (or players) get to choose their strategy. The model was originally introduced by von Stackelberg [91] in order to analyze competition among firms and first-mover advantage. Conitzer and Sandholm [24] showed that an optimal strategy for the leader to commit to can be computed in polynomial time for Stackelberg games. Since then, Stackelberg games have received considerable attention in computational game theory, largely because of applications to various security problems such as airport security [25], federal air marshal scheduling [31], preventing poaching and illegal fishing [26, 92], and several others.

In many applications, the opponent payoffs are not known with certainty. Uncertainty in parameters could occur due to limited scope in observable data, noise or prediction errors. For example, in the case of protecting wildlife from poaching, the goal of the park rangers (leader) is to defend a set of targets from being attacked by the poachers (follower) [92]. The rangers are trying to protect wildlife by patrolling locations where they frequently appear, while the poachers can conduct surveillance to learn about the patrol strategy and try to poach the animals by picking a target area [92]. we would not know the exact utility function of the follower (i.e., the poacher), but only have some model thereof. We would not know the exact utility function of the poacher (follower) but only some model thereof [26, 31, 92].

One approach for handling uncertainty about payoffs is to assume that each player has a publicly-known distribution over utility functions. The class of games called Bayesian Stackelberg games make the above assumption. Computing an optimal strategy for the leader in a Bayesian game setting with a finite set of follower types is NP-hard [24]. Nonetheless, there exist algorithms such as DOBBS [93] which can solve practical game instances. However, Bayesian games still require an exact specification of the distribution over possible follower types, putting strong assumptions on the modeling capacity of the leader who is committing to a strategy based on the supposed distribution.

Another approach to handling uncertainty is through robust optimization, where the goal is to compute a solution that maximizes utility given the worst-case parameter instantiation [32, 94]. In the Stackelberg game context, we typically interpret the above goal as computing an optimal strategy for the leader to commit to, given that a worst-case follower utility will be selected from an uncertainty set, and then the follower best responds based on this utility function. Robust Stackelberg models and regret-based methods for handling uncertainty about follower payoffs have been studied for security games [95, 96, 97] as well as *green security games* [98] and *extensive-form games [33].*

One issue with robust optimization models is that they can often lead to overly conservative solutions, due to the worst-case nature over a potentially large uncertainty set. In optimization, this can be ameliorated by considering *distributionally-robust optimization* (DRO).

We briefly describe DRO in a more generic optimization sense, then discuss how to apply DRO models to Stackelberg equilibria. In a standard stochastic optimization problem, the decision maker wishes to choose a strategy from a feasible region X, but the objective f depends on an unknown (at the time of decision making) quantity from a set θ of uncertain parameters. Thus, we wish to solve an optimization problem of the form

$$\min_{x \in X} \mathop{\mathbb{E}}_{\theta \sim \mu} f(x, \theta) \tag{3.1}$$

where μ is a distribution over the uncertain parameters. If the support on θ is finite then (3.1) is comparable to the Bayesian Stackelberg setting, where an exact distribution over parameters is required. In many settings, one may not know the true underlying distribution even with multiple realizations of the unknown parameter, and using a distribution different from μ could yield bad decisions [99].

A robust variation of the stochastic optimization problem would replace the expectation with a minimum over θ, where θ would come from some set that attempts to capture the uncertainty in θ. In DRO, a middle-ground between stochastic and robust optimization is struck. In DRO, we assume that there is a distribution over θ, but we do not know it exactly. Instead, that distribution is a worst-case instantiation from some *ambiguity set* of possible distributions. Thus, a generic DRO problem has the form

$$\min_{x \in X} \max_{\mu \in \mathcal{D}} \mathop{\mathbb{E}}_{\theta \sim \mu} f(x, \theta),$$

where $\mathcal{D}$ is the set of possible distributions over θ. A recent review of DRO can be found in Rahimian and Mehrotra [100]. Shapiro [94] also gives an overview of all these approaches to uncertainty.

Translating the DRO problem into the context of Stackelberg games, θ is the utility function of the follower, and $\mathcal{D}$ is the set of possible distributions over follower utilities.

Let us now see how DRO relaxes both the settings of robust Stackelberg games and Bayesian Stackelberg games. First, if the ambiguity set $\mathcal{D}$ is a singleton that contains only the true distribution of follower utility function, then DRO applied to Stackelberg games reduces to a Bayesian Stackelberg game. On the other hand, if $\mathcal{D}$ contains all the Dirac masses on the set of possible follower utilities, then DRO on Stackelberg games reduces to a robust Stackelberg model. Thus, a judicial choice of $\mathcal{D}$ can put DRO between Bayesian games and robust Stackelberg games. To the best of our knowledge, there has been no prior work on computing optimal distributionally-robust strategies to commit to.[1] Moreover, for the first time, when the set of utility functions could be infinite, we use the Wasserstein metric to construct the ambiguity set for which we have a MIP based algorithm.

3.1.1 Our contributions

We introduce the notion of a distributionally-robust Strong Stackelberg Equilibrium (DRSSE) for a normal-form game [90], allowing us to capture settings where there is some partial knowledge of the follower utility function. We focus on the simplest case of normal-form Stackelberg games. We prove that a DRSSE is guaranteed to exist for a broad class of ambiguity sets. Our result implies new existence results for some existing robust strong Stackelberg models as well.

We present two direct algorithms to compute DRSSE in the general case, when the set of follower utilities is finite, and with no assumptions on the ambiguity set. First, we show that in this case, it is possible to extend the classical algorithm that enumerates the set of possible best responses for each utility function, and solves a mathematical program for each choice, to the case of DRSSE. In the Bayesian games setting, each mathematical program is an LP, whereas in our setting it is a bilinear saddle-point problem (which can potentially be converted to an LP, depending on the structure of the ambiguity sets). As in the case of Bayesian games, this algorithms requires enumerating an exponentially-large set of mathematical programs, as there are m^k possible choices, where m is the number of follower actions and k is the number of follower utilities. To avoid this exponential search, we introduce binary variables that encode the choice of follower action for each utility function. Then, we show that it is possible to encode the constraints for inducing a given best response for the follower using linear constraints and these binary variables. Putting these things together, we get a mixed-integer program with a bilinear objective.

[1]Liu *et al.* [101] study distributionally-robust Nash and Stackelberg equilibria, but that setting is not about computing the optimal strategy to commit to under distributional uncertainty about the follower utility. Instead, they study a setting where each individual agent employs a DRO approach to their equilibrium behavior. This would not lead to a suitable solution concept when the goal is to compute an optimal leader strategy given uncertainty about the follower utility model, and thus is not suitable for e.g. security games or inspection games with uncertainty about the follower.

Next, we allow for an infinitely-large set of possible utility functions E^f, and we focus on a representation of the ambiguity sets as a *Wasserstein ball* around some finitely-supported nominal estimate of the probability distribution. Using recent characterizations of such ambiguity sets, we show that in this case we can still use duality theory to arrive at a mixed-integer program, albeit one with a robust optimization flavor which requires repeated MIP solving. Experiments substantiate the tractability of this MIP based algorithm on a classical Stackelberg game, showing that our approach scales to medium-sized games. We also discuss in detail the special case of finite utility functions in the Wasserstein setting along with three experiments showing tractability on classical Stackelberg games as well as synthetic data. Implementations of our algorithms and experiments may be found at `https://github.com/saimali/DRStackelberg`.

3.1.2 Organization of the chapter

The rest of this chapter is organized as follows. In Section 3.2, we introduce and prove the existence of distributionally-robust Strong Stackelberg equilibrium (DRSSE). Section 3.3, we propose two algorithms to compute DRSSE when there are a finite set of of follower utilities. We generalize to infinite sets of utility functions in Section 3.4 with a distance based ambiguity set and provide a repeated MIP solving based algorithm, with experiments on a classic Stackelberg game in Section 3.5. A special case of finite utility functions in the Wasserstein setting is discussed along with baselines and experiments in Section 3.6. We finally outline conclusions in 3.7.

3.2 Distributionally-robust Stackelberg Games

We consider a two player (leader and follower) general-sum game where the leader has a finite set of n actions A_l and the follower has a finite set of m actions A_f. Let Δ^l, Δ^f denote the set of probability distributions over the leader and follower actions. Let the utilities be $u_l : A_l \times A_f \rightarrow [0,1]$ for the leader and follower utilities $u_f : A_l \times A_f \rightarrow [0,1]$ which come from a compact set $E^f \subseteq [0,1]^{n\times m}$. The arguments inside the utility functions can be mixed strategies in which case, we consider the appropriate weighted sum of utilities, for example, $u_l(x, a_f) := \sum_{a\in A_l} x_a u_l(a, a_f)$

when $x \in \Delta^l$. Stackelberg equilibrium is a solution concept for this type of game where we want the leader's strategy to be optimal, assuming the follower will 'best respond' knowing the leader's strategy. Given a leader mixed strategy $x \in \Delta^l$, the best response for the follower given a particular utility function u_f is

$$BR(x, u_f) = \operatorname*{arg\,max}_{y \in \Delta^f} u_f(x, y). \tag{3.2}$$

One can see that $BR(x, u_f)$ need not be a singleton set. How ties are broken leads to different notions of Stackelberg equilibrium. A few commonly used tie breaking rules are:

- **Strong Stackelberg Equilibrium (SSE),** whenever the follower has multiple best responses, they break ties in favor of the leader. They play $y^* = \arg\max_{y \in BR(x,u_f)} u_l(x, y)$.
- **Weak Stackelberg Equilibrium (WSE),** whenever the follower has multiple best responses, they break ties adversarially with respect to the leader. They play $y^* = \arg\min_{y \in BR(x,u_f)} u_l(x, y)$.

Strong Stackelberg equilibrium has been studied by far the most. The SSE assumption is convenient because it usually leads to more tractable solution concepts, and existence is guaranteed, unlike for e.g. WSE [102]. From a practical standpoint, the assumption that the follower break ties in favor of the leader is accepted because in most cases the leader can induce the favorable strong equilibrium by selecting a strategy arbitrarily close to equilibrium that makes the follower strictly prefer the desired strategy [102] (though see Guo *et al.* [103] for a discussion on how this assumption can fail). More generally, tie-breaking rules rely on precise maximization by the follower, and expecting the follower to break ties in a certain way with respect to the leader may be counter-intuitive in adversarial games [96]. Therefore robust solutions (like the setting we consider) are useful so as not to depend too much on these rules.

An immediate consequence of the structure of the best response problem is that a pure-strategy best response always exists [93]. To that end, we will only consider pure-strategy best responses of the follower throughout the paper.

Now let us consider our robust game model. As mentioned, we assume that there is some set E^f of possible follower utility functions, and the leader does not know which utility function $u_f \in E^f$

the follower will have. Since $u_f(\cdot, \cdot) \in [0, 1]$, the set E^f is bounded under most common distance metrics. Strictly speaking we only need E^f to be a Polish space, i.e., a separable complete metric space. However, for ease of readability the reader can think of E^f as endowed with the standard Frobenius norm as a metric. We furthermore assume that there is some (unknown) probability distribution μ over the set E^f. While μ is unknown, it is assumed to come from some *ambiguity set* $\mathcal{D}_f$, and the goal of the leader is to maximize their worst-case utility over $\mathcal{D}_f$. Given a leader strategy $x \in \Delta^l$, the worst-case distribution $\mu \in \mathcal{D}_f$ is selected, and overall the leader utility is then

$$g(x) := \inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[\max_{y \in BR(x, u_f)} u_l(x, y) \right]. \tag{3.3}$$

The innermost maximization represents the fact that the follower breaks ties in favor of the leader: given x and u_f, the follower chooses a best response that is optimal for the leader. The $\inf_\mu \mathbb{E}_{u_f}$ part represents the fact that the follower utilities are distributed according to the worst-case distribution μ chosen from $\mathcal{D}_f$.

An important analytical object will be the inner maximization as a function of x and the follower utility u_f:

$$h(x, u_f) := \max_{y \in BR(x, u_f)} u_l(x, y). \tag{3.4}$$

Denote $h(x, \cdot)$ as notation for h to be considered a function of only the parameter x, for a fixed u_f.

Lemma 6. *The function $h(x, \cdot)$ is upper semicontinuous.*

Proof. First, for any $x \in \Delta^l$ and $u_f \in E^f$, the set $BR(x, u_f)$ is non-empty (recall the discussion above where we concluded a pure strategy best response always exists). Let

$$y(x, u_f) \in \mathop{\arg\max}_{y \in BR(x, u_f)} u_l(x, y)$$

be a mapping from any given pair (x, u_f) to a best response $y(x, u_f)$ that breaks ties in favor of the leader (if there are multiple best responses that break ties in favor of the leader, then we assume that there is a fixed ordering over the finitely-many follower actions, which is used to further break ties). Thus, we wish to show that

$$h(x, u_f) = \max_{y \in BR(x, u_f)} u_l(x, y) = u_l(x, y(x, u_f))$$

is upper semicontinuous (u.s.c.) in x for any fixed u_f. We use the following definition of upper semicontinuity: for every x_0 and $\epsilon > 0$, there exists a neighborhood $N(x_0)$ around x_0, such that $h(x, u_f) \leq h(x_0, u_f) + \epsilon$ for all $x \in N(x_0)$.

Now note that for any $y \notin BR(x_0, u_f)$, there exists a neighborhood around x_0 such that y is not a best response anywhere in that ball (this follows by the fact that u_f is continuous in x). Let $\hat{N}(x_0)$ be a neighborhood such that this holds for every $y \notin BR(x_0, u_f)$. Also by continuity, we may select a neighborhood $N'(x_0)$ such that $|u_l(x_0, y) - u_l(x, y)| < \epsilon$ for all $x \in N'(x_0)$ and each pure strategy y. Now we may select $N(x_0) = \hat{N}(x_0) \cap N'(x_0)$, which shows that $h(x, \cdot)$ is u.s.c. in x. $\square$

Lemma 6 helps in proving a crucial property satisfied by g: it is upper semicontinuous. This fact will later allow us to easily conclude that equilibria exist in our setting.

Lemma 7. *The leader utility function $g(x)$ is upper semicontinuous.*

Proof. Define $f(x, \mu) := \mathbb{E}_{u_f \sim \mu} h(x, u_f)$, and note that $g(x) = \inf_{\mu \in \mathcal{D}_f} f(x, \mu)$.

Upper semicontinuity (u.s.c.) is preserved under pointwise infimum (see e.g. Lemma 2.41 of Aliprantis and Border [104]). Therefore, $g(x) = \inf_\mu f(x, \mu)$ is u.s.c. as long as $f(x, \mu)$ is u.s.c. in x, for each μ.

Thus it remains to prove that $f(x, \mu)$ is u.s.c. in x for each μ. By the definition of u.s.c., we want to show that for any sequence $\{x_i\}$ which converges to x, $f(x, \mu) \geq \limsup_i f(x_i, \mu)$. Expanding the definition, we have

$$\limsup_i f(x_i, \mu) = \limsup_i \mathbb{E}_{u_f \sim \mu} h(x, u_f). \tag{3.5}$$

We now apply the Reverse Fatou lemma [105], stated below:

[Reverse Fatou Lemma] Let $\{f_i\}$ be a sequence of extended real-valued measurable functions defined on a measure space (S, Σ, η). If there exists a non-negative measurable function g on S such that (i) $\int_S g d\eta < \infty$ (ii) $f_i \leq g$ for all i, then

$$\limsup_i \int_S f_i d\eta \leq \int_S \limsup_i f_i \eta.$$

To apply the Reverse Fatou Lemma on the sequence $\{h(x_i, u_f)\}$, we need to check two things. First, that there exists a dominating measurable and integrable function. This is clearly true: we take a function which is equal to 1 on $[0,1]^{n\times m}$ and zero everywhere else. Second, that each $h(x_i, u_f)$ is measurable as a function of u_f when x_i is fixed. To check measurability, first consider that $h(x_i, u_f)$ takes on a finite set of values: $u_l(x_i, a_f)$ for each of the finitely-many $a_f \in A_f$. It suffices to show that for each a_f such that $a_f = y(x_i, u_f)$ for some u_f, the nonempty set

$$U(a_f, x_i) = \{u_f \in [0,1]^{n\times m} : y(x_i, u_f) = a_f\}$$

is measurable. Since $U(a_f, x_i)$ is the set of all u_f such that $u_f(x_i, a_f) \geq u_f(x_i, a'_f)$ for all other a'_f, and furthermore $u_f(x_i, a_f) > u_f(x_i, a'_f)$ for all a'_f such that a'_f would be chosen over a_f in case of tied follower utilities, we conclude U is measurable. Thus we fulfil the conditions to apply the reverse Fatou Lemma on (3.5),

$$\begin{aligned}
\limsup_i f(x_i, \mu) &= \limsup_i \mathbb{E}_{u_f\sim\mu} h(x_i, u_f) \\
&\leq \mathbb{E}_{u_f\sim\mu} \limsup_i h(x_i, u_f) \ \text{(by Reverse Fatou)} \\
&\leq \mathbb{E}_{u_f\sim\mu} h(x, u_f) \ (\because h(x, u_f) \text{ is u.s.c from Lemma 6}) \\
&= f(x, \mu).
\end{aligned}$$

□

A distributionally-robust Stackelberg solution is a strategy for the leader that maximizes the leader utility $g(x)$:

Definition 1. ***(DRSSS)*** *A distributionally-robust strong Stackelberg solution (DRSSS) is a mixed strategy $x_l \in \Delta^l$ such that:*

$$x_l \in \arg\max_{x \in \Delta^l} \inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[\max_{y \in BR(x, u_f)} u_l(x, y) \right] \tag{3.6}$$

Similar to the definition DRSSS of an optimal leader solution, we can define the corresponding equilibrium concept as well, where we also specify a best response for each utility function of the leader.

Definition 2. ***(DRSSE)*** *A strategy tuple (x, y_{u_f}) is said to form a distributionally-robust strong Stackelberg equilibrium (DRSSE) if x is a DRSSS and $y_{u_f} : E^f \to A_f$ is a best response mapping which breaks ties in favor of the leader.*

A crucial question is whether DRSSE is guaranteed to exist. One of our main theoretical results is that this is indeed the case. The heavy lifting is performed by Lemma 7, and with Lemma 7 in hand the existence result follows from the extreme value theorem for upper semicontinuous functions.

Theorem 1. *A DRSSE is guaranteed to exist.*

Proof. A pair $(x^*, y(x^*, \cdot))$ is a DRSSE, if $x^* \in \arg\max_{x \in \Delta^l} g(x)$. The best response set for the follower is always non-empty for a fixed x^* and u_f. Thus it suffices to show that $\arg\max_{x \in \Delta^l} g(x)$ is non-empty, or in other words, g attains a maximum over Δ^l. Using the fact that g is u.s.c. by Lemma 7, and the extreme value theorem for semicontinuous functions, we conclude g does attain a maximum. □

DRSSE generalizes several existing solution concepts, as the next proposition shows:

Proposition 5. *DRSSE generalizes SSE and robust strong Stackelberg equilibrium. DRSSE generalizes Bayesian strong Stackelberg equilibrium when there is only uncertainty about the follower payoff.*

Proof. To see why DRSSE generalizes robust Stackelberg, note that we can set the ambiguity set to be the set of point masses on each of the possible utility functions from the robust uncertainty set (as already mentioned in Section 1).

SSE is the special case of robust SSE where the uncertainty set consists of a single point.

DRSSE generalizes Bayesian strong Stackelberg equilibrium, since a special case of distributional uncertainty is where the ambiguity set consists of a single distribution. □

Combining Theorem 1 and Proposition 5 yields a fairly general existence result: since we can construct robust SSE and Bayesian Stackelberg SSE as special cases, our Theorem 1 shows existence for both. This is particularly useful for robust SSE. For example, the authors of Kroer *et al.* [33] left existence of robust SSE in *extensive-form games* an open problem. An extensive-form game has an equivalent normal-form representation that preserves utilities and the best-response relationship, and hence our Theorem 1 shows the first existence result for robust SSE in extensive form games.

3.3 Algorithms for Finite Sets of Follower Utilities

We present two algorithms to compute strategies that form a distributionally-robust SSE, in the setting where E^f is a finite set of k possible follower utilities.

Define $\mathcal{Z}$ to be the (finite) set of mappings from follower utility functions to actions. For a particular mapping z, we will use z_{u_f} to denote the action specified for utility function u_f under z. Given a leader strategy x, there is a at least one $z \in \mathcal{Z}$ that specifies a follower action which is a best response for each possible follower utility. Conversely, given a choice of z, we can consider the set $\mathcal{X}_z$ consisting of all strategies for the leader that make z a correct mapping from utility function to best response.

We first show a naive way of computing a DRSSE: we can enumerate all possible z, and for each z compute the best possible leader strategy that induces z. This constraint on the leader strategy can be captured by a set of linear inequalities, which gives us the following bilinear problem for a fixed z:

$$\text{OPT}_l(z) = \max_{x \in \Delta^l} \inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[u_l(x, z_{u_f}) \right], \tag{3.7}$$

$$\text{s.t. } u_f(x, z_{u_f}) \geq u_f(x, a'_f), \forall\, a'_f \in A_f,\ \forall\, u_f \in E^f$$

The set of constraints ensures that for any u_f and a'_f, z_{u_f} is among the set of best responses for a leader strategy x. The inner minimization term in the objective represents the fact that even after choosing a best response for each follower utility, the leader still faces the worst-case distribution over those utilities. Since there is a finite set of follower utilities, we can rewrite this as

$$\inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[u_l(x, z_{u_f}) \right] = \inf_{\mu \in \mathcal{D}_f} \left[\sum_{i=1}^{k} \mu_{f_i} u_l(x, z_{u_i}) \right],$$

where z_{u_i} is the follower best response for the i-th follower utility function.

Now, in order to find the optimal strategy to commit to, we may iterate over all $z \in \mathscr{Z}$, solve the mathematical program for each, and pick the optimal solution x^* associated to the program with the highest value. Note that if there are multiple best responses to x^*, then this approach corresponds to assuming that ties are broken in favor of the leader. Once we have the optimal strategy x^*, we may find the associated follower strategy simply by picking the best-response mapping z for which x^* was the solution. Then once an instantiation of the follower utilities is known, the follower plays the corresponding best action taken from this best z^*. This enumeration algorithm shows that DRSSE can be computed in exponential time in terms of m and k (we solve m^k math programs with linear constraints, each in n variables).

In practice, we do not want to enumerate all the exponentially-many possible best response mappings. Instead, we use binary variables to design a mixed-integer non-linear program for branching on the choice of z. Introduce binary variables δ_{a_f, u_f} for each pair (a_f, u_f) and these variables activate constraints whenever a_f is the BR to u_f. For a sufficiently large real M,

$$\text{OPT}_l = \max_{x,q} \inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[q_{u_f} \right] \tag{3.8}$$

$$\text{s.t. } u_f(x, a_f) \geq u_f(x, a'_f) + M(\delta_{a_f,u_f} - 1) \;\; \forall\, a_f, a'_f \in A_f, \;\; \forall u_f \in E^f \tag{3.9}$$

$$q_{u_f} \leq u_l(x, a_f) - M(\delta_{a_f,u_f} - 1) \;\; \forall\, a_f \in A_f, \; \forall\, u_f \in E^f \tag{3.10}$$

$$\sum_{a_f \in A_f} \delta_{a_f,u_f} = 1, \; \forall\, u_f \in E^f \tag{3.11}$$

$$\delta_{a_f,u_f} \in \{0, 1\} \;\; \forall\, a_f \in A_f, \; \forall\, u_f \in E^f \tag{3.12}$$

$$x \in \Delta^l, q \in \mathbb{R}^k. \tag{3.13}$$

The first set of constraints (3.9) relate to picking the best response follower action for each utility function (given a leader strategy x). The second set of constraints (3.10) ensure the leader utility corresponding to the best response follower action above shows up in the innermost term in the objective as q_{u_f}. The third set of constraints (3.11) guarantees that the first constraint is only activated once for each u_f , i.e., for a given u_f, exactly one follower action is assigned as best response. The other constraints (3.12) and (3.13) specify the domain of δ (binary variables), x (simplex) and q (real). We prove the math program (3.8) generates a DRSSE.

Theorem 2. *A solution (x^*, q^*) to the mathematical program (3.8) forms a DRSSE.*

Proof. Since q is being maximized and looking at the constraints, it is clear for any u_f, the optimal $q^*_{u_f} = u_l(x, a_f)$ for some $a_f \in A_f$. It suffices to show that for every u_f and any x,

$$q^*_{u_f} = u_l(x, a_f) = \max_{y \in BR(x,u_f)} u_l(x, y).$$

If not, there is another a'_f such that for some u_f, we have $a'_f \in BR(x, u_f)$ and

$$u_l(x, a'_f) = \max_{y \in BR(x,u_f)} u_l(x, y) > u_l(x, a_f) = q^*_{u_f},$$

which is a contradiction to maximality since

$$\inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[u_l(x^*, a'_f) \right] > \underbrace{\inf_{\mu \in \mathcal{D}_f} \mathop{\mathbb{E}}_{u_f \sim \mu} \left[q^*_{u_f} \right]}_{OPT_l} .$$

Thus the math program breaks ties in favor of the leader in the strong sense of Stackelberg equilibrium. □

The above mathematical program (3.8) gives an algorithm to compute a DRSSE. However, the objective having an inf in it, may not be easily computed for all type of ambiguity sets. Whether this bilinear objective is easy to handle depends on the form of $\mathcal{D}_f$. Note that here we are heavily exploiting the fact that the set of follower utility functions is finite, in order to encode z using integer variables. Thus we cannot surmise how the algorithm can be applied in practice without knowing more information about the ambiguity sets.

In the next section, we handle infinite sets of follower utility functions, as well as consider some structure on ambiguity sets so that we can obtain tractable algorithms.

3.4 Algorithms for Wasserstein Ambiguity Sets

We now move on to considering a specific type of ambiguity set $\mathcal{D}_f$ in the case where there is an infinitely-large set of possible utility functions E^f, and show that in this case we can still use duality theory to arrive at a mixed-integer program, albeit one with a robust optimization flavor which requires repeated MIP solving.

There are various different ways to deal with ambiguity sets (see discussion on ten common ambiguity sets in Keith and Ahner [23]). We would prefer the ambiguity sets to be as small as possible, and contain the true distribution with a good level of certainty [94, 100]. Two major ways of defining the ambiguity sets are:

- **Define the set using moment constraints**. For example, one can assume moment uncertainty conditions with additional assumptions [106] to handle robust optimization problems.

As discussed in Gao and Kleywegt [99], it has been shown that in many cases these moment based assumptions lets us formulate the problem as a conic quadratic or semi-definite program. However, the moment-based approach is based on the curious assumption that certain conditions on the moments are known exactly but that nothing else about the relevant distribution is known.

- **Distance from a nominal distribution**. A nominal probability distribution ν in $\mathcal{D}_f$ is given, and $\mathcal{D}_f$ is specified as a set of probability measures which are in some sense close to ν. Popular choices of the statistical distance are ϕ-divergences (which include Kullback-Leibler divergence and Total Variation distance as special cases), the Prokhorov metric, and Wasserstein distances [99, 107].

Here we will focus on the second setting, based on distance from a nominal distribution. We leave the question of whether similar results can be obtained for moment-based constraints for future work.

Recognizing the fact that the ambiguity set should be chosen judicially for the application at hand, Gao and Kleywegt [99] argue that by using the Wasserstein metric the resulting distributions hedged against are more reasonable than those resulting from other popular choices of sets, such as ϕ-divergence-based sets. Distributionally-robust stochastic optimization with Wasserstein distance has been empirically shown to resolve issues with ϕ-divergences, which do not address how close two points in the support are to each other. The integration involved in the definition of the Wasserstein metric is in a linear form of the joint distribution, whereas typical ϕ-divergences are nonlinear [108]. Moreover, Del Barrio *et al.* [109] show that as the number of samples increase, the empirical distribution under the Wasserstein metric almost surely converges to the true distribution. Fournier and Guillin [110] provide concentration results on the number of samples needed to guarantee a level of convergence. Likelihood-based divergence measures like Kullback-Leibler divergence do not satisfy the axioms of distance, unlike the Wasserstein metric. Blanchet and Murthy [111] also argue that Wasserstein distances do not restrict all the probability measures in the neighborhoods to share the same support as the nominal distribution.

For the above reasons, we focus on using the Wasserstein metric in our work (see Chapter 6 of Villani [112] for properties of Wasserstein distances). Our results will build on recent advances in duality theory for dealing with the infinite-dimensional optimization problem over $\mathcal{D}_f$ in the case of Wasserstein distances [99].

3.4.1 Wasserstein Distance

Let the set of potential follower utilities E^f be the set of all matrices in $[0,1]^{n\times m}$ specifying a mapping from a strategy pair a_l, a_f to a payoff. Let d be any distance metric between utility functions such that E^f is a Polish (separable complete metric) space. An example metric d would be the Frobenius norm of the difference between the follower payoff matrices:

$$d_F(u_{f_i}, u_{f_j}) = \left(\sum_{a\in A_l, a'\in A_f} (u_{f_i}(a,a') - u_{f_j}(a,a'))^2 \right)^{1/2}$$

Let $\mathcal{P}(E^f)$ be the set of Borel probability measures on E^f, and $\mathcal{P}_t(E^f)$ its subset with finite t-th moment ($t \geq 1$). If $\mu, \nu \in \mathcal{P}_t(E^f)$ (with any metric d on E^f), the Wasserstein distance between them is

$$W_t(\mu,\nu) := \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int_{E^f\times E^f} d(x,y)^t d\gamma(x,y) \right)^{1/t},$$

where $\Gamma(u,v)$ is the collection of all measures with marginals μ and ν on the first and second factors respectively. The Wasserstein distance between μ, ν is the minimum cost (in terms of d) of redistributing mass from ν to μ. For this reason, it is also called the "earth mover's distance" in the computer science literature. Wasserstein distance is a natural way of comparing two distributions when one is obtained from the other by perturbations. One sufficient condition for the infimum to be attained is d being lower semicontinuous (see Theorem 4.1 of Villani [112]).

Using the Wasserstein distance, we define our ambiguity set $\mathcal{D}_f$ as all distributions within a small radius (θ) of a nominal distribution ν:

$$\mathcal{D}_f := \{\mu \in \mathcal{P}(E^f) \mid W_t(\mu,\nu) \leq \theta\}. \tag{3.14}$$

The radius θ controls how far away from the nominal distribution ν the worst-case distribution can go. The parameter θ is also referred to as the *level of robustness*. By adjusting the radius of the ambiguity set, the modeler can thus control the degree of conservatism of the underlying optimization problem. If the radius drops to zero, then the ambiguity set shrinks to a singleton that contains only the nominal distribution, in which case the distributionally-robust problem reduces to an ambiguity-free stochastic problem.

Consider the restriction of the inner maximization function $h(x,\cdot) : E^f \to [0,1]$ for a fixed x (see (3.4)). We can characterize $h(x,\cdot)$ as a *simple* function, i.e., a measurable function that takes finitely many values $\{u_l(x,y) | y \in A_f\}$. Thus it is also L^1-measurable.

Next we wish to transform the leader utility defined in (3.3), into a finite dimensional problem (there could be infinitely many distributions in D_f and we cannot test them all) in the case of a general nominal distribution $\nu \in \mathcal{P}(E^f)$ and Wasserstein ambiguity set $\mathcal{D}_f$. Fix an arbitrary x and let $h(u_f) = h(x, u_f)$. We write this as the *primal* inner problem (akin to the primal problem in Gao and Kleywegt [99]), and it is equal to:

$$\nu_P := \inf_{\mu \in D^f} \left\{ \int_{E^f} h(u_f)\mu(du_f) : W_t(\mu,\nu) \ \leq \theta \right\}, \tag{3.15}$$

Following Gao and Kleywegt [99] the dual is an optimization problem on the dual variable λ,

$$\nu_D = \sup_{\lambda \geq 0} \left\{ -\lambda\theta^t + \int_{E^f} \inf_{u_f \in E^f} \left[\lambda d^t(u_f, u'_f) + h(u_f) \right] \nu(du'_f) \right\}$$

It is easily verified that we satisfy all the conditions needed for the strong duality theorem of Gao and Kleywegt [99], and thus we get $\nu_P = \nu_D < \infty$. The dual is a one dimensional convex minimization problem in λ, and always admits a minimizer (though in general the infima for each u'_f may or may not have a simple representation).

3.4.2 Nominal distribution with finite support

We now focus on the setting where the nominal distribution on E^f has finite support. Let the nominal distribution be written as $\nu = \sum_{j=1}^{k} \nu_j \delta_{\hat{u}_{f_j}}$ for some $\{\hat{u}_{f_j} \in E^f \mid j \in [k]\}$. The finite-support setting is practically important because it occurs when we have received a finite set of observations of follower utilities, and we treat those as an empirical distribution; the Wasserstein ball around this empirical distribution then gives robustness guarantees.

Specializing the primal and dual problems from the general nominal case, for $t \geq 1$ and $\theta > 0$ we get

$$\nu_P := \inf_{\mu \in \Delta^k} \left\{ \sum_{i=1}^{k} \mu_i h(u_{f_i}) : W_t(\mu, \nu) \ \leq \theta \right\} \tag{3.16}$$

$$\nu_D = \sup_{\lambda \geq 0} \left\{ -\lambda \theta^t + \sum_{j=1}^{k} \nu_j \inf_{u_f \in E^f} \left[\lambda d^t(u_f, \hat{u}_{f_j}) + h(u_f) \right] \right\}. \tag{3.17}$$

The overall problem of computing a DRSSS is then

$$\begin{aligned} \mathrm{OPT}_l(\theta) = \sup_{x \in \Delta^l, \lambda \geq 0} \Bigg\{ & -\lambda \theta^t + \sum_{j=1}^{k} \nu_j w_j : \\ & w_j \leq \lambda d^t(u_f, \hat{u}_{f_j}) + h(x, u_f) \ , \forall j \in [k], \ u_f \in E^f \Bigg\}. \end{aligned} \tag{3.18}$$

Computing OPT_l is not easy since there is an infinite number of constraints due to E^f generally being uncountably large.

We next propose an incremental MIP-generation approach that addresses this issue. The key idea behind the MIP is to leverage the structure of the function h in order to represent the constraint for a fixed u_f via several constraints and Boolean variables. Since there are infinitely-many u_f, we start with a small finite set of candidates, and iteratively expand this set. We proceed in iterations $\tau = 1, \ldots$ until convergence. For each nominal point $j \in [k]$, let $E^f_{\tau,j}$ be the set of utility functions that are considered for point j at iteration τ of the MIP. Define $E^f_\tau := \cup_j E^f_{\tau,j}$, whose cardinality is the number of utility functions generated so far. We introduce binary variables δ_{a_f, u_f} for each pair

(a_f, u_f) which denote whether a_f is the chosen best response to u_f. We solve the following MIP at iteration τ,

$$\text{OPT}_l^\tau = \min_{x,\delta,\lambda,w} \left\{ \lambda\theta^t - \sum_{j=1}^{k} \nu_j w_j \right\} \tag{3.19}$$

$$\text{s.t. } u_f(x, a_f) \geq u_f(x, a'_f) + M(\delta_{a_f,u_f} - 1) \quad \forall\, a_f, a'_f \in A_f,\ \forall\, u_f \in E_\tau^f \tag{3.20}$$

$$w_j \leq (1 - \delta_{a_f,u_f})M + \lambda d^t(u_f, \hat{u}_{f_j}) + u_l(x, a_f) \quad \forall\, a_f \in A_f,\ \forall\, u_f \in E_{\tau,j}^f,\ \forall j \in [k] \tag{3.21}$$

$$\sum_{a_f \in A_f} \delta_{a_f,u_f} = 1,\ \forall\, u_f \in E_\tau^f \tag{3.22}$$

$$x \in \Delta^l, \lambda \geq 0, w \in \mathbb{R}^k \tag{3.23}$$

$$\delta_{a_f,u_f} \in \{0, 1\}\ \forall\, a_f \in A_f,\ \forall\, u_f \in E_\tau^f \tag{3.24}$$

The first set of constraints (3.20) relate to picking the best response follower action for each utility function (given a leader strategy x). The second set of constraints (3.21) is the w_j constraint in OPT$_l(\theta)$. The third set of constraints (3.22) guarantees that the first constraint is only activated once for each u_f , i.e., for a given u_f, exactly one follower action is assigned as best response. The other constraints (3.23) and (3.24) specify the domain of δ (binary variables), x (simplex), λ and w (real).

Using the optimal variables $(x_\tau, \delta_\tau, \lambda_\tau, w_\tau)$ from solving OPT$_l^\tau$ at iteration τ, we construct a sub-problem that finds new utility functions. For each $j \in [k]$, we define the following subproblem which computes, over the infinitely-large set E^f, the utility function that most violates the second constraint (3.21):

$$\Gamma(\tau, j) = \min_{a_f \in A_f} \inf_{u_f \in BR^{-1}(x_\tau, a_f)} \lambda_\tau d^t(u_f, \hat{u}_{f_j}) + u_l(x_\tau, a_f), \tag{3.25}$$

Let $u_f^{(\tau,j)}$ be the utility function in E^f that gives us $\Gamma(\tau, j)$. Formally, let

$$\hat{a}_f := \arg\min_{a_f \in A_f} \inf_{u_f \in BR^{-1}(x_\tau, a_f)} \lambda_\tau d^t(u_f, \hat{u}_{f_j}) + u_l(x_\tau, a_f),$$
$$u_f^{(\tau,j)} := \arg\inf_{u_f \in BR^{-1}(x_\tau, \hat{a}_f)} \lambda_\tau d^t(u_f, \hat{u}_{f_j}) + u_l(x_\tau, \hat{a}_f).$$

Next, we then compute the most-violated of the k nominal points

$$\Gamma(\tau) := \min_{j \in [k]} \{\Gamma(\tau, j) - w_{\tau_j}\} \tag{3.26}$$

Since we can enumerate over a_f, the only hard part of solving these subproblems is to resolve the inner $\inf$ term, calculated over $BR^{-1}(x, a_f)$, which may be difficult depending on the structure of E^f. If we have an oracle that gives us the inner $\inf$ term for a fixed a_f, we can solve the subproblem in m oracle calls.

Next we show that when E^f is described by linear constraints, the inner $\inf$ problem of (3.25) can be solved via convex minimization as long as the distance metric d is nice (e.g. for the ℓ_1 or ℓ_2 distance). For a fixed $a_f \in A_f$, the inner problem can be written as,

$$\inf_{u_f} \ \lambda_\tau d^t(u_f, \hat{u}_{f_j}) + u_l(x_\tau, a_f) \tag{3.27}$$
$$\text{s.t.} \ \sum_{i=1}^{n} x_{\tau,i} \Big[u_f(a_i, a_f) - u_f(a_i, a_f') \Big] > 0 \ \ \forall a_f' \in B_{x_\tau}(a_f), \tag{3.28}$$
$$\sum_{i=1}^{n} x_{\tau,i} \Big[u_f(a_i, a_f) - u_f(a_i, a_f') \Big] \geq 0 \ \ \forall \ a_f' \in A_f \setminus B_{x_\tau}(a_f), \tag{3.29}$$
$$u_f \in E^f,$$

where $B_x(a) := \{a_f' \in A_f : u_l(x, a_f') > u_l(x, a)\}$. Each element a_f' of $B_x(a) \subset A_f$ (known deterministically for a given x and a) has leader utility bigger than a. Because we wish to enforce strong Stackelberg tie-breaking within the best response set, we need to ensure that any $a_f' \in B_{x_\tau}(a_f)$ is not picked instead of a_f, and hence, the first constraint (3.28) picks an u_f such that a_f

is **strictly** a best response to a given leader strategy x_τ compared to $a'_f \in B_{x_\tau}(a_f)$. In practice, strict inequalities are handled by industry-grade optimization solvers, typically by enforcing a very small epsilon gap. The second constraint (3.29) allows for $a'_f \notin B_{x_\tau}(a_f)$ to also be best responses along with a_f.

Thus for a nice distance metric d (such as ℓ_1 or ℓ_2 distance), the subproblem (3.27) is a convex minimization problem with linear constraints.

We can now summarize the algorithm to solve (3.18). We start with each $E^f_{1,j}$ to be just the nominal functions $\hat{u}_{f_j}$, and at each iteration τ, solve the program (3.19) for OPT$_l^\tau$. Use the optimal solution to solve the subproblem (3.25) for each $a_f \in A_f$ to add more utility functions to $E^f_{\tau+1,j}$, until there is an iteration where none can be added (thus $\Gamma(\tau) \geq 0$). Thus Algorithm 7 can be used

Algorithm 7: Iterative MIP-solving algorithm to compute DRSSS

Input: Leader actions A_l of size n, follower actions A_f of size m, leader utility function u_l, Wasserstein radius θ, finitely supported nominal distribution ν on k points, distanc metric d between follower utility functions.

Result: Optimal values $(x^*, \delta^*, \lambda^*, w^*)$ of DRSSS (3.18). In particular, x^* is the optimal distributionally-robust strategy for the leader to commit to.

```
1  $\tau \leftarrow 1$ // iteration counter
2  $E^f_{\tau,j} \leftarrow \{\hat{u}_{f_j}\}\ \forall j \in [k]$ // initialize to contain only nominal utility
       functions
3  $\Gamma \leftarrow -1$ // initialize to start the loop
   // As long as there is a new $u_f$ violating (3.21)
4  while $\Gamma < 0$ do
5      Solve the MIP (3.19) for OPT$_l^\tau$ and obtain the solution $(x_\tau, \delta_\tau, \lambda_\tau, w_\tau)$
6      for $j = 1, \ldots, k$ do
7          Compute solution $\Gamma(\tau, j)$ to the subproblem in (3.25) // Solve the subproblem
               for each $j$
8          if $\Gamma(\tau, j) < w_{\tau_j}$ then
               // if (3.21) can be violated
9              $E^f_{\tau+1,j} \leftarrow E^f_{\tau,j} \cup \{u_f^{(\tau,j)}\}$ // add new $u_f$ to the set of utility
                   functions for the next iteration's MIP
10         end
11     end
12     Update $\Gamma \leftarrow \Gamma(\tau)$ as defined in (3.26) // most violated among all $j$
13     Update $\tau \leftarrow \tau + 1$
14 end
15 Output $(x^*, \delta^*, \lambda^*, w^*) = (x_{\tau-1}, \delta_{\tau-1}, \lambda_{\tau-1}, w_{\tau-1})$
```

to compute the optimal leader strategy x^* and DRSSS.

3.5 Experiments

We now test the scalability of the MIP Algorithm 7 (henceforth referred to as *DR MIP*). We do not consider any baselines in our general setting with an infinite set of utility functions. A discussion of Wasserstein ambiguity sets with finite E^f, along with suitable baselines is provided in Section 3.6.

We present the experimental performance based on a classic Stackelberg game from GAMUT [113]. We vary different game parameters to investigate the scalability. All experiments are timed out at 1000 seconds, and run times reported as a function of the parameter being varied. All experiments were conducted using Gurobi 9 to solve MIPs and LPs (default internal parameters), on a Macintosh with 2.4 GHz Quad-Core intel Core $i5$ processor. Run times are reported in seconds. We consider a typical data-driven setting, where the nominal distribution ν is simply the discrete uniform distribution with weight $\frac{1}{k}$ on each of the k empirical observations. We set the Wasserstein radius $\theta = 0.1$, exponent $t = 2$ and a tight choice of $M = 2$ for the MIP (3.19).

Results on Inspection Game The Inspection Game [114] is a classic Stackelberg setting where an inspector tries to deter an inspectee from cheating. In the Simple Inspection Game setting in GAMUT, there is a set S of size $0 < s \leq 8$. The inspector (the leader) chooses a subset of S of size at most p $(0 < p \leq s)$. Hence the size of their action space is $n = \binom{s}{1} + \ldots + \binom{s}{p}$. Similarly the inspectee (the follower) chooses a subset of size at most q. Hence the size of their action space is $m = \binom{s}{1} + \ldots + \binom{s}{q}$. If there is no intersection in the chosen sets, the leader receives a payoff $-\alpha$ and the follower receives a payoff α for some $\alpha > 0$. Otherwise both players get zero payoff. The structure of E^f is helpful in tractably computing the subproblem (3.25) for the Inspection game. In fact, we can express the objective of the inner subproblem (3.27) as a quadratic function of two

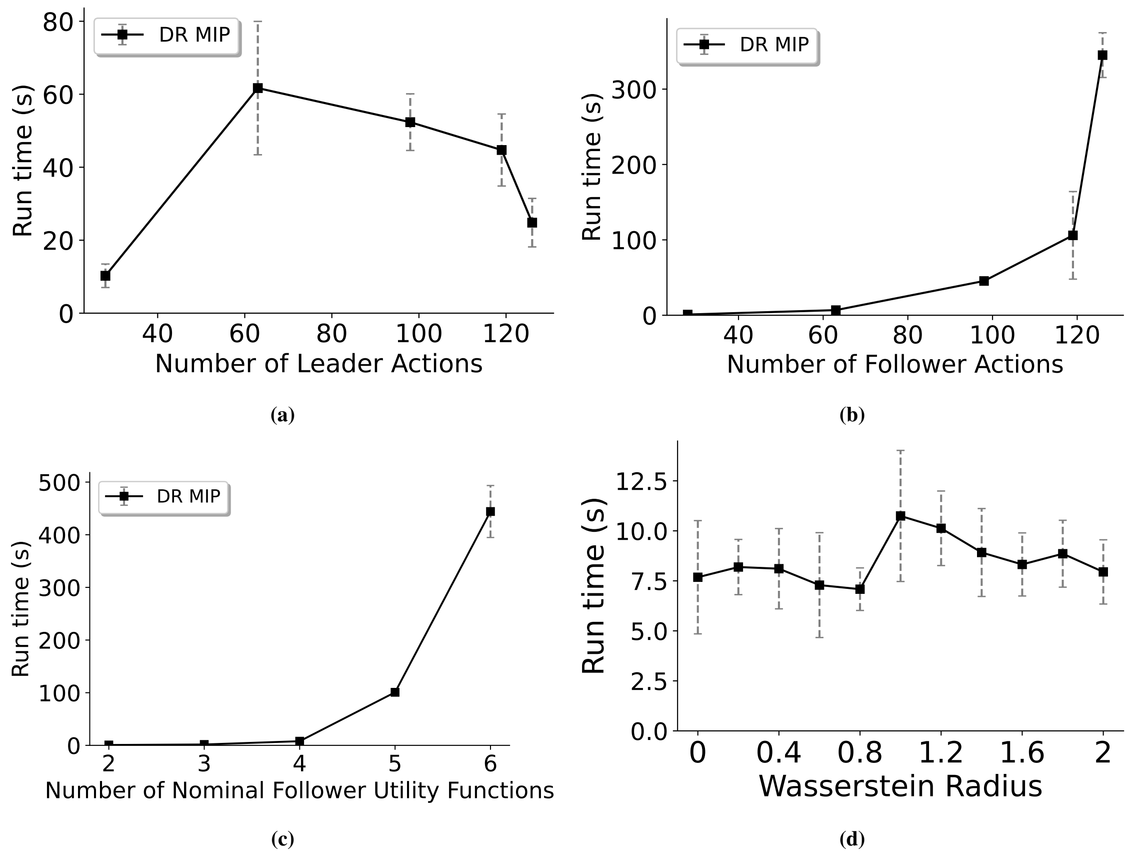


Figure 3.1: Performance of the DR MIP based Algorithm 1 on the Simple Inspection Game, averaged over 10 simulations with standard deviations displayed. **(a)** Runtime vs number of leader actions (n) with $s = 7, q = 2$ and $k = 4$. **(b)** Runtime vs number of follower actions (m) with $s = 7, p = 5$ and $k = 2$. **(c)** Runtime vs number of nominal follower functions (k) with $s = 7, p = 2$ and $q = 2$. **(d)** Runtime vs Wasserstein radius (θ) with $s = 7, p = 2, q = 2$ and $k = 4$.

variables for a ℓ_2 distance metric such as Frobenius norm d_F. Recall the definition,

$$d_F(u_{f_i}, u_{f_j}) = \left(\sum_{a \in A_l, a' \in A_f} (u_{f_i}(a, a') - u_{f_j}(a, a'))^2 \right)^{1/2}$$

The utility function of the inspection game has only two variables, the payoff α when the sets intersect and the payoff β when the sets do not intersect. For a given inspection game size, the positions of α in any follower utility matrix u_f is the same (resp. for β). And in the nm entries of the matrix, a fixed number (say c) of the entries have value α, and the rest have value β. Therefore the distance function in the objective of (3.27) can be written as,

$$\begin{aligned} d_F(u_f, \hat{u}_{f_j}) &= \left(\sum_{a \in A_l, a' \in A_f} (u_f(a,a') - \hat{u}_{f_j}(a,a'))^2 \right)^{1/2} \\ &= \left(c(\alpha_{u_f} - \alpha_{\hat{u}_f})^2 + (nm - c)(\beta_{u_f} - \beta_{\hat{u}_f})^2 \right)^{1/2} \end{aligned}$$

In our experiments on Inspection game, we choose $t = 2$, and hence the the inner subproblem (3.27) is a quadratic program in two variables α and β denoting the payoffs of the Inspection follower utility u_f. The objective is simply

$$\inf_{\alpha,\beta} \left\{ \lambda_\tau c(\alpha - \alpha_{\hat{u}_f})^2 + \lambda_\tau (nm - c)(\beta - \beta_{\hat{u}_f})^2 \right\},$$

and we can write the linear constraints similarly as well.

In the following experiment we normalize the payoffs to lie in $[0, 1]$ and set leader payoffs to lie in $\{0, 0.5\}$ instead of $\{-\alpha, 0\}$. For k different nominal follower utility functions, we set follower payoffs to be random variables that are uniformly distributed in $[0.3, 0.6)$ instead of 0 and uniformly distributed in $[0.7, 1)$ instead of α.

In Figure 3.1a, for $s = 7, q = 2$, and $k = 4$, we vary the number of leader actions (by varying the maximum size p of the leader set). Since the leader wishes for an intersection to happen, picking a larger set is good. As p increases, the size of the leader action space increases and the runtime of the DR MIP goes up. However for $p = 5$ and $p = 6$, larger sets are readily available and the DR MIP converges faster though the leader action space is large.

In Figure 3.1b, for $s = 7, p = 5$, and $k = 2$, we vary the number of follower actions (by varying the maximum size q of the follower set). Recall that the number of binary variables δ in the iterative MIP scales with the size of follower set, and hence the number of follower actions again has a moderate impact on scalability of the DR MIP.

In Figure 3.1c, for $s = 7, p = q = 2$, we vary the number of nominal utility functions which has an exponential impact on the scalability of the DR MIP, thus having access to a a higher of empirical distributions slows down the algorithm. Recall again that the number of binary variables is also dependent on k at each iteration of the algorithm.

In Figure 3.1d, for $s = 7, p = 2, q = 2$, and $k = 4$, we vary the Wasserstein radius θ from 0 to 2. There is little impact on the running time by considering an increase in this parameter.

3.6 Special case: Wasserstein ambiguity sets with a finite set of utility functions

So far in the chapter, we considered general Wasserstein ambiguity sets with infinitely many follower utility functions. The iterative MIP-solving Algorithm 7 involves solving MIPs at every iteration, with convergence dependent on the structure of the ambiguity set.

Let us analyze the case where the set of follower utility functions $E^f = \{u_{f_1}, \ldots, u_{f_k}\}$ is finite, and we show DRSSS can be computed with just a single MIP. We show the performance of the MIP on another classical Stackelberg game called Cournot Duopoly, as well as on a synthetic data set. Moreover, unlike the general case in Section 3.5, the finiteness of E^f allows us to construct two baselines that we can compare the DRSSS MIP to- one baseline is an enumeration approach and the other, a Bayesian Stackelberg MIP.

Since the support E^f of the distributions is finite, let the nominal distribution be written as $\nu = \sum_{j=1}^{k} \nu_j \delta_{u_{f_j}}$ and any other distribution as $\mu = \sum_{i=1}^{k} \mu_i \delta_{u_{f_i}}$, where δ_{u_f} denotes the unit mass on u_f. Since E^f is finite, the integral in the definition of Wasserstein metric simplifies to a summation and can be written as,

$$W_t^t(\mu, \nu) := \min_{\gamma_{ij} \geq 0} \left\{ \sum_{i=1}^{k} \sum_{j=1}^{k} d^t(u_{f_i}, u_{f_j}) \gamma_{ij} \quad : \sum_{j=1}^{k} \gamma_{ij} = \mu_i \; \forall\, i, \sum_{i=1}^{k} \gamma_{ij} = \nu_j \; \forall j \right\} \tag{3.30}$$

We also write the dual of the Wasserstein metric as,

$$W_t^t(\mu, \nu) := \max_{(r,s) \in \mathbb{R}^k \times \mathbb{R}^k} \left\{ r^T \mu + s^T \nu \quad : r_i + s_j \leq d^t(u_{f_i}, u_{f_j}) \; \forall\, i, j \right\} \tag{3.31}$$

Applying Theorem 1 from Gao and Kleywegt [99], we get strong duality and the overall problem of computing a DRSSS is

$$\mathrm{OPT}_l = \min_{\substack{z\in A_f^k\\ x\in\mathcal{X}_z}} \inf_{\lambda\geq 0} \left\{ \lambda\theta^t - \sum_{j=1}^{k} \nu_j w_j \quad : w_j \leq \lambda d^t(u_{f_i}, u_{f_j}) + u_l(x, z_{u_{f_i}}) \;\forall i, j \right\}, \tag{3.32}$$

where the set $\mathcal{X}_z$ is defined as follows: if we fix any $z \in \mathcal{Z}$, the set of feasible leader actions is restricted to the set

$$\mathcal{X}_z = \{x \in \Delta^l \;: u_{f_j}(x, z_j) \geq u_{f_j}(x, a_f) \;\; \forall\, 1 \leq j \leq k, \;\; \forall\, a_f \in A_f\}.$$

Notice in the program for OPT_l that while z is from a more general space, once a z is fixed, we have to pick x from the set $\mathcal{X}_z$, therefore the objective is not easy to compute. To deal with the constraints in $\mathcal{X}_z$, we define boolean variables δ_{a_f,u_f} for each pair (a_f, u_f) and use these variables to activate the constraints in the definition of $\mathcal{X}_z$, as well as the constraints involving w_j's. For a sufficiently large M,

$$\begin{aligned}
\mathrm{OPT}_l = \min_{x,y,\lambda,w} & \left\{ \lambda\theta^t - \sum_{j=1}^{k} \nu_j w_j \right\} \qquad (3.33)\\
\text{s.t. } & u_f(x, a_f) \geq u_f(x, a_f') + M(\delta_{a_f,u_f} - 1) \quad \forall\, a_f, a_f' \in A_f, \;\forall\, u_f \in E^f\\
& w_j \leq (1 - \delta_{a_f,u_f})M + \lambda d^t(u_f, u_{f_j}) + u_l(x, a_f) \quad \forall\, a_f \in A_f, \;\forall\, u_f \in E^f, \;\forall j \in [k]\\
& \sum_{a_f\in A_f} \delta_{a_f,u_f} = 1, \;\forall\, u_f \in E^f\\
& x \in \Delta^l, \lambda \geq 0, w \in \mathbb{R}^k\\
& \delta_{a_f,u_f} \in \{0, 1\} \;\; \forall\, a_f \in A_f, \;\forall\, u_f \in E^f
\end{aligned}$$

We obtain a MIP similar to iterative MIP (3.19) derived earlier, but due to the finiteness of E^f, we can solve (3.33) directly to get the leader strategy and compute DRSSS. The MIP has $n + k + 1$ continuous variables, mk binary variables, and $m^2k + mk^2 + k + 2 = O(mk(m + k))$ linear constraints.

3.6.1 Baselines

We illustrate experiments on the case where E^f is finite on two different games- complementing the Inspection Game considered in Section 3.5, we perform experiments on another classical Stackelberg game in GAMUT called Cournot Duopoly, as well as the synthetic dataset of all possible random matrices in $[0, 1]$, the most general ground set of the utility functions.

We compare the DR MIP (3.33) in both games on two baselines, explained below.

(a) the first baseline is an an enumeration approach where we enumerate all possible instances of z in (3.32) and for each z, we solve the LP and take the best result; Define the LP given a z (written as boolean variables δ),

$$\begin{aligned} \text{OPT-LP}(\delta) = \min_{x,\lambda,w} & \left\{ \lambda\theta^t - \sum_{j=1}^{k} \nu_j w_j \right\} \qquad (3.34) \\ \text{s.t. } & u_f(x, a_f) \geq u_f(x, a'_f) + M(\delta_{a_f,u_f} - 1) \quad \forall\, a_f, a'_f \in A_f,\ \forall\, u_f \in E^f \\ & w_j \leq (1 - \delta_{a_f,u_f})M + \lambda d^t(u_f, u_{f_j}) + u_l(x, a_f) \quad \forall\, a_f \in A_f,\ \forall\, u_f \in E^f,\ \forall j \\ & x \in \Delta^l, \lambda \geq 0, w \in \mathbb{R}^k \end{aligned}$$

If the set of all possible instances of δ is given by

$$\mathcal{Q} = \left\{ \delta \in \{0, 1\}^{m \times k} : \sum_i \delta_{ij} = 1,\ \forall\, j \right\},$$

then the enumeration approach is computing the best result among m^k LPs,

$$\text{OPT}_l = \max_{\delta \in \mathcal{Q}} \text{OPT-LP}(\delta).$$

(b) the second baseline is the Bayesian Stackelberg MIP which is non-robust, where the ambiguity set is a singleton set with only a nominal distribution.

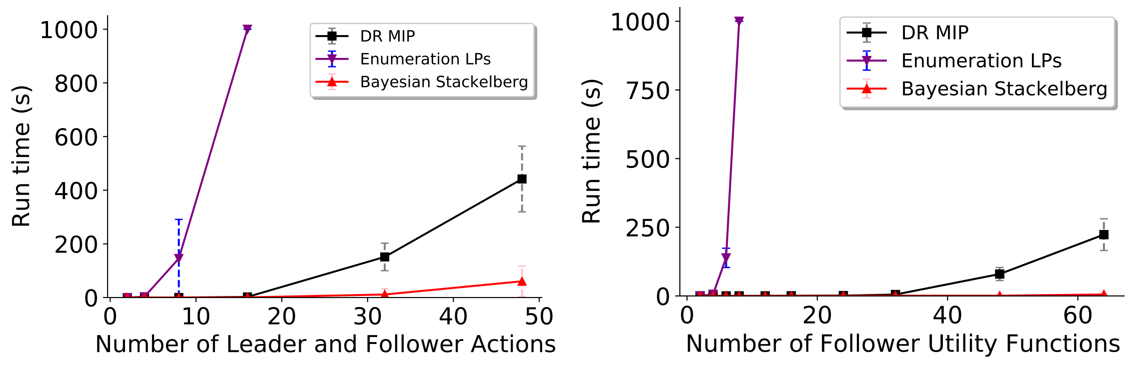
$$
\begin{aligned}
\text{OPT-Bayesian}(\nu) = \max_{x,\delta,w} & \sum_{j=1}^{k} \nu_j w_j \qquad (3.35)\\
\text{s.t. } & u_f(x,a_f) \geq u_f(x,a_f') + M(\delta_{a_f,u_f} - 1) \quad \forall\, a_f, a_f' \in A_f,\ \forall\, u_f \in E^f \\
& w_j \leq (1-\delta_{a_f,u_f})M + u_l(x,a_f) \quad \forall\, a_f \in A_f,\ \forall\, u_f \in E^f,\ \forall j \\
& \sum_{a_f \in A_f} \delta_{a_f,u_f} = 1,\ \forall\, u_f \in E^f \\
& x \in \Delta^l, w \in \mathbb{R}^k \\
& \delta_{a_f,u_f} \in \{0,1\}\ \forall\, a_f \in A_f,\ \forall\, u_f \in E^f.
\end{aligned}
$$

We now test the scalability of the distributionally-robust MIP (3.33) and use the two baselines (3.34) and (3.35). The nominal distribution ν is generated as a random probability vector, and the rest of the experimental setup is the same as previous experiments.

3.6.2 Results on Cournot Duopoly Game

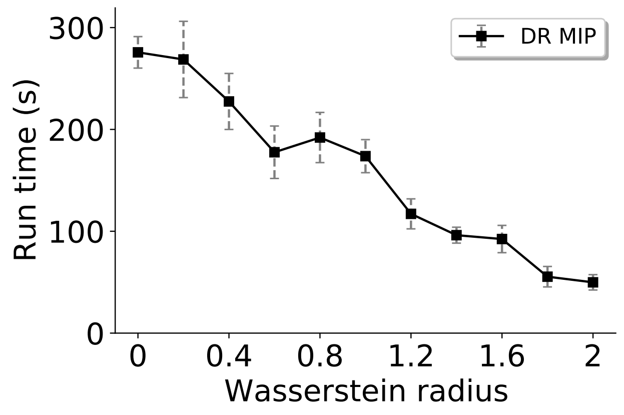
The Cournot duopoly [115] is a game that models two rival firms choosing the quantity of competing goods to produce at the same time. We consider the Stackelberg equlibrium setting of this game in GAMUT. Given an inverse demand function $P(\cdot)$ and increasing cost functions C_i for player i, the utility for player i given player actions (y_1, y_2) is $u_i(y_1, y_2) = P(y_1+y_2)\times y_i - C_i(y_i)$. With the notation $randint(a, b)$ to denote a random integer between a and b, in the following experiment we set $P(x) = 75 - randint(1, 10)x, C_1(y) = randint(10, 40) + randint(10, 20)y$ and $C_2(y) = randint(2, 20) + randint(1, 5)y$ to compute the utilities and normalize them to lie in $[0, 1]$. The number of actions is equal for both players in the game ($n = m$).

In Figure 3.2a, for 4 possible follower utility functions, we can compute DRSSE within the threshold for more than 50 leader and follower actions. The number of follower actions m is crucial to the scalability, affecting the number of integer variables and size of the constraints of the DR MIP (3.33). The enumeration approach hits the threshold very quickly ($n < 20$) and the



(a)

(b)



(c)

Figure 3.2: Performance of the DR MIP (3.33) and baselines on the Cournot Duopoly game, averaged over 10 simulations with standard deviations displayed. **(a)** Runtime vs number of leader actions (n) and follower actions (m) with $k = 4$. **(b)** Runtime vs number of follower utility functions (k) with $n = m = 4$. **(c)** Runtime vs Wasserstein radius (θ) with $n = m = 10$ and $k = 12$.

Bayesian Stackelberg (3.35) runs very fast compared to DRSSE, showing the computational cost of including robustness.

In Figure 3.2b, for 4 leader and follower actions, we can compute DRSSE within the threshold for more than 60 follower utility functions. The enumeration approach hits the threshold very quickly ($n < 10$) since we solve an exponential m^k LPs (one for each z). The Bayesian Stackelberg (3.35) runs very fast compared to DRSSE here as well. The number of follower utilities has a moderate impact on scalability of the DR MIP (3.33).

In Figure 3.2c, for 10 leader and follower actions, and 10 follower utilities, we vary the Wasserstein radius θ from 0 to 2. The DRSSE problem gets easier to solve with an increase in this parameter, as some small set of utility functions dominate eventually.

3.6.3 Results on Synthetic Data

We present the experimental performance based on a synthetic data set for the utilities: the leader utility and all follower utilities are iid random matrices in $[0, 1]$.

In Figure 3.3a, for $m = 12$ and $k = 4$, we can compute DRSSE within a minute for more than 900 leader actions. Unsurprisingly, the number of leader actions n is not crucial to the scalability, since these are reflected in continuous variables and not impacting the size of the constraints. The enumeration approach hits the threshold almost immediately due to the size of action spaces and hence is not plotted. The Bayesian Stackelberg MIP (3.35) runs fast (less than 10 seconds) compared to DRSSE illustrating the computational cost of including robustness.

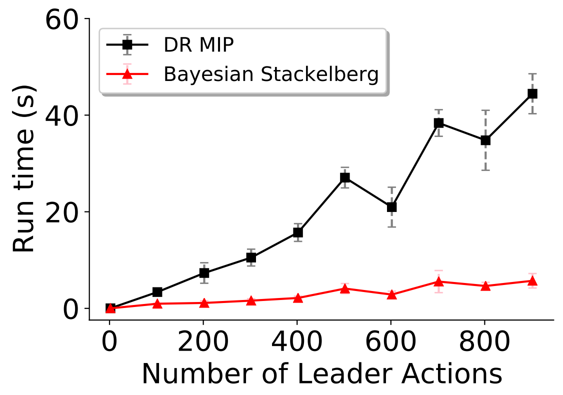
In Figure 3.3b, for $n = 50$, and $k = 4$, we can compute DRSSE within the threshold for more than 50 follower actions. The enumeration approach hits the threshold very quickly ($m < 10$) since we solve an exponential m^k LPs (one for each z). The Bayesian Stackelberg MIP runs very fast compared to DRSSE here as well. The number of follower actions again has a moderate impact on scalability of the DR MIP (3.33).

In Figure 3.3c, for $n = 8$ and $k = 4$, we can compute DRSSE within the threshold for more than 30 follower utility functions. The enumeration approach hits the threshold very quickly ($k < 8$) and the Bayesian Stackelberg MIP runs very fast compared to DRSSE here as well. The number of follower utility functions has an exponential impact on the scalability of the DR MIP (3.33), with k impacting the size of integer variables and integer constraints.

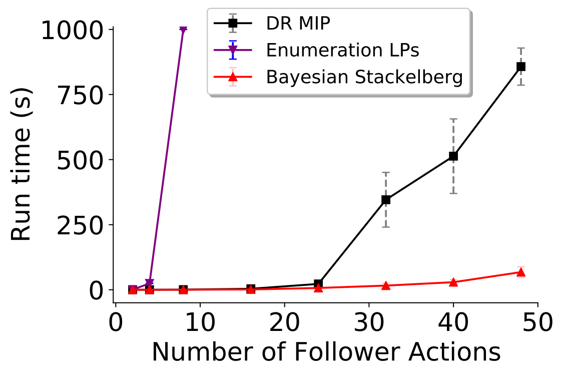
In Figure 3.3d, for $n = m = 10$, and $k = 12$, we vary the Wasserstein radius θ from 0 to 2. The DRSSE problem gets easier to solve with an increase in this parameter, similar to the Cournot game, as some small set of utility functions dominate eventually.

3.7 Conclusion

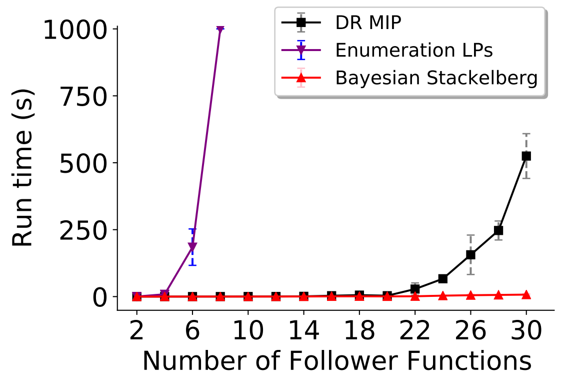
In this work, we initiated the study of computing optimally distributionally-robust strategies to commit to. We formalized the notion of a distributionally-robust strong Stackelberg solution



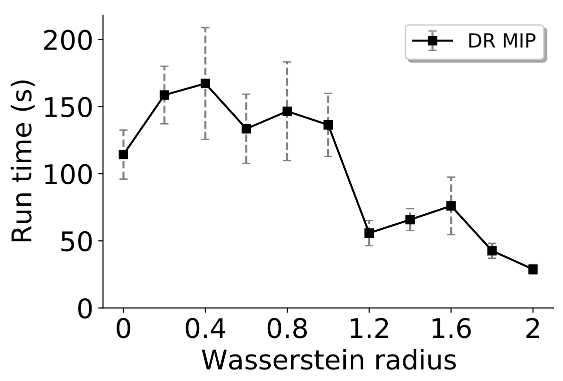
(a)



(b)



(c)



(d)

Figure 3.3: Performance of the DR MIP (3.33) and baselines on the synthetic data set, averaged over 10 simulations with standard deviations displayed. **(a)** Runtime vs number of leader actions (n) with $m = 12$ and $k = 4$. **(b)** Runtime vs number of follower actions (m) with $n = 50$ and $k = 4$. **(c)** Runtime vs number of follower functions (k) with $n = 8$ and $k = 4$. **(d)** Runtime vs Wasserstein radius (θ) with $n = m = 10$ and $k = 12$.

for the leader, and showed that these are guaranteed to exist in a wide number of settings. We presented two algorithms for computing a DRSSE using mathematical programs for any ambiguity set. One algorithm has only continuous variables and the other has mixed integer variables, and the constraints are linear. When the uncertainty is represented by Wasserstein uncertainty, we showed that the above programs can be solved with an incremental mixed-integer linear program. We performed computational experiments on the MIP in terms of different parameters on a classical Stackelberg game where the structure of the set of utility functions can be exploited to compute subproblem tractably. In the Inspection game, the MIP based algorithm scaled well for medium-sized games. We found that the runtime impact of the size of the leader action set is low, the number of nominal utility functions has high (exponential) impact, and the size of the follower action space has moderate impact.

One avenue for future work is to study ambiguity sets that are described by moment uncertainty conditions. These type of assumptions lead to conic quadratic or semi-definite programs in many settings [99]. For DRSSE, it would be interesting to see if it is possible to derive a mixed-integer conic program based on these results. Another promising avenue would be to take our general results on distributionally-robust Stackelberg equilibria and interpret them for popular applications of Stackelberg games like security games, where one could potentially exploit problem structure in order to get more scalable algorithms.

Conclusion

This dissertation focuses on two results in passenger scheduling for transportation systems, albeit with distinct flavors, and introduces a new equilibrium concept to handle uncertainty in a type of game called Stackelberg game. At a high level, our contributions are threefold across the chapters: (i) building a mathematical model to address an optimization problem with real-world applications, when there is uncertainty or unforeseen disruptions, (ii) developing novel algorithms new to the field, either mathematical programs or heuristics, substantiating with theory wherever applicable, and, (iii) Experimentally test our proposed algorithms either by constructing a simulation model or on medium to large sized data sets reflecting the real-world applications.

In Chapter 1, we consider a passenger scheduling problem in vertical transportation- elevator systems in high-rise buildings where passenger demand far exceeds the capacity of the system. We propose new interventions that group passengers together implicitly or explicitly according to their destinations and provide a model that can be applied to generic buildings, and provide theoretical support from queuing theory.

In Chapter 2, we consider a passenger scheduling problem in airline networks- recovering schedules affected by unforeseen or uncertain disruptions. We propose two approaches to reconstruct new schedules, that are high-quality solutions (in terms of minimizing costs) and fast (in order of minutes), and experimentally illustrate their performance on realistic airline network data.

In Chapter 3, we consider a type of game with applications in airport security, wildlife poaching, etc. called Stackelberg games, where players do not play simultaneously. One player (leader) has to commit to a strategy first, without knowing the payoff functions of the other player (follower) who can observe and best respond to the leader's strategy. We initiate the study of distributionally-robust models, providing existence results in general settings, as well as develop tractable algorithms in the case where there could be an infinite number of possible follower utilities, and the uncertainty comes from a type of set defined using Wasserstein distances. Experiments on classical Stackelberg games show tractability on medium-sized games.

References

[1] M. L. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[2] S. M. Ananthanarayanan, C. C. Branas, A. N. Elmachtoub, C. S. Stein, and Y. Zhou, "Queuing safely for elevator systems amidst a pandemic," *Production and Operations Management*, vol. 31, no. 5, pp. 2306–2323, 2022.

[3] L. Weber, *Office Elevator In COVID Times: Experts Weigh In On How To Stay Safe : NPR*, Available at `https://www.npr.org/sections/health-shots/2020/06/08/869595720/the-office-elevator-in-covid-times-experts-weigh-in-on-safer-ups-and-downs` (last accessed date: January 25, 2022), 2020.

[4] D. Swinarski, "Modelling elevator traffic with social distancing in a university classroom building," *Building Services Engineering Research and Technology*, vol. 42, no. 1, pp. 82–97, 2021.

[5] D. Swinarski, *Modeling Elevator Traffic With Social Distancing: Inside Higher Ed*, Available at `https://www.insidehighered.com/views/2020/12/14/advice-making-elevators-safe-possible-during-pandemic-opinion` (last accessed date: January 25, 2022), 2020.

[6] M. Wilson, *It Might Become the Scariest Part of Your Commute: The Elevator: New York Times*, Available at `https://www.nytimes.com/2020/10/26/nyregion/new-york-city-elevators-coronavirus.html` (last accessed date: January 25, 2022), 2020.

[7] P. Smith, *Distancing at Reopened Offices Will Mean Long Elevator Lines: Bloomberg Law*, Available at `https://news.bloomberglaw.com/daily-labor-report/elevator-questions-highlight-ups-and-downs-of-reopening-offices` (last accessed date: January 25, 2022), 2020.

[8] M. Bierlaire, N. Eggenberg, and M. Salani, "Column generation methods for disrupted airline schedules," in *Proceedings of the sixth triennial symposium on transportation analysis*, 2007.

[9] L. K. Hassan, B. F. Santos, and J. Vink, "Airline disruption management: A literature review and practical challenges," *Computers and Operations Research*, vol. 127, p. 105 137, 2021.

[10] J. A. Filar, P. Manyem, and K. White, "How airlines and airports recover from schedule perturbations: A survey," *Annals of operations research*, vol. 108, no. 1, pp. 315–333, 2001.

[11] S. Bisaillon, J. F. Cordeau, G. Laporte, and F. Pasin, "A large neighbourhood search heuristic for the aircraft and passenger recovery problem," *4or*, vol. 9, no. 2, pp. 139–157, 2011.

[12] D. Papiomytis, "[White Paper] Navigating through operational turbulence," Frost & Sullivan, Tech. Rep., 2019, Available at `https://www.frost.com/frost-perspectives/navigating-through-operational-turbulence/` (last accessed date: August 25, 2022).

[13] M. Ball, C. Barnhart, G. Nemhauser, and A. Odoni, "Air transportation: Irregular operations and control," *Handbooks in operations research and management science*, vol. 14, pp. 1–67, 2007.

[14] J. Clausen, A. Larsen, J. Larsen, and N. J. Rezanova, "Disruption management in the airline industry—concepts, models and methods," *Computers & Operations Research*, vol. 37, no. 5, pp. 809–821, 2010.

[15] M Palpant, M Boudia, C Robelin, S Gabteni, and F Laburthe, "ROADEF 2009 Challenge : Disruption Management for Commercial Aviation," *Operations Research*, no. 261, pp. 1–24, 2009.

[16] A. J. Castro, A. P. Rocha, and E. Oliveira, *A new approach for disruption management in airline operations control*. Springer, 2014, vol. 562.

[17] G. Righini and M. Salani, "New dynamic programming algorithms for the resource constrained elementary shortest path problem," *Networks: An International Journal*, vol. 51, no. 3, pp. 155–170, 2008.

[18] G. Barney and L. Al-Sharif, *Elevator traffic handbook: theory and practice*. Routledge, 2015.

[19] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, and A. Togawa, "An elevator group control system with floor-attribute control method and system optimization using genetic algorithms," *IEEE Transactions on Industrial Electronics*, vol. 44, no. 4, pp. 546–552, 1997.

[20] G. C. Barney and S. M. Dos Santos, "Improved traffic design methods for lift systems," *Building Science*, vol. 10, no. 4, pp. 277–285, 1975.

[21] Y. Lee, T. S. Kim, H. S. Cho, D. K. Sung, and B. D. Choi, "Performance analysis of an elevator system during up-peak," *Mathematical and Computer modelling*, vol. 49, no. 3-4, pp. 423–431, 2009.

[22] L. Al-Sharif, H. M. Aldahiyat, and L. M. Alkurdi, "The use of Monte Carlo simulation in evaluating the elevator round trip time under up-peak traffic conditions and conventional group control," *Building Services Engineering Research and Technology*, vol. 33, no. 3, pp. 319–338, 2012.

[23] A. J. Keith and D. K. Ahner, "A survey of decision making and optimization under uncertainty," *Annals of Operations Research*, vol. 300, no. 2, pp. 319–353, 2021.

[24] V. Conitzer and T. Sandholm, "Computing the optimal strategy to commit to," in *Proceedings of the 7th ACM conference on Electronic commerce*, 2006, pp. 82–90.

[25] J. Pita *et al.*, "Deployed armor protection: The application of a game theoretic model for security at the los angeles international airport," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, 2008, pp. 125–132.

[26] F. Fang, P. Stone, and M. Tambe, "When security games go green: Designing defender strategies to prevent poaching and illegal fishing," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[27] Y. Bai, C. Jin, H. Wang, and C. Xiong, "Sample-efficient learning of stackelberg equilibria in general-sum games," *Advances in Neural Information Processing Systems*, vol. 34, pp. 25 799–25 811, 2021.

[28] J. Nash, "Non-cooperative games," *Annals of mathematics*, pp. 286–295, 1951.

[29] B. Peng, W. Shen, P. Tang, and S. Zuo, "Learning optimal strategies to commit to," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2149–2156.

[30] M. Jain, D. Korzhyk, O. Vaněk, V. Conitzer, M. Pěchouček, and M. Tambe, "A double oracle algorithm for zero-sum security games on graphs," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2011, pp. 327–334.

[31] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordonez, and M. Tambe, "Iris-a tool for strategic security allocation in transportation networks," *AAMAS (Industry Track)*, pp. 37–44, 2009.

[32] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton University Press, 2009, vol. 28.

[33] C. Kroer, G. Farina, and T. Sandholm, "Robust stackelberg equilibria in extensive-form games and extension to limited lookahead," *arXiv preprint arXiv:1711.08080*, 2017.

[34] R. C. Larson, "OR Forum-Perspectives on Queues: Social Justice and the Psychology of Queueing," *Operations Research*, vol. 35, no. 6, pp. 895–905, 1987.

[35] C. van Rijn *et al.*, “Reducing aerosol transmission of SARS-CoV-2 in hospital elevators,” *Indoor Air*, vol. 30, no. 6, pp. 1065–1066, 2020.

[36] D. L. Pepyne and C. G. Cassandras, “Optimal dispatching control for elevator systems during uppeak traffic,” *IEEE Transactions on Control Systems Technology*, vol. 5, no. 6, pp. 629–643, 1997.

[37] S. Ross, *Simulation (Fifth Edition)*. Academic Press, 2013.

[38] N. A. Alexandris, “Statistical models in lift systems,” Ph.D. dissertation, The University of Manchester, 1977.

[39] L. Finschi, “State-of-the-art traffic analyses,” *Elevator Technology*, vol. 18, pp. 106–115, 2010.

[40] G. Al Sukkar *et al.*, “Reconciling the value of the elevator round trip time between calculation and simulation,” *Simulation*, vol. 93, no. 8, pp. 707–722, 2017.

[41] H. Hakonen and M. L. Siikonen, “Elevator traffic simulation procedure,” *Elevator World*, vol. 57, no. 9, pp. 180–190, 2008.

[42] J. Mulvany and R. S. Randhawa, *Fair Scheduling of Heterogeneous Customer Populations*, Available at SSRN 3803016, 2021.

[43] J. G. Dai and J. M. Harrison, *Processing Networks: Fluid Models and Stability*. Cambridge University Press, 2020.

[44] J. G. Dai and C. Li, “Stabilizing batch-processing networks,” *Operations Research*, vol. 51, no. 1, pp. 123–136, 2003.

[45] M. Bramson, “Convergence to equilibria for fluid models of FIFO queueing networks,” *Queueing Systems*, vol. 22, no. 1, pp. 5–45, 1996.

[46] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queueing algorithm,” *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 4, pp. 1–12, 1989.

[47] A. K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: the multiple node case,” *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 137–150, 1994.

[48] L. L. Berry, K. Seiders, and D. Grewal, “Understanding Service Convenience,” *Journal of Marketing*, vol. 66, no. 3, pp. 1–17, 2002.

[49] S. M. Ross *et al.*, *Stochastic processes*. Wiley New York, 1996, vol. 2.

[50] M. Bramson, “Instability of FIFO Queueing Networks with Quick Service Times,” *The Annals of Applied Probability*, vol. 4, no. 2, pp. 414–431, 1994.

[51] S. Tao, W. H. Cho, and J. Pender, *The Value of Flexible Customers via Join the Shortest of d Queues*, Cornell University, 2020.

[52] A. N. Elmachtoub, D. Yao, and Y. Zhou, *The Value of Flexibility from Opaque Selling*, Available at SSRN 3483872, 2019.

[53] N. Eggenberg, M. Salani, and M. Bierlaire, “Robust optimization with recovery: Application to shortest paths and airline scheduling,” in *Swiss Transport Research Conference*, 2007.

[54] S. Albers, “Online algorithms: A survey,” *Mathematical Programming*, vol. 97, no. 1, pp. 3–26, 2003.

[55] M. Grötschel, S. O. Krumke, and J. Rambau, *Online optimization of large scale systems*. Springer Science & Business Media, 2013.

[56] P. Kall, S. W. Wallace, and P. Kall, *Stochastic programming*. Springer, 1994, vol. 6.

[57] P. Kouvelis and G. Yu, *Robust discrete optimization and its applications*. Springer Science & Business Media, 2013, vol. 14.

[58] J. C. F. Guimarães and N. D. F. Gualda, “Math-heuristic to solve the airline recovery problem considering aircraft and passenger networks,” *Transportes*, vol. 29, no. 2, 2021.

[59] A. Kasirzadeh, M. Saddoune, and F. Soumis, “Airline crew scheduling: models, algorithms, and data sets,” *EURO Journal on Transportation and Logistics*, vol. 6, no. 2, pp. 111–137, 2017.

[60] D. Zhang, H. H. Lau, and C. Yu, “A two stage heuristic algorithm for the integrated aircraft and crew schedule recovery problems,” *Computers & Industrial Engineering*, vol. 87, pp. 436–453, 2015.

[61] N. Kohl, A. Larsen, J. Larsen, A. Ross, and S. Tiourine, “Airline disruption management—perspectives, experiences and outlook,” *Journal of Air Transport Management*, vol. 13, no. 3, pp. 149–162, 2007.

[62] D. Klabjan, “Large-scale models in the airline industry,” *Column Generation*, pp. 163–195, 2005.

[63] B. G. Thengvall, J. F. Bard, and G. Yu, “Balancing user preferences for aircraft schedule recovery during irregular operations,” *IIE Transactions (Institute of Industrial Engineers)*, vol. 32, no. 3, pp. 181–193, 2000.

[64] B. G. Thengvall, G. Yu, and J. F. Bard, "Multiple fleet aircraft schedule recovery following hub closures," *Transportation Research Part A: Policy and Practice*, vol. 35, no. 4, pp. 289–308, 2001.

[65] G. Stojković, F. Soumis, J. Desrosiers, and M. M. Solomon, "An optimization model for a real-time flight scheduling problem," *Transportation Research Part A: Policy and Practice*, vol. 36, no. 9, pp. 779–788, 2002.

[66] G. Yu, M. Argüello, G. Song, S. M. McCowan, and A. White, "A new era for crew recovery at Continental Airlines," *Interfaces*, vol. 33, no. 1, pp. 5–22, 2003.

[67] S. Bratu and C. Barnhart, "Flight operations recovery: New approaches considering passenger recovery," *Journal of Scheduling*, vol. 9, no. 3, pp. 279–298, 2006.

[68] J. Vink, B. F. Santos, W. J. Verhagen, I Medeiros, *et al.*, "Dynamic aircraft recovery problem-an operational decision support framework," *Computers & Operations Research*, vol. 117, p. 104 892, 2020.

[69] S. J. Maher, "A novel passenger recovery approach for the integrated airline recovery problem," *Computers & Operations Research*, vol. 57, pp. 123–137, 2015.

[70] S. J. Maher, "Solving the integrated airline recovery problem using column-and-row generation," *Transportation Science*, vol. 50, no. 1, pp. 216–239, 2016.

[71] N. Jafari and S. H. Zegordi, "Simultaneous recovery model for aircraft and passengers," *Journal of the Franklin Institute*, vol. 348, no. 7, pp. 1638–1655, 2011.

[72] N. Jozefowiez, C. Mancel, and F. Mora-Camino, "A heuristic approach based on shortest path problems for integrated flight, aircraft, and passenger rescheduling under disruptions," *Journal of the Operational Research Society*, vol. 64, no. 3, pp. 384–395, 2013.

[73] Y. Hu, B. Xu, J. F. Bard, H. Chi, *et al.*, "Optimization of multi-fleet aircraft routing considering passenger transiting under airline disruption," *Computers & Industrial Engineering*, vol. 80, pp. 132–144, 2015.

[74] U. Arıkan, S. Gürel, and M. S. Aktürk, "Integrated aircraft and passenger recovery with cruise time controllability," *Annals of Operations Research*, vol. 236, no. 2, pp. 295–317, 2016.

[75] U. Arıkan, S. Gürel, and M. S. Aktürk, "Flight network-based approach for integrated airline recovery with cruise speed control," *Transportation Science*, vol. 51, no. 4, pp. 1259–1287, 2017.

[76] B. F. Santos, M. M. Wormer, T. A. Achola, and R. Curran, “Airline delay management problem with airport capacity constraints and priority decisions,” *Journal of Air Transport Management*, vol. 63, pp. 34–44, 2017.

[77] A. Cook, G. Tanner, and A. Lawes, “The hidden cost of airline unpunctuality,” *Journal of Transport Economics and Policy (JTEP)*, vol. 46, no. 2, pp. 157–173, 2012.

[78] K. Sinclair, J. F. Cordeau, and G. Laporte, “Improvements to a large neighborhood search heuristic for an integrated aircraft and passenger recovery problem,” *European Journal of Operational Research*, vol. 233, no. 1, pp. 234–245, 2014.

[79] K. Sinclair, J.-F. Cordeau, and G. Laporte, “A column generation post-optimization heuristic for the integrated aircraft and passenger recovery problem,” *Computers & Operations Research*, vol. 65, pp. 42–52, 2016.

[80] T. Yang and Y. Hu, “Considering passenger preferences in integrated postdisruption recoveries of aircraft and passengers,” *Mathematical Problems in Engineering*, vol. 2019, 2019.

[81] N. Jafari and S. Hessameddin Zegordi, “Simultaneous recovery model for aircraft and passengers,” *Journal of the Franklin Institute*, vol. 348, no. 7, pp. 1638–1655, 2011.

[82] R. Acuna-Agost, P. Michelon, D. Feillet, and S. Gueye, “SAPI: Statistical Analysis of Propagation of Incidents. A new approach for rescheduling trains after disruptions,” *European Journal of Operational Research*, vol. 215, no. 1, pp. 227–243, 2011.

[83] C Eggermont, M Firat, C Hurkens, and M Modelski, “Roadef 2009 challenge: Description of the tue solution method,” *Nancy, France: ROADEF*, 2009.

[84] L. A. McCarty and A. E. Cohn, “Preemptive rerouting of airline passengers under uncertain delays,” *Computers and Operations Research*, vol. 90, pp. 1–11, 2018.

[85] M. Hotz, T. Chondrogiannis, L. Wörteler, and M. Grossniklaus, “Online landmark-based batch processing of shortest path queries,” in *33rd International Conference on Scientific and Statistical Database Management*, 2021, pp. 133–144.

[86] M. Zhang, L. Li, W. Hua, and X. Zhou, “Batch processing of shortest path queries in road networks,” in *Australasian Database Conference*, Springer, 2019, pp. 3–16.

[87] C. Artigues, E. Bourreau, H. M. Afsar, O. Briant, and M. Boudia, “Disruption management for commercial airlines: Methods and results for the ROADEF 2009 Challenge,” *European Journal of Industrial Engineering*, vol. 6, no. 6, pp. 669–689, 2012.

[88] T. Hartman, A. Hassidim, H. Kaplan, D. Raz, and M. Segalov, “How to split a flow?” In *2012 Proceedings IEEE INFOCOM*, IEEE, 2012, pp. 828–836.

[89] K. Sinclair, J. F. Cordeau, and G. Laporte, "A column generation post-optimization heuristic for the integrated aircraft and passenger recovery problem," *Computers and Operations Research*, vol. 65, pp. 42–52, 2016.

[90] S. M. Ananthanarayanan and C. Kroer, *Computing the optimal distributionally-robust strategy to commit to*, 2022.

[91] H. Von Stackelberg, *Marktform und gleichgewicht*. J. springer, 1934.

[92] R. Yang, B. J. Ford, M. Tambe, and A. Lemieux, "Adaptive resource allocation for wildlife protection against illegal poachers.," in *Aamas*, 2014, pp. 453–460.

[93] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 895–902.

[94] A. Shapiro, "Tutorial on risk neutral, distributionally robust and risk averse multistage stochastic programming," *European Journal of Operational Research*, pp. 1–31, 2020.

[95] C. Kiekintveld, M. Tambe, and J. Marecki, "Robust bayesian methods for stackelberg security games," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 2010, pp. 1467–1468.

[96] C. Kiekintveld, T. Islam, and V. Kreinovich, "Security games with interval uncertainty," *12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013*, vol. 1, pp. 231–238, 2013.

[97] T. H. Nguyen, A. Yadav, B. An, M. Tambe, and C. Boutilier, "Regret-based optimization and preference elicitation for stackelberg security games with uncertainty.," in *AAAI*, 2014, pp. 756–762.

[98] T. H. Nguyen *et al.*, "Making the most of our regrets: Regret-based solutions to handle payoff uncertainty and elicitation in green security games," in *International Conference on Decision and Game Theory for Security*, Springer, 2015, pp. 170–191.

[99] R. Gao and A. Kleywegt, "Distributionally robust stochastic optimization with wasserstein distance," *Mathematics of Operations Research*, 2022.

[100] H. Rahimian and S. Mehrotra, "Distributionally robust optimization: A review," *arXiv preprint arXiv:1908.05659*, 2019.

[101] Y. Liu, H. Xu, S.-J. S. Yang, and J. Zhang, "Distributionally robust equilibrium for continuous games: Nash and stackelberg models," *European Journal of Operational Research*, vol. 265, no. 2, pp. 631–643, 2018.

[102] B. Von Stengel and S. Zamir, "Leadership with commitment to mixed strategies," Citeseer, Tech. Rep., 2004.

[103] Q. Guo, J. Gan, F. Fang, L. Tran-Thanh, M. Tambe, and B. An, "On the inducibility of stackelberg equilibrium for security games," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2020–2028.

[104] C. D. Aliprantis and K. C. Border, *Infinite Dimensional Analysis A Hitchhiker's Guide, 3rd edition*. Springer, 2006.

[105] N. L. Carothers, *Real analysis*. Cambridge University Press, 2000.

[106] E. Delage and Y. Ye, "Distributionally robust optimization under moment uncertainty with application to data-driven problems," *Operations research*, vol. 58, no. 3, pp. 595–612, 2010.

[107] P. M. Esfahani and D. Kuhn, "Data-driven distributionally robust optimization using the wasserstein metric: Performance guarantees and tractable reformulations," *Mathematical Programming*, vol. 171, no. 1-2, pp. 115–166, 2018.

[108] F. Luo and S. Mehrotra, "Decomposition algorithm for distributionally robust optimization using wasserstein metric with an application to a class of regression models," *European Journal of Operational Research*, vol. 278, no. 1, pp. 20–35, 2019.

[109] E. Del Barrio, E. Giné, and C. Matrán, "Central limit theorems for the wasserstein distance between the empirical and the true distributions," *Annals of Probability*, pp. 1009–1071, 1999.

[110] N. Fournier and A. Guillin, "On the rate of convergence in wasserstein distance of the empirical measure," *Probability Theory and Related Fields*, vol. 162, no. 3, pp. 707–738, 2015.

[111] J. Blanchet and K. Murthy, "Quantifying distributional model risk via optimal transport," *Mathematics of Operations Research*, vol. 44, no. 2, pp. 565–600, 2019.

[112] C. Villani, *Optimal transport: old and new*. Springer, 2009, vol. 338.

[113] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown, "Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms," in *AAMAS*, vol. 4, 2004, pp. 880–887.

[114] R. Avenhaus, B. Von Stengel, and S. Zamir, “Inspection games,” *Handbook of game theory with economic applications*, vol. 3, pp. 1947–1987, 2002.

[115] A. A. Cournot, *Researches into the Mathematical Principles of the Theory of Wealth.* Macmillan, 1897.