

MODELS, ALGORITHMS, AND IMPLEMENTATIONS
FOR OPERATIONAL OPTIMIZATION OF RIDEPOOL
SERVICES

A Dissertation

Presented to the Faculty of the Graduate School
of Cornell University

in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by

Matthew Dean Notz Zalesak

August 2021

© 2021 Matthew Dean Notz Zalesak

ALL RIGHTS RESERVED

MODELS, ALGORITHMS, AND IMPLEMENTATIONS FOR OPERATIONAL
OPTIMIZATION OF RIDEPOOL SERVICES

Matthew Dean Notz Zalesak, Ph.D.

Cornell University 2021

Ridepooling/microtransit are special types of ridehailing that benefit from reduced pollution and lower costs to users by placing multiple unrelated customers that share similar itineraries in the same vehicle. This dissertation studies problems related to ridepooling from several angles, from algorithmic, numerical, theoretical, to real-world practical studies. The bulk of the work relates to Request-Trip-Vehicle decompositions for ridepool assignments problems, a recent technique that has gained popularity because it preserves optimal solutions, provides large flexibility to adapt to a variety of constraints, and because of good practical performance. In this dissertation, we start by presenting an algorithm for the operation of high-capacity electric vehicle fleets in ridepool systems where the vehicles need to recharge during the day. We show that our algorithm outperforms a naive approach with respect to the service rate metric and discuss how the results reinforce the importance of having demand forecasts in ridepool settings. Next, we present some novel formulations of the ridepool assignment problem based on inexact models. These formulations, as well as some from the literature, are benchmarked on New York City taxi data according to the service rate metric. After observing that the service rate is very similar for all of the algorithms we then demonstrate evidence that there may be a service rate barrier that many algorithms are hitting. We hypothesize that in order to break this barrier

algorithms must be adapted to take into account demand forecasts. On the front of microtransit, we first study a method for developing approximation algorithms for tail-risk minimization problems motivated by routing shared vehicles to maximize the probability of successful transfers to other transit. Then more concretely, we discuss work done in collaboration with transit agencies in Seattle and Minneapolis-St. Paul, using tools and algorithms we developed to perform a data-driven design of microtransit feeder services, with the first pilots slated for deployment in fall 2021. Finally, as a timely contribution we present a model and efficient implementation for studying infectious diseases on university campuses.

BIOGRAPHICAL SKETCH

Matthew Zalesak was born on April 1, 1993 in Durham, North Carolina. He has been interested in transportation since he was four years old, first being interested in trains, then planes, and finally in this work automobiles. He did his undergraduate studies at North Carolina State University majoring in Industrial and Systems Engineering with minor studies in Mathematics, Statistics, and Linguistics. He graduated with his bachelors in May of 2015.

He joined the School of Operations Research and Information Engineering at Cornell University in August of 2015. His research interests generally center around the operations of transportation systems. He has been working with his advisor, Samitha Samaranyake, since 2017 and since then has particularly focused in on problems in the shared mobility space. He got his M.S. degree in 2019.

Outside his academic life, his hobbies include music, playing piano and bassoon, baking, as well as running and playing squash. He was president of the Aikido Club at Cornell from 2019 through 2021. Upon completion of his degree, he will join Wayfair as an Operation Research Scientist.

This document is dedicated to my parents Rudy Zalesak and Lisa Notz, my brother Andrew Zalesak, and the memory of my older brother Christopher Zalesak.

ACKNOWLEDGEMENTS

I would like to start by giving a big thanks to my advisor Samitha Samaranayake. It has been nothing short of a true pleasure working with him over the past several years. He has provided me with much support both in starting my research career and in helping me realize my personal strengths.

In addition, I would like to thank my minor advisors David Shmoys and Sid Banerjee. I would like to give a special thanks to David Shmoys for bringing me onto a large project to help support the university in scheduling the Fall 2020 semester roster amidst the COVID-19 pandemic. It meant a lot to me to be able to make such a large contribution to an institution that had changed my life so much.

In my early years, I also received advising from Jamol Pender and Madeleine Udell who I would like to thank for helping me set my course in graduate school.

I would like to thank one of my close friends, Sam Gutekunst, whom I spent lots of time with over the past several years. During my time as PhD student not only did he support me as a friend, but he also gave me guidance in my academic life and job searching.

I must also give a particular thanks to Pat Steele for the way he inspired and influenced my programming style and skills and the way I think about algorithms.

There are numerous other students that have been great friends to me while I've been here at Cornell that I would like to thank. Just to name a few, Andrew Daw, Pamela Badian-Pessot, Carlos Martinez Mori, Ben Grimmer, Alyf Janmohammd, Woohyung Cho, Chamsi Hssaine, and Anders Wikum.

This work was partially supported by the National Science Foundation (grant

number 1839346), the United States Department of Energy (grant number DE-EE0008464 subaward RQ19-119R06), and United States Department of Transportation (grant 69A3551747119). The contents of this report reflect the views of the author, who is responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

TABLE OF CONTENTS

Biographical Sketch	iii
Dedication	iv
Acknowledgements	v
Table of Contents	vii
List of Tables	xi
List of Figures	xiv
1 Introduction	1
2 Real Time Operation of High-Capacity Electric Vehicle Ridesharing Fleets	11
2.1 Introduction	11
2.1.1 Literature Review	14
2.2 Preliminaries	19
2.2.1 Terminology	19
2.2.2 The Problem	20
2.2.3 Trip-Oriented ridepooling Framework	23
2.2.4 Extending Models for Electric Vehicles	25
2.2.5 Battery Model	27
2.3 Methods	31
2.3.1 Charge Scheduler Problem Formulation	33
2.3.2 Two Stage Planning Process	36
2.3.3 Long Horizon Planning	40
2.3.4 Short Horizon Planning	44
2.3.5 Heuristic Long Horizon Planning	48
2.4 Numerical Results	55
2.4.1 Charge Station Location	57
2.4.2 Benchmark Method	57
2.4.3 Producing Availability Requirements $R(t)$	58
2.4.4 Fleet Size and Reduced-Size Map for MILP Method Experiments	61
2.4.5 Battery Range and Expected Charge Duration	62
2.4.6 Data Processing and Simulator Implementation	63
2.4.7 Results	64
2.4.8 Sensitivity Analysis	72
2.5 Conclusions	79
2.6 Acknowledgements	82

3	Centrally Controlled Ridepool Algorithms: Efficiency and Performance	83
3.1	Introduction	83
3.2	Literature Review	89
3.3	Preliminaries	91
3.3.1	Problem Description	91
3.3.2	Common Concepts and Objectives	93
3.4	Formulations and Assignment Algorithms	96
3.4.1	Exact Formulations	97
3.4.2	Linear Assignment based methods	107
3.5	Stability of Solutions	120
3.5.1	Computing the Constrained TSP Subproblem	120
3.5.2	Notes on Stability in Rebalancing	126
3.6	Testing Framework	127
3.7	Numerical Comparisons of Assignment Methods	129
3.7.1	Comparison of Methods	130
3.7.2	Offline Analysis of Methods	132
3.7.3	Why do many methods perform similarly?	134
3.7.4	Alternate Primary Objectives: Revenue Miles Traveled	139
3.8	Value of Future Information and the cost of being Myopic	140
3.9	Conclusions	145
4	A Note on Transformations of Approximation Algorithms for Robust Combinatorial Optimization	148
4.1	Introduction	148
4.2	Approximating Robust Combinatorial Problems	150
4.2.1	Robust Combinatorial Problems	150
4.2.2	General procedure of Nikolova for Approximating Robust Combinatorial Problems	152
4.3	Limitations of the Direct Oracle Method	154
4.3.1	Normal distribution case	155
4.3.2	General case with Chebyshev bounds	157
4.4	An alternative approximation	160
4.4.1	Applicability of the bicriterion-budget constraint	162
4.4.2	Approximation guarantee	167
4.4.3	Issues in selecting γ	169
4.4.4	Some Geometric Intuition	172
4.5	Conclusions	175

5	The Benefits of Simulation-Intensive Microtransit Planning: Two Case Studies	177
5.1	Introduction and Motivation	177
5.2	Literature Review	180
5.3	Service Design Process	183
5.3.1	Demand Modeling	183
5.3.2	Policy Exploration	185
5.3.3	Communication and Dialogue Process	188
5.4	Model Background	190
5.4.1	General Ridepooling Framework	192
5.4.2	Modification for Microtransit Systems	196
5.5	Results	198
5.5.1	Initial sites in King County.	198
5.5.2	Site revision.	201
5.5.3	Demand modeling for North Kent.	204
5.5.4	Initial experiments for Minneapolis	210
5.5.5	Second round experiments for Minneapolis	216
5.5.6	Final round experiments for Minneapolis	219
5.5.7	Final round of experiments for King County.	221
5.6	Conclusions	226
6	SEIR-Campus: Modeling Infectious Diseases on University Campuses	229
6.1	Introduction	229
6.2	Epidemiological Model	233
6.2.1	Generalizations and possible future extensions	236
6.3	Model Details	237
6.3.1	States	238
6.3.2	Actions	238
6.3.3	Internal Structure	241
6.3.4	Order of Operations	242
6.3.5	Processing Daily Infections	242
6.3.6	Time Considerations	244
6.4	Usages Examples	246
6.4.1	Basic Usage	246
6.4.2	Asymptomatic and Symptomatic Individuals	252
6.4.3	Testing	254
6.4.4	Contact Tracing	259

6.4.5	Exploring Alternative Course Designs	261
6.4.6	Social Groups	264
6.4.7	Social meetings from groups.	268
6.4.8	Overview of other Parameters settings	273
6.5	Conclusions	274
A	Appendix for Chapter 1	276
A.1	Battery Section	276
A.1.1	Charging.	276
B	Appendix for Chapter 3	282
B.1	CTSP Heuristics	282
B.1.1	Fixing order of on-board passengers	283
B.1.2	Limit and recall prefix	284
	Bibliography	286

LIST OF TABLES

2.1	Description and abbreviations for statistics reported from experiment.	66
2.2	Simulation results for 220 Vehicles. The MILP and Heuristic methods had significantly higher services rates than the benchmark. The ICE baseline has no charging and is an upper bound on service rate performance.	66
2.3	Results for simulations with 1000 Vehicles. Compared to the benchmark, the heuristic method mitigated the loss in service rate compared to the upper bounding ICE baseline.	68
2.4	Results for simulations with 2000 Vehicles. As with the experiments with fewer vehicles, the heuristic method has a service rate much closer to the upper bounding ICE than the benchmark algorithm. . .	69
2.5	Vehicle simulations from midnight. When experiments start at midnight, the low demand in the early morning causes the distributions of charges in vehicles to spread. As a result, the service rate of the benchmark method is similar to that of the heuristic method.	71
2.6	Average Per Iteration Computation Time. Computation time increases with number of vehicles. MILP methods take a significant amount of time. The heuristic, benchmark, and ICE methods use roughly the same time. The changes in time are primarily driven by changes in the length of time required to solve the passenger assignment problem. This is affected by the number of vehicles that are online and available at different times throughout the day and the relation of that to the number of requests that need to be assigned. .	72
2.7	220 Vehicle Simulations for Availability Sensitivity. Even in the worst case with an inverted schedule, the service rate is significantly higher than the benchmark method, at 82.6% (see Table 2.2).	73
2.8	1000 Vehicle Simulations for Availability Sensitivity. While generating $R(t)$ using an inverted schedule yields the lowest service rate, it still only loses 0.6% service rate compared to the typical model we developed in section 2.4.3, much smaller than the 2.5-3% gap between all of these results and the benchmark method.	73
2.9	Vehicle Simulations Across Various Battery Ranges. Longer battery range typically resulted in better service rate since vehicles spent less time on average offline. The benchmark method did not strictly follow this trend, however, since it is also sensitive to the demand patterns at the time of day vehicles typically need to begin charging.	74

2.10	Analysis of Heuristic Method at Various Estimated Battery Lives. Estimated battery life has a weak effect on service rate.	75
2.11	Results for 220 vehicles when charge stations are placed using a greedy algorithm. The benchmark method increased in performance by about 0.6%, but the heuristic and MILP methods are largely unchanged. See Table 2.2 to compare these results with the default charge station layout.	76
2.12	Results for 1000 vehicle when charge stations are placed using a greedy algorithm. Both the heuristic and benchmark methods have lower service rate when compared to the default charge station layout, shown in Table 2.3, with the benchmark method impacted slightly more.	76
2.13	1000 Vehicle Simulations with Reduced Station Count. Reducing the number of charging stations has a larger impact on the service rate of the benchmark algorithm than the heuristic method.	77
2.14	Comparison of Benchmark Method Maximum Charge Station Travel Time. Allowing the greedy station assignment algorithm to assign vehicles to stations nearer or farther away did not change the service rate enough to be competitive with the heuristic method. See Tables 2.2 and 2.3.	77
2.15	Results for 220 vehicles when capacity is reduced to 4. Compare to Table 2.2 which shows the same simulations when vehicle capacity is 10. The ICE method suffers little impact from the change in capacity, but the other methods lose around a percentage point of service rate.	78
2.16	Results for 1000 vehicles when capacity is reduced to 4. Compare to Table 2.3 which shows the same simulations when vehicle capacity is 10. Roughly all methods in this case were negatively impacted by the lower capacity, typically losing about two and a half percentage points of service rate.	78
2.17	Results for 2000 vehicles when capacity is reduced to 4. Compare to Table 2.4 which shows the same simulations when vehicle capacity is 10. All methods were negatively impacted by the lower capacity, with the ICE method taking the small hit to service rate with a 3.4% loss, and the benchmark method taking the worst hit with a 4.5% loss in service rate.	79
3.1	Service rate comparison for various demand levels under each method. Note there are only half as many vehicles used in experiments with quarter-level demand.	130

3.2	Computation time (minutes) on initial runs for each method. The results are only approximate since some of the trials were run in parallel, potentially negatively impacting their runtime. Note there are only half as many vehicles used in experiments with quarter-level demand.	130
3.3	Average rejections and VMT across full day for each method under identical inputs.	133
3.4	Comparison of service rates, revenue miles traveled, and vehicle used by method and objective type.	140
5.1	Service rates.	201
5.2	Service rates (percent of requests served) for the nested service zones with radius of one, two, three, and four miles. While having additional vehicles is better, it is a struggle to achieve good service rates in the largest service area.	201

LIST OF FIGURES

2.1	Flow of each iteration. First new requests arrive. Constrained on the existing charge schedule, passenger assignments are made. Finally, conditioned on the new passenger assignments the charging schedule is updated.	26
2.2	In the long horizon, the schedule more than T_{SL} time in the future can be modified. The short horizon station problem only assigns vehicles that are scheduled to charge within $T_{SL} + \Delta$ of current time. Δ can be 0, but theorem 2.3.1 shows it cannot be too large.	36
2.3	Benchmark method for 220 vehicles. Notice that vehicle charging, in blue, is concentrated during a single period in late afternoon. The number of vehicle miles traveled during this time, in orange, plummets. The 30 minute service rate during the period, shown in red, also dips resulting in a reduction of the total cumulative service rate.	67
2.4	Heuristic method on 220 vehicles. The fractional availability used, shown in blue, is defined to include both charging and the 15 minute period prior to charging in which it is assumed that vehicles lack full ability to serve customers due to constraints on reaching the charging station. The maximum allowed offline availability, shown in yellow, indicates the limit by time of day of unavailable vehicles.	68
2.5	Benchmark method on 1000 vehicles. This shows that the the vehicles all needed to charge at a similar time. Some of the effect is hidden since only charging vehicles are counted in blue. A larger number of vehicles are offline waiting to be charged during that time, only visible through the great reduction in vehicle miles traveled shown in orange.	69
2.6	Heuristic method on 1000 vehicles. This shows that the times the vehicles charged at were better spread, as evidenced by the small reduction in vehicles miles shown in orange.	70
2.7	Benchmark method on 1000 vehicles starting at midnight. By coincidence it turns out that the vehicles all finish charging before the day's peak demand, a fortunate gain for this particular run.	71
3.1	Example demonstrating instability of insertion CTSP heuristic.	125
3.2	Number of requests served per iteration, plotted in pairs for fixed lag durations. LA method shown in blue, Full-RTV method shown in orange. Regression lines shown over plot.	135

3.3	Slope of regression line for lagging pairs of number of requests served per iteration, varied by the duration of the lag between the pairs. . .	137
3.4	Steady state service rate when there are 180 requests per minute. In each iteration, the Full-RTV method is restricted to assign at most x requests, for x given on the x -axis.	138
3.5	With demand reduced by a factor of 16, and with 125 vehicles, the service rate is shown when we have different visibility into the future.	142
3.6	With demand reduced by a factor of 16, and with 125 vehicles, the service rate is shown when we have noisy information about the future.	143
4.1	Visualization of the geometric approach of Nikolova. On the left, the feasible region is shown projected on the mean-variance plane, $\mu^T x$ and $\tau^T x$ respectively. The optimal solution can be found by solving for the linear objective shown in the center of the feasible region P and the optimal level set λ^* is shown. On the right, an approximate linear oracle is used with the same linear objective. The set of possible solutions is shaded in gray, bounded by a line parallel to the tangent line at the optimal solution resulting from the δ -approximation guarantee of the oracle. The worst case level set is λ	153
4.2	Visualization of the bicriterion method. On the left, the feasible region is shown projected on the the mean and variance plane, $\mu^T x$ and $\tau^T x$ respectively. The optimal solution can be found by solving for the linear objective shown in the center of the feasible region and the optimal level set λ^* is shown. On the right, an approximate oracle along with the bicriterion approximator of [47] is used. The set of possible solutions is shaded in gray, bounded on the right by the budget $(1 + \gamma)\delta m^*$ and above by the standard tangent line for a δ -approximation oracle. The worst case level set is λ	163
4.3	Approximation factors for the newly proposed algorithm (blue) and the algorithm proposed by Nikolova (orange), both using $\xi = (t - m^*)/2t$. The key feature of the new algorithm is that it continues to provide an approximation guarantee even as δ grows large, though it is sometimes outperformed by Nikolova's algorithm for smaller δ . . .	170
4.4	Comparing the approximation guarantee when the mean value of the optimal solution is not known. The figures compare perfect knowledge (orange) with a lower bound on the guarantee given when we use $m = 0.8m^*$	171

4.5	Geometric demonstration for bicriteria method (left) and simple linear oracle (right). In this case the mean value is large, with $m^* = 0.5t$ and $\gamma = 0.14$. The regions of possible returned solutions are shaded in gray.	173
4.6	Geometric demonstration for bicriteria method (left) and simple linear oracle (right). In this case the mean has an intermediate value, with $m^* = 0.17t$ and $\gamma = 1$. The regions of possible returned solutions are shaded in gray.	174
4.7	Geometric demonstration for bicriteria method (left) and simple linear oracle (right). In this case the mean value is small, with $m^* = 0.002t$ and $\gamma = 25$. The regions of possible returned solutions are shaded in gray.	175
5.1	An illustration of comparing alternative scenarios. This y -axis of this plot shows the percentage of customers served in a variety of experiments. Each color line represents a different scenario. The x -axis represents the average daily demand so the difference in each scenario can be understood in the face of uncertainty about the actual demand level.	187
5.2	The initial service area was drawn from OpenStreetMap. The service centered on Kent railway station. A simple technique to create demand regions was chosen by forming concentric rings around the station and defining a demand intensity within each one.	200
5.3	Service area north of Kent. The region can be partitioned into residential areas, highlighted in orange, and industrial areas, highlighted in blue. The railway station is near the bottom left. Credit Andrea Broaddus.	203
5.4	Number of transit riders by time of day, split into residential demand (blue) and industrial demand (orange). Note that the AM and PM peaks are each three hours long while the midday period is seven hours long. This means that demand intensity during midday is significantly lower than during peak hours.	206
5.5	Break down of relative demand by region, comparing full residential option (left) and the smaller residential option (right).	206
5.6	Directional trend of demand by time of day. Residential demand shown in blue, industrial demand shown in orange.	207

5.7	While increasing the number of vehicle makes a considerable improvement in service rate (left), it has little effect on the total delay time for passenger (right).	208
5.8	Comparison of delay constraints. 30 minute delay performed best, followed by door-to-door + 15 minutes delay and door-to-door delay. The ordering is not surprising. The constraints that effectively allow more delay time perform better.	209
5.9	Rejection patterns for two delay constraints. While the distribution of rejections is fairly uniform in the 30 minute delay model (left), in the case of 2x door-to-door time (right) the density of rejections is higher in the lower portion of service area, corresponding with locality to the hub. In many ways this is expected since the delay constraints would require very fast service in those areas.	210
5.10	Two possible service area options proposed for the Minneapolis pilot after initial site selection. The smaller service option (left) is contained within the larger service option (right).	212
5.11	Comparing the larger (blue) and smaller (orange) service area option. The smaller service area may be performing better at high demand levels since the shorter average distances make matches between requests and vehicle more likely.	214
5.12	Evaluations of capacity vs fleet size indicate that fleet size is more important. In the left figure, the 5-vehicle fleet (orange) does not suffer from degraded service as demand increases anywhere near as much as the 3-vehicle fleet (blue) does. In the case of vehicle capacity, shown in the right figure, a capacity-6 vehicle (orange) offers only a small benefit over a capacity-3 vehicle (blue) option.	214
5.13	Service rate of the system under various delay constraints. The best was the Fixed Detour option (red) which overlapped the Hybrid option (blue). The Hybrid option turned out to be equivalent to the Fixed Detour option.	215
5.14	Illustration of the Minneapolis service area, with orange dots indicating intersections requests can be made from. Two blue triangles show the location of the two fixed route stations considered to be hubs.	217
5.15	Map of Minneapolis service area, showing roads and intersections. Three blue triangles indicate the location of three fixed-route stations under consideration for the final round of experiments.	220

5.16	Evaluation of adding a third station. While the service rate (left) does not noticeably change, the in-vehicle time (right) is reduced for the three station option. This is likely since the average distance between hub and destination is lower when there are more hubs to choose from.	220
5.17	Effect of rejection constraints. The average first-mile waiting time grows substantially when there are no rejections (orange) compared to rejections (blue). The in-vehicle time decreases when there are no rejections, attributable to more efficient assignments.	222
5.18	Customer behavior. Alice and Bob want to connect to a 10am bus departure. If Alice requests 10-20 minutes in advance of the bus departure she is less likely to be served than Bob, who requests 15-25 minutes in advance, since there are fewer options to serve tighter connections than looser ones. When generating demand profiles, we must consider whether customers are more like Alice or more like Bob.	224
5.19	Service rate by how far in advance of a connection requests are made. Generally, booking earlier is better.	225
6.1	State flow chart. Susceptible individuals (S) may be quarantined (Q), but then return to being susceptible afterwards. When infected, susceptible individuals become exposed (E). An exposed individual eventually becomes infectious, either asymptomatic (I_a) or symptomatic (I_s). Infectious individuals, if not subject to any interventions, recover and enter the removed state (R). Exposed (E) or infectious (I_a, I_s) individuals may be quarantined in (Q_e), (Q_a), or (Q_s) respectively. Since these individuals are infected before they enter quarantine, they only leave once they have recovered and are moved to the removed state (R).	239
6.2	Results from running a basic simulation. Each simulation outputs three charts. The first shows the number of susceptible, exposed, infectious, recovered, and quarantined individuals by date. The second shows the percentiles for the number of exposed individuals by date across all repetitions. The final chart shows the percentiles for the number of infectious individuals by date across all repetitions.	252
6.3	The three output charts for the basic simulation with the modification that none of the individuals report/have symptoms.	255

6.4	The three output charts for the basic simulation with the modification that each weekday, $1/5$ of the student body is tested for infection. Those that test positive are placed in quarantine the next day when the result comes back.	258
6.5	The three output charts for the basic simulation with the modification that when individuals show symptoms and take a test with a positive result, everyone they have been near in the past three days is placed in quarantine.	261
6.6	The three output charts for the basic simulation with the modification that courses are split into two groups, each groups alternating each week between meeting in-person and meeting online.	264
6.7	The three output charts from the basic simulation with the modification that social groups have been added via random social clusters. These groups meet for 2 hours on weekdays and 3 hours on weekends.	268
6.8	The three output charts for the basic simulation with the modification that 25% of individuals have been randomly assigned to be pairs that meet for the equivalent of 20 hours each day, which accounts for hypothesized lack of social distance and not wearing face masks. . . .	269
6.9	The three output charts for the basic simulation with the modification that members of varsity teams spend time with each other each day, with clusters changing within each team each day.	271
6.10	The three output charts for the basic simulation with the modification that members of varsity teams spend time with each other each day. In this version, clusters do not change over the course of the semester.	272
A.1	Simple model of charge in an initially empty battery as a function of charging time. The curve has two components: in blue, the charge in the battery initially increases linearly until time ρ , at which point the battery has charge Q and charging continues as the orange curve which asymptotically approaches the battery's theoretical maximum charge at an exponentially decreasing rate. Denoted in green are the lines $x = \tau$ and $y = 1$. The asymptotic limit is shown as a red line. In this example, linear charging lasts for the first $\rho = 15$ minutes at which point the battery has charge $Q = 0.7$ and it takes $T = 30$ minutes for a full charge.	278

A.2 Assume a vehicle takes 30 minutes to fully charge and receives 70% of its charge from empty in the first 15 minutes. Suppose that on average the vehicle wastes 5 minutes of time traveling to the charge station while not holding any passengers, a full battery lasts 400 minutes, and that offline time costs the operator \$1 per minute. Figure (A.2a) plots $k(q')$ with $d = 5$ and Figure (A.2b) plots $k(q')$ with $d = 0$. Optimal value in general case is $q = 78.2\%$ and optimal value when $d = 0$ is any value less than Q 280

CHAPTER 1

INTRODUCTION

Innovations in technology and the prevalence of smart phones has created a revolution in on-demand transportation. With new mobile technologies allowing users to request rides anywhere at any time, new transportation options range from high-availability city-scale ridehail services to smaller, neighborhood scale services connecting with fixed route transit options. Users are attracted to these services as they reduce individual dependence on vehicles and parking, increase options for those with limited mobility, and abate gaps of fixed route public transit coverage. While similar in concept to a traditional taxi, these services are distinguished by the central control afforded to the dispatcher using the new mobile communications technology as well as the ability to leverage new algorithms to efficiently create shared rides, which offer benefits both in cost and in environmental impact.

There are several forms these services can take on. Some of the most well known existing on-demand services run by centrally matching two sides of the market. In this setting, drivers and riders each separately join and leave the platform and the dispatcher's task is to pair riders with drivers and to determine fair cost allocation. Not all systems operate like this, however. The services studied in this dissertation run one-sided markets where drivers are regularly scheduled employees, a setup common in government sponsored systems and a natural choice for services run by autonomous vehicles.

A common element appearing in both types of systems is sharing, where multiple unrelated riders travel in the same vehicle at the same time. This gives rise to some important terminology which will be used in this dissertation, though it is important to note that this terminology is not universally standardized. In the most general sense, *ridesharing* refers to systems that provide transportation services to passengers riding in vehicles that they do not own. When we use the related term *ridehailing*, we are emphasizing the fact that the driver is working rather than traveling, as might be the case in an organized carpooling system. The term *ridepooling* draws attention to situations where multiple riders simultaneously ride in the same vehicle. This last term, ridepooling, drives the central motivation of this dissertation.

The advantages ridepooling offers over traditional taxis come from the efficiency afforded by serving multiple riders simultaneously. To see this, first consider the impact of single-occupancy ridehail services compared to the use of individual vehicle usage. Suppose that all travelers in an area use their own personal vehicle to get from an origin to a destination of choice. To serve that same group of people using a single-occupancy ridehail service the system must drive at least as many miles, assuming people drive on the shortest path, and additionally must add extra miles for when a vehicle moves from the destination of one passenger to the origin of another. So while the total number of vehicles required may have decreased from an individual usage scenario, the total miles traveled, and accordingly the associated pollution and environmental impact, can actually rise. This is why ridepool systems are of growing interest.

In the field of operations research, ridepool systems are of interest because of the many mathematical and algorithmic challenges associated with on-demand operation. Challenges include matching potential riders to vehicles, determining routes for the vehicles, modeling travel times on road networks and predicting traffic, and forecasting customer demand, among others. This dissertation focuses on several of these topics, ranging from theoretical coverage all the way down to real-world applications showing how these algorithms and models can be used to design transit projects in data-drive manners.

This work is divided into five chapters, with the bulk of the work focusing on algorithms for on-demand operations of centrally controlled on-demand fleets of vehicles. The first chapter will show how to exploit a decomposition of the vehicle routing problem to control large-scale fleets of electric vehicles. This is followed by a chapter that gives a more in depth study of the assignment process itself. Then a detour will be taken to address a mathematical question motivated by work on routing problems where robust variants of combinatorial problems are solved by systematically modifying existing tool for deterministic optimization. Next, a real world case study is presented where the tools and framework for on-demand systems is used to provide a data-driven approach to planning two pilot first/last mile microtransit systems. Finally, the work is rounded off by a small detour with a timely contribution to fighting the COVID-19 pandemic by creating an efficient simulation framework for modeling the spread of infectious diseases in university environments.

The algorithms studied in this work for on-demand operations all fall under the

Request-Vehicle-Trip (RTV) decomposition framework. To explain what this is, first consider the background of these problems. Centrally controlled, on-demand ridepooling systems have a fleet of vehicles V and a set of requests R that become visible in real-time. Each request is defined by an origin and destination and wishes to be served immediately upon becoming visible to the system, meaning there is no booking in advance. Often, making assignments in these systems is done in batches ranging from a few seconds to a minute depending on the application. The operator must then choose, from the set of all possible assignments, a solution that maximizes their objective. The Request-Vehicle-Trip decomposition splits this problem into two parts. First, it develops a candidate set of potential assignments by defining a trip $t = (v, \bar{r})$ as a vehicle $v \in V$ and a set of requests $\bar{r} \subseteq R$. A trip can be evaluated to determine whether it can be routed in a way that meets system constraints as well as the cost of the trip. Then, in a second step an optimization problem can be solved that chooses from the set of candidate trips the subset that yields the best value.

The first chapter of this dissertation studies the way the properties of this decomposition can be used to operate high-capacity fleets of electric vehicles in ridepooling systems. Research prior to this has generally either focused on the large-scale, high-capacity ridepooling aspect of the problem or studied systems with electric vehicles in the context of smaller scale and lower-capacity systems. To the best of our knowledge, this is the first work that focuses on the combination of both. The combination introduces the challenge of managing vehicles that must go offline during the day to charge since many larger electric vehicles on the market that are affordable have ranges that are significantly smaller than seen on the high-end personal vehi-

cle market. This chapter shows the versatility provided by the RTV decomposition framework for integrating the constraints of an electric fleet into real-time system management.

The reason the decomposition framework helps in this case is the flexible modularity of developing candidate trips in the assignment optimization problem. The solution we propose is iterative: at each iteration, first assign incoming requests to vehicle subject to respecting the charging schedule, then update the charging schedule given the new vehicle paths. While each of the optimizations is run in a separate step, it is important to note that there is significant interplay between the two problems in the form of constraints. The constraints for the charge scheduling problem are naturally developed in the chapter along with that portion of the algorithm since it is new. But as for the assignment process, the flexibility of the RTV decomposition comes from the separation of the trip selection problem from the candidate trip generation problem, which uses a blackbox function called the *travel* function. This function takes in a trip $t = (v, \bar{r})$ and searches for the best route that meets all constraints. Even when employing heuristics to evaluate this, it is a trivial addition to reject routes that fail to bring a vehicle to its scheduled charging station on time and with all passengers already dropped off.

The first chapter gives a formal introduction to the RTV framework as well as the concept for the iterative solution. It then gives an overview of the mathematical model used for batteries in the electric vehicle fleet. The solutions section proposes three approaches to scheduling charging: naive on-demand charging, an exact

scheduling model based on a mixed integer linear program (MIP), and a heuristic for generating schedules. All of the approaches are compared in numerical results. The results show that the decomposition in combination with either the heuristic schedule or exact, MIP generated schedule are able to reduce the loss of served passengers compared to a naive, on-demand charging approach.

The second chapter of this dissertation moves away from exploiting the decomposition framework for new capabilities and instead studies in depth the core of the assignment process itself. The goals of this chapter are twofold. First, the chapter aims to provide a detailed reference to those who may want to build real implementations of RTV ridepooling algorithms. It introduces the framework, gives detailed descriptions of several assignment algorithms, gives important remarks about solving the routing subproblems necessary to implement such systems, and comments on the gap in performance left for future research caused by the myopic nature of the assignment problem. Second, it explores the trade off between algorithm run-time and performance. Performance results for ridepool assignment algorithms are typically reported in the literature at the day level, and the most popular metric is the service rate. The service rate indicates the percentage of requests that were served, as opposed to the percentage that were rejected because a route could not be found to serve them that met all of the system's constraints.

The trade off between run-time and performance is not a new topic in itself. From the seminal work of [3] that uses full enumeration to lighter-weight solutions of [53] and [70], solutions have been proposed within the RTV framework that compete

by showing off their run-time and performance. The critical missing component is understanding why these algorithms are giving performance that they do and comparing them under fair and equal settings. As this question is explored in the chapter, we also introduce several new variants of assignment algorithm and benchmark them against solutions in the literature. In fact, we find a solution that gives nearly the same full day performance in our experiments than nearly all existing solutions while utilizing less run-time than full-enumeration style algorithms.

Moving deeper into the theory side, the third chapter studies certain types of robust combinatorial optimization problems. The motivation to study this comes from a proposed microtransit system pilot that seeks to match first mile riders with fixed route services, such as train or bus departures. As is natural in the RTV framework, there needs to be an algorithm that can find an optimal route for any trip. While a deterministic setting is generally satisfactory for the problems in research settings, in real world settings we must also consider the way that traffic patterns and other events can cause a vehicle to not reach certain targets on time. In particular, if a system promises a customer that they will be delivered to a fixed route service before it departs there is a significant reputation cost if the service fails to deliver the promised connection on time. Treating the travel times on the road network as random variables, the goal is now to find a route for the trip that maximizes the probability that the connection is made on time.

The question asked in the third chapter is actually a slight simplification of the real problem: given a set of nodes to visit, find a circuit that visits all of the nodes

whose probability of exceeding a certain time budget is minimized. This simplification eliminates a major difficulty in studying the proper routing problem caused by the possibility of time windows imposed for each request. When time windows enter the equation, it becomes NP-complete to determine whether a feasible route exists. The relaxation of this constraint allows us to analyze the problem in an approximation context. What we look for is an α -approximation for the probability maximization problem, a polynomial time algorithm that is guaranteed to find a solution whose probability of success is at least α times the optimal probability, for $0 \leq \alpha \leq 1$.

The third chapter builds off of the work of [51] who studies a general tool for creating approximation algorithms for this space of robust optimization problems by systematically modifying existing approximation algorithms for deterministic variants of the problem. In our case, the metric traveling salesman problem (metric TSP) is the deterministic analogue of our problem. However, analysis shown in the chapter reveals a weakness of the tool when the only known approximation algorithm for the deterministic problem has a poor approximation ratio. We propose an alternative way of analyzing the problem that avoids several of the weaknesses, though we cannot overcome all of them.

The fourth chapter goes in the opposite direction from the third chapter and instead looks at two case studies showing how ridepooling algorithms and the RTV framework can be used by transit agencies to design real-world pilot systems. Microtransit systems are not in and of themselves a new concept. Several such systems

have been run historically. The reason these particular case studies are important is because of the shift made to a flexible simulation testing framework that allowed for major decisions on system policy and design to be guided in data-driven manner. The chapter covers the entire process from site selection, gathering data, comparing alternative options for evaluation, and the communication and dialogue process between the the agencies and the research team on simulations and algorithms.

Finally, in chapter five we make a timely contribution to fighting the COVID-19 pandemic by creating an efficient simulation framework for modeling the spread of infectious diseases in university environments. COVID-19 took the world by surprise and changed the way people lived, greatly affecting the operations of every organization that relies on drawing large populations together, such as universities. The last year and a half of the work done in this dissertation was completed under the unusual and unprecedented reality brought on by fighting this disease. Early on in the pandemic, many researchers in technical areas were looking for ways to apply their skills to help with the pandemic response. One component of that response was giving the university tools and insights related to reopening for the 2020-2021 academic year. While the student body in general was considered low risk for severe complications from the disease, uncontrolled spread in the student population could affect faculty, staff, the local community, and members of the student body with higher individual risk.

The contribution in chapter five is a general tool that was developed to model and study the spread of infectious diseases, such as COVID-19, through an agent

based model. Stepping back for a moment, at a high level there are two related, but different, approaches to studying disease spread in a population. The first way is known as a compartment model. In this case, at each point in time the total number of people in the population who are susceptible, exposed, infectious, and recovered are known in the aggregate and a procedure is used to calculate how many people from each state transition to another state at each time. The second way, which we use here, is agent based modeling. Here, each member of the population is tracked individually. While this contains some extra overhead in storing the extra information, it is useful in this context because it allows us to use class roster based interaction graphs to determine how classes affect spread. This enables us to, among other things, evaluate how potential alternatives to a traditional class schedule can be used to mitigate and control the spread of infection.

Beyond just proposing a model, chapter five also presents a Python package that efficiently implements the model and can simulate infection spread in a university setting with nearly 20,000 agents in a matter of seconds. It also allows for easy study of several common interventions such as contact tracing, quarantine, and asymptomatic testing. These are all interventions that have commonly been used during the pandemic.

Each of the five chapters is derived from papers where the author of this dissertation was the primary author. They are included here with minor modification.

CHAPTER 2

**REAL TIME OPERATION OF HIGH-CAPACITY ELECTRIC
VEHICLE RIDESHARING FLEETS**

2.1 Introduction

Recent innovations in communications technology and mobile devices have enabled the emergence of large-scale ridesharing¹ services which provide on-demand, door-to-door transportation services. These services are attractive to users as they provide alternatives to the cost and hassle of finding parking, overcome the the problems of slow or unreliable transit options, and offer service when no other transit options are available [13]. These new services are possible as private operators begin to take advantage of new mobile technologies and methodologies to match riders and drivers together with higher speed and efficiency than has been done before. Cities also stand to benefit from ridehailing services with the hopes that they may act as an extension to public transportation, reduce congestion, and have a positive impact on pollution.

While private ridehailing services have made a debut in many countries around the world, in many cities these services have actually been found to contribute to higher congestion [18] and green house gas emissions [59]. One cause for this is asymmetry in demand patterns causing local mismatches between supply and demand,

¹We use the term *ridesharing* to refer to any system that allows riders to travel in vehicles not owned by themselves. For an explanation of terminology used in this chapter, refer to section 2.2.1.

leading many vehicles in ridehailing services to deadhead (also known as rebalancing) in order to relocate to other areas [27]. This process increases the number of vehicle miles traveled required to serve the traveling population compared to the same users driving in their own vehicles. A solution to this problem that has received increased attention recently is high-capacity ridepooling, where a large number of passengers can share a trip in a single vehicle in order to increase the system's efficiency (ratio of passenger miles traveled to vehicle miles traveled). An ever growing area of research, shared ridepooling has the potential to offer mass mobility as a complement to existing public transit options. As a consequence of serving riders using fewer vehicle miles traveled, shared ridepooling has the potential to reduce roadway congestion and reduce pressure in municipal parking areas [74].

In addition to using shared ridepooling as a tool to reduce pollution, total greenhouse gas emissions can be further reduced by operating ridepool systems with fleets of hybrid or electric vehicles rather than conventional internal combustion engine (ICE) vehicles, with most of the benefits of electric vehicles in particular depending on the availability of green energy sources [52]. Not only do electric vehicles help reduce smog and other pollutants in cities, empirically they have been found to be more energy efficient in urban settings due to the ability of regenerative braking to reclaim energy. With ongoing research on autonomous vehicles making large autonomous electric fleets a possibility, it is necessary that algorithms for ridepooling systems adapt to handle the new constraints introduced by electric fleets.

Past research on ridesharing systems has generally focused on either large-scale,

high-capacity ridepooling or smaller scale and lower-capacity ridehail systems with electric vehicles, but few have focused on both. Our contribution with this work is a model and accompanying methods for the control and operation of large-scale, high-capacity ridepool systems with a fleet comprised of electric vehicles ². Unlike models that have assumed the use of ICE vehicles, electric vehicle fleets must be able to go out of service to charge mid-day to maintain quality levels of service. Even as maximum battery ranges continue to grow in higher end cars, the actual number of miles that electric vehicles can travel can be significantly affected by outside factors such as temperature [34]. In addition, time dependent consumption of power from air and auxiliary systems mixed with slow travel in congested city roadways can negatively impact range [57].

The chapter is organized as follows. Section 2.1.1 reviews the literature on ride-hailing and ridepooling algorithms as well as studies concerning electric vehicles. Section 2.2 formally defines the setting of the problem and reviews an algorithm for ridepool management that we extend for electric vehicles. Section 2.3 presents the formulation of the charge scheduling subproblem and proposes two solution methods. Finally, section 2.4 presents the results of simulations of large-scale ridepool systems and compares the performance of the proposed solutions with benchmark and ICE vehicle baselines.

²Our approach can handle the case of mixed fleets by setting the battery capacity of ICE vehicles to be infinite or explicitly modeling refueling if necessary.

2.1.1 Literature Review

Literature on operational algorithms for high-capacity ridepooling and management of electric vehicles in ridepooling systems has largely been separate. Much of the work on the management of large-scale on-demand ridepooling assumes the use of conventional internal combustion engine (ICE) vehicles.

Many early ideas for large-scale ridepooling employed greedy assignment algorithms. This group of methodologies consider requests sequentially, usually in chronological order, and assigns them to vehicles in a way that minimizes some criteria subject to constraints. [45] presented T-Share, a framework that manages a real-time taxi system where multiple users can share a ride. The system sequentially handles incoming travel requests by matching each request with the vehicle that would have to travel the least additional distance to serve them. The focus is on how to efficiently compute this greedy assignment. In follow up work, [46] demonstrated simulations of the system with over 7000 taxis on a map in Beijing. To measure the performance of the system, they introduce the taxi-share rate and seat occupancy rate statistics, which other works use as well, to benchmark the performance gains of ride-sharing algorithms. The taxi-share rate is the percentage of requests that share a ride while the seat occupancy rate is the ratio of passenger-minutes to the total number of available seat-minutes in the system.

[53] proposed the STaRS simulation framework which, similar to T-share, incorporates quality of service requirements limiting the maximum amount of waiting

time for customers that are served as well as the total amount of delay that can be incurred due to sharing rides. Requests arrive into the system online and are processed sequentially, each request is greedily assigned to the best vehicle, and vehicles are not rebalanced at the end of each iteration.

Another line of methodologies instead rely on batching requests over a time period and processing requests simultaneously. Many of these methods aim for exhaustive searches over the space of vehicle and request matches during each batch. Starting this line of work, [60] introduced the notion of a shareability network. The shareability network is a graph that contains a node for every request in the system. If two requests are mutually shareable under quality of service requirement such as maximum waiting time and detour time, an edge is drawn between them. For groups of 3 or more requests, shareability can be represented by checking group feasibility and then creating hyperedges connecting the request nodes.

[3] built upon the concept of shareability network in a batch-optimization framework. In this framework, requests arrive to the system in batches. A shareability network is produced on the set of requests that arrived in the batch. Unlike the work of [60], it also includes nodes representing the vehicles in the shareability network. By modifying the definition of a trip to be a collection of passengers along with a vehicle, every feasible shareable trip forms a clique in the shareability network including exactly one vehicle. By fixing a particular vehicle and group of requests it is possible to compute optimal routing, costs, and revenues associated with various assignment and routing decisions. Using the results of the extended shareability net-

work, the set of trips and costs are used as inputs to integer linear program. The result of decomposing the problem into trip generation and selection preserves optimality in the solution up to computational limits that apply at each step. The results is a system that makes routing decisions that are optimal at all times given request currently in the system.

Other work proposes ideas that straddle the space between greedy assignment and batching. A variant proposed by [70] allows for significant speed gains at the cost of optimality. To avoid searching for cliques in a sharability graph, they restrict the definition of a trip to be only a vehicle and a single request. At each time step, the optimization only requires solving a linear assignment problem. Once a passenger is assigned to a vehicle, they cannot be reassigned to a different vehicle later even if the new matching would lead to better performance. In addition, [42] consider decomposing the problem into paths rather than into trips which allows for faster computation at the cost of exact constraint evaluation.

Work on electric vehicle systems address two main challenges: where to locate charging infrastructure and how to operate fleets of electric vehicles in real time.

On the front of locating charging infrastructure, most researchers propose two phase procedures where a first stage simulates a system initialized without charging infrastructure, and then use these results to make decisions about locating the charging infrastructure via various methods. For example, the works of [11], [20], and [41] simulate a ridehail system with electric vehicles with a first phase that dynamically creates charge stations at the location of each vehicle if the vehicle is running too

low on charge and no previously created charge station is in range. Using a different approach, [83] instead saves the locations of the charge requests when vehicles report they are low on charge and then solves a K-means problem to locate the charging stations. Going beyond the use of simulations to provide the two phases, [31] studies where to place charging stations by reviewing GPS data for vehicle trajectories from real world systems and using where they tend to park and spend time idling to guide station location decisions.

Some researchers have studied charge station location outside of the context of a two-phase simulation. For example, [35] looks to choose the best location for charge stations out of a set of candidates by solving a MIP model that minimizes the cost of constructing charge stations subject to coverage constraints. Similarly, [36] chooses charge station locations from a set of candidates by solving a p-median problem minimizing the average distance from nodes in the map to charge stations.

On the electric vehicle operations side of the problem, many studies have focused on simple myopic policies [21, 8, 37, 9] while others have attempted to incorporate planning for future demand [2, 68, 33, 29], though these methods do not necessarily scale to operational size.

Many prior works have used myopic threshold charging policies or policies activated by special events. These policies allow the vehicles to operate without regard to future charging needs until vehicle charge goes below a threshold value or the range is low enough that it cannot reach a charge station after serving a possible next request. The method of [21] assigns passengers to the nearest vehicle and charges

the vehicles when they get below 25%, or a different threshold at some times of day based on the cost of electricity. In a similar vein, [8] also uses a greedy passenger assignment algorithm and a greedy algorithm to assign vehicles to charge stations when they need to charge. Using a slightly different mechanism, [37] implements charging as part of a rebalancing process where the vehicle charges when it cannot reach its rebalancing target with its current level of charge. Planning charging as part of a vehicle’s travel assignment, [9] sequentially processes requests that arrive to the system and attempts to insert them into the vehicle’s schedule, with strategies to add, extend, or modify the vehicle’s charge station visits if necessary.

Some studies have used methods that try to be aware of the future, either directly or indirectly. One approach is to use approximate dynamic programming (ADP), such as [2] which uses ADP to determine when vehicles get new passengers and whether vehicles should charge. In [68] and [33], deep reinforcement learning is used to develop policies for vehicles to determine when to accept new customers and when to charge. They suggest that the learning process allows the system to anticipate future demand. In [29], two model predictive control optimization algorithms are used to both solve for routing and charging decisions, demonstrating their method on up to 30 vehicles.

A few studies have used other methods that were applied at small scale, such as [69] which controls 4 single capacity vehicles and determines assignments and charging times using a MIP model, [56] which looks for a policy to operate a single electric vehicle using reinforcement learning, and [10] which uses a MIP to create

routing for all visible requests subject to the routes satisfying charge constraints.

In this work we will derive methods from the batching, exhaustive search framework of [3] to show performance under optimal routing with high-capacity electric vehicles is possible in real time without having to use a myopic charging strategy. Nothing we present prevents adapting the underlying fleet assignment and routing algorithm to other methods in the literature as our electric vehicle additions are modular in this respect.

2.2 Preliminaries

2.2.1 Terminology

The area of shared transportation does not, unfortunately, have universally standardized terminology and so we briefly explain our usage in this chapter. We use the term *ridesharing* to refer to any system that allows riders to travel in vehicles not owned by themselves. This includes carpooling systems where the driver of the vehicle is traveling to the same or similar destination as the passengers as well as settings where the driver is strictly offering services to passengers. When we wish to draw attention to the fact that the driver of a vehicle is working as a driver rather than making personal travel, we call it a *ridehailing* system. When we wish to draw attention to the fact that multiple passengers can travel in a vehicle simultaneously, we call it a *ridepool* system.

2.2.2 The Problem

In the general on-demand ridepool problem, the inputs given are a graph $G = (N, A)$ representing a road network, a set of vehicles V with possible heterogeneous capacities that are centrally controlled by the operator, and a set of passengers \mathcal{R} that arrive online. Each passenger request $r \in \mathcal{R}$ is specified by a system entry time, an origin, and a destination.

The system is constrained to provide a set of minimum quality of service (QoS) requirements to all requests that are accepted into the system. These requirements are general, can be request specific, and can be chosen differently by each operator. For example, an operator may impose a maximum waiting time and a maximum detour constraint, ensuring that all accepted passengers are picked up within t_w^{\max} time of submitting their requests and that the extra time traveling in the vehicle due to a shared ride is no more than t_s^{\max} . An operator may opt to choose settings such as $t_w^{\max} = 5$ minutes and $t_s^{\max} = 15$ minutes.

If a request cannot be served using a trip that meets the QoS requirements, it must be rejected. The feasibility of a trip is determined by solving a constrained traveling salesman problem, where among other constraints we must consider the capacity of the vehicle at each point in time. Thus, matching a passenger to a vehicle in a particular batch does not require that the passenger will actually board the vehicle during that batch or that the vehicle has capacity for the new passenger at that

precise moment in time. In deterministic settings³, since we gain no new information over time that reduces the set of constraints in the system (i.e., no information that allows for previously infeasible pickups to be feasible) if a request is rejected in one batch, it will be infeasible to find a trip to serve them in later batches as well.

In our problem we assume that vehicles are electric and identical with battery properties that will be discussed in section 2.2.5. A subset of the nodes in the graph are designated as charge stations, $S \subset N$. Each charge station $s \in S$ has a maximum capacity K_s denoting the maximum number of vehicles that can simultaneously charge at s . The charge stations are private and are for the exclusive use of the operator; no consideration is given to others using the stations.

Operationally, the system is different from typical ridepool systems since 1) electric vehicles need to go out of service during the middle of the day to charge and 2) charging electric vehicles takes significantly longer than filling an internal combustion engine vehicle with gasoline.⁴ It is possible that with poor planning many of the charge stations could become congested and not be able to serve all the vehicles that are assigned to charge at them. If it is not possible for a vehicle to continue operation due to low charge, it may go to a charge station and wait. When vehicles charge, we assume they will always charge up to level q_{\max} , which may or may not be the vehicle battery's theoretical maximum capacity.

³deterministic travel times and no dropouts

⁴Operationally, refueling does not have a large impact on the performance of ICE vehicles since they usually have a long fuel range, can refuel in a short time, and have a large number of options for getting fuel.

Though the system operates online, without a prediction of future demand we cannot make meaningful decisions about when to charge vehicles, especially since some may need to be preemptively charged. For example, during rush hour (when demand is at its highest) it would be unwise to maximally utilize the charging infrastructure. To this extent, we allow the vehicle operator to create, in advance, a requirement function $R(t)$ for each time t designating the number of vehicles that must be online rather than charging. The operator can compute this function in any way, be it from demand quantity alone, value of customers at certain times of day, etc. It can also be created in more complicated ways such as considering the cost of electricity by time of day and making the requirement function so as to limit vehicles being offline and charging during the most expensive times of day.

The computation of $R(t)$ may hide a great deal of work itself. However, not only are these values computed offline, these values can involve a large number of practical considerations from the operator that are beyond the scope of this chapter. In this work we limit the construction of $R(t)$ to simple functions of expected demand, as discussed in section 2.4.3.

The objective of the operator is to assign requests to vehicles and assign routes to vehicles to maximize some function of the assignments. If the function is linear in the selection of request and vehicle assignment decisions, this leads to an ILP formulation for the problem.

2.2.3 Trip-Oriented ridepooling Framework

The underlying passenger assignment framework used in this work is based on the trip-oriented formulation of [3]. The online arrival of requests is split into batches of uniform length and computations are performed once for every batch. For any vehicle $v \in V$ and any set of requests $r_b \subseteq \mathcal{R}_b$ all in the same batch, a trip $t = (v, r_b)$ is a pairing of the vehicle and the requests. For any given trip, an optimal route for the vehicle and associated cost c_t can be found by solving a constrained traveling salesman problem (CTSP). The method used in [3] for trip generation is very general and allows for the insertion of additional constraints, something we will take advantage of as we extend the problem to electric vehicles.

The formulation of [3] is presented as an optimization formulation that first assigns a large penalty for requests that are not assigned and second penalizes the cost of serving the assigned trips. They use a binary variable ϵ_r to track which requests are rejected and then apply a large penalty, M . For the secondary objective, the cost c_t is left unchanged. The optimization problem remains general, however, since one could choose $M = 0$ and then use an arbitrary function to assign cost c_t , possibly negative, for each trip. The operator's assignment problem in a non-EV ridepool

system would thus be the solution of

$$\begin{aligned}
& \min_x \sum_{t \in \mathit{Trips}} c_t x_t + M \sum_r \epsilon_r \\
& \text{subject to } \sum_{\substack{t \in \mathit{Trips}: \\ v \in t}} x_t \leq 1 & \forall v \in V \\
& \sum_{\substack{t \in \mathit{Trips}: \\ r \in t}} x_t + \epsilon_r = 1 & \forall r \in \mathcal{R}_b \\
& x \in \{0, 1\}^{|\mathit{Trips}|}, \epsilon \in \{0, 1\}^{|\mathcal{R}_b|}
\end{aligned}$$

where here Trips is the set of feasible trips and the solution is constrained to assign at most one trip to each vehicle and choose at most one trip containing each request. Notice that this relies on full enumeration of all feasible trips. The authors of [3] observe that when the quality of service requirements are tight, as is typical in ridepooling applications, the set of feasible trips is small enough to approximately enumerate in real-time.

Trips are found using a shareability graph, a concept introduced in [60]. A shareability graph, as extended by [3], is an undirected graph $G = (N, E)$ in which the node set $N = V \cup \mathcal{R}$ is comprised of nodes for each vehicle and request. The edge set E contains all pairs (v, r) , $v \in V, r \in \mathcal{R}$ if vehicle v can serve r and all its current passengers while satisfying the quality of service constraints. E also contains all pairs $r_1, r_2 \in \mathcal{R}$ if it is possible for an ideal hypothetical vehicle to serve both requests (for example, if the requests are 5 minute apart and the maximum waiting time is 2 minutes even an ideally located vehicle cannot serve both).

Every feasible trip induces a clique in the shareability graph containing exactly

one vehicle node. This is the case since the edge set was constructed from necessary conditions for the feasibility of a trip. While the general problem of finding cliques is hard, [3] proposes a method for searching for the cliques in the shareability graph that is exact and, when heuristics are used such as restricting requests to be paired with the nearest 30 vehicles, can be solved in real time.

It may be the case that after the assignment process some requests are not served and some vehicles are left unassigned. If this is the case, a minimum cost matching based on distance is performed between the vehicles and the rejected requests. Those vehicles are then routed to the locations of the rejected requests as a heuristic for rebalancing to areas that require additional coverage. This is only one strategy for rebalancing; more sophisticated rebalancing strategies have also been proposed [40, 76].

2.2.4 Extending Models for Electric Vehicles

Algorithms for managing ridepooling fleets with electric vehicles are differentiated from typical ridepooling algorithms in that we must now additionally plan for when and where the vehicles will charge according to our operational objective. While many ridehailing operators use business models where individuals choose when to enter and exit the market, something that would naturally provide charge scheduling outside of the scope of the operator, our work considers algorithms that scale for use with high-capacity vehicles, possibly up to capacity 10. Higher capacity vehicles

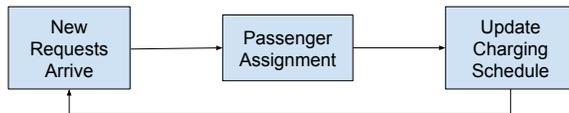


Figure 2.1: Flow or each iteration. First new requests arrive. Constrained on the existing charge schedule, passenger assignments are made. Finally, conditioned on the new passenger assignments the charging schedule is updated.

increase the potential benefits of pooling, but come with the caveat that these vehicles are unlikely to be owned by individuals. Our choice to use a centrally controlled system where drivers always follow the directions of the operator, as is the case in the model we build off of from [3], reflects this expectation. Therefore, we aim to create a charging schedule for the vehicles given fixed charging infrastructure with the assumption that vehicles will respect their assignments.

Rather than computing charging schedules and passenger assignments in a single computation, we separate the operations of the system into two steps. The first component, from the trip-oriented framework, assigns requests to vehicles constrained on not violating the current charging schedule. The second component plans when and where vehicles will charge constrained on the previously created request assignments. As shown in Figure (2.1) and algorithm (1), they work in successive steps with the interplay between the two components controlled by constraints that restrict each to decisions fully consistent with the previous output of the other.

Implementing the constraints on passenger assignments is straightforward in the trip-oriented assignment framework. All that must be done is add extra constraints in the trip generation phase to exclude possible tours that violate the charging con-

Algorithm 1 Basic Electric Vehicle Ridepooling

- 1: Initially run charging planner.
 - 2: **for** each iteration (occurs once every T^B time) **do**
 - 3: Collect list of new requests and vehicle states.
 - 4: Run the request assignment algorithm.
 - 5: Run the charge planning algorithm.
 - 6: **end for**
-

straints while solving the constrained-TSP. Because of this, the rest of the chapter is dedicated to formulating the last step - where the charge schedule is generated and updated.

2.2.5 Battery Model

An important part of running any system with electric vehicles is to understand how the batteries behave. In this subsection we describe the assumptions we make in our models. While more sophisticated models can be used, especially for charging, we use simplified linear models that can be justified for typical everyday use in the context of operational optimization of ridepooling fleets. A detailed justification for the simplification of the charging model, as well as a brief literature review on batteries, can be found in A.1.

Discharging.

Discharging involves drawing energy from the battery to power the vehicle’s drive train and auxiliary systems. Detailed research has been done on the manner in which electric vehicles consume power based on topography, acceleration, regenerative braking, etc. For high accuracy modeling of power consumption, one may refer to works such as [22].

We model discharge, the decrease in battery state of charge, with a linear function of the distance traveled. This is a reasonable first order approximation since the drive train of the vehicle consumes more power than the auxiliary systems. If one wishes to go further, models of power consumption could be used to define a custom arc “distance” using as a measure the typical power consumption of vehicles over those arcs, as done in works such as [38]. More generally, since we always know the path the vehicle has traveled on or is planned to travel on, any general path-based model can be used in our work, though we continue to use the simple distance model for the purposes of our numerical results.

In order to schedule vehicles to charge one must be able to estimate when the vehicle will run out of power. The primary variable source of power consumption in a vehicle is the engine, whose energy consumption is a function of distance and terrain. However, in an online system it is difficult to predict the mileage and terrain vehicles will travel on. Therefore we use predicted or historical data to estimate the typical rate, q_{est} , at which vehicles consume power over time from both time

dependent auxiliary services and non-time dependent components such as the engine. This estimate is based not only on the particular fleet of vehicles but also on the operator's experience with the particular road network they operate on. Since actual vs predicted power consumption is always stochastic, it is preferable for q_{est} to err on the side of over estimating power consumption over time. This helps alleviate the risks associated with sudden bursts of power usage not accounted for by the scheduling algorithm.

Charging.

While it may be natural to assume that batteries gain charge linearly over time when they are plugged into charge stations, this turns out to not be the case [23]. Many batteries charge using a procedure called Constant-Current Constant-Voltage. Under this system, the charge in the battery initially increases linearly and after reaching a specific charge threshold gains charge progressively slower [54]. While the resulting battery charging functions are non-linear, under optimal operating policies they are nearly linear. Since charging is less time efficient as the battery nears full charge, it is natural from an economic standpoint to consider a policy that vehicles should only charge up to some level less than 100% instead of fully charging due to the opportunity costs associated with charging, such as inability to carry passengers during charging. In the general case, the optimal behavior is to allow the vehicles to charge slightly beyond the constant-voltage portion of the charging curve. The process of charging just beyond the constant voltage point can be approximately

modeled as a linear function with little loss of accuracy.

In the rest of this chapter we assume that charging times are linear with rate η (% / minute). A more detailed discussion is provided in the Appendix.

Battery Conclusions

In the rest of this chapter we assume that charging and discharging are linear, as described previously. In our models we will treat vehicles as charging up to 100% and needing charge before reaching 0%, though we assume that the operator's charge policy actually means that 0% and 100% are not exactly those values. For instance, in the Subsection on charging we argued that an optimal policy does not fully charge the battery but we will consider it to be 100% for the purposes of the calculations here. On the other side, we also assume that 0% is not quite zero as described in Assumption 2.2.1.

Assumption 2.2.1. *The operator fixes a buffer level q_{\min} which is the lowest charge vehicles are allowed to have.*

For example, this might be 15%. If the vehicle accidentally goes slightly below this level it will still be able to reach a charge station without being towed. Not only does this help extend the long-term longevity of the battery, this assumption is important since it will give us a buffer in scheduling vehicles to charge as battery usage under our discharge model cannot be fully predicted since vehicle routing can

constantly be changed and updated. For convenience, we will denote this as 0% since it is the lowest we intend to let the vehicle discharge to.

2.3 Methods

In this section we propose algorithms for solving the charge scheduler’s problem. Recall from section 2.2.4 that the scheduler needs to determine when each vehicle will charge and where. Before we present our proposed algorithms, we first make some notes on the interaction between passenger assignment and charge scheduling.

The charge scheduler and the passenger assigner interact through the necessity of mutually compatible solutions - passenger assignments cannot prevent a vehicle from charging and an updated charge schedule cannot interfere with a passenger’s ride. In terms of scheduling, it is sufficient to have a mutually agreed method for computing release dates for each vehicle. The release date for a vehicle is the earliest time the vehicle can charge given its current location and passenger assignments. When scheduling vehicles for charging, this calculation is used to avoid scheduling charging while the vehicle is serving a passenger. When considering possible passenger assignments, this determines if a potential trip routing violates the charging schedule.

We define release dates for two cases - when the vehicle’s assigned charging station is known and when the assigned charging station is unknown.

Suppose vehicle v is assigned to charge at station s . Let t_c and n denote the time and location v drops off the last passenger under a potential or actual trip. The earliest time the vehicle v can be scheduled to charge at station s is

$$e_{v,s} = t_c + t(n, s),$$

where $t(a, b)$ denotes the time needed to travel from node a to b in the graph.

In the other case, if the charge station is unknown we have to intelligently add a time buffer. Let s^{opt} be the closest charge station to the vehicle's ending location n . Given buffer time D , we can conservatively estimate the earliest charging time as

$$e_v = t_c + \max\{t(n, s^{opt}), D\}.$$

The choice of D directly impacts the system as it limits which requests vehicles can be paired with whenever the charge station assignment is not yet known. It would be conservative to choose D to be the worst case travel time between any charge station and node in the graph that are the furthest apart since this will always guarantee each vehicle can be assigned a station to charge at at the assigned time. In practice one would likely choose a smaller value such as the time sufficient to reach the nearest k charge stations since this would allow more riders to be matched at the small risk that vehicles periodically cannot reach their assigned station. In such cases, the vehicle could pause and charge after being reassigned stations or, in the worst case, be directed to a closer charge station where it can charge later after some delay.

2.3.1 Charge Scheduler Problem Formulation

The objective of the charge scheduler is to charge vehicles so that, to the extent possible, there are at least $R(t)$ vehicles on the road at any given time. Let the day be separated into a set of discrete time intervals T . Assume that at each time t we expect we need $R(t)$ vehicles available and the actual number of vehicles available is $B(t)$. Using the positive part operator $(x)^+ = \max\{0, x\}$, the objective of minimizing total short fall in supply is

$$\min \sum_{t \in T} (R(t) - B(t))^+. \quad (2.1)$$

We allow the loss to be linear in the number of vehicles short we are in the solution in contrast to making a hard constraint. This is to ensure that a feasible solution always exists. Additionally, the soft penalty takes a linear form since it is more efficiently computed than other penalties, such as quadratic penalties.

To make the objective well defined, we must specify when a vehicle is and is not available. Notice that as a vehicle approaches the time it needs to charge, the set of requests it can be assigned to becomes continually more restricted. This means vehicles on the road may be neither fully available nor unavailable.

Consider the following example:

Example 2.3.1. *Suppose there is a vehicle with a single passenger who will be dropped off in 5 minutes. When the vehicle drops off the passenger it will be 5 minutes away from the charge station it has been assigned to go to. There are currently*

15 minute remaining until the vehicle must be at the charge station.

In this example, after the vehicle drops off the passenger it will have 5 minutes of idle time where one is naturally tempted to say the vehicle is available. However, if a request arrives that wishes to be driven 10 minutes in a direction away from the charge station, the vehicle will have to reject this request. On the other hand, if a request arrives asking to travel 4 minutes and wishes to be dropped off close to the charge station the vehicle may be able to serve it. Therefore the vehicle is neither fully available or fully unavailable.

This justifies the introduction of fractional availability. Fractional availability is a value between 0 and 1 that is a function of the time remaining until a vehicle is scheduled to charge and of the time remaining since a vehicle completed its previous charging. When a vehicle is charging the availability is 0. Likewise, when the previous charging and next charging times are far in the past and future, the availability should be 1. It is also reasonable that for a vehicle that charges exactly once, the availability should be monotone non-increasing as the charging time approaches, and be monotone non-decreasing afterwards as the charging time has passed.

To incorporate such a general function into an MILP formulation of the scheduling problem, we let the function $a(t)$ denote the availability of a vehicle at any point in time t . To adapt $a(t)$ to be suitable for insertion in an MILP formulation we break vehicle charging apart into multiple time periods and then introduce the helper function $A(t)$ to denote the availability of a vehicle that charges for one time period starting at time $t = 0$, ending at $t = \Pi_c$. It is an intentional design choice that a full

charge could require multiple time periods since this allows for preempting charging and heterogeneous battery capacities. Now, for any charging schedule the actual availability of each vehicle can be computed as the minimum over such functions with appropriately adjusted arguments.

Formally, the helper function $A(t) : \mathbb{R} \rightarrow [0, 1]$ is monotone non-increasing for $t < 0$, is zero for $0 \leq t \leq \Pi_c$, and is monotone non-decreasing for $t > \Pi_c$. Intuitively, the monotonicity is natural since as we move further away from the scheduled charging time, both earlier and later, the vehicle has more flexibility to serve passengers. This notion of availability only considers the charge in the vehicle indirectly through the time and duration the vehicle is scheduled to charge. Let the time periods a vehicle is scheduled to charge at be given by the binary variables x_t , for times $t \in T$. The availability of the vehicle at any time t is given by the equation

$$a(t) = \min_{s \in T} 1 - (1 - A(t - s))x_s.$$

In the context of a MILP, suppose for each vehicle $v \in V$ and each discrete time $t \in T$, the continuous variable $a_{v,t}$ denotes the availability of vehicle v at time t and the binary decision variable $c_{v,t}$ determines if v charges at time t . Since without loss any optimal solution will maximize availability, it is sufficient to use the constraint

$$a_{v,t} \leq \min_{s \in T} 1 - (1 - A(t - s))c_{v,s}.$$

Thus, since the value of $A(t - s)$ is not dependent on any decision variable even general nonlinear availability functions can be incorporated into the MILP.

2.3.2 Two Stage Planning Process

Our method for charging management is split into two parts: time scheduling and location scheduling. Here we will briefly justify and explain the details of this separation.

Scheduling vehicles to charge has two components: determining when vehicles will charge and where they will charge. Since ridepooling systems operate online, choosing the charging location for vehicles many hours in advance cannot be meaningfully done in any ‘optimal’ way, other than ensuring each vehicle will have a station to charge at. With this observation, we split the charge scheduling process into two separate parts: a long horizon where we only consider changing the time at which vehicles charge and a short horizon where we only consider the locations at which

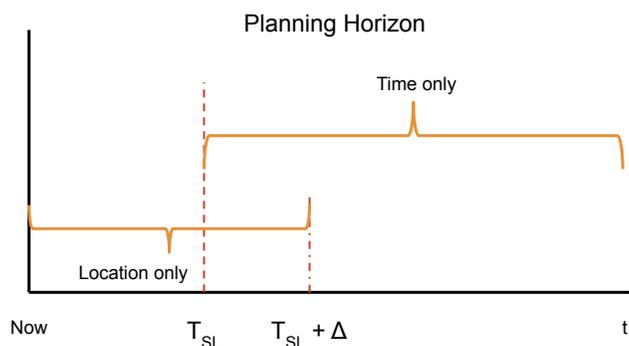


Figure 2.2: In the long horizon, the schedule more than T_{SL} time in the future can be modified. The short horizon station problem only assigns vehicles that are scheduled to charge within $T_{SL} + \Delta$ of current time. Δ can be 0, but theorem 2.3.1 shows it cannot be too large.

vehicles charge. More formally, the long horizon only modifies charge schedules at least T_{SL} time in the future while the short horizon assigns charge stations to vehicles scheduled to charge within $T_{SL} + \Delta$ for some chosen T_{SL} and parameter Δ controlling the maximum overlap. For example, one might choose $T_{SL} = 45$ minutes and $\Delta = 15$ minutes. See Figure (2.2). It is not necessary to choose $\Delta > 0$, but an upper limit on Δ exists as will be shown in theorem 2.3.1.

While the proposed split reduces computation complexity of the problem, the solutions it produces may be suboptimal. For example, it may be possible to choose better station locations in the short horizon if charge times can be shifted by a small amount. However, since the short horizon problem is defined only over a small subset of times, the flexibility to reschedule vehicle charging times is greatly restricted unless the full complexity of the long horizon problem is reintroduced.

With the short and long horizon problems being solved separately, it is necessary to take steps to ensure the times vehicles are scheduled to charge in the long horizon problem admit feasible charge station assignments. The choice of D in the release date equation $e_v = t_c + \max\{t(n, s^{opt}), D\}$ provides the trade off between guarantees of feasibility with conservative, large D and flexibility to serve more requests with aggressive, smaller D . Sufficiently large D guarantee that feasible solutions will exist, but come at the cost of being restrictive on which requests a vehicle can serve.

Since an operator may not feel that updating time and location information for vehicle charging is necessary at the same frequency as passenger assignments are made, it is possible to run these planning algorithms at only a subset of the iterations.

For example, if requests arrive in batches every T^B seconds, then one could run the short horizon station location problem every T^S seconds (divisible by T^B) and run the long horizon time problem every T^ℓ second (divisible by T^S). As a variant of algorithm (1), this modified procedure is shown in algorithm (2).

Algorithm 2 Electric Vehicle Ridepooling

```

1: Initially run Long horizon planning algorithm.
2: for each iteration (occurs once every  $T^B$  time) do
3:   if  $T^\ell$  time has passed since last run of short horizon planner then
4:     if  $T^S$  time has passed since last run of long horizon planner then
5:       Run the long horizon planning algorithm.
6:     end if
7:     Run the short horizon planning algorithm.
8:   end if
9:   Run the request assignment algorithm.
10: end for

```

As a final note, the choice of overlap Δ must be less than or equal to $2T^\ell$, as explained in the following theorem.

Theorem 2.3.1. *It is necessary and sufficient that $\Delta \leq 2T^\ell$ to guarantee that feasible charge station assignments will exist for all vehicles.*

Proof. Without loss of generality, assume the system clock is currently set to time 0.

1. Case 1 : $\Delta = 2T^\ell$.

Consider the set of vehicles the long horizon problem schedules to charge in time periods T_{SL} and $T_{SL} + \Delta$. Those in the former will have their assignments fixed

and will not appear in the next round of long horizon optimization since the long horizon only modifies schedules at least T_{SL} time in the future. The later will appear one more time in the next iteration of long horizon optimization.

Any vehicle in the next round of longest optimization will either be a vehicle previously assigned in time slot $T_{SL} + \Delta$, in which case it already has a station assignment, or a vehicle that has no previous station assignment. In the case of no assignment, the buffer time $e_v = t_c + \max\{t(n, s^{opt}), D\}$ guarantees these vehicles have passenger assignments that are compatible with being assigned any station. Thus, a feasible charge station assignment always exists.

2. Case 2 : $\Delta > 2T^\ell$

Let $\Delta = 3T^\ell$. Suppose there are two charge stations of capacity one that are far apart. In the first iteration, one vehicle is assigned to charge for a single time period at time $T_{SL} + \Delta$ and another is assigned to charge at time $T_{SL} + 2\Delta$. Assume both vehicles are assigned the same charge station, call it c_1 . Since both of these vehicles now have an assigned station, their buffer time calculation changes to $e_v = t_c + t(n, c_1)$. This means that it is possible that each vehicle, in the next iteration, is assigned passengers that want to travel to c_1 and who will be dropped off just as the vehicles' assigned charge times begins.

In the next iteration, it is feasible for the long horizon problem to now assign both vehicles to the same charge time, $T_{SL} + 2\Delta$. However, if the distance between c_1 and the other station, c_2 , is greater than Δ , then both vehicles can still only reach c_1 due to the passenger assignments. But they are scheduled to

charge at the same time and each station has capacity 1. So there is no longer a feasible charge station assignment.

□

2.3.3 Long Horizon Planning

The long horizon problem sets vehicle charging times that are scheduled at least T_{SL} beyond the present time. We propose solving this problem using a MILP that assigns charging times for the full day. In addition to the objective of minimizing short falls given in equation (2.1), we also include a soft penalty if the schedule expects any vehicles to reach or go below zero charge. The soft penalty on negative charge levels guarantees there is always a feasible solution to the long horizon problem for any input.⁵ Additionally, recall from earlier that we assume the operator chooses a charge level $q_{\min} > 0$ that we refer to as 0% charge. Therefore, even if a vehicle goes slightly below this level it will still be able to operate.

Since we cannot know the exact charge in the vehicles in the future, we estimate the charge by computing the charge in the vehicle when it is done serving its current passengers at time e_v , which we call $Q_{v,0}$. We then use the estimated linear discharge

⁵In practice, the model penalizes vehicles for having negative charge but not for being close to negative charge, so many vehicles will run out of charge slightly before the scheduled charge time due to the discharge function depending on the vehicle’s routing, which is not fully known in advance. This is not a problem since either vehicles can idle at the charge station prior to charging or the ‘zero’ charge mark is actually a safe buffer away from true empty.

rate q_{est} discussed in the section on batteries for the time periods the vehicle does not charge and η to account for when the vehicle does charge.

The total capacity of all charge stations is denoted by $K = \sum_{s \in S} K_s$. Since the fractional availability function may be less than one for vehicles that have charged, or are planning to charge, prior to the first time period considered in the long horizon problem, we introduce the value $P_{v,t}$ which equals $A(t-t')$ where t' is the most recent time period vehicle v charged.

There are three decision variables used in the problem.

1. $c \in \{0, 1\}^{|V||T|}$ - binary decision to charge vehicle v at time t .
2. $a \in \mathbb{R}_+^{|V||T|}$ - decision of fractional availability of vehicle v at time t .
3. $q \in \mathbb{R}^{|V||T|}$ - decision of estimated charge in vehicle v at time t .

Using the above, the MILP model⁶ is defined by

⁶Let $(x)^+ = \max\{0, x\}$, $(x)^- = \max\{0, -x\}$.

$$\min \sum_{t \in T} \left(R(t) - \sum_{v \in V} a_{vt} \right)^+ + M \sum_{v \in V} \sum_{t \in T} (q_{vt})^- \quad (2.2)$$

$$\text{s.t. } \sum_{v \in V} c_{vt} \leq K \quad \forall t \in T \quad (2.3)$$

$$c_{vt} = 0 \quad \forall t < e_v \quad \forall v \in V \quad (2.4)$$

$$q_{v,t_v} = Q_{v0} \quad \forall v \in V \quad (2.5)$$

$$q_{vt} \leq q_{v,t-1} - q_{est}(1 - c_{v,t-1}) + \eta c_{v,t-1} \quad \forall t > e_v \quad \forall v \in V \quad (2.6)$$

$$a_{vt} \leq \min_{s \in T} (1 - (1 - f(t - s))c_{vs}) \quad \forall t \in T \quad \forall v \in V \quad (2.7)$$

$$a_{vt} \leq A_{vt} \quad \forall t \in T \quad \forall v \in V \quad (2.8)$$

$$0 \leq a \leq 1, \quad c \in \{0, 1\}, \quad q \leq 1, \quad (2.9)$$

where e_v is as defined at the beginning of Section 2.3.

The objective (2.2) penalizes having fewer than the required number of vehicles at each point in time and also penalizes for vehicles having negative charge. M allows control between supply shortfall and avoiding negative charge as the primary objective. M is typically large since it is most important that charge is non-negative whenever possible.

Constraint (2.3) forces the number of vehicles charging at any one time to be at most the total available capacity across all charging stations. Constraint (2.4) ensures that no vehicle is scheduled to charge before the release date, which accounts for final passenger drop-off and the minimum time needed to reach charge stations.

Constraint (2.5) defines the charge in the vehicle at the vehicle’s release date. Constraint (2.6) then describes how the charge value evolves over time, decrementing by q_{est} if it is not charging, and increasing by η if it is. The constraint prevents vehicles from gaining more charge in a single period than is possible using the charging infrastructure and works in conjunction with constraint (2.9) which limits the maximum charge in vehicles to 100%.

Constraint (2.7) defines the fractional availability of each vehicle in terms of the function $f(t)$, though it does not account for the possibility of reduced availability due to prior charging. Finally, constraint (2.8) restricts the value of availability to account for the prior charging.

While non-preemptive solutions are preferred, the model is preemptive since there are no constraints that require selected charging periods to be of the length necessary to fully charge. The solution is encouraged to be non-preemptive through the penalty of availability introduced in constraint (2.7). Since the availability function $a(t)$ reduces the amount of availability a vehicle is considered to have just before and after charging, when a vehicle’s charging is preempted the system cannot fully reclaim the vehicle’s availability. Between it and the vehicle that preempted it, this will detract from the systems’ total amount of availability and potentially lead to a worse objective value. In schedules where many vehicles need to charge and the system is running near minimum availability in all time periods, non-preemptive solutions allow the model to charge more vehicles while maintaining the required availability. Therefore, we expect that typical solutions will not contain many preemptive

elements.

2.3.4 Short Horizon Planning

Short horizon planning refers to the problem of scheduling where vehicles should charge when the vehicles' scheduled charge times are within a *short* window, i.e. less than $T_{SL} + \Delta$. Short horizon planning will make final assignments of vehicles to stations minimizing the impact of charging on the vehicles. This is done by associating a cost $d_{v,s}$ for matching vehicles $v \in V$ to stations $s \in S$. For example, the cost $d_{v,s}$ might be the distance from the vehicle's last drop-off point to station s . When it is not feasible to match a particular vehicle and station we let $d_{v,s} = \infty$.

Although the short horizon problem does not select charging times, this problem must nonetheless consider both space and time. Let $V' \subseteq V$ be the set of vehicles to be assigned charge stations, let T refer to the set of time intervals during which any vehicle in V' is assigned to charge, and recall that K_s is the capacity of each station. For any time $t \in T$, let V_t be the set of vehicles that charge at time t . For any vehicle $v \in V'$, let S_v be the set of stations the vehicle can reach by its scheduled charge time.

Since vehicles may have heterogeneous battery capacities or states of charge, note that the charging times will be vehicle specific. In other words, given some time discretization step Π_c , $t_v = |\{t \mid v \in V_t\}|$ denotes the number of time periods needed to charge vehicle v , which leads to a charging time of $\Pi_c \cdot t_v$. This differentiates the

short horizon problem from a bipartite matching problem since the vehicle charges at the same location during each of the successive time periods it's assigned to charge.

The sole binary decision variables is $P \in \{0, 1\}^{|V' \times S|}$ which is the indicator for whether vehicle $v \in V'$ charges at station $s \in S$. The binary programming formulation is

$$\min \sum_{s \in S, v \in V'} d_{vs} P_{vs} \quad (2.10)$$

$$\text{s.t.} \sum_{v \in V_t} P_{vs} \leq K_s \quad \forall s \in S, \forall t \in T \quad (2.11)$$

$$\sum_{s \in S_v} P_{vs} = 1 \quad \forall v \in V' \quad (2.12)$$

$$P \in \{0, 1\}^{|V' \times S|} \quad (2.13)$$

Constraint (2.11) ensures that each station is operating at or below capacity during each time period and constraint (2.12) ensures each vehicle receives exactly one valid station assignment.

The station assignment problem can be shown to be NP-complete by reduction to the hierarchical interval scheduling (HIS) problem with single machine capacities which was introduced and shown to be hard by [32]

Definition 2.3.1 (from Kolen et al. 2007). *Given K continuously available machines and n jobs with starting times s_j and finishing time f_j such that each job can*

only be processed on machine $\{0, \dots, r_j\}$, it is NP-hard to determine if there exists a feasible schedule.

Theorem 2.3.2. *The short horizon station location problem is NP-complete.*

Proof. To show the problem is in NP it suffices to show that the number of constraints is polynomial in $|V|$ and $|S|$. Constraints (2.12) clearly satisfy this. It is sufficient for constraint (2.11) to satisfy this if the number of discrete time periods T is $O(|V|)$. Since the starting and ending time of each charging job is fixed, an interval graph can be created with vehicles as vertices and edges connecting nodes that represent vehicles with overlapping charging time. Simultaneously charging vehicles form cliques in the graph and it is sufficient to consider the maximal cliques in the graph. It is an old result that interval graphs have at most linearly many maximal cliques all of which can be found in linear time [66]. Thus, the problem is in NP.

To show it is NP-complete, start by letting there be K charging stations, each with capacity 1. For each job j , create a vehicle that needs to charge from s_j to f_j . Let the cost d_{vs} of pairing station s with vehicle v be 0 if $s \leq r_j$ and 1 otherwise.

There exists a feasible schedule to the HIS problem if and only if there exists a solution to the ILP of zero total cost. Thus the short horizon problem is NP-complete. \square

While solving the ILP is NP-complete, as with many problems in practice commercial solvers were able to solve sufficiently large instances in less than a tenth of

a second. Earlier it was argued that release dates for the long horizon problem can be computed so that this problem will always have a solution. But what happens if under more aggressive parameterizations of D there is no feasible solution in some iteration? Since all vehicles that were present in the previous iteration could be assigned charge stations, we know the infeasibility of the problem must stem from vehicles that were just added to V' and that have the latest charge times of all vehicles in V' . One way to resolve this is by solving the short horizon problem without these vehicles, using a greedy assignment on the new vehicles, and then forcing any remaining unassigned vehicles to be rescheduled in the long horizon problem in the next iteration. In simulations, this step was never needed.

In the real world, the simulated prediction of the mechanics of the vehicle could be incorrect and charge could be used faster than expected. This could cause a vehicle to be unable to reach the charge station it was assigned previously because its charge level is now expected to be below zero at its assigned charging time. Without compromising the practical applicability of this method, there are a variety of model properties that allow this problem to be addressed.

In this case, it is typically sufficient to make a hard constraint that any vehicle that was assigned to charge station s in the previous assignment period must also have that station included in S_v in the next iteration as well. Recall from section 2.2.5 that for battery longevity reasons there is a buffer between what the operator calls 0 charge and the actual empty level for the battery (assumption 2.2.1). Thus a vehicle might have a charge level that is too low for any stations to be included

in S_v though in reality it has sufficient range to reach at least some stations. All else failing, vehicles can travel to the nearest charge station and wait for the next available slot, even if it is outside of the assigned time window. A variety of other resolutions could be used, such as letting the vehicle wait and be assigned a new charging time similar to the resolution proposed for aggressive parameterizations of D .

2.3.5 Heuristic Long Horizon Planning

While the short horizon planning problem can be solved very efficiently in practice, the long horizon MILP is not practical for real-time use. This is not surprising since it contains $O(VT)$ variables and constraints making it pseudo polynomial in size with the resolution of time used. Time indexed formulations are known to be large and intractable in many settings. While in the machine scheduling literature there exist exact techniques, such as column generation approaches [72], for solving similar problems, these methods do not yield strong worst-case runtime guarantees and thus instead of adapting exact techniques for this novel MILP formulation we propose an efficient heuristic to update the charge schedule for vehicles.

As motivation for the heuristic, consider a system with a single vehicle and a single charge station. The vehicle is always either operating or charging. Supposing discharge and charge are linear function of time, the time for the vehicle to charge after driving for t minutes can be modeled as $at + b$, where b is the offline time spent

driving to the charge station. If the cost of operation is proportional to time offline for charging, then the average cost over the course of the day

$$g(t) = \frac{at + b}{at + b + t} = \frac{at + b}{(a + 1)t + b}$$

is decreasing in t . Meaning, the impact of being offline is minimized when the battery is used to the maximum extent and charging occurs as infrequently as possible. This directly motivates our heuristic:

Heuristic 2.3.1. *The vehicle with most remaining charge should be scheduled to charge as late as possible. Other vehicles can be scheduled iteratively.*

In the online model we would like to only consider the next time each vehicle charges. Even though vehicles must charge multiple times we hope that the objective of maximizing time until charging, as suggested in the example, will accomplish this goal. However, choosing a charging schedule that waits as long as possible has a downside - it maximizes the amount of work that needs to be done in the single charge time horizon we consider.

It is well known in the literature of time dependent job scheduling that processing the shortest job first minimizes the makespan (total work) of the schedule. In fact, the problem closely resembles the time dependent jobs with deadlines problem studied by [12]. However, by scheduling vehicles with the most remaining charge last we are doing the opposite - we are placing the shortest jobs at the end, resulting in a final schedule where realized processing times are not in any sorted order.

We believe the justification for our heuristic stands even in light of this. The

total amount of work to be done recharging is first the fixed cost of detouring to a charge station and second charging time, which we assume is proportional to the miles the vehicle have driven. Assuming the vehicle drive the same number of miles under any scheduling scheme, the best schedule will be one the aims to minimize the number of times vehicles charge. This is what our heuristic does.

This conflict between minimizing work in a single scheduling horizon versus minimizing work over the entire day also illustrates why it was important for the long horizon MILP to produce schedules for the entire day rather than just find the next time each vehicle charges. Under the shortest work rule used in time dependent job scheduling (in its strictest sense) vehicles that finish charging would have to immediately turn around and charge again while vehicle with depleted batteries would never be charged!

Algorithm.

In order to implement the planning heuristic we will first define some objects to store static data and the dynamically updated state of slack in the problem's constraints and then describe how they are used in the algorithm.

We start by creating an object `Constraints` which can be queried for each time of day to know the slack on capacity constraints (total capacity K) and availability constraints (given by $R(t)$). As decisions are made, changes can be written to the `Constraints` to show reduction in slack.

The basic algorithm works as follows. For each vehicle, we compute the value of e_v and store these values in the discrete map **Earliest**. Similar to $A_{v,t}$ in the MILP formulation, for each previous assignment, we update our **Constraints** object to reflect reduction in availability so that fewer jobs are scheduled at that time.

Since we can compute the charge in vehicles $Q_{v,0}$ at time e_v , we can use q_{est} to extrapolate the first time each vehicle is expected to run out of charge. This is the deadline for the vehicle, which we store in the discrete map **Deadlines**.

Next, we sort the list of vehicles according to their **Deadlines**. Vehicles with later deadlines are given the highest priority, they will be scheduled as far in the future as possible. We store this sorted list as **Priorities**.

Using this priority based scheduling, sometimes it is not possible to schedule all the vehicles to charge given capacity, availability, and deadline constraints. When this happens, all elements of the schedule are pushed into the future until sufficient space is made, using algorithm (3) **PushBack**. This operation is done by first pushing back vehicles with lowest priority - minimizing the maximum extent any vehicle is moved back compared to its current assignment. This operation can be performed efficiently using doubly linked lists to avoid redundant computations for groups of vehicles scheduled to charge at the same time.

This could cause some vehicles to have charge times after they would be expected to run out of charge. Since q_{est} is a slightly pessimistic estimate of charge usage, it is often the case that the vehicles that are pushed back will not actually run out of

power before charging. In addition, since we assume that the 0% mark is actually a safe buffer q_{\min} away from absolute zero this will not be a significant issue for the vehicle. In any case, if this poses a practical issue for the operator modifications can easily be made to passenger assignment and vehicle routing algorithm to halt a vehicle at a nearby charge station if it is nearing empty charge.

The `PushBack` algorithm is shown in algorithm 3 and the full algorithm for charge scheduling is given in algorithm 4. The `PushBack` algorithm uses two special functions: `ClearAvailability` and `ClearCapacity`. Each of these takes as an input the working copy of `Constraints`, the `Schedule`, the vehicle’s priority, and time for which the vehicle to be scheduled is blocked either by availability or capacity constraints. For ease of notation, the `Constraints` and `Schedule` arguments are omitted in the algorithms.

`ClearAvailability` and `ClearCapacity` do the following: if there is a set of vehicles of higher priority that can be removed that will clear sufficient capacity or availability, the smallest set of such vehicles is returned. Keeping the recursive nature of the `PushBack` algorithm in mind, the functions always return the vehicle of highest priority among all that would clear the constraints in order to minimize work in future iterations.

Lemma 2.3.1. *The complexity of the `PushBack` algorithm (algorithm 3) is $O(T)$.*

Proof. Note that the first loop runs over $O(T)$ time frames. Assume that the maximum number of time windows a vehicle may need for a full charge, Needed-

Algorithm 3 PushBack($\mathbb{N} \rightarrow (V, T)$ Items, Schedule, Constraints)

```
1: if Items =  $\emptyset$  then
2:   return Schedule.
3: end if
4:  $(v, t) = \text{Items.pop}()$  (lowest priority item).
5: for time  $s = t, \dots, T$  do
6:   Constraints' = Constraints, Schedule' = Schedule.
7:   BumpList :  $\mathbb{N} \rightarrow (V, T) = \{\}$ .
8:   for increment  $i = 0, \dots, \text{NeededCharge}$  do
9:     if Constraints' availability insufficient at time  $(s + i)$  then
10:      obstruction  $[v'] = \text{ClearAvailability}(\text{Priorities}_v, s + i)$ .
11:      if no obstruction then
12:        Continue to iteration  $s = s + 1$ .
13:      else
14:        for  $v' \in [v']$  do
15:          Unschedule  $v'$  in Schedule', Update Constraints', Schedule'.
16:          BumpList $\cup = \{\text{Priorities}_{v'} : (v', s + i)\}$ .
17:        end for
18:      end if
19:    end if
20:    if Constraints' capacity insufficient at time  $(s + i)$  then
21:      obstruction  $[v'] = \text{ClearCapacity}(\text{Priorities}_v, s + i)$ .
22:      if no obstruction then
23:        Continue to iteration  $s = s + 1$ .
24:      else
25:        for  $v' \in [v']$  do
26:          Unschedule  $v'$  in Schedule', Update Constraints', Schedule'.
27:          BumpList $\cup = \{\text{Priorities}_{v'} : (v', s + i)\}$ .
28:        end for
29:      end if
30:    end if
31:  end for
32:  Schedule vehicle  $v$  at time  $(s + i)$  in Schedule'.
33:  Update Schedule' and Constraints'.
34:  return PushBack(Items  $\cup$  BumpList, Schedule', Constraints').
35: end for
```

Charge, are $O(1)$. Within each loop, we have a constant number of steps. The `ClearAvailability` and `ClearCapacity` functions take constant time by looking up the time index in an array and then inspecting at most the first c vehicles assigned at that time, with c being the number of time periods needed for charging and assumed to be $O(1)$. Since the `ClearAvailability` and `ClearCapacity` function prefer to select vehicles with the highest priority among all vehicle eligible at each point in time, running the `Unschedule` and `Schedule` routines takes constant time by storing elements for each vehicle at each time in a doubly linked list structure. Updating constraints is also constant time since we have $O(1)$ removals to clear space and $O(1)$ insertions to add the current vehicle.

To see that the algorithm is $O(T)$ including recursion, notice that when vehicles are removed they are only considered for rescheduling at a later time. This is because we always choose to remove the vehicle scheduled at that time with the highest priority and we can only bump vehicles of higher priority when choosing a new charge time for the vehicle. Thus each time period is considered at most a constant number of times and the recursive algorithm is $O(T)$. \square

Theorem 2.3.3. *The complexity of the Long Horizon Heuristic (algorithm 4) is*

$$O(n \log(n) + nT).$$

Proof. The algorithm sorts all vehicles by priority using $O(n \log(n))$ time. Then the algorithm tries to schedule the vehicle in one of $O(T)$ time periods with each period requiring a constant amount of work to verify whether there is space or not. Insertion

into the schedule takes constant time if the duration is $O(1)$, which we assume, and the schedule is implemented as an doubly linked list.

If the `PushBack` algorithm is required, this adds at most $O(T)$ time to the loop. Thus the loop runs in time $O(nT)$. Since the algorithm begins with sorting, the total running time is $O(n \log(n) + nT)$. \square

Algorithm 4 Long Horizon Heuristic

```

1: Input:  $(V, \text{Constraints}, \text{Deadlines}, \text{Earliest}, \text{Priorities})$ .
2:  $\text{Schedule} : V \rightarrow [T] = \{\}$ .
3: for each vehicle  $v \in \text{Priorities}$  in descending order do
4:   for each time  $s$  from  $\text{Deadlines}_v$  down to  $\text{Earliest}_v$  do
5:     duration = number of time windows  $v$  needs if start charging at  $s$ .
6:     if Can schedule  $v$  at time  $s$  then
7:        $\text{Schedule} = \text{Schedule} \cup \{v : [s, \dots, s + \text{duration}]\}$ .
8:       break
9:     end if
10:  end for
11:  if not yet scheduled then
12:     $vs : \mathbb{N} \rightarrow (V, T) = \{\text{Priorities}_v : (v, \text{Earliest}_v)\}$ .
13:     $\text{Schedule} = \text{PushBack}(vs, \text{Schedule}, \text{Constraints})$ .
14:  end if
15: end for

```

2.4 Numerical Results

To test the performance of our proposed methods, simulations were performed using publicly available taxi ridership data in Manhattan from August 2013 [15] We use a threshold of $T_{SL} = 45$ minutes and let requests arrive in batches every $T^B = 60$

seconds. We weighted the objective function to first maximize the number of requests served and then, second, minimize total vehicle travel time.

The charging rate η was chosen so that vehicles would take 30 minutes for a full recharge. The estimated discharge rate over time q_{est} was estimated using distance over time simulation results from ICE simulations whose results were dependent on the choice of request data set used (e.g., which day). Recall, discharge does not have to be computed as a function of distance but we choose to do so for simplicity.

The simulations were performed on a desktop using an Intel Core i7-6700 CPU with 8 cores and 16GB of RAM. The simulation was written in C++ and optimization was performed using Mosek version 8.

The experiments tested four different methods:

1. MILP: Long horizon MILP model and short horizon MILP model.
2. Heuristic: Long horizon heuristic in combination with the short horizon MILP model.
3. Benchmark: Simple algorithm that does not use information about future demand, defined in section 2.4.2.
4. ICE Baseline: None of the vehicles are EV.

In the rest of this section we discuss how the inputs to the simulations were produced and discuss the results.

2.4.1 Charge Station Location

As discussed in the literature review, a variety of approaches have been proposed to choose locations for charging stations. Following the spirit of the methods that use K-means problems, we considered locating the charge stations using a weighted k-means clustering algorithm. As a proxy for determining where the vehicles would have no passengers, and thus available to charge, we referred to the taxi data set which consists of historical request data for Manhattan in the form of origin-destination pairs. Every time a vehicle charges it is between its last drop-off and next pickup and so this data set can be interpreted as samples of possible locations at which vehicles will be empty.

Combining the two ideas, we performed weighted k-means clustering where weights were proportional to the number of originations/destinations at each node and normalized so the total weight of the graph was equal to the desired total charging capacity. The central node of each cluster was chosen to be the charge station and the total capacity of the charging infrastructure was distributed in proportion to the corresponding cluster weights.

2.4.2 Benchmark Method

As a benchmark, we created a simple algorithm to schedule vehicles that does not take into account any information about future demand. The method is similar to

the method used in [21] which greedily assigns vehicles to the closest charge station, though we additionally allow the vehicle to go to further stations if it is able to charge sooner there. In addition, we allow charge going below a threshold to be the signal for charging similar to [21], though rather than implementing this as a check whenever a passenger is dropped off we also check before assigning a new passenger to a vehicle, a necessary change since [21] only considers single capacity vehicles.

In this algorithm, vehicles make individual decisions of when to charge. Vehicles operate all day until their charge goes below the threshold q_{\min} . Then the vehicles continue to serve requests that have been assigned to them, but refuse to accept any new passengers. Once the final passenger has been dropped off, the vehicle requests space at a charge station that will allow the vehicle to charge the soonest. To help keep vehicles near areas of high demand, we limit the set of charge stations the vehicle can charge at to be within a specified time radius of its location, which we choose to be 15 minutes. Once the vehicle is finished charging it returns to service.

This is described in Algorithm 5. Additionally, the Algorithm 6 describes how a vehicle that is empty and is below the charge threshold is assigned to a charging station in a greedy manner.

2.4.3 Producing Availability Requirements $R(t)$

To compute the vehicle availability requirement vector $R(t)$ for methods that are aware of information about future demand, we employed a simple procedure that

Algorithm 5 Benchmark Charging Algorithm

```
1: for each vehicle  $v$  do
2:   if charge in  $v$  is below threshold then
3:     Forbid vehicle from accepting new passengers in next assignment iteration.
4:     if vehicle is empty and has no charge station assignment then
5:       Assign charge station using Greedy Assignment algorithm.
6:     end if
7:   else
8:     Clear vehicle's charge station assignment.
9:   end if
10: end for
```

Algorithm 6 Greedy Station Assignment(**vehicle**)

```
1:  $\ell \leftarrow$  current location of vehicle.
2:  $S \leftarrow$  set of charging stations within distance threshold.
3: Times  $\leftarrow$  {time(vehicle,  $s$ ) for  $s \in$  Stations}.
4: Earliest  $\leftarrow$   $\{(s, \text{first available slot time at } s \text{ for } )s \in S\}$ .
5: Assignment  $\leftarrow$   $\arg \min_{s \in S} \max\{\text{Times}_s, \text{Earliest}_s\}$ .
6: Update  $S$  with Assignment.
7: return  $s, \max\{\text{Times}_s, \text{Earliest}_s\}$ .
```

roughly models the required number of vehicles by time of day as a weighted average of the entire fleet and a scaled multiple of the customer demand profile. Letting $d(t)$ be an expected demand profile vector and λ be a scalar parameter, our model for $R(t)$ is described by the equation

$$R(t) = |V| \left(\lambda d(t) + (1 - \lambda) \right).$$

First, to select $d(t)$ we started by considering all requests in the input data file and associated each with an interval starting when the request is made and with duration equal to the travel time from the request's origin to destination. For every thirty minute block of time during the day we counted the number of requests whose

intervals overlap any portion of that block. We used the results to form $d(t)$ as a piecewise constant function and normalized so the largest value was 1.

Next, to select λ we started by computing an estimate for the number of hours of charging that must be done given the size of the fleet and the range of the vehicles. We first tried using a λ that made the number of offline vehicle hours equal to the total charging demand. This turned out to not be sufficient since vehicles have limitations on when charging can be done, such as not running out of power and being limited to 100% charge. The peaks and troughs in the demand function were sometimes too spread out or too heavily concentrated in the morning to yield an effective requirement function. Thus we decided another method was needed to determine a reasonable λ .

We used a greedy algorithm to create a naive charging schedule for the day given the expected battery discharge rate and $\lambda = 0$, so no feasible solution could be found. Then λ was continuously raised until it was possible to find a feasible schedule using the greedy algorithm. This λ was then used for the experiments. Since this method is only a heuristic, it is possible that optimal solutions may exist for tighter (smaller) λ , meaning more vehicles required on the road at all times. In practice an operator would have many factors to consider beyond the scope of this dissertation and so we leave open the space of possible procedures for constructing $R(t)$. We explored some of the possibilities of other constructions in $R(t)$ in our experiments as well, see tables 2.7 and 2.8.

2.4.4 Fleet Size and Reduced-Size Map for MILP Method Experiments

Simulations with ICE vehicles and those using computationally efficient methods, such as the benchmark and long horizon heuristic methods, were tested with up to 2000 vehicles. Since the long horizon MILP models are large and do not scale well in simulations with many vehicles, in order to run experiments with the MILP method we reduced the number of vehicles and produced a reduced size graph for testing.

To make the reduced-size graph we selected a portion of Manhattan roughly from 19th Street to 59th street. This reduced the total number of intersections in the graph by about 78%. All requests that did not originate and terminate in this region were removed from the input data set. Calibrating the experimental service rate by randomly filtering out requests was put off until the number of vehicles had been chosen.

To determine the number of vehicles to use in the reduced size experiment, we ran a small number of iterations of the long horizon MILP at various vehicle counts. We chose 220 vehicles since it was near the limit of how many vehicles the MILP solver could handle in a reasonable period of time. Based on observations, we choose two time limits for the MILP solver to use in the final experiments of 180 and 55 seconds, after which in each case we use the best solution found so far. Giving the MILP solver more than 180 seconds did not result in significantly lower objective values.

To select the demand level, we ran the ICE method on the reduced size system at various levels of demand with 220 vehicles and reduced the number of requests until the service rate reached 90%. This resulted in filtering 2/3 of the requests from the input data set.

In summary, we chose 220 vehicles to use on the reduced-size map on which we perform experiments on all four methods. In addition, we ran experiments using 1000 and 2000 vehicles for the heuristic method, benchmark method, and ICE baseline.

2.4.5 Battery Range and Expected Charge Duration

While battery ranges on small personal vehicles have grown significantly in recent years, albeit with high price tags, ranges for larger commercial vehicles lag behind. Since our target system is a high-capacity ridepool system, we surveyed information about charge range for electric service and passenger vans. When reviewing the data, we assume that tests performed using NEDC have real-world energy consumption that is 38% higher [50] and tests that do not list a method were performed using the newer WLTP standard with resulting real-world energy consumption that is 14% higher [17]. By reviewing a list of electric vans that are to be available in 2020 [19], we find that the average listed electric van has a range of 160 km using the default or lower capacity battery option and 200 km using the default or higher capacity battery option. From this, we selected 180 km as the range for our hypothetical vehicles in simulation.

Both the MILP method and the heuristic for charge scheduling require an estimate of when batteries in vehicle will need to be charged. However, the assumption that energy usage is linear in distance traveled requires us to produce a separate estimate for how long batter charges last. For each of the 3 fleet sizes we first run a simulation using ICE vehicle. Then, using the number of miles driven during the simulated day we produce an estimate of the number of hours until discharged. For 220 vehicles, the result was 13 hours and for 1000 and 2000 vehicles it was 10 hours, different values due to differences in request density and map size discussed in 2.4.4. See Table 2.10 to see how sensitive these estimates were.

2.4.6 Data Processing and Simulator Implementation

The Manhattan taxi ridership data [15] was converted into a format usable by the simulator by map matching the origin and destination coordinates onto the road network used in the simulator. The time that each request was submitted in the ridership data was preserved and the resulting combinations of time and locations were stored in a file. To run the simulation, the file is loaded and each request is presented to the assignment algorithm at the end of the batch interval during which it actually arrived in the ridership data.

At the end of each batch of processing, the central operator gives each vehicle an updated assignment list. The operator can update the assignment list at any time without regard for the state of the vehicle, with the exception that the assignment

must be feasible given the quality of service constraints and passengers that are already onboard must remain in the assignment list until the vehicle drops them off. After each batch of assignments, the simulator executes the following steps for each vehicle:

1. From the trip assigned from the RTV graph, get optimal ordering of passenger pickup and drop off, given current vehicle position, onboard passengers, time remaining for quality of service constraints for each request, and any electric constraints on the vehicle.
2. Expand the ordering with an edge-by-edge path.
3. Follow the path, making pickups and drop offs as appropriate. Pause when the end of the batching interval is reached, even if the vehicle has not completed its assignment or has only partially traversed the current edge. Travel will resume after the next batch of assignment updates have been processed.

2.4.7 Results

We primarily evaluated the performance of each method by the percentage of requests served, though we also report vehicle miles traveled as well as other statistics. The statistics we report are similar to those used by [3], [46], and [53]. A summary of all statistics used and their abbreviations are given below. See Table 2.1 for a more compact reference.

- *Service rate*, abbreviated as *Rate*, is the ratio of the number of requests that were served to the total number of requests that were made.
- *Waiting time*, denoted *WT*, is the average amount of time, among requests that were served, between system entry time and vehicle pickup.
- *Riding time*, denoted *RT*, is the average amount of time between requests boarding vehicles and alighting from them.
- *Total delay*, denoted *delay*, is the average difference between shortest path travel time from requests' origins to destinations and the actual time from request system entry to drop-off.
- *Absolute utilization*, denoted *Abs*, is the ratio of the total number of rider-minutes in the system to the total number of vehicle-operating minutes in the system.
- *Rider share rate*, denoted *rider*, is the ratio of the total number of rider-minutes to the total number of vehicle minutes while at least one rider was in the vehicle.
- The *shared rate*, denoted, *shared*, is the percentage of riders who shared their ride with another rider at any point in their journey.
- *Distance* is the total number of kilometers driven by the entire fleet of vehicles over the course of the day.

The results for experiments with 220 vehicles are shown in Table 2.2. Compared to the simulation with internal combustion engine vehicles (ICE), the benchmark method has the lowest service rate, which performs about 8% worse. All of the non

trivial methods for electric vehicles perform well, recovering around 80% of the lost service rate compared to the upper bounding ICE vehicle simulation.

The reason for the poor performance of the benchmark method is largely due to the offline time of the vehicles. Figure (2.3) shows that the vehicles charge in a single short period. Although the number of vehicles charging is capped at the number of charging stations in the simulation, the drop of the vehicle miles traveled (shown in orange) indicates congestion at the charge stations and that many vehicles are waiting offline. Vehicles wait an average of 76 minutes before they can begin charging due to vehicles running out of charge at similar times.

Table 2.1: Description and abbreviations for statistics reported from experiment.

Statistic	Abbr.	Description
Service Rate	Rate	Requests served / requests made
Waiting Time	WT	Average of pickup time minus request time
Riding time	RT	Average time in vehicle
Total delay	delay	Average ride time minus direct travel time
Abs utilization	Abs	Rider minutes / total vehicle minutes
Rider share rate	rider	Rider minutes / non-empty vehicle minutes
Shared rate	shared	Percent of riders that shared a ride
Distance	distance	Average distance vehicles travel

Table 2.2: Simulation results for 220 Vehicles. The MILP and Heuristic methods had significantly higher services rates than the benchmark. The ICE baseline has no charging and is an upper bound on service rate performance.

Method	ILP Time Limit	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE		90.99	125.96	534.68	86.77	1.35	1.63	92.06	256.68
MILP	(180 seconds)	89.19	125.93	540.15	93.23	1.34	1.67	92.99	248.85
MILP	(55 seconds)	89.37	126.71	540.74	93.62	1.34	1.67	93.17	249.33
Heuristic		89.35	126.88	541.47	94.16	1.34	1.67	93.21	248.73
Benchmark		82.56	125.3	532.56	88.39	1.22	1.64	91.83	233.22

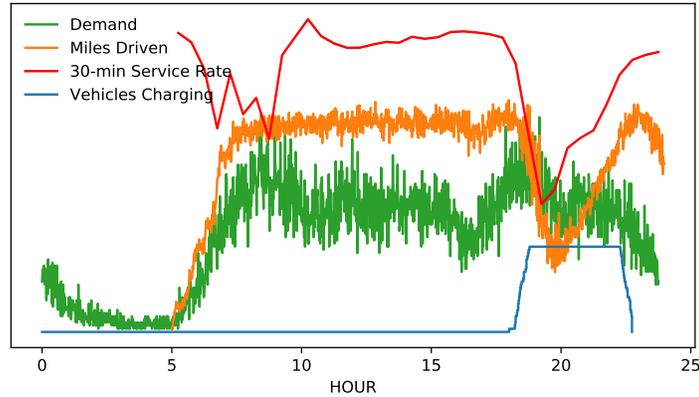


Figure 2.3: Benchmark method for 220 vehicles. Notice that vehicle charging, in blue, is concentrated during a single period in late afternoon. The number of vehicle miles traveled during this time, in orange, plummets. The 30 minute service rate during the period, shown in red, also dips resulting in a reduction of the total cumulative service rate.

The variation in performance of the heuristic and MILP methods comes down to the quality of the availability vector $A(t)$ given as an input to the problem. We see that the MILP and heuristic methods perform similarly. In Figure (2.4), we can see that avoiding congestion at charge stations played a major role in reducing the negative impact on service rate.

We can only assume the heuristic method performs similarly to the MILP methods in larger cases since the long horizon MILP cannot be solved in reasonable time.

The results of the simulation for 1000 vehicles are shown in Table 2.3. Here again we see the heuristic method outperform the benchmark method and recover more than 50% of the lost service rate as compared to the ICE vehicles simulation. Vehicles in the benchmark method spent an average of 60 minutes waiting for an

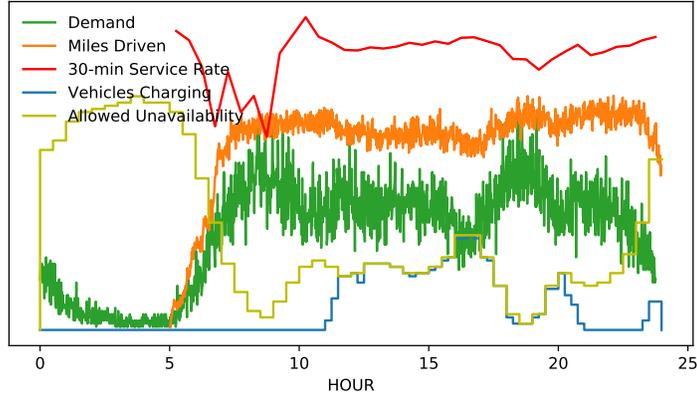


Figure 2.4: Heuristic method on 220 vehicles. The fractional availability used, shown in blue, is defined to include both charging and the 15 minute period prior to charging in which it is assumed that vehicles lack full ability to serve customers due to constraints on reaching the charging station. The maximum allowed offline availability, shown in yellow, indicates the limit by time of day of unavailable vehicles.

Table 2.3: Results for simulations with 1000 Vehicles. Compared to the benchmark, the heuristic method mitigated the loss in service rate compared to the upper bounding ICE baseline.

Method	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE	88.44	176.87	793.97	190.46	2.1	2.37	98.19	342.59
Heuristic	86.38	177.71	796.34	196.81	2.06	2.43	98.17	327.51
Benchmark	83.35	175.79	791.68	196.61	1.98	2.44	98.22	315.77

available charging station. Figures (2.5) and (2.6) show that a similar concentration of charging demand affected the benchmark method again, similar to the case with 220 vehicles. Results for simulations with 2000 vehicles, shown in Table 2.4, show the heuristic method similarly performing well with respect to the gap between the ICE method and benchmark.

There are situations, however, where the performance of the benchmark method

Table 2.4: Results for simulations with 2000 Vehicles. As with the experiments with fewer vehicles, the heuristic method has a service rate much closer to the upper bounding ICE than the benchmark algorithm.

Method	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE	93.11	171.38	781.63	174.22	2.18	2.56	98.25	331.68
Heuristic	92.51	172.71	785.88	179.93	2.18	2.61	98.21	323.7
Benchmark	90.08	171.01	781.02	180.7	2.11	2.63	98.34	313.25

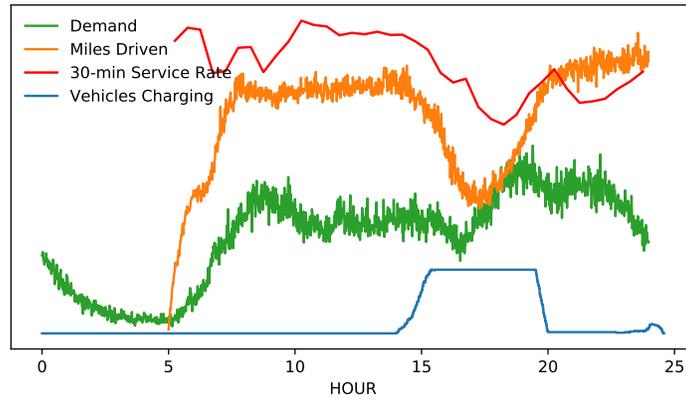


Figure 2.5: Benchmark method on 1000 vehicles. This shows that the the vehicles all needed to charge at a similar time. Some of the effect is hidden since only charging vehicles are counted in blue. A larger number of vehicles are offline waiting to be charged during that time, only visible through the great reduction in vehicle miles traveled shown in orange.

is comparable to that of the heuristic method. As an example, consider again the simulation with 1000 vehicles and suppose they are initially fully charged at midnight. The low levels of demand in the early morning do not require all vehicles be used and so the distribution of charges among vehicles is widely spread by 7am. This has the effect of preventing vehicles from needing to charge at the same time. In fact, on average vehicles only wait 5 and a half minutes before getting a charging assignment. Figure (2.7) shows this. Table 2.5 shows the results of some 220 and

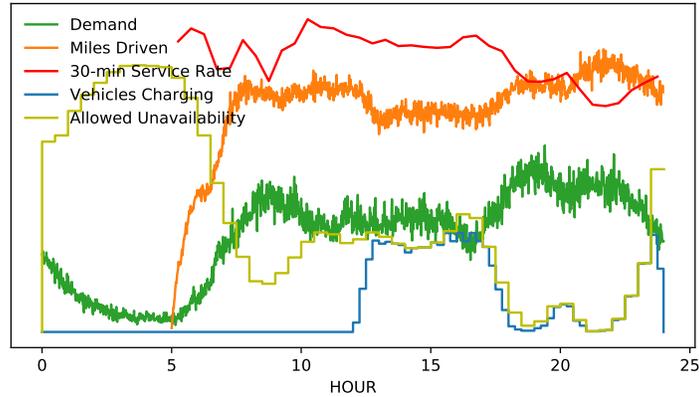


Figure 2.6: Heuristic method on 1000 vehicles. This shows that the times the vehicles charged at were better spread, as evidenced by the small reduction in vehicles miles shown in orange.

1000 vehicle simulations starting at midnight. But in practice we expect the fleet to charge overnight for the early morning and thus the simulations starting at midnight are likely not the best representative of an actual system.

In this particular case for 1000 vehicles, not only were the charging times well spread by a fortunate coincidence, most of the vehicles were in service during the peak demand period with charging levels slowly resuming as demand falls at the end of the day.

The computation times for the experiments are shown in Figure 2.6. As expected, the average computation time increased as the number of vehicles increased. In the case of 220 vehicles, the per iteration cost using the MILP long horizon method took significantly more time since there were many iterations where the solver would spend the maximum allowed time performing branch and bound. In general, the

Table 2.5: Vehicle simulations from midnight. When experiments start at midnight, the low demand in the early morning causes the distributions of charges in vehicles to spread. As a result, the service rate of the benchmark method is similar to that of the heuristic method.

Veh. Count	Method	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
220	ICE	91.06	124.92	532.58	83.93	1.11	1.59	89.98	272.93
220	Heuristic	89.59	125.95	538.75	91.0	1.1	1.63	91.25	264.23
220	Benchmark	86.79	125.25	533.4	86.34	1.06	1.6	90.51	259.96
1000	ICE	88.01	177.05	795.88	189.11	1.78	2.33	97.83	381.81
1000	Heuristic	85.73	178.23	800.15	197.43	1.74	2.39	97.86	364.01
1000	Benchmark	85.94	177.68	801.98	198.08	1.75	2.39	98.03	367.71

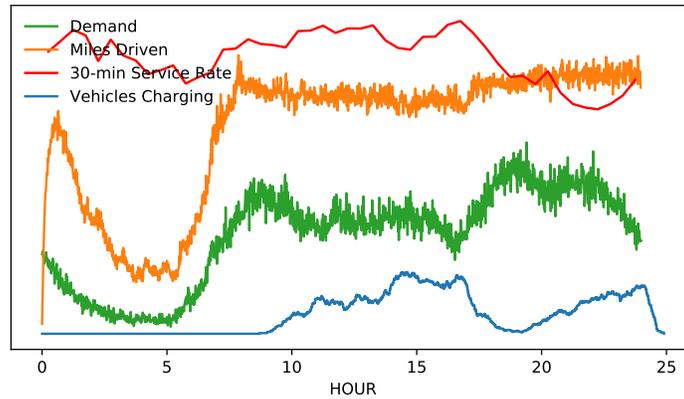


Figure 2.7: Benchmark method on 1000 vehicles starting at midnight. By coincidence it turns out that the vehicles all finish charging before the day’s peak demand, a fortunate gain for this particular run.

long horizon heuristic and benchmark method did not add significant computational overhead. In fact, there is a counterintuitive effect where the benchmark method seems to be spending less time per iteration even compared to the ICE method. The changes in time are primarily driven by changes in the length of time required to solve the passenger assignment problem, which is affected by the number of vehicles that are online and available at different times throughout the day.

Table 2.6: Average Per Iteration Computation Time. Computation time increases with number of vehicles. MILP methods take a significant amount of time. The heuristic, benchmark, and ICE methods use roughly the same time. The changes in time are primarily driven by changes in the length of time required to solve the passenger assignment problem. This is affected by the number of vehicles that are online and available at different times throughout the day and the relation of that to the number of requests that need to be assigned.

	220	1000	2000
MIP (180 sec)	4.41 sec		
MIP (55 sec)	1.95 sec		
Heuristic	0.38 sec	6.14 sec	33.3 sec
Benchmark	0.35 sec	5.31 sec	29.7 sec
ICE	0.37 sec	5.79 sec	31.9 sec

2.4.8 Sensitivity Analysis

The results presented above require many assumptions about the practical setting in which a high capacity electric ridepool fleet might operate. To add credence to the conclusions suggested by the results, here we will perturb a variety of the input parameters and discuss how the outputs change compared with the default settings. First we vary the preferred availability schedule $A(t)$, then we look at battery capacity, estimated range, number and location of charge stations, maximum distance the benchmark method can send vehicles to get to a charge station, and vehicle capacity.

Since the proposed methods rely heavily on the choice of the availability function, first we compare three different choices of $A(t)$: the preferred availability schedule, a uniform availability schedule, and an intentionally counterproductive schedule that emphasizes charging during peak periods. For the heuristic method, the service rate

Table 2.7: 220 Vehicle Simulations for Availability Sensitivity. Even in the worst case with an inverted schedule, the service rate is significantly higher than the benchmark method, at 82.6% (see Table 2.2).

$R(t)$ version	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
Typical	89.35	126.88	541.47	94.16	1.34	1.67	93.21	248.73
Uniform Demands	88.47	125.99	537.28	90.6	1.32	1.66	92.89	249.12
Inverted Schedule	88.9	125.61	539.39	92.7	1.33	1.67	92.84	247.49

Table 2.8: 1000 Vehicle Simulations for Availability Sensitivity. While generating $R(t)$ using an inverted schedule yields the lowest service rate, it still only loses 0.6% service rate compared to the typical model we developed in section 2.4.3, much smaller than the 2.5-3% gap between all of these results and the benchmark method.

$R(t)$ version	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
Typical	86.38	177.71	796.34	196.81	2.06	2.43	98.17	327.51
Uniform Demands	86.05	177.54	796.41	198.04	2.05	2.44	98.16	324.23
Inverted Schedule	85.8	176.37	793.63	196.1	2.04	2.42	98.09	325.7

is not very sensitive to the choice of availability function. As shown in Tables 2.7 and 2.8, the impact of various choices of availability function yield similar results. Choices that place too much availability at the wrong time of day do moderately worse. All experiments had sufficient availability for the system to maintain fully charged vehicles. This suggests that the majority of benefits of the heuristic charging method over the benchmark, which only served around 82.6% and 83.4% of requests in the case of 220 and 1000 vehicles, respectively, come from its ability to look ahead and appropriately spread the timing of vehicle charging. As a secondary effect, choosing the availability function with consideration of demand patterns offer smaller, positive improvements.

Next we vary the battery range used in the simulations, whose default value was 180km. In Table 2.9, simulation results are shown for three different battery

Table 2.9: Vehicle Simulations Across Various Battery Ranges. Longer battery range typically resulted in better service rate since vehicles spent less time on average offline. The benchmark method did not strictly follow this trend, however, since it is also sensitive to the demand patterns at the time of day vehicles typically need to begin charging.

Veh. Count	Method	Battery Range	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
220	Heuristic	150km	88.83	126.63	541.79	95.55	1.33	1.68	93.49	245.85
220	Heuristic	180km	89.35	126.88	541.47	94.16	1.34	1.67	93.21	248.73
220	Heuristic	210km	89.67	126.5	538.8	91.5	1.34	1.66	93.28	250.7
1000	Heuristic	150km	85.44	177.32	796.77	199.2	2.04	2.45	98.07	321.92
1000	Heuristic	180km	86.38	177.71	796.34	196.81	2.06	2.43	98.17	327.51
1000	Heuristic	210km	86.76	177.42	797.72	197.27	2.08	2.43	98.17	329.63
220	Benchmark	150km	83.03	125.93	536.04	90.74	1.23	1.66	92.85	233.46
220	Benchmark	180km	82.56	125.3	532.56	88.39	1.22	1.64	91.83	233.22
220	Benchmark	210km	84.31	125.66	532.86	89.7	1.25	1.66	92.65	235.65
1000	Benchmark	150km	82.63	176.1	793.19	199.75	1.97	2.46	98.38	310.12
1000	Benchmark	180km	83.35	175.79	791.68	196.61	1.98	2.44	98.22	315.77
1000	Benchmark	210km	82.9	175.88	788.48	194.79	1.96	2.4	98.17	317.72

capacities: 150km, 180km, and 210km. In general, longer battery ranges resulted in better performance. This is not surprising since we know that ICE vehicles, effectively vehicles with infinite charge, have the highest performance. There are some unexpected trends in the data under the benchmark method. While the general trend preferring longer battery range appears for the high and low end of the battery range, with the default 180km we do not get strictly better or worse performance than a longer or shorter battery capacity. This is likely due to the effects of timing since the benchmark method is sensitive to the dictates of the timing of battery depletion. If by coincidence the vehicles run out of charge during a period of lighter demand there is less impact than if they run out during high demand. So in some sense, the battery capacity determines the time of day the vehicle runs out of power. The heuristic methods seem less dependent on this and thus has fairly steady variations in performance when the range is perturbed.

Table 2.10: Analysis of Heuristic Method at Various Estimated Battery Lives. Estimated battery life has a weak effect on service rate.

Veh. Count	Battery Life Estimate	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
220	11hr	89.18	126.3	541.28	94.45	1.34	1.67	93.4	248.6
220	15hr	89.56	125.93	540.68	93.59	1.34	1.66	93.53	250.35
1000	8hr	85.79	177.6	796.32	198.08	2.05	2.44	98.12	323.64
1000	12hr	86.6	177.43	796.91	197.07	2.07	2.43	98.13	328.27

Next, recall the heuristic method is also dependent on the estimate for the expected duration that vehicle batteries last for. In Table 2.10 we compare the default 13 hour estimate for 220 vehicle and default 10 hour estimate for 1000 vehicles used in the simulations, as discussed in Section 2.4.5, with a variation of 2 hours, up and down. The sensitivity appeared fairly low. Although service rate increases with longer expected battery duration, recall that this means that it is more likely that vehicles will approach true zero charge, which is below what we consider to be zero in the planning model, based on the assumption of a buffer (Assumption 2.2.1).

We consider sensitivity of the system to the placement of charge station infrastructure in two ways. First, to test the sensitivity of the default charge station location layout, we compare the scenarios with 220 vehicles and 1000 vehicles to a layout made by a simple greedy algorithm that places stations of capacity one. First we randomly place the first charge station, then subsequently place stations so that the each placed station maximizes the distance to the nearest placed charge station. The results are shown in Tables 2.11 and 2.12, which order the result in the same order as they were presented for the default charge station placement scenarios shown in Tables 2.2 and 2.3, respectively. Generally, the performance of the heuristic and MILP methods we not affected by the change in charge station placement; in the

Table 2.11: Results for 220 vehicles when charge stations are placed using a greedy algorithm. The benchmark method increased in performance by about 0.6%, but the heuristic and MILP methods are largely unchanged. See Table 2.2 to compare these results with the default charge station layout.

Method	ILP Time Limit	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE		90.99	125.96	534.68	86.77	1.35	1.63	92.06	256.68
MILP	(180 seconds)	89.36	126.45	540.86	93.36	1.34	1.67	93.24	247.78
MILP	(55 seconds)	89.36	126.45	540.86	93.36	1.34	1.67	93.24	247.78
Heuristic		89.46	126.36	540.85	93.49	1.34	1.67	93.31	248.3
Benchmark		83.15	125.44	530.94	87.46	1.22	1.64	91.72	233.36

Table 2.12: Results for 1000 vehicle when charge stations are placed using a greedy algorithm. Both the heuristic and benchmark methods have lower service rate when compared to the default charge station layout, shown in Table 2.3, with the benchmark method impacted slightly more.

Method	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE	88.44	176.87	793.97	190.46	2.1	2.37	98.19	342.59
Heuristic	86.0	177.88	795.7	196.18	2.05	2.43	98.09	328.53
Benchmark	82.84	176.46	792.51	196.66	1.97	2.43	98.18	317.52

case of 220 vehicles the service rate stayed roughly the same or increased marginally, in the case of 1000 vehicle the service rate decreased marginally. The benchmark method had a slightly larger change benefiting by about 0.6% in the case of 220 vehicles and losing about 0.5% in the case of 1000 vehicles. In all cases, the heuristic and MILP methods outperformed the benchmark method.

Second, we consider the sensitivity of the methods to the number of available charge stations. In the above experiments there were 101 stations available for the 1000 vehicles. In Table 2.13 we compare the outputs from the experiments had that number been reduced to 90 charge stations, using the same procedure as outlined in section 2.4.1 for placement. The heuristic method is hardly affected by this change

Table 2.13: 1000 Vehicle Simulations with Reduced Station Count. Reducing the number of charging stations has a larger impact on the service rate of the benchmark algorithm than the heuristic method.

Method	Charge Station Count	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
Heuristic	Typical 101	86.38	177.71	796.34	196.81	2.06	2.43	98.17	327.51
Heuristic	90	86.23	177.38	796.79	197.19	2.06	2.43	98.15	327.52
Benchmark	Typical 101	83.35	175.79	791.68	196.61	1.98	2.44	98.22	315.77
Benchmark	90	82.12	175.87	790.67	196.65	1.94	2.44	98.22	311.64

Table 2.14: Comparison of Benchmark Method Maximum Charge Station Travel Time. Allowing the greedy station assignment algorithm to assign vehicles to stations nearer or farther away did not change the service rate enough to be competitive with the heuristic method. See Tables 2.2 and 2.3.

Veh. Count	Greedy Search Bound	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
220	600 sec	80.9	125.48	532.08	89.13	1.19	1.65	92.13	225.03
220	1200 sec	82.48	125.42	532.15	87.97	1.22	1.64	91.79	233.82
1000	600 sec	83.28	176.09	792.16	196.9	1.98	2.44	98.25	314.69
1000	1200 sec	83.3	176.08	791.89	196.91	1.98	2.44	98.2	316.73

while the benchmark method loses more than a full percent of service rate under reduced stations. This is expected since the greedy assignment algorithm will not be able to give all vehicles that need to charge immediately available assignments while the heuristic can look ahead and avoid congestion situations all together.

Next, we look at a parameter the benchmark method has for the maximum distance between a vehicle’s position and the charge station it is assigned to. This parameter controls the trade off between keeping vehicles near areas with high demand and making sufficient charging options available. We use a default of 900 seconds of travel time, but we compare this with 600 second and 1200 second options. The choice of 900 seconds outperformed both the 600 and 1200 second bound. See Table (2.14).

Table 2.15: Results for 220 vehicles when capacity is reduced to 4. Compare to Table 2.2 which shows the same simulations when vehicle capacity is 10. The ICE method suffers little impact from the change in capacity, but the other methods lose around a percentage point of service rate.

Method	ILP Time Limit	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE		90.32	125.96	535.6	213.93	1.07	1.76	91.76	262.55
MILP	(180 Seconds)	88.15	126.67	541.04	221.03	1.05	1.8	92.72	253.9
MILP	(55 Seconds)	88.22	126.89	542.11	222.36	1.05	1.81	93.08	253.85
Heuristic		88.65	126.23	541.01	220.95	1.06	1.74	92.88	254.69
Benchmark		81.72	125.43	533.75	214.76	0.96	1.79	91.42	239.56

Table 2.16: Results for 1000 vehicles when capacity is reduced to 4. Compare to Table 2.3 which shows the same simulations when vehicle capacity is 10. Roughly all methods in this case were negatively impacted by the lower capacity, typically losing about two and a half percentage points of service rate.

Method	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE	86.17	183.73	784.4	369.87	1.62	1.83	97.57	345.8
Heuristic	83.99	183.99	786.45	376.5	1.58	1.99	97.61	329.38
Benchmark	80.83	183.21	780.52	374.55	1.51	2.4	97.58	318.73

Finally, we consider the sensitivity of the system to our choice of using capacity 10 vehicles by comparing against scenarios where the capacity is reduced to 4. We compared the vehicle capacity across all vehicles counts, shown in Tables 2.15, 2.16, and 2.17. As expected, the service decreases when vehicle have lower capacity. The impact of lower capacity was strongest among simulations with larger fleet sizes. The general relationships between the ICE, MILP, heuristic, and benchmark methods appears to be the same in the capacity 4 case as it was in the default capacity 10 case.

In summary, the sensitivity analysis has shown that changes in the input parameters can make marginal changes to the performance of the system in terms of

Table 2.17: Results for 2000 vehicles when capacity is reduced to 4. Compare to Table 2.4 which shows the same simulations when vehicle capacity is 10. All methods were negatively impacted by the lower capacity, with the ICE method taking the small hit to service rate with a 3.4% loss, and the benchmark method taking the worst hit with a 4.5% loss in service rate.

Method	Rate	WT	RT	Delay	Abs	Rider	Shared	Distance
ICE	89.73	180.95	770.48	349.7	1.66	2.07	97.71	338.21
Heuristic	88.74	183.06	775.29	358.8	1.65	2.56	97.76	328
Benchmark	85.57	182.14	768.51	358.11	1.58	2.48	97.76	315.49

service rate. However, despite the differences in performances under the various perturbations a few features remained constant. First, the ICE method always served as an upper bound for the performance of any system that required a charging scheme. Second, the benchmark method was always outperformed by the MILP and heuristic methods, supporting the intuition that algorithms that foresee future conditions can be beneficial in electric ridepool systems.

2.5 Conclusions

Results from experiments demonstrated that high-capacity ridepool systems can be run at scale with electric vehicles in real time. Unlike the case with ICE vehicles which do not need refueling over the course of the day, the requirement for electric vehicles to recharge added a new set of constraints to the already hard problem of high-capacity ridepooling. This work presented a two-phase method that allowed the operator to manage the fleet by separating the problem of passenger assignment and charge scheduling into separate optimization procedures. While our method

required estimating new parameters, such as estimates of daily vehicle availability requirements, reasonable estimates we ran experiments on performed similarly and well suggesting such an approach may work well in a real-world setting.

The benefits of the methods presented are significant since strategies that act purely online without making any estimates of future demand, such as the benchmark method studied here, pose risk when the distribution of charges is initially uniform across vehicles. These risks are most prominent when the entire fleet is fully charged at the beginning of the work day, though experimental results showed that night service can sometimes mitigate this risk by causing the charge in vehicles to spread. The spread caused by night time service is different from regular service as the variability is not caused by the vehicles traveling on roads with different speeds but rather by the large portion of downtime experienced by the vehicles. In cases when night traffic is especially low, as seen in the experiments with 220 vehicles, this effect is smaller. In realistic settings, operators would likely use dynamic fleet sizing which would concentrate utilization of vehicles and again raise similar risks as with the fully charged fleet.

Although our experiments assumed the distribution of vehicle miles traveled is uniform over the course of the day, it may be beneficial not to make this assumption. For example, if one were to operate a system 24 hours a day, the night time may be better accounted for with lower expected battery usage. This change can easily be implemented in our framework since expected battery usage is entirely accounted for per-period; one could estimate and utilize separate $q_{est,t}$ in the long horizon MILP

for each time period.

If the operator rents rather than owns charge station capacity, the model can further be adapted to allow for different numbers of vehicles to be assigned at different times of day. This implementation has the caveat that the long horizon problem will only take into account whether the schedule is feasible for the total number of vehicles charging, not whether the schedule is feasible for the particular set of stations that are available at different times. For example, suppose that before noon the operator has access to one set of stations and then after noon they have access to a mutually exclusive set of stations. The solution to the long horizon problem may still schedule a vehicle to charge from 11:45 until 12:15 since sufficient capacity seems to exist, but it is not feasible since no station assignment could accomplish that. While this problem can be avoided in both the heuristic and MILP formulations by forbidding scheduling vehicle times that cross certain time boundaries, this solution is too strong for most cases since it is rare that the time constraint would apply to all charging stations at a single time step.

In the future we would like to extend the model to allow for dynamic fleet sizing, where vehicles that do not need to be available are allowed to stop. The model used in this chapter assumes that vehicles must always be able to be in service unless they are charging. But in practice it may be advantageous to have the capability to keep more vehicles online during busy hours while having fewer vehicles available during off peak hours.

2.6 Acknowledgements

The authors would like to thank Vindula Jayawardana for his help in coding the original simulator that was extended to produce the computational results in this chapter.

CHAPTER 3
CENTRALLY CONTROLLED RIDEPOOL ALGORITHMS:
EFFICIENCY AND PERFORMANCE

3.1 Introduction

Ridepooling is a growing and popular form of urban transportation which aims to overcome the efficiency limitations of taxi-like, single-rider systems by grouping multiple riders with similar trips together on the same vehicle. From an operations perspective, ridepool systems can have varying levels of central control, with some platforms creating a two-sided market between passengers and vehicle operators and others simply providing an interface for passengers to interact with a centrally controlled system. In either case, a central decision maker has to suggest or make assignments between the passengers and the vehicles in order to achieve an operational objective. In the case of on-demand, centrally controlled systems there has been much recent work proposing a variety of methods [3, 43, 53, 58, 70].

The basic setting is as follows: an operator has a fleet of vehicles under their control. In real time, requests are sent to the operator for trips that give an origin and destination pair. The requests wish to complete their journey as soon as possible and can be served immediately following the submission of their request. The operator in these scenarios has two options: either guarantee the passenger a trip that meets certain requirements, such as maximum time until pick up or maximum on-board

delay, or otherwise reject the passenger from the system. These requirements are generally referred to as quality of service (QoS) constraints. The requirement that trips satisfy certain criteria is an advantage computationally and leads to popular performance metrics such as the service rate, also known as the percentage of requests served.

This chapter focuses on the assignment process of ridepooling, deciding which requests get paired with which vehicles. In this setting, since all requests that are assigned must be given a trip satisfying QoS constraints it is common for the results of these methods to be presented as the service rate at the day level, among other metrics, reporting the number of feasible trips assigned. While metrics are reported at the day level, since assignments are made on-demand throughout the day this means that the performance of each of the methods is more than just a function of the quality of the solution in each step in time. Rather, it includes an unquantified effect resulting from the sequence of assignments made over the full day, with the system state at each time being a function of the solution found in the previous epoch. The hallmark of a good method by this metric would be serving the highest percentage of people during the day or achieving a good service rate with small computation time.

There is a wide spectrum of methods in the literature that span the tradeoffs between run-time and performance, ranging from greedy algorithms to full enumeration techniques. Full enumeration in this context refers to finding all possible trips in the Request-Trip-Vehicle (RTV) type decomposition for ridepool assignment algorithms,

a category we will restrict ourselves to in this chapter. RTV based algorithms are a method of decomposing the assignment problem by first generating candidate combinations of requests and vehicles, called trips, and then in a second step choosing the combination of trips to use in the final solution. If each of the two steps are performed exactly, then this is equivalent to solving the complete assignment problem. The methods we study will frequently use heuristics for many of these steps. In contrast to RTV based algorithms, methods we do not study include those that define the assignment (and implicitly routing) problem over a graph nor those that use other types of decompositions, such as the zone-path model of [42].

Despite all the work developing methods from many directions to solve ridepool problems, there are gaps left by these works that may be helpful for algorithm researchers to address. First, the methods presented by different authors are difficult to compare to each other in terms of performance. While many authors report algorithmic performance in terms of service rate, this by itself is not sufficient to make comparisons. By using different data sets and running tests on different cities it becomes difficult to rule out the impact of the demand profiles on the ultimate results as it's possible that certain demand patterns are more compatible with certain algorithmic procedures. Even when authors make direct comparisons of their methods with others from the literature it can be difficult to determine whether authors are interpreting and implementing methods the same as each other or are solving for the same objectives. And in fact, in many cases, they are not.

Part of the reason authors interpret and implement each other's methods differ-

ently is that publications often do not have space to discuss all of the nuances of the problem that would be necessary to exactly replicate the results. From the perspective of researchers or engineers that wish to implement or use any of these methods, replicating much of the existing literature requires searching for additional details or making assumptions about how certain subproblems required in the methods are being solved.

We address these problems in this study in several ways. First, the software tool used to make all of the results presented here will be released publicly. This should be considered a best practice, and is done by some authors, as the code itself serves as a complete description of the methods tested. Next, we describe several methods and place them side by side in the same simulation environment to make direct comparisons and study features of the results.

We also make new contributions to the literature motivated by the direct comparisons. After reviewing several methods from the literature, we present some extensions of weighted bipartite matching based methods and demonstrate that they can yield very similar performance to full enumeration based methods. In fact, we show that many of the existing methods yield similar performance as well. There is currently no work in the literature that studies why the performance of many of the proposed methods have performance at the day level that is much more similar than would be expected at the single time period level. Meaning, there have not been studies about the way these methods affect themselves on an epoch-by-epoch basis. By running these methods in a controlled, offline settings we can demonstrate that

the effect of time has a significant impact on these full day results. In a synthetic demand environment, we show evidence that these methods may have similar performance since the system might have a throughput barrier, whose value depends on several factors.

While the main focus of this chapter is on myopic versions of the assignment problem, as a consequence of this the results over the course of a full day may not be optimal even when running exact algorithms for each time period. Having shown that there might be a throughput barrier in ridepool systems, we next provide analysis that both tries to quantify the gap between non-myopic solutions compared to myopic solutions and that shows that incorporating forecasts of the future into the assignment process may raise the throughput barrier. It is not in the scope of this chapter to propose algorithms that close the myopic/non-myopic gap, but the interested reader may refer to work such as [42] to see example approaches that try to overcome this obstacle. However, we argue that this is the best direction for future research on these types of assignment algorithms.

In short, this chapter makes several contributions to the ridepool literature. First, we provide comparable benchmarks of some of the major methods in the literature used for ridepooling. We also propose new methods that aim to provide a lower runtime solution that yields similar service rates to slower, full enumeration style algorithms. In addition, we provide the first analysis we are aware of comparing the methods directly in solving the assignment problems on a per-iteration basis rather than only summarizing performance over the course of an entire day. We

then provide in-depth analysis of the solutions produced by the methods to explain why there is a performance disconnect between the offline and online problems. For each of the methods we test, we provide detailed descriptions and discuss some of the subproblems required to implement them, such as the function used to find an optimal route for a vehicle given a candidate set of passengers.

The chapter is organized as follows: Section 3.3 introduces the ridepool problem setting, assumptions, and nomenclature used in the chapter. Section 3.4 formally defines several assignment methods from the literature, introduces a new assignment method based on very large neighborhood search, and discusses the intuition behind them. Section 3.5.1 provides a discussion of solution methods for the constrained traveling salesman problem, an essential component of all ridepool assignment algorithms. Section 3.6 introduces the testing framework we use to run all experiments in this chapter. The source code for the framework is available online at GitHub. Section 3.7 evaluates the performance of each of the methods both in full-day, online settings and in an offline, iteration-per-iteration comparison. It also looks at the impact of optimizing service rate vs miles traveled and uses a numerical study to answer the question of why a wide variety of methods proposed in the literature perform similarly and well. Finally, Section 3.8 investigates the cost of being myopic in ridepool problems.

3.2 Literature Review

The VRP problem has been studied for a long time as its applications are more general than that of the ridepooling problem. Early techniques used to solve the ridepooling problem include greedy solutions [73] and [61] and MILP formulations that include the road network graph as some of the variables [39] and [28]. An important shift in approach used for ridepooling problems came with the creation of the shareability graph in [60] which suggested a view of feasibility of rider groupings as a separate step from optimal matching selection.

Papers have shown a wide variety of methods for this problem. A scalable but greedy solution was proposed in [53] which considers requests in chronological order by arrival and greedily searches for the best match. Much work stemmed from the seminal paper of [3] which presented full framework for a ridepool system, including vehicle-passenger assignments as well as an unassigned vehicle rebalancing algorithm. Other methods for a similar framework that are particularly relevant for this chapter include the work of [70] which uses a similar assignment system but reduces the computational demand of the solution by reducing the possible matching graph to a bipartite matching problem. In addition, [58] use a column generation heuristic to select candidate trips that exploits the tightness of the LP relaxation of the assignment problem. Building off these works, we also refer to general techniques for improving solutions for hard problems by building off work of [1] on very large neighborhood search methods.

There are also works that are mostly similar such as [42] which use an alternative computational heuristic that simplifies path construction and selection, but we don't include them in the comparison of methods here since they do not guarantee that the QoS constraints will be satisfied.

An important part of the assignment problem is handling the subproblem associated with ordering pick ups/drop offs and checking feasibility. Some literature places significant focus on this portion of the problem. The work of T-Share [45] focuses on solving the shortest path problem from the perspective of shortest paths. They come up with a heuristic/algorithm for quickly finding the best vehicle to match with a request by using a spatio-temporal index. The work of [71] discusses search procedures for finding paths using insertion heuristics. While not optimal, they are practical and the procedure has low complexity. The work of [67] focuses on constructing valid ridepool paths through merge operations. They start by identifying the shortest paths for each request and then build trip sets through sequences of merge operations.

There has been some work on identifying competitive algorithms for myopic settings and work that tries to use machine learning to improve myopic algorithms. Rather than using future information to change the final assignment stage, [81] instead seeks to serve requests without using shortest paths - biasing paths to follow routes that are expected to enable future matching of demand. The challenges faced by myopic algorithms are discussed directly in [5] which studies matching pairs of requests in time windows with differing matching values. They show that the best

possible competitive ratio in such a setting is $1/2$. To overcome the challenges of the myopic nature of the assignment problem, [62] use neural approximate dynamic programming to modify the objective costs associated with candidate trips. Historical information has also been leveraged directly in the method proposed by [4] which incorporates virtual future requests into the optimization procedure. The virtual requests are intended to guide the system towards more productive paths and assignments. Although using a slightly different framework, a follow up [43] to the zone path model presents adaptations for a non-myopic optimization procedure.

3.3 Preliminaries

3.3.1 Problem Description

We are given a road network $G = (N, A)$ with distance metric $d(\cdot, \cdot)$, a system starting time and end time, and a collection of centrally controlled vehicles V , each associated with a capacity v_c . Each vehicle has an initial position $v_n \in N$ and is assumed to be empty. Vehicles can only move to nodes in the network adjacent to their current location. When a vehicle moves from some node a to b with travel time $d(a, b)$, we say the vehicle is at node b with offset $0 < t \leq d(a, b)$ describing the time remaining before the vehicle reaches node b . Vehicles can only pickup and drop off requests at nodes in the network.

Not visible to the operator, there is a set of requests $r \in \mathcal{R}$ that spans all requests

made to the system over the entire day. Each request $r = (o, d, t_e, l_b, l_a)$ is defined by an origin $o \in N$, a destination $d \in N$, a time the request becomes visible to the operator t_e , the latest time the request will tolerate boarding l_b , and the latest time the request will tolerate being dropped off l_a . For simplicity, we often choose l_b and l_a in terms of the request's direct travel time: $l_b = t_e + t_b^{\max}$, $l_a = t_e + d(o, d) + t_a^{\max}$. In this case, t_b^{\max} is referred to as the maximum waiting time and t_a^{\max} is referred to as the maximum total delay, which includes the waiting time.

The day is partitioned into a set of decision epochs, typically uniformly spaced. At each decision epoch $e \in E$, the operator can view all requests that have become visible at that time or earlier, formally $\mathcal{R}_e = \{r \in \mathcal{R} \mid t_e(r) \leq e \leq l_b(r)\}$. The operator then makes routing decisions for the vehicles and assigns each requests $r \in \mathcal{R}_e$ to a vehicle, reassigns them to different vehicles if allowed to, or leaves them unassigned. Assignment of a request to a vehicle can happen under any routing plan the operator chooses that satisfies vehicle capacity constraints, and each request's constraints l_b and l_a . The operator makes these decisions in order to maximize a private objective function. Given a notification window t_n^{\max} , any request r that is not given an assignment before time $t_e(r) + t_n^{\max}$ is permanently rejected from the system (because we are unable to serve them). We typically assume that t_n^{\max} is equal to the epoch length so that all requests not assigned in an epoch are automatically rejected. In deterministic settings, if we are unable to guarantee a vehicle assignment to a request in one iteration, the same will be true in all future iterations as well.¹

¹In deterministic settings, we can compute the future states of all vehicles under every possible assignment. If we find we cannot serve a request in one iteration, no new assignment possibilities will appear in future iterations and so the request will still not be assigned.

In this setting, the operator’s problem is to determine how to make vehicle assignments to requests and how to route vehicles at each epoch in order to maximize their objective over the span of the day, though they can only observe their objective in that epoch.

3.3.2 Common Concepts and Objectives

Routing instructions to vehicles are referred to as trips. A trip is defined by a vehicle, and an ordered set of NodeStops to visit. A NodeStop consists of a location node, and possibly other data such as a requests to pickup or drop off, or in some instances a delay time (in order to allow routing instructions to be general). Under most circumstances, since requests are available to be picked up as soon as we are aware of their existence, there is no value in delaying at a location. In some special circumstances, such as model extensions that allow requests to walk to other nodes to be picked up, delay may be necessary to wait for requests to be available for pickup at a particular place.

We classify trips in two ways: those that serve requests, and those that do not. Trips without requests are called rebalancing trips. These trips are used to move vehicles that are not assigned any requests to other locations with the intention that the new location is more likely to yield a profitable match with requests in future epochs.

Assumption 3.3.1. *The operator’s objective function is separable as the sum of*

objectives over each vehicle.

In this chapter, we assume that the operator’s objective function is separable across vehicles. For example, the objective may be a reward for the total number of requests served minus a penalty for vehicle minutes travelled serving those requests. This would be separated across vehicles as the reward for the number of requests assigned to the individual vehicle minus a penalty for the number of vehicle minutes travelled serving those requests. The advantage of Assumption 3.3.1 is that we can now compute the optimal routes for vehicles independently of each other. In order to find a route, it is sufficient to be given a vehicle and a set of requests. Because of this, we often denote trips using the notation $t = (v, \bar{r})$ where $v \in V$ is the vehicle, and $\bar{r} \subseteq \mathcal{R}_e$ is the set of requests. While this is a common assumption used in the literature, it leads to certain limitations. For example, travel times have to be exogenous and cannot depend on other trip assignments since that would make the objective non-separable.

Since the costs of rebalancing are often omitted from the operator’s objective, under certain objective functions, such as vehicle miles traveled, the optimal assignment of requests to vehicles may not be the same across epochs even if no new requests become available. This is formally shown in Lemma 3.3.1. This issue can appear in many methods presented in the literature since rebalancing is often performed as a separate problem from request assignments.

Remark 3.3.1 (Objective Instability from Rebalancing). *Assume that the operator’s objective function does not include the cost of rebalancing. Then there exist objective*

functions for which the optimal assignments may change between consecutive epochs even if no new requests are available.

To gain intuition for this, consider a request that can be assigned to two vehicles. Perhaps one of the vehicles is farther away but is assigned to this request since the extra cost of assignment is lower due to a passenger already on board the vehicle. Suppose that the closer vehicle then starts rebalancing towards a nearby point. After enough iterations pass, the closer vehicle is so close that it can serve the request with less additional cost than the vehicle it was already assigned to and therefore the assignments change. In essence, the price of moving the vehicle was paid by the system even though the assignment problem did not account for it. In other words, the objective value of the assignment problem was wrongly discounted and this caused the solution to change. A formal proof of this remark is included in the Appendix.

Proof of Remark 3.3.1. Consider a network defined by a rectangular lattice with uniform speeds. Suppose there is a request that wishes to travel from $(0, 0)$ to $(0, 1)$. Vehicle A is located at $(-3, 0)$ carrying one passenger destined for location $(2, 0)$. Vehicle B is empty and is located at $(0, -3)$. If the new request is assigned to vehicle A , the vehicle would incur an extra 3 units of travel distance. If assigned to vehicle B , the vehicle would incur an extra 4 units of travel distance. The optimal solution is to assign the request to vehicle A .

During the rebalancing phase, assume that vehicle B is assigned to move towards

$(0, 0)$.

At the next decision epoch, both vehicles have moved two units. Vehicle A is now at $(-1, 0)$ and vehicle B is at $(0, -1)$. Consider changing the request's assignment from vehicle A to vehicle B . Unassigning the request will reduce vehicle A 's travel distance by 3 units. The cost of assigning the request to vehicle B is now only 2. So swapping the request to vehicle B will result in a savings of one unit of total distance. Since the optimal assignment has changed without any new requests arriving, the claim has been proven. \square

3.4 Formulations and Assignment Algorithms

In this section we first review an exact algorithm for solving the epoch-wise ride-pooling problem, as formulated in [3]. As the size of the exact formulation can be intractable to solve, we identify some heuristics, including those used in [3], that can be used in conjunction with the algorithm. Striking the right balance between achieving the best objective at each epoch computation time can be a difficult tradeoff. Beyond just applying heuristics to the exact algorithm, some heuristics are methods in their own right. After the exact algorithm we will present several non-exact approaches to the problem that are similarly used to reduce computational burden. The first alternate approach we cover uses column generation on a linear relaxation of the assignment problem to find a promising set of candidate trips. Then we cover a variety of approaches based on bipartite matching with extensions for locally im-

proving the solution. Finally, we cover an methods that combines efficient matching based assignment with a sophisticated local improvement algorithm.

3.4.1 Exact Formulations

The first set of methods we study are based on exact formulations. Even when heuristics are used to solve the resulting problem, a good solution for the model is necessarily a good solution for the assignment problem in that particular epoch. This will contrast with the formulations studied in the next subsection which solve inexact models of the assignment problem. While the optimal solutions to the exact formulation give the best epoch-wise assignment, the computational effort required to find them can make alternative heuristics attractive in practice.

Epoch-Wise Exact Solution via RTV Graphs

In the Epoch-Wise Exact Solution, at each epoch we choose the assignment of requests to vehicles that maximizes our reward. Because of the Separability Assumption (3.3.1), it is sufficient to use a decomposition of the problem that enumerates all feasible trips $t = (v, \bar{r})$ and then selects a set of trips of maximum value. Given a set of trips, the optimal choice of trips can be determined through an integer linear program (ILP).

Problem Formulation. The objective, as presented by [3], assumes that the cost function is the sum of the the cost for making the assignment to the vehicles and the cost of rejecting the passengers that were not matched. Assuming that the cost of rejecting each passenger is M , this means the reward for selecting a trip $t = (v, \bar{r})$ is $w_t = M|\bar{r}| - c_t$. In this formulation the cost for rejecting requests is separated from the particular trip that serves it, though it could also be written as a reward maximization problem by ignoring this separation. Written as a minimization problem, the separated formulation is

$$\begin{aligned}
& \min_x \sum_{t \in \text{Trips}} c_t x_t + M \sum_r \epsilon_r \\
& \text{subject to } \sum_{\substack{t \in \text{Trips}: \\ v \in t}} x_t \leq 1 & \quad \forall v \in V \\
& \sum_{\substack{t \in \text{Trips}: \\ r \in t}} x_t + \epsilon_r = 1 & \quad \forall r \in \mathcal{R}_e \\
& x \in \{0, 1\}^{|\text{Trips}|}, \epsilon \in \{0, 1\}^{|\mathcal{R}_e|}.
\end{aligned} \tag{3.1}$$

Solution method. While the formulation is NP-hard (since it generalizes weighted set packing), in practice commercial solvers can find near-optimal solutions to real-world size instances. What would remain for solving the higher-level, decomposed problem is to determine the set of trips that are given as inputs to the optimization problem.

An exact approach for doing this was presented by [3]. They note that due to QoS constraints for requests (l_b, l_a) , the set of possible trips can be feasibly enumerated. They produce the full set trips using the following three steps:

1. Create Request-Request graph on the node set \mathcal{R}_e . An edge is added between pairs of requests if the two requests could in theory be served by the same vehicle, assuming that a hypothetical empty vehicle could be placed in an ideal location. This is the same as the shareability graph introduced in [60].
2. Create Vehicle-Request graph on node set V and \mathcal{R}_e . An edge is added between a vehicle v and a request r if the trip $t = (v, \{r\})$ is feasible. Passengers already on the vehicle are considered implicitly.
3. Let the Request-Trip-Vehicle (RTV) graph be defined by starting with the union of the Vehicle-Request graph and Request-Request graph. Begin by defining a trip for each edge connecting a request to a vehicle, effectively enumerating all trips with exactly one request. To enumerate all trips with $n \geq 2$ requests, consider all pairs of trips with $n - 1$ requests associated with the same vehicle. If the union of the requests \bar{r} contains n requests and the set of nodes $\{v\} \cup \bar{r}$ forms a clique in the RTV graph, add the trip to the list of feasible trips if a feasible routing exists.

While using the RTV graph is not strictly necessary to build the set of trips, checking if a trip corresponds to a clique in the RTV graph can help identify trips that will be infeasible without needing to perform expensive feasibility checks using many more constrained traveling salesman problems. It also creates a natural termination condition for the algorithm.

The Full-RTV method, without any algorithmic optimizations, is described in Algorithm 7.

Algorithm 7 Full-RTV Assignment(V, R)

- 1: $Rr \leftarrow \{(r_1, r_2) \mid r_1, r_2 \in \mathcal{R}_e, v^* \text{ empty}, loc(v^*) \in \{o(r_1), o(r_2)\}, c(v, \{r_1, r_2\}) < \infty\}$. \triangleright RR Graph
- 2: $Vr \leftarrow \{(v, r) \mid v \in V, r \in \mathcal{R}_e, c(t = (v, \{r\})) < \infty\}$. \triangleright Vehicle-Request Graph
- 3: $Rtv \leftarrow Rr \cup Vr$.
- 4: $\text{Trips} : \mathbb{N} \rightarrow \{\text{Trip}\} = \{\}$.
- 5: $\text{Trips}[0] \leftarrow \{(v, \emptyset) \mid v \in V\}$.
- 6: $\text{Trips}[1] \leftarrow Vr$.
- 7: $k \leftarrow 1$.
- 8: **while** $|\text{Trips}[k]| > 0$ **do**
- 9: $\text{Trips}[k+1] \leftarrow \{t' = (v, \bar{r}_1 \cup \bar{r}_2) \mid (v, \bar{r}_1), (v, \bar{r}_2) \in \text{Trips}[k], |\bar{r}_1 \cup \bar{r}_2| = k + 1, \{v\} \cup \bar{r} \text{ a clique}, c(t) < \infty\}$.
- 10: $k \leftarrow k + 1$.
- 11: **end while**
- 12: $P = \bigcup_k \text{Trips}[k]$.
- 13: $x \leftarrow \text{opt}(P)$.
- 14: $T : V \rightarrow \text{Trip} = \{\}$.
- 15: **for** $t = (v, \bar{r}) \in P$ **do**
- 16: **if** $x_t = 1$ **then**
- 17: $T[v] = t$.
- 18: **end if**
- 19: **end for**
- 20: **return** T .

The need for heuristics. While vehicle routing problems are hard to solve in general, the reason this problem is tractable in practice for ridepooling settings is because the quality of service constraints, latest boarding and alighting time constraints (l_b and l_a), considerably limit the number of feasible trips. Despite this, in the case of larger instances, producing an exact solution using Full-RTV is impractical because these constraints do not prune the number of trips enough. In order to combat these problems, there are several places where exact computations in the Full-RTV method can be replaced by heuristic approximations, though the solution

is no longer guaranteed to be epoch-wise optimal. Several such heuristics found in the literature can be used to reduce the solving time:

1. Use a heuristic CTSP solver to determine feasibility of trips, especially when large. This heuristic is general and can be useful for reducing computation time of any trip-based method. See, for example, [3].
2. Create only an approximation of the Request-Request graph [3, 25].
3. Create only an approximation of the Vehicle-Request graph [3].
4. Enforce computing time limits on trip generation by fixing a maximum search time or a maximum number of queries [3].
5. Under computational limits, build the trip list in a heuristic order that increases the chances of good trips being found within the time budget [58].
6. Don't allow requests to be reassigned to other vehicles. Compute RR, VR, and RTV graphs only using newly observed, unassigned requests. Let previously assigned requests be implicit in the cost of routing each vehicle even if they have not yet boarded [70].

In our numerical results, we use heuristic 1 in all cases. While we don't use heuristic (2), heuristic (3) is used for building RTV graphs in all results except for the Section 3.8 where we study non-myopic settings. When heuristic (4) is applied to the RTV graph generation process we refer to the resulting method as Fast-RTV. Heuristic (5) is implemented in the Column Generation Approach, which is discussed next. Heuristic (6) is used by the Linear Assignment method, discussed later.

Column Generation Approach

An alternative approach for solving the decomposed problem was proposed by [58] using *Column Generation* [7], which we will abbreviate as CG. Still a trip-based formulation, the CG method is based on ILP 3.1 and is developed as an alternative procedure to select the candidate trip list. The aim of using column generation is to save computation time by only evaluating a fraction of the possible trip combinations that are likely to be useful in the final solution.

Problem formulation. Column generation is a technique for solving linear programs when the number of variables is large, such as the case here where there is a variable for each candidate trip. The technique takes advantage of the fact that in a basic optimal solution the number of non-zero variables can be considerably smaller than the total number of variables. First, the linear program is solved with only a subset of the variables. Since a minimization linear program has no negative-costs associated with any variable at an optimal solution, the technique evaluates whether the solution to the linear program with the subset of variables is optimal by using an oracle that either states that the solution is optimal or returns a variable that can be added that has a negative reduced-cost. Such oracles, referred to as separation oracles, need to be tailored to each problem and in an ideal setting run efficiently.

The name separation oracle comes from the equivalence of finding primal variables to enter the basis and finding violated constraints in the dual linear program. Column generation can be used for linear programs but not for integer programs

since the integrality constraints in the primal are not explicitly expressed, meaning the dual formulation can only be used to separate the linear relaxation. As such, the term column generation in this context refers to a heuristic approach for solving integer programs where the linear relaxation of the problem is solved and then an integer program is solved that only uses variables that were introduced by the linear program's column generation procedure. Such a heuristic is not guaranteed to return an optimal solution but is the approach covered here for the assignment problem as suggested by [58].

A limitation of the heuristic is that even when a feasible solution exists that can serve all of the requests, the set of columns produced might not be able to serve all of those requests when the integrality constraint is added. In fact, as the following lemma shows, even if one allows multiple trips serving the same request to be selected, an integer solution may not be able to serve as many requests as were served in the solution to the relaxed problem. This is simply the cost of using this heuristic.

Lemma 3.4.1. *Let x^* be an optimal solution to the linear relaxation of ILP 3.1. In general, there does not exist an integer solution covering as many requests as x^* , even if we are allowed to choose multiple trips serving the same request.*

Proof. Suppose we are working in \mathbb{R}^2 with the usual Euclidean distance. Suppose there are requests at $w = (0, 0)$, $x = (0, 1)$, $y = (1, 0)$, and $z = (1, 1)$. Each request wants to be transported to a point at distance ϵ from their pickup point, for some small ϵ . Suppose that vehicle A is at $(2, 1/2)$ with a passenger bound for $(-1, 1/2)$. Further, suppose vehicle B is at $(1/2, -1)$ with a passenger bound for $(1/2, 2)$. Each

request can tolerate up to 1 unit of delay.

In this setting, vehicle A can serve requests w and y or x and z . Similarly, vehicle B can serve requests w and x or y and z . The optimal linear solution selects each trip with value $1/2$, resulting in each request being completely covered. Using complete enumeration, it is possible to see that the maximum number of requests served in the integer setting is 3. \square

To find a separation oracle, start by writing the dual of the linear relaxation of ILP 3.1 which in this case is

$$\begin{aligned}
& \max_x \sum_{r \in \mathcal{R}_e} \pi_r + \sum_{v \in V} \sigma_v \\
& \text{subject to } \pi_r \leq M \quad \forall r \in \mathcal{R}_e \\
& \sum_{r \in t} \pi_r + \sigma_{v_t} \leq c_t \quad \forall t \in \text{Trips} \\
& \pi \geq 0, \sigma_v \leq 0,
\end{aligned} \tag{3.2}$$

where dual variables π_r correspond to constraints on requests and dual variables σ_v correspond with the requirement vehicles are assigned at most one trip.

Given a solution to the LP relaxation of ILP 3.1, the separation problem is to search for trips such that $c_t - \sigma_v < \sum_{r \in t} \pi_r$. If we interpret $-\sigma_v$ to be the benefit of having more or the cost of having less access to resource v , we can interpret the separation problem as a search for trips such that the cost of the trip plus the value of the vehicle are less than the price we would be willing to pay to not have to serve the requests in the trip. Notice that the separation problem does not change the

subproblem of finding the optimal route and cost for trip t with vehicle v .

Solution method. Solving the separation problem requires finding a vehicle v and group of requests \bar{r} so that the resulting trip $t = (v, \bar{r})$ solves the maximization problem

$$\max_{t=(v,\bar{r})} \left\{ \sigma_v + \sum_{r \in \bar{r}} \pi_r - c_t \right\}.$$

Solving this problem is NP-hard since determining the cost of the trip c_t is NP-hard. In fact, it is hypothesized that even approximating this problem is hard [48]. However, in some ways this is not the main difficulty associated with the separation problem since as before with the full RTV method we are willing to let the CTSP subproblem be solved by a call to an oracle. Thus the problem can be solved by making an oracle call for every combination of vehicle and group of requests. The issue we are concerned with is whether this uses fewer oracle calls than the Full-RTV method. Surely, if one is willing to search over every combination of vehicle and requests to identify columns then one can skip the column generation procedure and use the Full-RTV method instead. Thus, instead one must use an inexact heuristic to solve the separation problem .

At a high level, the heuristic separation procedure used in [58] starts by checking for columns with the smallest cardinality of requests first. If at least one trip with k requests exists that should be added to the basis, they suggest producing a list of request-disjoint trips all of size at most k at the same time. This process is repeated until there is an iteration in which no new trips are found. The two questions this

leaves are how to determine which columns to add and identifying a termination condition.

The authors of [58] suggest the following heuristic procedure to search for columns. Given a solution x^* to the relaxation, and a cardinality k , search for new columns as follows:

1. Select the request r with largest dual value π_r .
2. Select the vehicle that can match with r at the lowest dual adjusted price.
3. Iteratively add additional requests to the set, choosing the set that minimizes the dual adjusted price at each step. If we cannot find k of these, delete r and go back to step 1.
4. Of the k trips generated, select the one with the best dual adjusted price.
5. Delete the requests used in the selected trip and go back to step 1, if there are any requests remaining.

There is not a natural termination condition for this procedure if there is not a bound on the number of requests that can be included in each trip and so [58] use an arbitrary time budget. Compare this to the RTV graph generation process where if there no feasible trip for a vehicle with k requests there cannot be a feasible trip with $k + 1$ requests with that same vehicle. This same logic cannot be directly applied to the column generation process since the value $\sigma_v + \sum_{r \in \bar{r}} \pi_r - c_t$ is not monotone decreasing when considering a trip with one additional request.

Comments. Note that this procedure can benefit greatly by caching the requests of trip pricing queries (meaning the price before adjustment with the dual variables), both for the trips that are feasible and those that are infeasible.

3.4.2 Linear Assignment based methods

While the methods above use heuristics to solve an exact model of request assignment, here we present methods that instead make assignments by solving heuristically simplified assignment models. We will start with a simple bipartite matching algorithm, called Linear Assignment, which can be solved in polynomial time. Then we show some extensions of the matching algorithm to explore the tradeoff between computation time and solution quality, with some extensions coming from the literature and others being newly presented here.

Linear Assignment (LA)

The linear assignment method (LA), also referred to as single-matching assignment, and first demonstrated in [70], approaches the problem from a different angle that results in significant time savings by avoiding the costly trip generation phases used in the Full-RTV and CG methods. It can be solved in polynomial time with an algorithm for weighted bipartite matching, assuming usual access to a CTSP oracle.

Problem formulation. This method saves computation time by only generating the RV graph rather than the entire RTV graph. Here, the assignment problem is modeled as a matching problem on the bipartite RV graph, with edges weighted by the change in cost incurred if the request is matched with the vehicle. The cost on each edge is determined by the change in cost for the associated vehicle plus a factor rewarding serving a previously unserved passenger.

To formalize the approach, we first define a δ trip which will capture the concept of making a single change to an existing trip. For a trip $t = (v, \bar{r})$, the δ trip associated with adding request r is $\delta(t, r) = (v, \bar{r} \cup \{r\})$. The cost associated with a δ trip $\delta(t, r)$ is $c(t, r) = c(\delta(t, r)) - c(t)$. Given the previous trip assignments T to vehicles, let the set of all single-addition trips we consider form a bipartite graph with edge set given by

$$\mathcal{E} = \{\delta(t, r) \mid c(t, r) < \infty, \forall r \in R, \forall t \in T\}.$$

Solution method. The bipartite matching problem is solved. For each edge selected, the request corresponding to the edge is added to the set of requests assigned to the vehicle. All of the assignments made to the vehicle previously are preserved. Optionally, requests not assigned in an epoch may be saved and considered again at the next epoch since assigning these requests may still be possible, something not true from an exact solution of the Full-RTV method.

Comments. Unlike the previous two methods, LA never reassigns requests that have not yet boarded to different vehicles. Additionally, requests might not be assigned to a vehicle even if there exists a trip that could serve them since only one new request can be added to a vehicle in each epoch. Because of this, it is necessary to consider storing unserved requests in future epochs.

On the most naive end, one might have started with a greedy algorithm that considers each request in turn and assigns them to the best possible vehicle, regardless of any other requests that may come into the system, as is used in [53]. It may seem as if LA imposes an artificial relationship on the assignments since a request that pairs well with another could be assigned to a different, suboptimal vehicle, something both the Full-RTV method and a greedy method could avoid. Suppose for a moment that we are using a simplified model and each vehicle can only accept one request each epoch. Now the LA method finds the exact solution to the problem. Even only considering the number of requests that are assigned, we know that the greedy algorithm is only a $1/2$ -approximation to the maximum cardinality bipartite matching problem. Even though LA can only assign at most $|V|$ requests per epoch, this does not imply that it cannot achieve high service rates as long as the epoch length is shorter than the average request's travel time, divided by the optimal average capacity of the vehicles.

Multi-Round Assignment (LA-MR)

In this section we present the first extension of the LA method, first considered in joint work [16] and modified here. Motivated by the consideration in LA of carrying unassigned requests from one epoch to the next, this method considers performing multiple rounds (hence LA-MR) of assignment to address the same issue.

Problem formulation. The basic formulation is the bipartite graph used in the LA method. While not strictly necessary, here we incorporate an additional heuristic that might increase the chances that two compatible requests are assigned to the same vehicle, at the cost of some extra iterations. See the Comments for a discussion. In short, this heuristic says that if a request is assigned to a vehicle, then any other request that could have shared a ride with that request must not get assigned in that round. Once a request is assigned to a vehicle, any other request that could be assigned to that vehicle cannot be assigned in the same round. A pair of requests that can both be assigned are referred to as independent. The concept is formally defined as follows.

Definition 3.4.1 (Independent Requests). *Two requests are considered independent if there is no vehicle such that both requests can feasibly be assigned to the vehicle. In other words, two requests are considered independent if there is no vehicle that has an edge between it and both of the requests in the bipartite graph.*

Solution method. Rather than add constraints to the matching problem to restrict assignment according to Definition 3.4.1, the bipartite matching problem is solved in the normal way and then a subset of the edges in the solution are chosen and used to make assignments. These edges are drawn from the edges in the solution in an arbitrary order and edges that correspond to assignments with a request that is not independent of another request made in the same iteration are not chosen. These requests are saved and used in the next round. The process ends when there is a round where no new assignments are made.

Comment. The use of a sub-solution is motivated by the following example: Suppose there are two travelers with the same origin and destination that both request a trip at the same time. Intuitively, an efficient way to serve the two travelers would be to place them in the same vehicle. When we use the bipartite matching problem to find a matching, if there is more than one vehicle that can serve the passengers then each traveler will be assigned to a different vehicle. This possibly undesirable situation can be avoided if only one of those assignments is accepted in the round. By definition, since both travelers could have been assigned to the same vehicle they are not considered independent. Thus, one of the travelers will have to wait until the next round to be assigned. In this next round, it is possible to assign that traveler to the vehicle the first traveler was matched with. Although only a heuristic, it creates potential to better serve requests that would do well matched together.

Linear Assignment with Naive Swapping (LA-MR-S)

For a first try at swapping, we create a new method that adds an additional layer on top of the LA-MR method in a naive way that preserves the efficiency of the bipartite matching. This method is naive in the sense that it is lightweight to compute but the objective value of the solution may not correspond with the actual realized objective gain. The objective value is not accurate since swapping involves two vehicles and a request while each edge in the bipartite graph only connects a request with a single vehicle.

Problem formulation. LA-MR-S starts with the bipartite matching graph used in each round of the LA-MR method. For every request that has already been assigned, a node is added to the bipartite graph representing the request. An edge is added between this node and any vehicle it can be assigned to, other than its current assignment. Each such edge represents removing the request from the vehicle it's currently assigned to and adding it to the new vehicle. The cost associated with the edge is the benefit to the vehicle that no longer needs to serve the request less the cost to the vehicle that must now serve it. This is formalized in the concept of a δ -swap.

Definition 3.4.2 (δ -swap). *Given trips $t_1 = (v_1, \bar{r}_1 \cup \{r\})$ and $t_2 = (v_2, \bar{r}_2)$, with $v_1 \neq v_2$ and $r \notin \bar{r}_2$, a δ swap of r from t_1 to t_2 is denoted by $\delta(t_1, t_2, r)$. A δ swap results in the creation of two new trips, $t'_1 = (v_1, \bar{r}_1)$ and $t'_2 = (v_2, \bar{r}_2 \cup \{r\})$. The associated cost is $c(t_1, t_2, r) = c(t'_1) + c(t'_2) - c(t_1) - c(t_2)$.*

In addition, as mentioned above this sense of swapping is naive since each edge in the bipartite graph is adjacent to at most one vehicle even though swapping involves two vehicles. Related to this reason, we here add the concept of independent vehicles for use in drawing a sub-solution. While Definition 3.4.1 was used as a heuristic, here the constraint is also necessary to guarantee the algorithm terminates, as discussed in the Comments. The formal definition is given as follows.

Definition 3.4.3 (Independent Vehicles). *Two vehicles are considered independent if there is no request currently assigned to one of the vehicles that can feasible be assigned to the other.*

Solution method. The problem is solved as in the LA-MR method using the extended bipartite graph with edges for δ -swaps along with extra notion of independence given in Definition 3.4.3 added to the sub-solution drawing step. Rounds of assignment continue until no more changes are made.

Comments. Unlike the case of LA-MR where independence is used as a heuristic for good assignments, in the case of LA-MR-S we need independence to ensure the method terminates. Consider the case where there are two vehicles, each with one request assigned to them. Further assume that each request has the same origin and destination. In this case it would be most efficient to serve both requests in the same vehicle, thus using half of the VMT to serve them. Under the formulation used for LA-MR-S, there would be a significant positive weight assigned to the each of the edges denoting the requests being swapped to the other vehicle. But if both

of those swaps are actually made we actually end in the same situation we started with. Using the notion of independent vehicles as we choose our sub-solution only allows one of those swaps to be made, meaning that each vehicle whose assignment is updated must be making a change that strictly improves the total objective value. Thus, eventually the procedure must terminate.

This example further calls attention to why the notion of swapping here is naive — the objective value of the solution doesn't consider the cost of assigning a request to a vehicle if another request is simultaneously being removed from it via swapping.

Proper Swapping (LA-MR-PS)

In this section we again consider swapping, but propose an approach whose objective value is accurate. The weakness of LA-MR-S was that the objective value in the bipartite matching problem could not accurately capture the case when a vehicle had both an assignment and one or more reassignments. Here, we avoid the issue while preserving polynomial time solvability by converting the model into a general matching problem. As with LA-MR-S, this model is larger than LA and requires more CTSP queries to build the graph the matching is performed on.

Problem formulation. As before, this method uses a graph as the underlying mechanism for decision making. Start by creating the bipartite graph used in the LA and LA-MR methods. Since every swap of a request involves two vehicles, we extend the model by adding new edges connecting vehicle nodes if there is at least

on request in one of the vehicles that can feasibly be assigned to the other. Each such edge connects two vehicle nodes and has value equal to the value of swapping a single request from one vehicle to the other.

Note that since we can compute the best single swap between any pair of vehicles, and a matching would always choose the best arc between any pair of nodes, with out loss of generality we can make the graph extension undirected.

Solution method. This method works in rounds, similar to LA-MR. Each round begins by finding an optimal maximum weight matching in the graph. As before with LA-MR, as a heuristic we draw a subset of the edges from the solution using the notion of independence in Definition 3.4.1. Edges are drawn in an arbitrary order, rejecting edges if any request the edge is adjacent to is not independent of previously drawn requests.

Comments. Since every edge in the model is adjacent to each vehicle modified by selecting it (in contrast to the LA-MR-S model), the use of Definition 3.4.1 is not required for the method to terminate. The solution at each step only selects groups of edges that each individually contribute a strict improvement in objective value since we can omit zero valued edges. This was not the case in LA-MR-S since two edges could have been selected that correspond to assignment modifications for the same vehicle while still being a valid matching. The effect of taking a sub-solution does not defeat the strict improvement property, rather it serves only as a heuristic

to get better matches.

Very Large Scale Neighborhood Search (LA-MR-CE)

Here we introduce another new method that pushes further into the tradeoff between execution time and solution quality by adding a more sophisticated swapping mechanism. This method extends the LA-MR method with swapping based on the cyclic exchange very large-scale neighborhood search algorithm proposed by [1]. As with the other linear assignment based methods, this is only a heuristic for solving the epoch-wise optimal assignment problem. Still running in polynomial time, the advantage of this method is the ability to swap the assignments of multiple requests at the same time. It may be an attractive use of extra computation time to do this sophisticated swapping under the hypothesis that swapping has the largest marginal benefit in assignment algorithm heuristics.

Problem formulation. In this method, we think of each vehicle as a partition. The objective is total value given by each partition, each of whose value is determined by a general cost function as a function of the vehicle and the set of requests assigned to the partition. In the cyclic exchange method, we will move requests between the partitions (vehicles) in order to reduce the total cost of the solution. It should be noted that this method will allow us to serve the requests in a more efficient manner, but we are not assigning requests that were previously not assigned to any vehicle.

In the cyclic exchange method, we create a graph G with a node for each request

and a dummy node for each vehicle. A directed arc leads from request r_i to request r_j in the graph if the requests are assigned to different vehicles and it is feasible to route the vehicle that r_j is currently on if r_j were to be replaced by r_i . A directed arc leads from request r_i to dummy vehicle node v if it is feasible to route v when r_i is added as an assignment. An arc leads from every dummy vehicle node v to each request node in the graph representing removing the request from its vehicle without replacement. The cost associated with an arc (i, j) is the net benefit (which is sometimes negative) experienced by the vehicle corresponding to node j .

The crux of this method is identifying valid cycles in G , as defined here.

Definition 3.4.4 (Valid cycle). *A cycle in G is considered valid if no two vertices in the cycle are associated with the same vehicle (partition) and at most one node in the cycle is a dummy vehicle node.*

The restriction on dummy vehicle nodes is without loss since they effectively allow us to create cycles that are paths. Any cycle with more than one dummy vehicle node can be recast as two separate cycles with identical total objective value.

The cyclic exchange procedure requires the use of two additional data structures. First, a set P which stores the set of nodes that need to be processed by the algorithm. Second, an associative array T which maps nodes in the graph to a set of nodes, to be updated during the course of the algorithm to track which nodes must be re-evaluated.

Solution method. The method begins by assigning new requests using the LA-MR method. Then, a second stage is begun that executes the swapping through repeated use of the cyclic exchange procedure.

The cyclic exchange method begins by adding all nodes other than dummy nodes to the set P and by letting T map nodes to the empty set. Each iteration of the cyclic exchange method removes an element from P and processes it by searching for valid cycles starting at that node. If a cycle with positive-value is found, updates are made to assignment and the set T is updated and used add some nodes back to P . If none is found, T is updated but no new elements are added to P . The procedure terminates when P is empty. Each step will now be explained in detail.

Each iteration begins by selecting a starting vertex from P and searching for a maximum-benefit cycle. The process of finding a maximum-weight cycle in a simple graph is known to be NP-hard and the process of finding a maximum-weight cycle in G could be equivalent to this, and therefore is hard as well. To find the exact maximum weight valid cycle in G we would need to use a vertex labeling algorithm that tracks the best objective observed at each vertex with respect to each possible subset of predecessor vertices. This in and of itself is computationally hard as it may require us to track an exponential number of states. As a heuristic, we make two simplifications to the problem.

First, without loss of generality we require that the running sum of benefits along the arcs be strictly greater than the potential benefit of removing the origin request from the starting vehicle. This is without loss since every cycle will have at least one

feasible starting point under this condition.

Second, to directly address the problem of searching for maximum-benefit cycles we use the heuristic suggested by [1]. In this heuristic, we search for the cycles with a node labeling algorithm where each time we visit a vertex we check that the partial cycle is valid (not visiting the same vehicle multiple times) and then if this partial cycle has the best objective of all partial cycles that have visited this vertex so far we discard all other partial cycles that have visited it and save the newest one.

In the case where a positive-benefit cycle is found, we make all request swaps that are denoted by the edges in the cycle. Then we update the weights on all arc corresponding to swaps into and out of the vehicles that were affected by the cycle, possibly creating or destroying some arcs in the process as appropriate. For every node in the cycle, we look up the set of request nodes mapped to in T , add them to P , and then clear the set of nodes from T . We also add the origin node to P in case there are more cycles that originate there.

In the case where no positive-benefit cycle is found, for each node touched by the search procedure the origin node is added to the corresponding set in T .

Comments. This is the most heavy-weight of the linear assignment based methods. It is interesting to note that this method performs the assignment and swapping in two disjoint steps. The focus on swapping in the second step emphasizes efficiently serving requests and clearing them out of the system in minimal time. Since the fleet size is fixed and vehicles have limited capacity, swapping is essential to high through-

put. This method may be most attractive when system performance causes more of a bottleneck than the ability of the first step to find initial assignments for new requests.

3.5 Stability of Solutions

Stability refers to the ability to replicate parts of a solution across subsequent epochs. This applies both to recalculating the routes that vehicles should take as well as to the optimality of the assignments between vehicles and requests. This section discusses stability in both of these contexts.

3.5.1 Computing the Constrained TSP Subproblem

No matter which method is used to make assignments, the route the vehicle takes must be computed by a constrained traveling salesman problem which takes into account the objective value of a route and feasibility with respect to the QoS constraints. When the number of requests assigned to a vehicle is not too large, exact algorithms are usually sufficient to find optimal routes. However, as the number of requests grows it becomes infeasible to explore all patterns. Here it is natural to utilize heuristics for larger problem instances. Despite the convenience offered by heuristics, when routes are recomputed each epoch both with new sets of requests, possibly adding more or removing some, the heuristic may not give a feasible solution

to the new problem. In light of this, here we will discuss the important property of stability for heuristics used in CTSP algorithm.

Stability in heuristics

Epoch-wise approaches to vehicle-request assignment must constantly run CTSP subproblems as a normal part of determining whether requests can be swapped or if new requests can be added to the assignment of a vehicle. When using a heuristic for the CTSP subproblem, we hope that the heuristic will frequently yield good solutions and in particular think about issues related to updating vehicle assignments. Since assignments can be constantly updated it is natural to consider recalculating the optimal route each epoch. Unlike the case of the standard metric TSP problem, it is NP-hard to determine whether a feasible route exists for the CTSP problem. The first issue this brings up is whether heuristic can find a feasible route for a vehicle given an assignment in a previous even when the no new requests have been added, a possible concern depending on the sensitivity of the heuristic to requests being picked up/dropped off and vehicle motion. While this problem can be directly addressed by saving the vehicle's route in memory, if a heuristic cannot reproduce a route we may be concerned that it might also be less likely to be able to find a way to add additional requests to the vehicle in future epochs. This is an issue since the ability to maintain several requests assigned to the same vehicle is a key focus of yielding performance in ridepooling systems.

Heuristics that avoid these problem, and in particular are able to give a vehicle

the same assignment from epoch to epoch, can be thought of a stable in the sense that they provide a constant output. This is formally defined as follows.

Definition 3.5.1 (Stable heuristic). *A heuristic is (strictly) stable if for any vehicle and set of requests that is given a feasible routing under the heuristic, at all points as the vehicle follows the given route, picking up and dropping off requests as assigned, the heuristic always produces an (identical) feasible route for the vehicle in its new state and the remaining set of requests.*

A simple type of heuristic is an insertion heuristic, for example see [3]. An insertion heuristic builds a route in stages, though there are several ways to implement them. Insertion heuristics can sometimes start with a base trip, such as a copy of the vehicle's previously assigned route from another epoch. Then additional requests can be added either sequentially or as a group, at each point preserving the order of requests previously added. Sometimes heuristics are used in conjunction with a threshold policy, which provides a simple mechanism to alternate between a brute-force solution and heuristic.

Definition 3.5.2 (Threshold policy). *Given a vehicle, a set of requests, a routing heuristic, and a threshold value, a threshold policy is a rule that produces a number as a function of the vehicle and requests, and if the number exceeds the threshold returns a path made by the routing heuristic, otherwise returning a path made by an exact search.*

Threshold policies are in effect a way to make a new heuristic from an existing heuristic by adding the ability to use an exact search for some instances. These

policies need to meet some conditions to ensure that the resulting policy is stable, in the sense of Definition 3.5.1.

Definition 3.5.3 (Stable threshold policy). *A threshold policy is stable if it is built on a stable heuristic and for any vehicle and set of requests that is considered below-threshold, that same vehicle and set of requests is considered below-threshold when*

1. *Any of the requests are removed*
2. *Any of the requests become onboard passengers*
3. *Any of the onboard passengers are removed.*

As discussion of some heuristics that are stables is included in Appendix B.1.

Not all TSP heuristics are stable for CTSP

Given that the CTSP must find a route that satisfies QoS constraints, heuristics that are natural for the TSP are not necessarily stable for CTSP. This is unsurprising since finding a feasible solution to the CTSP is NP-hard. As an example of this, consider the use of a simple insertion heuristic. This heuristic starts with no requests assigned to the vehicle. Then, one by one the requests are inserted into a schedule for the vehicle with the pickup (if it applies) and drop off point of the request being inserted at the locally optimal locations with the order of all previously inserted requests held constant.

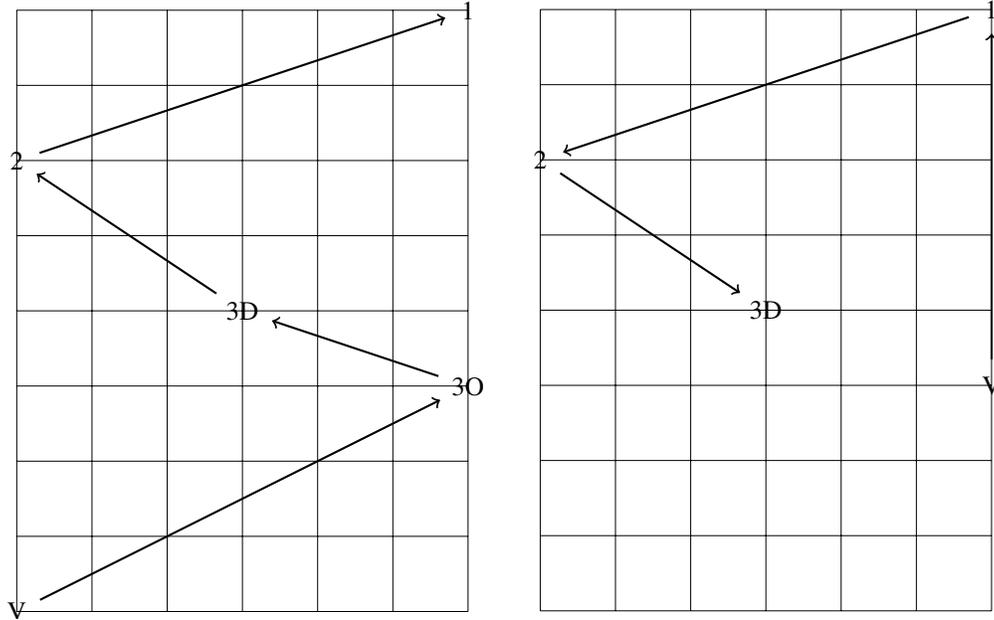
The following Lemma shows that this heuristic is not stable.

Lemma 3.5.1. *The insertion heuristic is not stable.*

Proof. To prove the heuristic is not stable it suffices to show an example where the heuristic produces a feasible solution but then fails to produce a feasible solution at a later portion of the journey.

Suppose there is a vehicle with two passengers and a new request. Due to the time the requests arrived, they must all be dropped off before time 20 and request 3 must be picked up before time 10. The destination of of passengers 1 and 2, as well as the origin and destination of new request 3, are shown in Subfigure 3.1a. Using the heuristic, first passengers 1 and 2 are added to the path, resulting in the partial path $\{V, 2, 1\}$. Next, request 3 is inserted. The optimal way to do this is the path $\{V, 3O, 3D, 2, 1\}$ which is a path of length 19.8 units. Assuming the vehicle travels at unit speed, this means that all constraints will be satisfied and we have a feasible path.

After the vehicle reaches the origin of request 3, suppose we recalculate the vehicle's path using the heuristic. After the first two steps considering passengers 1 and 2, the partially built insertion path is $\{V, 1, 2\}$. Notice that this is already different from the first situation - the order of the first two passengers has been flipped. There are three locations that request 3 can be inserted into the path. None of these insertions yield a feasible path since the length of the path, plus the time already traveled, exceeds 20 units. The shortest path, $\{V, 1, 2, 3D\}$, gives a total time of 21.6.



(a) Initial problem instance and heuristic solution for vehicle, currently with passengers 1 and 2. (b) New problem after vehicle picks up request 3. Example shortest, but infeasible, solution shown.

Figure 3.1: Example demonstrating instability of insertion CTSP heuristic.

Thus the heuristic is not stable.

□

In short, this example illustrates that unless one uses a stable CTSP algorithm then one must save the previous path assigned to each vehicle at each epoch in order to guarantee that a feasible path can be found when recalculating routes in the following epoch, though relying on saved paths may inhibit assigning new requests to the vehicle. This is different than the observation that heuristics cannot always find a feasible solution to CTSP, for which it is NP hard to determine if a feasible

solution exists. Rather, it points out that even when a feasible solution is found the heuristic may not be able to solve the residual CTSP problems as the vehicle follows the assigned route.

3.5.2 Notes on Stability in Rebalancing

Stability not only applies to vehicle routes but also to solutions overall. In the methods presented here, vehicle rebalancing is performed as a second step that occurs after the assignment process has completed. Rebalancing refers to the process of moving empty vehicles without an assignment to other locations in anticipation that the new location will be more likely to lead to a successful assignment. There are a wide variety of rebalancing algorithms used in the Ridepooling literature. While we will not comment here on which one has the best performance in any particular situation, we have found that memory across iterations is an important aspect of the successful rebalancing algorithms we have used.

Consider the rebalancing algorithm used in [3]. With slight modification, it can be written as a bipartite matching problem between the set of vehicles with no passengers or assignment and the set of requests in the current iteration that were not assigned to a vehicle. Rebalancing assignments are determined as the solution to a minimum cost matching in the graph subject to either all requests being matched or all vehicles being matched, depending on which set is smaller. As presented, there is no clear instruction for what vehicles given a rebalance assignment should do in the

following epoch if they are once again not given an assignment. Some authors have chosen the interpretation that the vehicles must receive a new rebalancing assignment each epoch.

The limitation of this approach is that it is sensitive to the duration of the batching interval. If, for example, the batching interval is short then most vehicles will not be matched in the rebalancing process since smaller intervals generally include fewer requests.

A better alternative is for vehicles to save their rebalancing assignments across epochs. In this model, any vehicle can always be assigned requests to serve. However, in the rebalancing process vehicles that were heading towards a rebalance target in the previous iteration are excluded and continue towards the same target as before. By excluding these vehicles, the rebalance process gains some independence from the length of the batching interval since the quantity of requests that can be matched as rebalance targets over any period of time is no longer attached to the batch interval length.

3.6 Testing Framework

We develop a testing framework in C++ that simulates the full day operation of a ridepool system. The inputs to the simulator are a directed graph whose arc weights represent travel times, the initial location of all vehicles, and a file listing the entry

time, origin, and destination of all requests that will arrive during the day. Requests can only have origins and destinations at nodes in the graph, not at midpoints of arcs. Other parameters can be selected as well if the defaults are not desired. Such parameters include the batching interval, which assignment method to use, which routing heuristics use, QoS constraint values, fleet sizing, and output folder location.

The output of the framework includes results summaries from each iteration noting computation time, basic information about the system state, and key output statistics including the service rate, shared rate, and average times associated with each of the QoS constraints. It also outputs a vehicle action log file that tracks each vehicle's movements throughout the network and records the times each served request was picked up and dropped off.

The choice of C++ as a programming language is based on the efficiency with which it can process the large computational demand placed on it by several methods, especially those that require building large RTV graphs. C++ allows the program to be built in a modular way, which along with clean variable scoping rules makes it easy to verify the program as a decomposition of the entire problem. It also gives a high degree of control to the data structures used and ways to prevent unnecessary duplication of data when passing arguments to functions. The typing system in C++ also gives clarity throughout the program of the purpose and functionality of various functions.

In building the testing framework, trade offs were encountered between the generality of the framework and computational efficiency. While generalization is typically

beneficial to large computer projects due the modularity they provide, making such a framework for all features sometimes creates an unnecessary computational burden when such features are not being used. For example, if the travel function is to handle objectives that are functions of the sum of the waiting times for all passengers served then not only must we find a way to pass such metadata in recursive function calls, we must also go through the effort of computing it. Since this is not necessary in the most common scenario we have, where we only search for the feasible path that minimizes VMT, this leads to a trade off where it can be more beneficial to have duplicate functions that differ in metadata and computation for the sake of saving computation time typically. The result is harder to read since there is more code, however.

3.7 Numerical Comparisons of Assignment Methods

In this section, we will look at a comparison of the assignment methods in two settings: in full day performance and in instance-per-instance performance. It is important to consider these two cases separately as they give different results since in full-day performance the problem instances solved at each epoch are a function of the decisions made in previous epochs. We discuss the results and conduct further tests to explain the differences in some of the methods. Finally, we explore the difference between using service rate and RMT as a primary objective.

Table 3.1: Service rate comparison for various demand levels under each method. Note there are only half as many vehicles used in experiments with quarter-level demand.

	LA	LA-MR	LA-MR-S	LA-MR-PS	CG	Fast-RTV	Full-RTV	LA-MR-CE
Full	56.83	56.85	57.46	57.14	57.21	57.29		57.99
Two-thirds	74.22	74.34	75.09	75.03	71.29	75.46	75.4902	75.83
Half	86.47	86.31	86.83	87.07	83.54	87.47	87.56	87.41
Quarter	79.49	79.58	80.41	80.54	75.12	81.09	80.85	81.11

Table 3.2: Computation time (minutes) on initial runs for each method. The results are only approximate since some of the trials were run in parallel, potentially negatively impacting their runtime. Note there are only half as many vehicles used in experiments with quarter-level demand.

	LA	LA-MR	LA-MR-S	LA-MR-PS	CG	Fast-RTV	Full-RTV	LA-MR-CE
Full	26.9	38	129.7	92.7	1567.9	63.7		238.7
Two-thirds	21.6	31.2	97.6	77.5	1048.1	53.9	5838.1	199.4
Half	19.3	29.1	77.8	71.8	814.9	48.8	535.9	169.1
Quarter	6.4	10.2	17	15.6	141.7	8.3	10.5	15.5

3.7.1 Comparison of Methods

We begin by comparing the methods by running each for a full day on identical data sets. Here, we will use New York City taxi data for our inputs. We use the data set at four levels: full (including all requests), two-thirds (removing one-third of the requests from the full data set), half, and quarter. We use 1000 vehicles for each trial except for the quarter simulation, for which we used 500 vehicles. Results are shown in Table 3.1 and the average per-epoch running times are shown in Table 3.2.

The first thing to note from the table is that the Full-RTV method outperforms most of the other methods in the trials, which would seem to be intuitive given it finds the optimal solution to the myopic problem. Among the methods that build

solutions iteratively, without using swapping LA and LA-MR consistently get lower service rates than any of the other methods that do allow swapping. Between LA and LA-MR, the service rates are nearly identical, with LA-MR performing on average 0.02% better on the full data. With the basic swapping methods, LA-MR-S and LA-MR-PS, LA-MR-PS performed better when the demand levels were lower while LA-MR-S performed better when demand levels were higher though performance was similar. LA-MR-CE performed exceptionally well. Except in the case of the half demand level (and the full demand level which was not tested for Full-RTV), LA-MR-CE achieved a higher service rate than Full-RTV.

The CG and Fast-RTV methods are both timed and thus their performance is not strictly deterministic as it is a function of the performance of the computer's CPU. The strong performance of CG was most pronounced on the full data, with its performance notably dropped at lower demand levels. This is likely due to a combination of it producing fewer candidate trips at lower demand levels together with the general fact that column generation optimizes for the linear relaxation, which might not translate to a good integer solution. The Fast-RTV method also performed quite well and actually outperformed the regular Full-RTV method in the case of the lowest demand.

3.7.2 Offline Analysis of Methods

While it is useful to compare assignment methods by evaluating their full-day performance, to gain a better understanding of why each of the methods performs differently we give one of the first direct comparisons of the methods on a series of identical states. First we run the Full-RTV method for a full day on the two-thirds data set and at each time step save a copy of the RTV graph as well as the state of the requests and vehicles. Then, the offline problem for each method is run as a type of replay: each method observes each of the states and creates a solution and we report the average performance for each of the methods across all of the saved states. This creates a fair comparison for each of the methods since all of the methods were presented the same states at each step.

Unlike with the online problem, the column generation procedure used in the offline problem instances is not timed. The timing feature is not necessary when the full set of possible trips has already be precomputed in the saved RTV graph state. Thus, CG performance represented optimal performance under that method.

The results are shown in Table 3.3. Although CG performed quite well in the offline tests, remember that the computation duration shown hides the time needed to evaluate feasible combinations of trips. An exhaustive list of feasible trips was available (effectively an oracle) for all methods. Since we know that the Full-RTV method can take a long time due to the time required to produce the RTV graph, we wonder to what extent CG would need to query the same number of trips in order

Table 3.3: Average rejections and VMT across full day for each method under identical inputs.

Algorithm	Rejected Trips	Distance Assigned
LA	48.229	756840
LA-MR	45.401	757965
LA-MR-S	45.397	757965
LA-MR-PS	44.972	755949
LA-MR-CE	45.401	753559
CG	39.063	753659
Full-RTV	39.075	753802

to achieve the objective found here. In the online tests, a heuristic was used that limited the columns found.

In general, we can see a few trends in the results that seem reminiscent of the full-day performances of each of the methods. First we notice that LA and LA-MR perform worse than LA-MR-S and LA-MR-PS, respectively, showing benefits from allowing swapping. Further, we see what while LA-MR-CE did not bring significant gains in terms of rejecting fewer trips it did reduce the average assigned VMT compared the those four previous methods. Perhaps this is partly why it had such strong online performance.

CG and Full-RTV, which in this case were untimed, rejected the fewest trips. It may be misleading to consider the average VMT they assign compared to other methods since some of the extra distance is accounted for by virtue of not rejecting as many requests.

3.7.3 Why do many methods perform similarly?

While the offline analysis made it clear that each of the methods have different abilities to achieve low rejection rates, the range of outcomes varied considerably more than they did in the online case. Our claim is that this is because there is certain amount of feedback that naturally occurs in the system. For example, if an assignment method does a good job of assigning as many requests as possible in one iteration, then in the next iteration it will be harder to assign as many of the new requests because more of the system capacity has already been reserved. On the flip side, if an method does a poor job at achieving its objective in one iteration then the system is likely to have more capacity in future iterations, giving it a chance to catch up.

To test this hypothesis we compare two of the methods, LA and Full-RTV, against each other when the demand levels are uniform over the course of the day. The purpose of this trial is to determine if good performance of an assignment method (assigning many of the available requests) is subsequently being counteracted by worse options in the following iterations, and vice versa when they assign only a few of the available requests. By using uniform demand levels, we expect in general that the number of people assigned in each iteration should have no correlation over time if the hypothesis is false. If, in contrast, the performance of one iteration does impact future iterations then we would expect there to be a negative correlation between the number of people assigned over some time interval. This would capture good performance followed by bad and bad performance followed by good.

To do this, we ran a steady state simulation under the two methods. First we consider how the number of requests accepted changes over time by plotting lagging pairs on a scatter plot for lag duration 1, 10, 20, and 30 minutes, as shown in Figure 3.2. As the regression lines show, there is a positive correlation between the number of requests served over subsequent, 1 minute intervals, while over 20 minute intervals there is a negative correlation. For 30 minute and longer, the regression line appears fairly level.

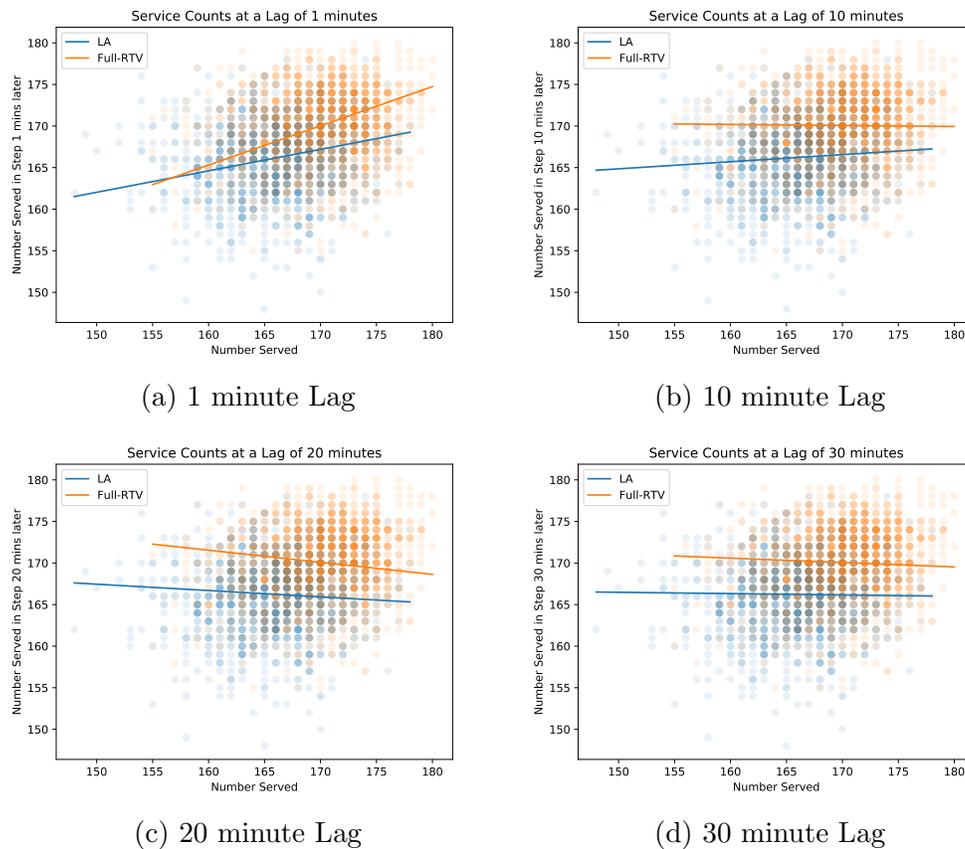


Figure 3.2: Number of requests served per iteration, plotted in pairs for fixed lag durations. LA method shown in blue, Full-RTV method shown in orange. Regression lines shown over plot.

The 1 minute lag positive correlation means that the number of requests that are assigned at each iteration is very similar across immediately adjacent iterations. Meaning, if a large number of requests are assigned in this iteration, then in the next iteration we will also tend to assign many requests. If on the other hand a smaller number of requests are assigned, then in the next iteration a smaller number will typically also be assigned. At the 20 minute lag level we see a negative correlation which means that if many requests are assigned right now, fewer will be assigned in an iteration 20 minutes from now, and vice versa.

To show the effect more directly, the slope of the regression lines are plotted against lag durations in Figure 3.3. The results in the Figure show that the number of passengers assigned during each iteration has a positive correlation over short time periods. This means that if a large (small) number of requests are assigned in one iteration, then in the following iteration there will also likely be many (few) requests assigned. The Full-RTV method in particular shows negative correlation around the 20 minute lag window. This means that if many (few) requests are assigned in a particular iteration, then 20 minutes later there will likely be fewer (more) requests served per iteration. An interpretation of this might be that the system's capacity is being packed and relieved over cycles of roughly 40 minutes.

This behavior is possible is due to a combination of two reasons. First, the system has a maximum capacity it is capable of handling at each point in time. For example, given that the average request rides in a vehicle for about 14 minutes, serving 90% of the demand in this scenario (180 requests per minute) means each vehicle must

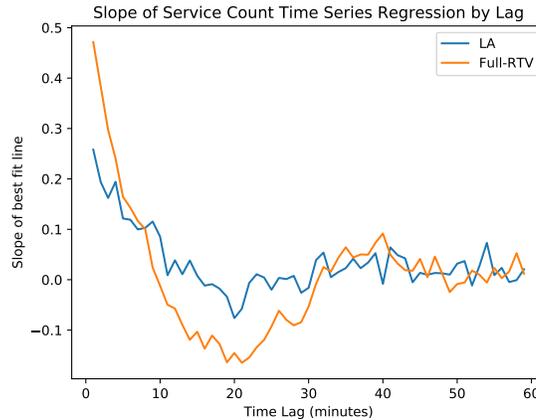


Figure 3.3: Slope of regression line for lagging pairs of number of requests served per iteration, varied by the duration of the lag between the pairs.

sustain an average of 2.3 passengers at all times. The second reason is that some optimization methods, such as Full-RTV and LA, can assign more requests in a single iteration than the system can handle, given its long term maximum capacity. So any method that can assign requests at a rate at least that of the system’s maximum capacity will be able to achieve a similar steady state service rate, except to the extent that an method might use extra information such as that about future demand.

One might hypothesize that the cyclic behavior of over-assigning requests and then hitting a capacity bottleneck may be bad for system performance. To test this idea, we rerun the steady state simulations with the restriction that in each iteration, at most x requests can be assigned, for some choice of x . This means that the method will be forced to choose the subset of requests at each iteration that are most beneficial for performance while also placing a damper on using system capacity too fast. In Figure 3.4, we show the steady state service rate resulting from

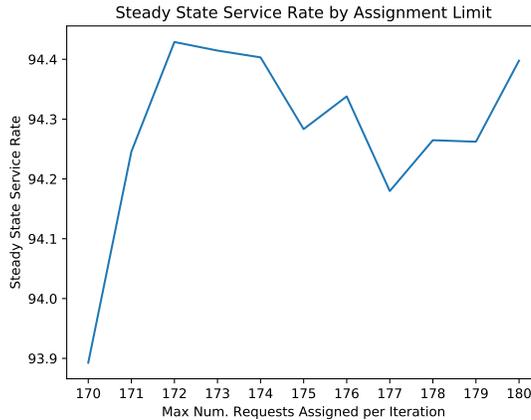


Figure 3.4: Steady state service rate when there are 180 requests per minute. In each iteration, the Full-RTV method is restricted to assign at most x requests, for x given on the x -axis.

limiting the Full-RTV method with various values of x . The results show a mixed response applying the limit. The maximum steady state service rate of 94.27% is achieved when the method can assign at most 172 requests per minute (out of 180), meaning a theoretical maximum of 95.6% service rate. Below 172, the steady state service rate drops off rapidly. While this experiment may benefit from additional randomized trials to help account for some of the variability in observed rates, it does suggest that the packing behavior of the method may have been hurting it.

These results show that methods can have a wide range of performances on the same input at any given time period while still providing high overall service rates over full days. In practical settings, this could suggest that a wide variety of methods can provide high service rates. However, the results from the static simulations showed that these methods might nonetheless have widely varying capabilities on a

per-instance basis. This could be relevant in situations where the objective function is significantly different from maximum service rate, such as if the reward for making the assignments were determined under an alternative cost model. A simple method such as LA that fails to capture the full epoch-wise benefits in the offline instances may fail to achieve the same catch-up that was observed under the service rate metric here. In short, when an method yields a competitive service rate this does not show that it will perform competitively with regards to the specific objective an operator might have.

3.7.4 Alternate Primary Objectives: Revenue Miles Traveled

We analyze what happens when we optimize for RMT instead of service rate. Table 3.4 shows the results from three methods: LA, LA-MR-PS, and Full-RTV. Each method was run twice, once using service rate as the primary objective and in the second case the total revenue miles traveled. RMT is computed as the sum of the direct travel distances of all assigned requests. In both cases, the secondary objective was to minimize VMT used to serve the requests.

In general, while optimizing for RMT improved the objective with respect to RMT compared to service rate optimization, the increase in RMT was marginal at less than a half of a percent improvement in performance. The service rate, however, decreased considerably ranging from just over a one percent decrease for the Full-RTV

Table 3.4: Comparison of service rates, revenue miles traveled, and vehicle used by method and objective type.

Half Dataset	Objective	SR	RMT	Idle	Enroute	Rebalancing	Inuse
LA	SR	86.47%	118355593	18172581	2221909	4174005	61771505
LA	RMT	83.98%	118664785	17576010	2225211	4507789	62030990
LA-MR-PS	SR	87.07%	119502377	19114910	2126998	4431663	60666429
LA-MR-PS	RMT	85.02%	119986511	18586687	2095150	4732277	60925886
Full-RTV	SR	87.56%	120308750	19915552	1960652	4111796	60352000
Full-RTV	RMT	86.20%	120510567	19420534	1988165	4401003	60530298

method to nearly two and a half percent decrease for LA. Optimizing for RMT, in general appears to reduce vehicle idle time. Most of this time is now being consumed by the rebalancing process and in-use time. The decrease in service rate might not be entirely unexpected since several trips in the data set had extremely short length.

3.8 Value of Future Information and the cost of being Myopic

The previous sections suggest that there could be a significant difference between solving the myopic problem and the full-day problem. To know the value of exploring this space it is perhaps more important to directly address the question of how much we stand to gain from knowing the future. Or equivalently, what is the cost of being myopic in ridepool assignment problems? The answer to this question is important since it has implications on how much we can gain by intelligently changing our objective functions, and thus how much value we can get from future research in this area.

While computational limitations prevent us from solving the exact problem, we can instead study what happens when we have partial visibility into the future. We employ one sixteenth-level demand with 125 vehicles. We use a variant of the limit and recall CTSP method described in section B.1.2 whereby we allow an excess number of requests to be considered if the extra requests are future requests that will arrive after the current epoch’s nominal time. After building a solution using the limit and recall CTSP method with the excess future requests removed, the future requests are added sequentially to the route, ordered so requests that arrive sooner are considered first. While this CTSP variant might not be stable across time periods, this is mitigated by removing the requirement that a future request that is assigned must also be assigned in subsequent assignment epochs.

We begin by considering up to 8 minutes of future requests visibility. The results, shown in Figure 3.5 for both the LA and Full-RTV methods, show a substantial increase in service rate with even a small amount of future visibility. In this particular experiment, 8 minutes of future visibility raised the service rate from 63.5% to 70.6% for Full-RTV and from 62.3% to 69.5% for LA. A possible explanation for the observed increase in performance may have to do with coverage. If all requests must be picked up within 5 minutes of entering the system, then each vehicle acts as a 5-minute radius cover of the map. When we can see 8 minutes into the future the radius of coverage expands to 13-minutes. While still limited by the ability to pack passengers onto vehicles, this allows the planning problem to find routes that utilize the expanded coverage that we could not have identified otherwise.

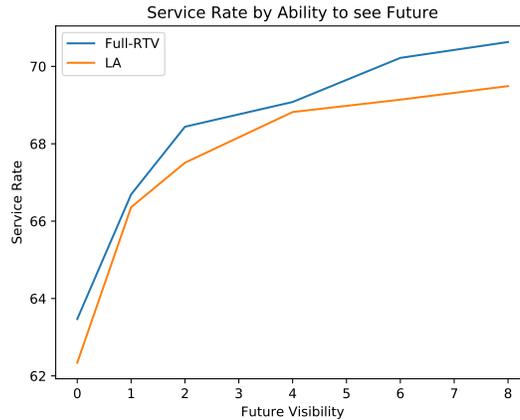
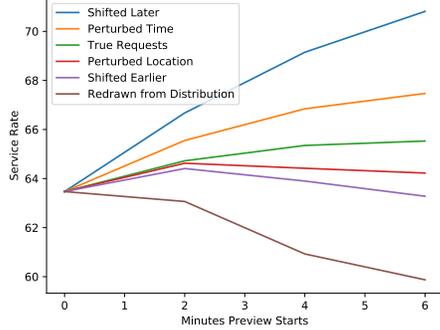


Figure 3.5: With demand reduced by a factor of 16, and with 125 vehicles, the service rate is shown when we have different visibility into the future.

The improvement from using future information is not limited to cases where the service rate is low. For example, in the case of one-eighth level demand with the Full-RTV method we see an improvement in service rate from 71.8% without future information improving to 78.0% with 6 minutes of future information.

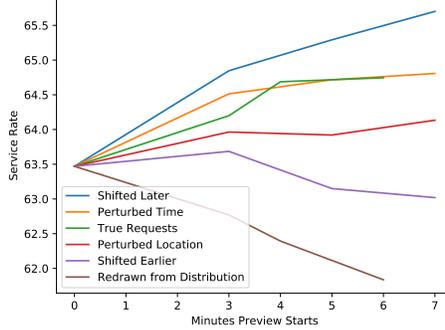
Despite the benefits of using future information, in practice utilizing it is harder since we typically only have a partial view of what comes next. To demonstrate this sensitivity we develop one possible simple procedure one could use to take partial or noisy future information and incorporate it into routing. We allow the system to view future requests as before, but then when the time until the future request arrives is too soon, we delete it from the system. This is a necessary step since it does not make sense for a vehicle to arrive at a request that does not actually exist. The theory would be that although the request expires when it reaches a certain threshold the vehicle in the meantime will have roughly moved to an area near where demand

Impact on Perturbations for Future Predictions with 1 Minute Threshold



(a) 1-minute threshold

Impact on Perturbations for Future Predictions with 2 Minute Threshold



(b) 2-minute threshold

Figure 3.6: With demand reduced by a factor of 16, and with 125 vehicles, the service rate is shown when we have noisy information about the future.

actually occurs so that actual demand is more likely to be served. We do not claim this to be an optimal approach - rather it is just a simple one we can test.

First we experiment with what happens when we choose a threshold of 60 and 120 seconds. This means that we keep the virtual future requests in the system until 60 (120) seconds before they would have arrived. We then vary how long into the future the requests first become visible. We give future requests the same weighting in the assignment process as real requests. The experiment is run on both the true data and various perturbations of the data, ranging from shifting time/location information all the way to resampling the data from the same distribution as the true data. The results are shown in Figure 3.6. While using the method with the true data produces a positive net benefit, albeit much smaller than when the information is available continuously, the cases with alternative data show wide variation.

In the case of simple perturbations of the time and location, altering the location

had a small negative impact on service rate compared to the true data. In this case the locations of the origins and destinations of each request was randomly drawn from a ball of radius 45 second from the true origin/destination of the request. This radius was chosen since it at least 95% of requests had at least one node within 45 seconds of their origin.

As for perturbing time, surprisingly the performance is actually better than in the case with the true data. This data was created by shifting the apparent arrival times of the future data by up to 59 second earlier or later. To better understand the effect, we create two separate experiments: one with all future requests reporting a 59 second earlier arrival time and one with all reporting a 59 second later arrival time. The scenario with the arrival times shifted later performs extremely well, nearly as good as with the true data when we don't delete the future requests. In contrast, shifting the apparent times earlier performed slightly worse than when the locations were perturbed. The conclusion we draw is that the gap between when the future virtual request is removed and the real request appears is giving time for the vehicles to be given alternative assignments that move them away from the requests actual location. When we falsely report the request as arriving 59 seconds later, we effectively shorten this gap at the cost of making the matched vehicle in the solution arrive up to a minute later than would otherwise have happened.

In contrast, creating future information by resampling from the requests' distribution performs very poorly. Perhaps, though, this is more of a function of the simple method we are using to incorporate the future information rather than a weakness

of the information itself.

In contrast to the 60 second threshold, the 120 second threshold shows more muted results. The extra gap appears to be reducing the impact of all forms of future information, both good and bad. We attribute this to the extra gap between the virtual requests being removed and the real ones appearing allowing the vehicles to deviate from reported future information.

These results, taken together, show that utilizing knowledge of the future is important and can improve performance but also shows that performance can be very sensitive to the quality and way it is used. While our simple resampling method performed poorly, some researchers have developed stochastic optimization models in similar frameworks that perform quite well. Perhaps an adaptation of the method to the trip based framework could provide a future direction towards achieving the optimal performance in real-world systems. However, finding the best possible non-myopic method is out of the scope of this work.

3.9 Conclusions

In this work we have shown several key points, the most import of which is that there is evidence of a throughput ceiling for requests in ridepool systems that was similar for a wide range of assignment methods. We showed this by demonstrating the cycling behavior of the Full-RTV and LA methods on a synthetic dataset that

had uniform demand distribution over time. This suggests that a similar effect could explain why the service rates for all of the methods we tested were so similar on the NYC taxi data.

In part because of this effect, we showed that a wide range of methods from full enumeration based techniques to simple extensions of weighted bipartite matching can yield nearly the same service rate on a full day of request data. This was a surprising result for us as each of the more sophisticated methods would have been expected to perform significantly better when considering the solution at a single iteration. Confirming that intuition, the offline experiments showed average assignments rates varying by up to 23% between the best and worst method. Even here though it is impressive that the bipartite assignment process of LA could assign as many requests as it did, likely indicating that the structure of the typical assignment problem is admitting a large number of feasible trips that do not require incoming requests to share rides with each other. Extending simple bipartite matching, swapping was shown to be a computationally cheap way to improve the quality of the solution.

Looking forwards, we believe that the next big space for improving service rates for ridepool systems lies in the modeling of future information. We showed in Section 3.8 that allowing both the Full-RTV and LA methods to look eight minutes into the future resulted in significant gains in service rates. Importantly, both methods enjoyed a similar improvement in service rate. This suggests to us that the limitations of myopic optimization may be more significant now that the difference between the

methods that actually solve the problem to make assignments.

CHAPTER 4

A NOTE ON TRANSFORMATIONS OF APPROXIMATION ALGORITHMS FOR ROBUST COMBINATORIAL OPTIMIZATION

4.1 Introduction

In combinatorial optimization many problems may never be solvable by algorithms that are guaranteed to run in polynomial time. The field of approximation algorithms has sought to get around this problem by defining a class of algorithms that always run in polynomial time and return a solution guaranteed to not be too much larger than the optimal solution (in the case of minimization). Formally, for an α -approximation algorithm for a minimization problem is a polynomial time algorithm that finds a feasible solution with objective no greater than α times larger than the optimal solution.

While much research has focused on solving problems with deterministic formulations, here we are interested in certain robust optimization problems whose form closely resembles deterministic combinatorial problems. These problems can introduce new difficulties by virtue of introducing a probability requirement in the formulation, all the more daunting when the closest deterministic problem is sometimes already hard to solve. In an ideal world, we would not have to start from scratch finding new algorithms to solve these problems.

This note is concerned with methods to automatically transform approximation

algorithms for deterministic settings into approximations for robust optimization problems. A method introduced by [51] provides such an automated transformation for three robust formulations of interest, tail risk, value-at-risk, and mean-variance models. However, the method they provide for solving tail risk formulations has a limitation on which approximation algorithms and which problem instances it can transform.

The motivation for this note is a tail-risk minimization problem related to the metric traveling salesman problem known as the stochastic on-time tour (SOTT) problem. In this problem we are given a graph $G = (N, A)$ where the cost of traversing an arc is given by a normal random variable with known mean and standard deviation. Given a budget B , the objective is find a tour that minimizes the probability that the length of the tour exceeds the budget. This problem has a natural deterministic parallel with the traveling salesman problem, which seeks to find a tour in a deterministic graph of minimum length. The best known deterministic approximation algorithm for this problem is the Christofides-Serdyukov which provides a $3/2$ -approximation¹. The limitations of the method proposed by [51] cannot transform any approximation algorithm with an approximation factor this large, thus motivating our search for an alternative.

The main contributions of this chapter are an analysis of the algorithm of [51] showing its limitations and the proposal of a new algorithm that circumvents these limitations to a degree.

¹A randomized algorithm with approximation factor less than $3/2 - 10^{-36}$ has been found in [30], but since we want a deterministic algorithm we use the $3/2$ factor instead

The note starts in Section 4.2 by reviewing the problem setting and the algorithm of [51], including a discussion of its limitations. Then in Section 4.4 we introduce a new algorithm and compare it with the algorithm of [51] and show its limitations as well.

4.2 Approximating Robust Combinatorial Problems

In this section we introduce the robust problems setting used throughout the note. Then we review the algorithm proposed by [51] for transforming approximation algorithms for deterministic problems into new algorithms that can solve the robust variants. We comment on the transformations in the context of the motivating problem, the SOTT problem and its deterministic parallel the metric TSP.

4.2.1 Robust Combinatorial Problems

Let $\mathcal{F} \subseteq \{0, 1\}^n$ be a feasible region to some combinatorial problem. Let W be a stochastic cost vector, with non-negative mean and variance vector given by $\mu \in \mathbb{R}_+^n$ and $\tau \in \mathbb{R}_+^n$ respectively. We make the following assumption.

Assumption 4.2.1. *Each element W_i is independent.*

The robust problem we are interested in is tail-risk minimization, seeking to find the solution that minimizes the probability the cost exceeds a given budget. For a

budget t , the tail-risk problem is given by

$$\begin{aligned} \min_x \quad & P(W^T x > t) \\ \text{subject to} \quad & x \in \mathcal{F}. \end{aligned} \tag{4.1}$$

In two natural cases this problem can be written in terms of the mean and variance of the solution.

1. $W_i \sim \mathcal{N}$ for all i .

In this case, we can rewrite the tail-risk minimization problem as

$$\begin{aligned} \max_x \quad & \frac{t - \mu^T x}{\sqrt{\tau^T x}} \\ \text{subject to} \quad & x \in \mathcal{F} \end{aligned} \tag{4.2}$$

2. W_i have an arbitrary distribution, with known mean and variance.

In this case there cannot be an explicit form for the tail-risk, but [51] shows that a one-sided Chebyshev bound can be written in terms of mean and variance as

$$\begin{aligned} \max_x \quad & \frac{(t - \mu^T x)^2}{(t - \mu^T x)^2 + \tau^T x} \\ \text{subject to} \quad & x \in \mathcal{F}. \end{aligned} \tag{4.3}$$

We will comment on the algorithm of [51] with respect to these two cases.

4.2.2 General procedure of Nikolova for Approximating Robust Combinatorial Problems

The algorithm proposed in [51] is geometrically inspired. First, since the objective functions for the two cases in the previous subsection can be written in terms of the mean and variance of the solution alone, they consider the projection of \mathcal{F} on the mean variance plane. This projection is possible since each W_i is independent, meaning the means and variances add. In this projection we can superimpose level-lines for each of the robust formulations. The objective is now to find the point in the projected feasible region that intersects the level-set line of maximum value.

This concept is illustrated in Figure 4.1. Suppose we have an oracle that can find solutions in \mathcal{F} minimizing any linear objective. One approach to finding the optimal solution to the robust problems would be to choose the linear objective to be perpendicular to a tangent line where the feasible region \mathcal{F} intersects the optimal level-line, assuming there is a way to find the linear objective. Figure 4.1(a) shows the projection of \mathcal{F} as the polygon P . The optimal robust solution, (m^*, s^*) , is given by choosing the linear objective denoted by the arrow and the optimal level set is given by λ^* .

The first thing to notice is that for many combinatorial problems of interest we only have access to a δ -approximate oracle. This means that even with the correct choice of linear objective we may get a suboptimal solution. This is illustrated in Figure 4.1(b). The shaded region shows all of the possible return values of the ap-

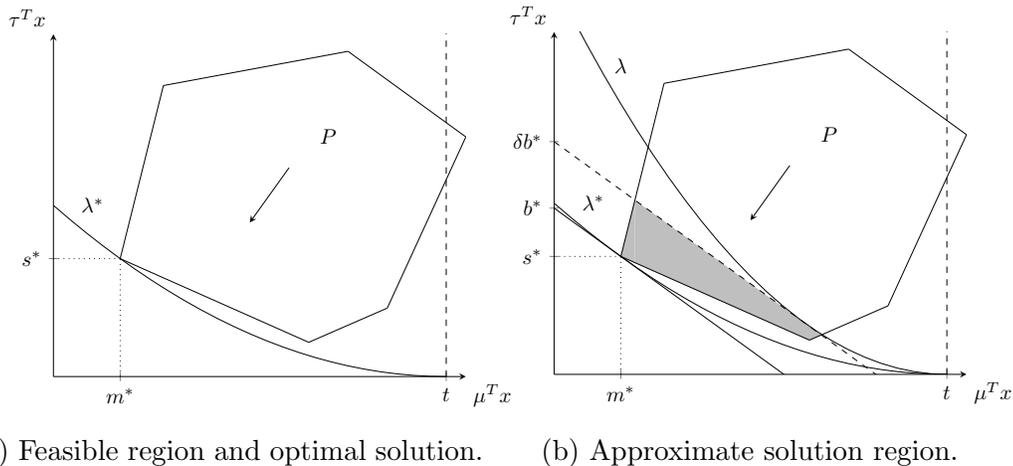


Figure 4.1: Visualization of the geometric approach of Nikolova. On the left, the feasible region is shown projected on the mean-variance plane, $\mu^T x$ and $\tau^T x$ respectively. The optimal solution can be found by solving for the linear objective shown in the center of the feasible region P and the optimal level set λ^* is shown. On the right, an approximate linear oracle is used with the same linear objective. The set of possible solutions is shaded in gray, bounded by a line parallel to the tangent line at the optimal solution resulting from the δ -approximation guarantee of the oracle. The worst case level set is λ .

proximate oracle. The level set λ is the worst-case level set among all that correspond to solutions in the shaded area. The difference between λ and the optimal level set λ^* defines the approximation guarantee for the robust problem.

The second thing to notice is that we cannot know a priori the correct linear objective to use. This is resolved in [51] by sampling several linear objectives and taking the best solution found.

No approximation guarantee is given in [51] for the general case with Chebyshev bounds, but in the case of W having normal distribution, they show that when the optimal solution has mean length at most $(1 - \epsilon)t$ and there is access to a δ -

approximate oracle for the deterministic problem, the robust problem can be solved within a factor of

$$\sqrt{1 - \left[\frac{\delta - (1 - \epsilon^2/4)}{(2 + \epsilon)\epsilon/4} \right]}. \quad (4.4)$$

4.3 Limitations of the Direct Oracle Method

In this section we will identify some of the limitations of the direct oracle method for solving robust problems, guided by our motivation to solve the stochastic on-time tour problem. The primary cause of limitations we study are approximation oracles with large approximation factors. Recall the deterministic parallel of SOTT is the metric TSP problem with a $3/2$ -approximation oracle.

To start to see the limitations of the direct oracle method, consider first the approximation guarantee given in Equation 4.4. The approximation factor is only well defined when the value in the square root is non-negative. Some algebraic manipulation shows that this implies that $\epsilon > 2\delta - 2$. When $\delta = 3/2$, as is the case for SOTT, this means that $\epsilon > 1$. However, combining this with the requirement that the mean value of the optimal solution is no greater than $(1 - \epsilon)t$, we can see that this gives no guarantee unless the optimal tour length is zero.

Intuition for this limitation can be developed from Figure 4.1(b). In general, we do not know where the projected feasible region P lies. This means that in our analysis we must assume the set of possible solutions returned by the approximate

oracle is any point in the gray region, which must be defined to be all points between the tangent line with intercept b^* and the line with intercept δb^* . If the mean value of the solution or the approximation factor δ are too large, this shaded region could cross beyond the budget line t . This means that the approximate oracle could return solutions with arbitrarily low probability of satisfying the budget.

This intuition will be formalized through analysis for both the robust formulations.

4.3.1 Normal distribution case

In this section we will prove the following theorem:

Theorem 4.3.1. *Suppose we have a δ -approximate oracle for solving the deterministic combinatorial problem. Suppose W_i are independent, normally distributed. Then the direct oracle method provides no approximation guarantee for the tail-risk problem unless the optimal solution value m^* is at most $\frac{2-\delta}{\delta}t$.*

Proof. Let the optimal solution be denoted as (m^*, s^*) with objective value λ^* , where m^* and s^* are the mean and variance of the solution. Let b^* be the y-intercept of the line that is tangent to the level set λ^* at the solution (m^*, s^*) . Suppose we use a δ -approximate oracle for the underlying combinatorial problem. Denote the returned solution as (m, s) with objective value λ and tangent line y-intercept b . In all cases assume that $m < t$.

From Lemma 4.3 in [51], we know that

$$\left(\frac{\lambda}{\lambda^*}\right)^2 = 1 - \frac{b - b^*}{s^*}.$$

Since the oracle is linear, we also know that $\frac{b}{b^*} \leq \delta$, which by definition is tight.

Therefore,

$$\left(\frac{\lambda}{\lambda^*}\right)^2 = 1 - \left(\frac{b - b^*}{b^*}\right) \left(\frac{b^*}{s^*}\right) = 1 - \left(\frac{b}{b^*} - 1\right) \left(\frac{b^*}{s^*}\right) \geq 1 - (\delta - 1) \left(\frac{b^*}{s^*}\right),$$

which is tight. The last inequality follows from $b \leq \delta b^*$.

In [51], lemma 4.4 shows that

$$\frac{b^*}{s^*} = 1 + \frac{2m^*}{t - m^*}.$$

Continuing from above,

$$\frac{\lambda}{\lambda^*} \geq \sqrt{1 - (1 - \delta) \left(1 + \frac{2m^*}{t - m^*}\right)},$$

which is tight.

Using the fact that $\delta > 1$, in order for the square root to be real valued, we must have

$$\frac{2 - \delta}{\delta} t > m^*.$$

□

As a natural consequence, we get the following lemma for optimization problems with weak approximations.

Corollary 4.3.1. *There are no risk-averse guarantees under the method of [51] for the normal probability tail model if the best approximation algorithm for the underlying optimization problem has $\delta \geq 2$.*

Further, notice that for the special case of the Christofides-Serdyukov algorithm with a $3/2$ -approximation using the last line of the proof we find that we require $m^* < t/3$. This means we can only operate in this regime if the optimal path is very short compared to our limit. This limit, however, is much too small for practical use in problems of our interest and therefore we say that there is no practical approximation guarantee for normally distributed edge weights for the SOTT problem.

4.3.2 General case with Chebyshev bounds

In this section we will show the perhaps more surprising claim that the direct oracle method does not provide an approximation guarantee for maximizing the Chebyshev bound, regardless of the approximation factor $\delta > 1$ or the mean value of the optimal solution m^* .

As before, let the optimal solution be denoted as (m^*, s^*) with objective value λ^* , where m^* and s^* are the mean and variance of the solution. Let b^* be the y-intercept of the line that is tangent to the level set λ^* at the solution (m^*, s^*) and let $-a$

be the slope of the tangent line. Using a δ -approximate oracle for the underlying combinatorial problem, denote the returned solution as (m, s) with objective value λ and tangent line y-intercept b .

Recall that the Chebyshev bound, in terms of the mean-variance plane, is denoted

$$\max \frac{(m - t)^2}{(m - t)^2 + s}.$$

We will derive a formula for the worst case objective value λ in terms of the other parameters of the problem. Then we will argue that the value can be arbitrarily small. First, we need some supporting facts.

First we will find the value of y-intercepts b in terms of a , t , and λ .

Recall that the y-intercepts are $b = s + am$. To start, we need to find the values of s and m in terms of our input λ , t , and a .

Lemma 4.3.1. *The slope, $-a$, of the line tangent to the optimal solution is $-\frac{2s}{t-m}$.*

Proof. To compute the slope of the tangent line we need to rearrange the objective.

Let $x = t - m$.

$$\lambda = \frac{(m - t)^2}{(m - t)^2 + s} = \frac{x^2}{x^2 + s}.$$

This implies $s = x^2 \frac{(1-\lambda)}{\lambda}$. The derivative is then $\frac{\partial s}{\partial x} = -a = \frac{2x(1-\lambda)}{\lambda}$. Substituting for λ gives the result

$$-a = -\frac{2s}{t - m}.$$

□

Lemma 4.3.2. *The intercept of the tangent line to a solution of value λ with slope $-a$ is given by*

$$b = at - \frac{a^2}{4} \frac{\lambda}{1 - \lambda}.$$

Proof. First we find the value of s . Starting with the result from Lemma 4.3.1 that shows $a = \frac{2s}{t-m}$, we find that $s = \frac{1}{2}a(t-m)$. Using the substitution $a = \frac{2s}{t-m}$ from Lemma 4.3.1 again, we get $s = \frac{1}{4}a^2(t-m)^2$. Solving the objective equation for $(m-t)$ yields $(m-t) = \sqrt{\frac{\lambda s}{1-\lambda}}$. By substitution, we get

$$s = \frac{1}{4}a^2 \frac{\lambda}{1 - \lambda}.$$

Now we find m continuing from the above computations. Starting from the value of s , we solve for a and again apply Lemma 4.3.1 to get

$$m = t - \frac{1}{2} \frac{\lambda}{1 - \lambda} a.$$

Substituting these computations in for the definition of b , we get

$$b = s + am = \frac{1}{4}a^2 \frac{\lambda}{1 - \lambda} + a \left(t - \frac{1}{2} \frac{\lambda}{1 - \lambda} a \right) = at - \frac{a^2}{4} \frac{\lambda}{1 - \lambda}.$$

□

Now we are ready to prove the desired result.

Theorem 4.3.2. *Suppose we have a δ -approximate oracle for solving the deterministic combinatorial problem with $\delta > 1$. The direct oracle method provides no approximation guarantee for the Chebyshev bound maximization problem regardless of optimal mean value m^* or approximation oracle δ .*

Proof. The tangent line to the worst case solution has y -intercept $b = \delta b^*$. From manipulating the result from Lemma 4.3.2, we find that

$$\frac{\lambda}{1-\lambda} = \frac{4t}{a} - 4ba^2.$$

Considering the relationship $b = \delta b^*$ and substituting in the above equation, we get

$$\frac{\lambda}{1-\lambda} = \frac{4t}{a} - \delta \frac{4}{a^2} \left(at - \frac{a^2}{4} \frac{\lambda^*}{1-\lambda^*} \right) = (1-\delta) \frac{4t}{a} + \delta \frac{\lambda^*}{1-\lambda^*}.$$

The value of λ is then

$$\lambda = \frac{(1-\delta) \frac{4t}{a} + \delta \frac{\lambda^*}{1-\lambda^*}}{(1-\delta) \frac{4t}{a} + \delta \frac{\lambda^*}{1-\lambda^*} + 1}.$$

Now let $x = (1-\delta) \frac{4t}{a} + \delta \frac{\lambda_1}{1-\lambda_1}$ so that

$$\lambda = \frac{x}{x+1}.$$

Since $\delta > 1$, $x \in (-\infty, y)$ with $y = \delta \frac{\lambda_1}{1-\lambda_1} > 0$. With a being a function of s^* , for any m^* there exists an s^* so that $x = 0$ and $\lambda = 0$. This concludes the proof that there is no approximation guarantee regardless of m^* and δ . \square

4.4 An alternative approximation

Here we show that in certain cases we can use an approach inspired by a different orientation in the space to get better approximations to the stochastic problems under certain circumstances. This uses a different geometric approach using a bicriterion-optimization approximation algorithm introduced by [47]. The theorem they provide that we will use, with minor modification, is

Theorem 4.4.1. *Let there be a ρ -approximation algorithm for a combinatorial problem. Let there be two objective vectors, c and d . Given a budget \mathcal{C} on metric c , for any γ there exists an algorithm that finds a solution to the combinatorial problem such that the solution exceeds the budget by at most a factor of $(1 + \gamma)\rho$ and exceeds the optimal value with respect to d under the budget restriction on c by at most $(1 + 1/\gamma)\rho$.*

We will use this theorem to derive an alternative procedure to find a new approximation algorithm to maximize the probability of on time arrival problem. Although the theorem was written in terms of solving problems on subgraphs, this is corrected by replacing the phrase “subgraph” with “solution to the combinatorial problem” in their proof.

To introduce how we use the bicriterion approximation algorithm to create an approximation in the risk-averse setting, suppose the optimal solution is (m^*, s^*) with objective value λ^* , where m^* and s^* are the mean and variance of the solution. Let b^* be the y-intercept of the line that is tangent to the level set λ^* at the solution (m^*, s^*) . Given a δ -approximate oracle for the underlying combinatorial problem, suppose we use the procedure from Theorem 4.4.1 to get a solution with mean length at most $(1 + \gamma)\delta m^*$ and that achieves the linear objective within $(1 + 1/\gamma)\delta$ of optimal. Under certain conditions, the worst case approximation guarantee comes from the case where the returned solutions has the largest possible mean value and highest variance of any solutions with that mean value.

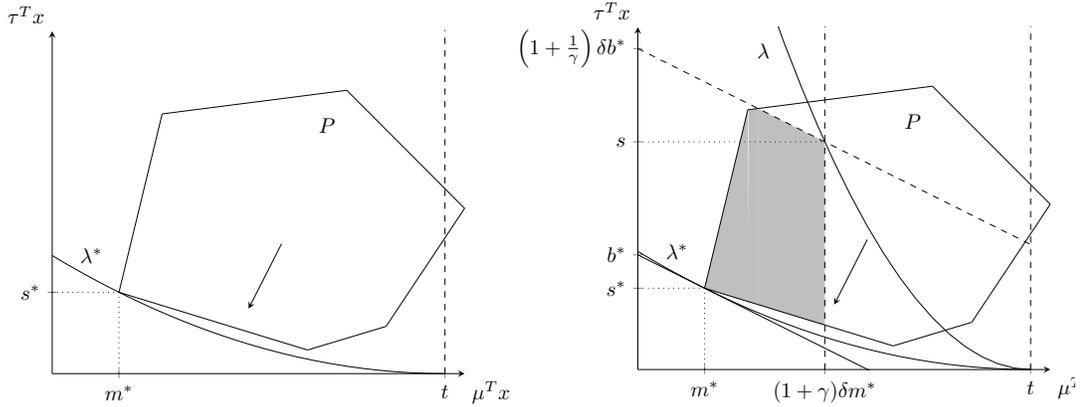
An illustration of the procedure is shown in Figure 4.2. On the left in Subfigure

4.2(a) the feasible region is shown, projected onto the mean-variance plane in the same way as the approximation in the previous sections. The optimal linear objective is shown with an arrow and the optimal objective level set λ^* is shown. Since finding the optimal solution requires an exact oracle, suppose now we use the bicriterion method to search for a solution. The method will give us two things: an upper limit on the the budget with respect to one objective criterion, in this case the mean value, at the cost of a weaker linear oracle that now only guarantees a solution within $(1+1/\gamma)\delta$ of optimal. The set of possible solutions this will return is shown in gray in the Figure, bounded by the vertical budget line, the weakened linear approximation line, and the feasible region P .

4.4.1 Applicability of the bicriterion-budget constraint

As the main differentiating component of this new approach, we make the assumption that the worst possible solution will be the one with largest linear objective and largest mean value, meaning the budget constraint is tight. Of course, this is not true for all problems and possible optimal solutions. Here we derive bounds on what values of m^* are guaranteed to satisfy this requirement as function of the problem parameters, t , δ , ϵ and γ .

In Figure 4.2(b) notice that the level set line λ touches the set of possible solutions in the upper right corner, meaning the budget constraint is active. However, it is possible that in some conditions the worst case level set may touch the solution set



(a) Optimal level set along with optimal linear objective. (b) Region of possible solutions from bi-criterion oracle.

Figure 4.2: Visualization of the bicriterion method. On the left, the feasible region is shown projected on the the mean and variance plane, $\mu^T x$ and $\tau^T x$ respectively. The optimal solution can be found by solving for the linear objective shown in the center of the feasible region and the optimal level set λ^* is shown. On the right, an approximate oracle along with the bicriterion approximator of [47] is used. The set of possible solutions is shaded in gray, bounded on the right by the budget $(1 + \gamma)\delta m^*$ and above by the standard tangent line for a δ -approximation oracle. The worst case level set is λ .

somewhere along the interior of the upper edge of the solution set, similar to what can be seen in Figure 4.1. Informally, to ensure the worst case level set touches on the right hand side, it is sufficient for us to constrain that the point on the level set with slope equal to the slope of the upper diagonal line of the solution set has mean value no less than the value of the budget $(1 + \gamma)\delta m^*$.

As before, we start by considering the tangent line to the solution (m^*, s^*) and consider the parallel line that corresponds to a worse linear objective, in this case a factor of $(1 + 1/\gamma)\delta$ larger. We will also assume that we only have approximate knowledge of the optimal linear objective, only within a factor of $1 - \xi$. We start

with a lemma about the set of points for which the local level sets have equal tangent slope.

Lemma 4.4.1. *For any solution (m, s) , the slope tangent to the level set at that point is given by $-2s/(t - m)$.*

Proof. Since $s = \lambda(t - m)^2$ for objective value λ ,

$$\frac{\partial s}{\partial m} = -a = 2\lambda(t - m) = \frac{2s}{t - m}.$$

□

Lemma 4.4.2. *Let the slope of the tangent line at the optimal solution be $-a$. Suppose that the actual slope our algorithm assumes is $-(1 - \xi)a$. Then for $0 \leq m \leq t$, all points on the line $s = s^*(1 - \xi)(1 - \frac{m - m^*}{t - m^*})$ also lie on level sets whose tangent line has slope $-(1 - \xi)a$ at the respective points.*

Proof. Substitute the value of s into the equation of slope shown in Lemma 4.4.1.

$$(1 - \xi)a = \frac{2s}{t - m} = \frac{2s^*(1 - \xi)(1 - \frac{m - m^*}{t - m^*})}{t - m} = (1 - \xi)\frac{2s^*}{t - m^*}.$$

Thus, the slopes are constant. □

Next, we want to know the slope of the line corresponding to the worst possible linear objective returned.

Lemma 4.4.3. *The line upper bounding the returned worst-case variance from the δ -approximate oracle as a function of m is given by*

$$s = \left(1 + \frac{1}{\gamma}\right) \delta s^* \frac{t + m^* - 2\xi m^*}{t - m^*} - (1 - \xi) \frac{2s^*}{t - m^*} m.$$

Proof. In general, we know that the line corresponding to a suboptimal linear objective is of the form $s = (1 + \frac{1}{\gamma})\delta b^* - (1 - \xi)am$. In this case,

$$b^* = s^* + (1 - \xi)am^* = s^* + (1 - \xi)\frac{2s^*m^*}{t - m^*} = s^*\frac{t + m^* - 2\xi m^*}{t - m^*}.$$

Using the above substitution for b^* and Lemma 4.4.1 gives the result. \square

The intersection of the variance upper bound given by Lemma 4.4.3 and the line given by Lemma 4.4.2 give the point at which the line representing the variance upper bound is the tangent line for the worst case level set. We want to constrain the budget for the bicriterion algorithm to always be less than or equal to the value of m at the intersection. This guarantees that the worst possible solution is the one with the highest mean value, and highest variance conditioned on maximal mean value.

Lemma 4.4.4. *The worst case solution returned by the bicriterion algorithm is the solution of largest variance among solutions with mean value equal to the budget B when*

$$B < \frac{1}{1 - \xi} \left(1 + \frac{1}{\gamma}\right) \delta(t + m^* - 2\xi m^*) - t.$$

Proof. We set the value of the line found in Lemma 4.4.2 to be less than the line corresponding to the worst case variance found in Lemma 4.4.3. Using algebraic manipulation we find that

$$\begin{aligned} s^*(1 - \xi) \left(1 - \frac{m - m^*}{t - m^*}\right) &< \left(1 + \frac{1}{\gamma}\right) \delta s^* \frac{t + m^* - 2\xi m^*}{t - m^*} - (1 - \xi) \frac{2s^*}{t - m^*} m \\ m &< \frac{1}{1 - \xi} \left(1 + \frac{1}{\gamma}\right) \delta(t + m^* - 2\xi m^*) - t. \end{aligned} \quad \square$$

In order to get the best possible approximation of the minimum tail-risk solution, we next need to find the setting of γ in the algorithm that will produce the best approximation guarantee. Before we can do that, first we need to establish the domain of γ that can be used in the approximation.

Lemma 4.4.5. *To guarantee the worst case solution is the solution with largest variance with mean equal to the budget when the optimal mean solution has value m^* and $\xi < 1/2$, γ must satisfy*

$$0 \leq \gamma \leq \min \left\{ \frac{t}{m^*\delta} - 1, \frac{-\xi m^*\delta + t\delta + t - t\xi + \sqrt{(\xi m^*\delta - t\delta - t + t\xi)^2 + 4(1 - \xi)\delta^2 m^*(m^* + t - 2\xi m^*)}}{2(1 - \xi)m^*\delta} \right\}.$$

Proof. On the one hand, since the mean value of the solution returned by the algorithm can be as large as $(1 + \gamma)\delta m^*$ and we need solutions with value at most t , it must be the case that

$$\gamma \leq \frac{t}{m^*\delta} - 1.$$

On the other hand, we also need to be sure that the mean value of the solution produced lies in the range of m such that the upper right solution is the worst case, as shown in Lemma 4.4.4. Substituting the worst case mean value $(1 + \gamma)\delta m^*$ for m in the equation above, we require that

$$(1 + \gamma)\delta m^* \leq \frac{1}{1 - \xi} \left(1 + \frac{1}{\gamma} \right) \delta (t + m^* - 2\xi m^*) - t.$$

Multiplying both sides by γ and solving for γ using the quadratic formula yields the desired result. Since the expression is an inequality we only need to give the upper

critical point (the other one is found by placing a negative sign in front of the square root). It can be seen that the interval between the two critical points is correct by considering $\gamma \rightarrow \infty$ makes the left hand side too large. \square

4.4.2 Approximation guarantee

Now we will determine the approximation factor generated by the algorithm.

Start by supposing we know the optimal solution (m^*, s^*) . Instead of considering the tangent line with slope $-a^*$, suppose we use our near-linear objective so that the tangent line has slope with $a = (1 - \xi)a^*$. Recall that for any solution (m, s) , the slope tangent to the level set at that point is given by $2s/(t - m)$, per Lemma 4.4.1. We will show the following Theorem, whose result is shown in Figure 4.3.

Theorem 4.4.2. *Given δ, m^*, ξ , and a γ satisfying the conditions of Lemma 4.4.5, the bicriterion algorithm provides an approximation ratio of*

$$\sqrt{\frac{1}{\delta(t - m^*)} \cdot \frac{(t - (1 + \gamma)\delta m^*)^2}{\left(1 + \frac{1}{\gamma}\right)(t - m^* + 2(1 - \xi)m^*) - 2(1 + \gamma)(1 - \xi)m^*}}.$$

We will now establish a few facts before proving the theorem.

Lemma 4.4.6. *When $m = (1 + \gamma)\delta m^*$, the variance of the worst case solution is*

$$s = \delta s^* \left(\left(1 + \frac{1}{\gamma}\right) \left(1 + \frac{2(1 - \xi)m^*}{t - m^*}\right) - (1 - \xi) \left(\frac{2}{t - m^*}\right) (1 + \gamma)m^* \right).$$

Proof. The lemma will be proved in three steps. First we determine the y -intercept for the level set of the linear objective passing through the optimal solution (m^*, s^*) .

Using that, we find the equation for the line of the level set of the worst case linear objective. Finally, we combine that with the budget to compute the worst case variance.

First, to find the y -intercept of the level set of the linear objective passing through the optimal solution we use the point and slope to yield

$$b^* = s^* + (1 - \xi)am^* = s^* + \frac{2(1 - \xi)s^*m^*}{t - m^*}.$$

Next, the worst case y -intercept for the actual realized solution is

$$b = \left(1 + \frac{1}{\gamma}\right) \delta b^* = \left(1 + \frac{1}{\gamma}\right) \delta \left(s^* + \frac{2(1 - \epsilon)s^*m^*}{t - m^*}\right).$$

Since the worst case solution has mean value $m = (1 + \gamma)\delta m^*$, the result can be obtained by substituting the above in $s = b - (1 - \xi)m$. \square

Now we are ready to prove the theorem.

Proof of Theorem 4.4.2. By definition, the optimal level set is

$$\lambda_*^2 = \frac{(t - m^*)^2}{s^*}.$$

Similarly, the value for the worst case level set, as given by $m = (1 + \gamma)\delta m^*$ and s found in Lemma 4.4.6, is

$$\begin{aligned} \lambda_n^2 &= \frac{(t - m_n)^2}{s_n} \\ &= \frac{1}{\delta s^*} \cdot \frac{(t - (1 + \gamma)\delta m^*)^2}{\left(1 + \frac{1}{\gamma}\right) \left(1 + \frac{2(1 - \epsilon)m^*}{t - m^*}\right) - (1 - \epsilon) \left(\frac{2}{t - m^*}\right) (1 + \gamma)m^*}. \end{aligned}$$

The result is then given by substituting the two previous values in $\frac{\lambda}{\lambda^*}$. \square

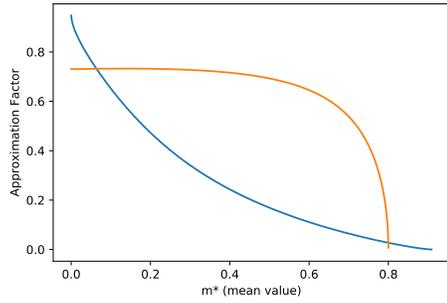
Thus it can be seen that the optimal choice of γ depends only on the mean value of the optimal solution m^* , the approximation factor δ , and the sampling term ξ . Though we do not solve the equation explicitly for γ , we are able to numerically select the best value of γ for any given m^* , δ , and ξ by searching the domain of γ defined in Lemma 4.4.5.

Figure 4.3 shows the tail bound approximation guarantees given both by this new algorithm as well as that of Nikolova. There are two notable advantages of the new algorithm that can be seen in the figures. First, the range of mean values m^* for which an approximation factor can be given is larger, especially as δ gets larger. This is clearly evident when $\delta \geq 2$ since the algorithm proposed by Nikolova cannot make any approximation guarantee in this range, even when $\xi = 0$.

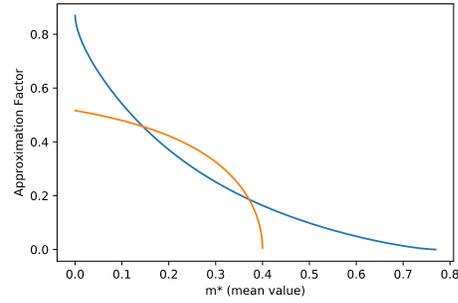
Note that the Figures show the case when $\xi = (t - m^*)/2t$, as selected in Nikolova's paper.

4.4.3 Issues in selecting γ

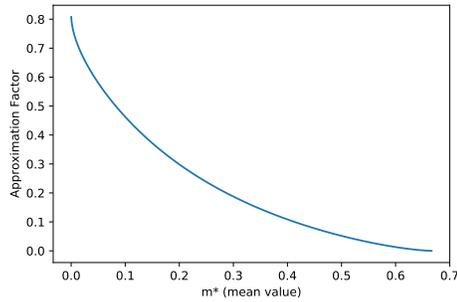
While the previous section gives an approximation guarantee for the problem when we know the optimal m^* , without an exact oracle we do not have access to m^* and therefore do not know the optimal γ . Here we will show that despite this, we can get good approximations in nearly all cases.



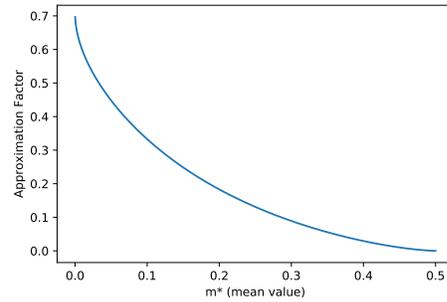
(a) Case of $\delta = 1.1$.



(b) Case of $\delta = 1.3$.



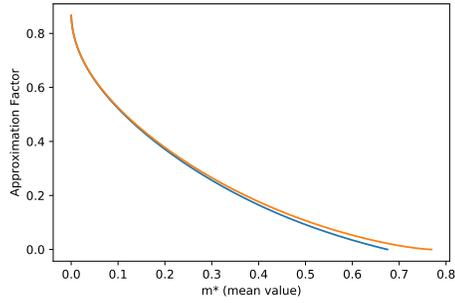
(c) Case of $\delta = 1.5$.



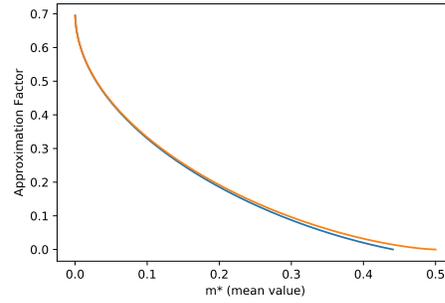
(d) Case of $\delta = 2.0$.

Figure 4.3: Approximation factors for the newly proposed algorithm (blue) and the algorithm proposed by Nikolova (orange), both using $\xi = (t - m^*)/2t$. The key feature of the new algorithm is that it continues to provide an approximation guarantee even as δ grows large, though it is sometimes outperformed by Nikolova's algorithm for smaller δ .

The basic approach is to check many values of m^* along a logarithmic scale. For upper and lower bounds m_u and m_ℓ on the value of m^* , we choose γ by assuming that $m^* = m_u(1 - \alpha)^i$ for some $\alpha \in (0, 1]$ and for i starting at zero and up to the first value such that $m_u(1 - \alpha)^i < m_\ell$. This means we choose the optimal value of γ for almost the correct value of m^* . As a consequence, in the worst case we may get a lower approximation guarantee for the problem than may have been theoretically possible given knowledge of the optimal mean value, or even have no approximation



(a) Case of $\delta = 1.3$.



(b) Case of $\delta = 2.0$.

Figure 4.4: Comparing the approximation guarantee when the mean value of the optimal solution is not known. The figures compare perfect knowledge (orange) with a lower bound on the guarantee given when we use $m = 0.8m^*$.

guarantee at all.

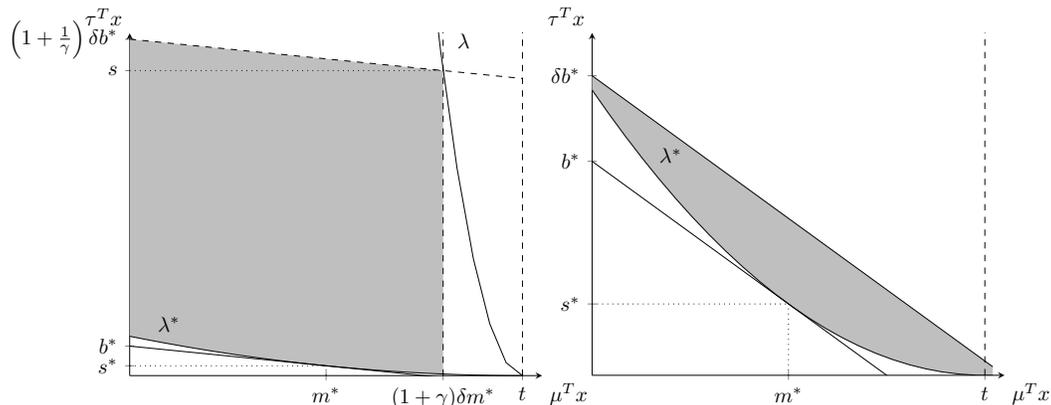
To get a lower bound for the guarantee given when approximating the optimal mean value within a factor of $(1-\alpha)$, we plot in Figure 4.4 the performance guarantees for all m^* assuming the use of the optimal γ for a solution a factor of $(1-\alpha)$ smaller. The top two figures show the guarantee made when we have the approximate value of m^* , shown in blue, and the optimal guarantee when we know m^* , shown in orange. The subfigures below them show the corresponding ratios of the two guarantees. While the ratios are disfavorable when the mean value is larger, the ratio is nearly 1.0 when the mean value is even just a little smaller.

4.4.4 Some Geometric Intuition

In this subsection we cover some of the intuition behind why the algorithm works better (and sometimes worse) than the algorithm of [51] depending on the mean value of the optimal solution. Recall that the bicriterion method has two components, a linear objective that is being optimized and a budget constraint that limits the maximum mean solution value returned. Depending on the mean value of the optimal solution a wide range of values for γ can be used, with increases and decreases in γ having opposite effects on the resulting approximation guarantee for the linear objective and the budget constraint. In general, the best solution falls into one of three categories:

1. When the mean value is large, using small γ for a tight budget constraint.
2. For lower/intermediate mean values, medium values of γ balance between budget and linear objective.
3. For small mean values, even for large γ the budget constraint becomes more powerful than the linear objective itself.

We present the geometric intuition through a sequence of figures. In each figure we give a geometric demonstration of the bicriteria method and the simple linear approximate oracle method side-by-side on the same problem. Shading illustrates the set of possible solutions that might be returned using each method. The optimal level set λ^* is shown as well as the worst case level set λ when it exists. In all of



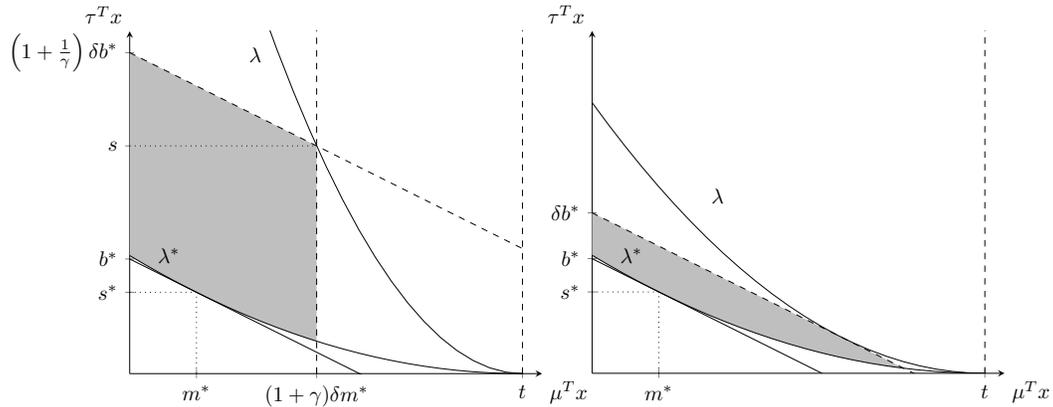
(a) Bicriterion method has an approximation guarantee. (b) Simple linear oracle cannot make a guarantee.

Figure 4.5: Geometric demonstration for bicriteria method (left) and simple linear oracle (right). In this case the mean value is large, with $m^* = 0.5t$ and $\gamma = 0.14$. The regions of possible returned solutions are shaded in gray.

the examples we suppose we have a combinatorial problem for which we only have a $\delta = 1.4$ -approximate linear oracle for the deterministic problem.

First, in Figure 4.5 we suppose a case where the optimal solution has large mean value $m^* = 0.5t$. Using the bicriteria method with $\gamma = 0.14$ we are able to show visually that the set of feasible returned solutions, in the worst case, have tail-risk corresponding to the curve λ . However, in the case of the linear oracle the set of solutions crosses the budget line, meaning we can get solutions whose mean value is larger than the budget. Because of this, it can be seen that no non-trivial approximation of the tail-risk can be given under such a procedure.

Next, in Figure 4.6 we consider a case where the optimal solution has an intermediate-magnitude mean value $m^* = 0.17t$. In this case, the simple linear

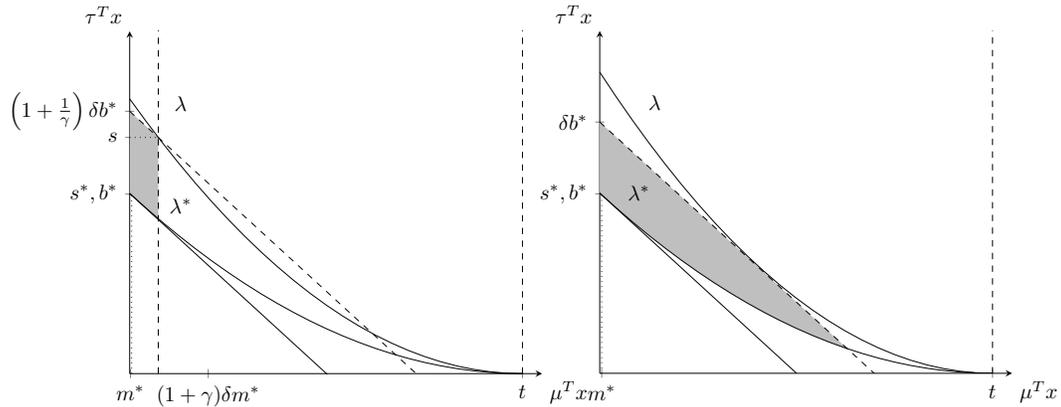


(a) Bicriteria method has a weak approximation guarantee. (b) Simple linear oracle method has a stronger guarantee.

Figure 4.6: Geometric demonstration for bicriteria method (left) and simple linear oracle (right). In this case the mean has an intermediate value, with $m^* = 0.17t$ and $\gamma = 1$. The regions of possible returned solutions are shaded in gray.

oracle outperforms the bicriteria method. This can be seen since the y -axis of both graphs spans an equal range and the ratio of the scale parameters for the level set curves λ^* and λ is clearly larger in the left subfigure.

Finally, in Figure 4.7 we consider a case where the optimal solution has very small mean value $m^* = 0.002t$. In this case the bicriteria method gives a slightly stronger guarantee, powered by the budget constraint removing the possibility of solution with larger mean value.



(a) Bicriteria method takes advantage of (b) Simple linear oracle has slightly weaker guarantee.

Figure 4.7: Geometric demonstration for bicriteria method (left) and simple linear oracle (right). In this case the mean value is small, with $m^* = 0.002t$ and $\gamma = 25$. The regions of possible returned solutions are shaded in gray.

4.5 Conclusions

In this note, we reviewed a method proposed by [51] for approximating the tail-risk minimization problem. We showed that the method has limitations with respect to the mean value of the optimal solution and the approximation ratio of the best available δ -approximate oracle. Using bicriterion approximations, we gave an algorithm that overcame these limitations in some contexts.

Unfortunately, some of these limitations may be inherent in the problem. For example, consider the way that both methods we discuss must not return solutions with mean value greater than the budget constraint. This is a limitation of our geometric approach to solving the problem since in those cases we cannot rule out low variance solutions, whose tail-risk can be arbitrarily close to 1. This means that

it is NP-hard to give an approximation guarantee to the tail-risk problem unless the mean value of the optimal solution is at most t/δ .

While there is likely no algorithm can give an approximation guarantee for tail-risk when the optimal solution has mean value greater than t/δ within the geometric framework we have presented, we cannot rule out the possibility that such algorithms might exist under another approach or for special cases of interest. Future work may be able to address this area of the problem as well as provide stronger approximation guarantees for instances whose optimal mean value is close to t/δ .

CHAPTER 5

THE BENEFITS OF SIMULATION-INTENSIVE MICROTRANSIT PLANNING: TWO CASE STUDIES

5.1 Introduction and Motivation

The past decade has seen a surge of interest in on-demand ridehailing services, allowing users to request rides in real-time using apps on their phones. These services improve upon older system designs that required users to book by phone and sometimes book significantly in advance. By generating schedules and routes dynamically throughout the day, ridehailing services add convenience beyond what can be offered by most forms of public transit. While the largest ridehailing services can cover entire metropolitan areas, small services, typically referred to as microtransit systems, have become the subject of study and aim to either connect users to existing fixed route transit, fill transit gaps in off-peak times or low-density areas, or serve as paratransit providing accessibility services [44]. In general, microtransit refers to on-demand shared mobility services using smaller vehicles (compared to buses). These services can serve predefined pickup points, can dynamically make new pickup points for individual passengers, or can ask riders to walk to shared pickup points. These smaller systems typically take advantage of ridepooling for efficiency, the practice whereby multiple riders with similar itineraries can share the same vehicle.

Public transit agencies have a growing interest in microtransit as they view it as an

opportunity to supplement or increase accessibility to their existing services. Increasingly, these services are implemented with the help of public-private partnerships. These private organization can provide a range of services from fleet acquisition, customer facing phone-apps, to running the real-time components of operations. Despite private organizations offering a means for operating a given service, the scarce availability of planning tools limits the ability of agencies to plan new services, especially when the proposal contains new service elements. Microtransit services require planning not normally required for traditional fixed route services, including service region, user facing apps and operational technology, and a wide variety of possibilities in the space of service design due to their tailoring to their smaller, use-case specific setting. Without tools to aid in the planning process, agencies are left to create their own processes for designing the services, possibly spending effort rediscovering what has already been learned by other agencies.

The two case studies presented in this chapter begin to address this problem using a simulation model to test potential configurations of the potential microtransit services offering first- and last-mile connections to regional transit services. While the aim of the simulation model is not to provide novel estimates of demand or customer behavior, it provides a new and flexible tool to understand the impact on system performance caused by a wide range of potential configurations and demand assumptions not previously available to transit agencies, including limits on waiting, riding time, maximum distances customers can be required to walk, frequency of accessible lift deployments, and more. The simulation work was conducted by a research team on simulation and algorithms at Cornell University in partnership

with Ford, King County Metro (KCM) in Seattle, Washington, and Metro Transit in the Minneapolis-Saint Paul metropolitan area.

The goal in each of the case studies is to create FMLM shuttles services that allow riders in areas underserved by public transit to connect with the transit network. Each of the transit agencies presents a unique setting. For KCM, the project aims to cover an area with mixed use in Kent, part of the service area being residential while the other part containing a large employment area. The goal of the service is to provide access to jobs by connecting the residential and industrial areas, and connecting both of those areas to Kent railway station, which hosts regular commuter trains and many fixed route bus lines. For Metro Transit, the project aims to create a feeder service for a bus rapid transit (BRT) service. The BRT, as well as traditional bus lines, serve the area and the goal is for a FMLM service to allow riders to travel to stations that connect to the bus routes.

In each case, the final service that is being considered is the result of a multi-year planning process, working from site location all the way to service model and design. While this is not the first project that seeks to create a microtransit service, a particular goal is to illustrate the way a flexible simulation tool can be used inform the design process. Throughout the case studies we will go into detail about which questions the simulation sought to answer, what the results were, and how the agencies reacted. Although the simulation tools can compare alternatives, ultimately the agencies may end up choosing against some of the recommendations made. In these cases, its important to note that the decision is still informed by the data.

This chapter is organized as follows: Section 5.2 provides a background on microtransit deployments in general and introduces the need and purpose of this work. Section 5.3 discusses the general process used to plan the services and the form of communications used between the agencies and the research team. Next, Section 5.4 presents the modeling framework used for passenger assignment and discusses changes to the model that were needed to adapt to the demands of the case studies. Section 5.5 describes the results of the iterative process of working with the agencies, showing the progress and development of the systems across several meetings held with both agencies. Finally, in Section 5.6 we give concluding remarks.

5.2 Literature Review

This literature review, and portions of the introduction, have been adapted from a technical report [82] by the members of the project team, including the author of this dissertation.

Microtransit sits within the category of shared mobility along with jitneys, paratransit, dollar-vans, and employer-provided shuttles [63, 64]. In its most general form, microtransit can be defined as an app-enabled, pooled ride service in a small shuttle or van for which routes, schedules, or both vary based on demand [14, 44, 64, 65, 63, 75, 80]. While early research defined microtransit as a form of private transit, similar to a jitney but enhanced with information technology [63], microtransit now also encompasses public-private partnerships with goals such as

improving transportation quality for elderly, disabled, and low-income populations, serving new transit riders, and enabling public transit agencies to redistribute resources more efficiently [44].

Through a study of more than 60 microtransit deployments across the United States, Lucken et al. [44] developed a typology categorizing microtransit private-public partnerships. They found four categories: first-mile/last-mile (FMLM) to or from public transit stops, serving low-density areas that cannot support fixed route service, supplementing coverage during off-peak hours, and paratransit applications to supplement or replace the transit agency's existing paratransit service. Each typology was found to have advantages and disadvantages. FMLM models benefit transit by effectively subsidizing rides to and from transit stations, though it can compete with other FMLM options such as walking, biking, and riding the bus. Serving low-density zones provides better service than infrequent bus services but can directly compete with any existing transit services in the area. Off-peak models can provide trips at a lower cost to the agency than fixed route services but these services can add coverage at times not normally served by transit, thus representing a possible cost increase for the agency. Paratransit models lower per-trip costs, but this may stem from less driver training and agencies may not have wheelchair accessible vehicles ready in their fleet.

In a review of case studies, Curtis et al. [14] found that the dominant target market of microtransit deployments was FMLM, providing transit connections when other means of access were limited. Volinski et al. [75] found that microtransit

service areas are often in the range of five square miles, though their study found systems ranging from one up to thirty square miles. Service area design was affected by a large number of factors including demand levels and desired travel times, though sometimes political boundaries were used for simplicity. The large flexibility in service design afforded to microtransit systems can be both a benefit as well as a cost. On the one hand, it enables operators to optimize the service structure by time of day. In one case study, customers in peak service hours cared most about directness and reliability while in off-peak hours customers cared most about convenience and flexibility [55]. The optimization engine in that study was able to take advantage of this by ensuring certain connection stops were visited on a frequent basis during peak hours while allowing vehicles to roam more freely during off-peak hours. As Perera et al [55] note, having too much flexibility can be a liability when spatial and temporal freedom hinder the ability to group trips. Nonetheless, as illustrated by New York-based Via, when done properly fully dynamic services can be successful [64].

Understanding the right way to design and plan microtransit services is important because of the wide range of benefits microtransit can offer, from reduced parking demand and emissions to increased access to public transit in underserved areas. However, future access to these benefits needs established frameworks for modeling and evaluating new services [26] and organized, intentional planning between agencies and the service provider [14]. The case studies in this work illustrate a way to overcome problems experienced by previous services by using an approach that actively engages the transit agency with a simulation and modeling team.

5.3 Service Design Process

Planning a new service requires more than just an objective. While agencies may have little trouble forming a goal and purpose for a new service, launching a successful service requires a detailed understanding of the service area, customer demand, system design parameters, and the capabilities of the technology used to operate it. Because of this, it is essential that the agency planning the service have a simulation environment for the design process itself. Simulation environments are useful both by giving the agencies a risk-free way to explore design decisions, such as fleet size or service area, as well by clearly communicating the capabilities of the technology which may be provided by a third-party vendor, an important step in developing RFPs prior to deployment.

The service design process has to include a combination of local knowledge, general goals for the service, data to support analysis of potential service areas, and a simulation tool that can demonstrate the relative impacts of changes in policies. This section will discuss some of the steps in the design process and how they fit into the larger planning picture.

5.3.1 Demand Modeling

Predicting demand can be difficult when planning a new type of service since there are limited sources of data available to extrapolate demand levels. For a given service

area, there are two important dimensions of demand: time and space.

In the two case studies, the intention is to design a first/last mile transit system that feeds into existing fixed route services. One natural way to estimate the demand by time of day is to look at the temporal distribution of demand, both in and out, of fixed route services that serve each hub. Some agencies have the ability to report the number of passengers boarding and alighting from the fixed route services at each stop. This data can either be used to create demand intensities split into buckets, perhaps an hour or half an hour in length, or can be used to generate random numbers of passengers that enter the system at times roughly corresponding to the time the bus serves the hub.

Predicting demand over space can be difficult with limited data. For lack of more information, assuming a uniform spread of demand may be the best option though demand is unlikely to be truly uniform in the real world. In all likelihood, demand will have spots of higher concentration and lower concentration. Concentrated areas can generally be served more efficiently by a shared service since there is less time overhead in picking up multiple requests that spatially closer together. Uniform distributions in that sense give a conservative view of the distribution of demands since it will be more challenging to serve it. However, if the demand is actually concentrated near the boundary of the service area the uniform distribution may create demand that is easier to serve than real demand.

Another way to plan for service demand is by dividing the space into regions that are expected to have different ridership characteristics. In King County, for

example, the service area studied could be partitioned into a distinct residential zone and industrial zone. The dynamics of each space were estimated by aggregating data from fixed route lines that passed through each zone. A distinction was found in the time varying component in the fraction of riders heading into and out of the hub.

5.3.2 Policy Exploration

Microtransit systems are defined by many parameters and design choices. Simulation tools can help inform decisions that need to be made including

- Service Area. Service area impacts a service through request density and travel times on roads. Larger areas may capture a larger user base while hurting key performance metrics for the system.
- Waiting Time/Max Delays. In contrast to serving all riders on a first-come first-serve basis, instead the system can aim to maximize ridership throughput subject to hard constraints. If these constraints cannot be satisfied for a request, the system must reject the request so they can seek other transit options. Waiting time refers to the maximum amount of time that can elapse between a request being submitted and the vehicle picking them up. Delay means the excess time from when a request is submitted until they are dropped off at their destination beyond what would be experienced by someone leaving in a personal vehicle at the time the request was submitted.
- Connection constraints. When a first mile trip wishes to connect to a particular

fixed route, there are several possible constraints to consider. For example, there can be a requirement that passengers that are accepted to the system must get to their transit connection at least five minutes before its departure. Or, perhaps instead we accept all passengers but must simply warn them if a connection cannot be made.

- Number of shuttles. Determining the number, types, and capacities of vehicles to use introduces a tradeoff between cost and quality of service. Fewer vehicles cost less while more vehicles serve more people with lower waiting times.
- Dynamic vs Static fleet sizing. A static fleet plan uses the same number of vehicles all day. A dynamic fleet plan may add or remove vehicles during peak and off-peaks hours.
- Additional parameters. This captures anything else of interest to the agency. It may include creating different maximum waiting times depending on whether a trip is first- or last-mile, or it could be parameters surrounding how far riders can be asked to walk to meet a vehicle.

When testing the effects of different policies, we start each round of simulations by building a base case. Then, we extend the base case by varying one of the dimensions of the problem. Often we present the results in a way that let's the x -axis be the demand level. We do this since the demand level is difficult to accurately predict for a new type of service. This allows the impact of the effect to be seen in a variety of possible settings. A typical representation of experimental results is illustrate in Figure 5.1.

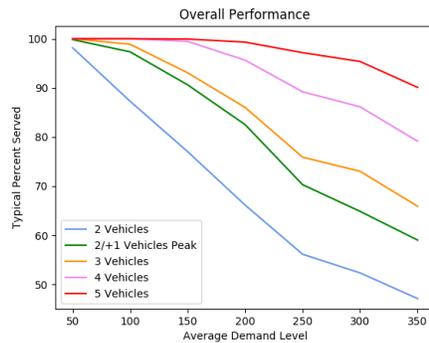


Figure 5.1: An illustration of comparing alternative scenarios. This y -axis of this plot shows the percentage of customers served in a variety of experiments. Each color line represents a different scenario. The x -axis represents the average daily demand so the difference in each scenario can be understood in the face of uncertainty about the actual demand level.

Presentation of results is very important in exploring possible policies since small nuances in the way an experiment is being performed or the way a constraint is being modeled may not be understood well by all parties. For example, there can be a difference between rejecting rides that would require waiting more than a specified amount of time versus operating a system that achieves a satisfactory average waiting time. Or a constraint may have unclear impact, such as the connection constraint. This constraint allows first mile riders to request a specific fixed route connection they want to reach. An important decision is whether we should reject requests when we cannot get them to the fixed route connection in time or whether we should allow it but give users a warning. Clear presentation of results is essential for leading conversations on these questions.

5.3.3 Communication and Dialogue Process

The study of microtransit service relies on the expertise of both a transit agency and a simulation experiment specialist. Transit agencies have unique knowledge in their customer base and the behavior and results desired from the system. The specialist may have a deep knowledge of the computational side of the problem or available technology while not knowing the particular goals or demographics of the area. While the details of this process are discussed in detail in the results presented in Section 5.5, here we give a summary of some key lessons learned.

One major hurdle that slowed down the process was not knowing which data to explore. Transit agencies have the capability to collect vast amounts of data on their existing services but were not be sure which data would be helpful for us to use in designing experiments for the pilots deployments. We found that useful data included boarding and alighting data for existing fixed route services in the candidate areas as well as data on any previous pilots that may have been run by the agency, even if they were in a different part of their jurisdiction. We used this to estimate the relationship between fixed route ridership and first/last mile service used, though an exact demand forecast was difficult due to localized differences in employment, demographics, and service and connection quality.

Another difficulty we encountered was the agency not knowing or understanding the range of capabilities offered by algorithms we intended to provide. Reasonably, this can be an issue when an agency begins work with any external operations

provider. This made early dialogue difficult between our group and the agencies when they didn't know which questions need to be asked or what types of impacts to be looking for.

In the case studies presented in this work, we found utility in adopting an iterative process. First, time would be spent working on developing either a new demand model, algorithmic capabilities, or set of experiments. Then the results would be reviewed and a decision would be made about which questions needed to be addressed by the agencies. Sometimes these questions were ones the agencies had already explicitly raised, and sometimes these questions related to details of system performance that the agency may not have been aware of or had not thought of asking due to their lack of familiarity with our particular simulation tool. Then a presentation would be made that explained the impacts of different ways of answering the questions, utilizing concrete language to describe the impact to strengthen the solicitation for feedback from the agency. Sometimes we would even present ideas that we reasonably expected to be rejected when we felt they would help solicit comments or illustrate another recommendation we had through comparison and contrast.

We repeated this process several times, as can be seen in Section 5.5. Over time, as the research team and agencies gained familiarity with the process and design of the system developed, the purpose of the meetings changed. Early meetings concerned general goals and selecting a general location for the service. Later, meetings focused on the boundaries of the service and discussions around modeling demand. Last, the final round of meetings concerned particulars of the system, such as hours, fleet size,

and other quality of service and user experience choices.

5.4 Model Background

At a high level, the ridepooling model that we study is an centrally-controlled, on-demand service. Centrally-controlled means that all of the vehicles obey the instructions given by the operator of the service. This is in contrast with other models that treat vehicle operators as independent contractors that can accept or reject each job. The assumption of central control may be reasonable for settings with either autonomous vehicles or when vehicles are managed by a single organization, such as government transit agencies. The on-demand aspect means that we do not allow early booking. Rather, we cannot know in advance when requests will be made and when a request is made we assume the rider wants to start their trip as soon as possible.

In order to operate a microtransit service there needs to be an algorithmic framework that can process the demands and service policies and output decisions about how to match riders and vehicles, as well as computing the paths that vehicles take. As a tool for operation, the assignment algorithm is also an important component of a simulation environment that allows agencies to study the relative performance of hypothetical new services under a variety of designs. In other words, the simulator provides a sandbox in which we can study the impact made by changing service fundamentals and the way we assign passengers to vehicles. While there are many

flavors of simulation tool used in the transportation community, here we will study a simulator that works with the following three core components:

1. Demand profiles
2. Environment simulator
3. Assignment algorithm

To start an experiment, we input a pre-prepared file with the demand profiles and service policies. A demand profile is a list of requests where each request has a stated origin, destination, and time of request. When an experiment is run, the demand profile is used to mimic the real-time arrival of requests to the system. The environment simulator knows how many vehicles there are, where they are and who is on board, has a copy of the road network, and serves the purpose of keeping track of the state of the system. When a vehicle is told to follow a planned path to pick up a group of requests, the environment simulator will begin to move the vehicle along the planned path, according to the travel times on the road network, and perform the pick ups and drop offs. The assignment algorithm is the final piece that, at regular intervals, inspects the current state of the environment, takes in the newly visible requests from the demand profiles, and generates a new set of assignments and paths for the requests and vehicles. Under some circumstances some requests can be rejected. Further, since this is a shared system, the assignment algorithm is allowed to preempt the current path of any vehicle and insert new requests. But, once a request is assigned to a vehicle it can never be rejected in the future by new computations.

The algorithms we use come from the framework of [3], which is designed for large-scale, high-capacity systems. While the microtransit systems we study utilize high capacity vehicles, there are many differences caused by the change in scale of the systems as well as special constraints that may be imposed on such systems. In addition, while ridepool frameworks generally target systems where requests have complete freedom on choosing origin and destination pairs, we study first/last mile systems where all rides are to or from a fixed set of hubs.

The rest of this section introduces the general ridepooling framework of [3] and then describes how its properties lend to modification for first/last mile microtransit settings.

5.4.1 General Ridepooling Framework

In the general on-demand ridepool problem, the inputs given are a graph $G = (N, A)$ representing a road network, a set of vehicles V with possible heterogeneous capacities that are centrally controlled by the operator, and a set of passengers \mathcal{R} that arrive online. Each passenger request $r \in \mathcal{R}$ is specified by a system entry time, an origin, and a destination.

The system is constrained to provide a set of minimum quality of service (QoS) requirements to all requests that are accepted into the system. These requirements are general, can be request specific, and can be chosen differently by each operator. For example, an operator may impose a maximum waiting time and a maximum

delay constraint, ensuring that all accepted passengers are picked up within t_w^{\max} time of submitting their requests and that the extra time traveling in the vehicle due to a shared ride is no more than t_s^{\max} . An operator may opt to choose settings such as $t_w^{\max} = 5$ minutes and $t_s^{\max} = 15$ minutes.

If a request cannot be served using a trip that meets the QoS requirements, it must be rejected. The feasibility of a trip is determined by solving a constrained traveling salesman problem, where among other constraints we must consider the capacity of the vehicle at each point in time. Thus, matching a passenger to a vehicle in a particular batch does not require that the passenger will actually board the vehicle during that batch or that the vehicle has capacity for the new passenger at that precise moment in time. In deterministic settings¹, since we gain no new information over time that reduces the set of constraints in the system (i.e., no information that allows for previously infeasible pickups to be feasible) if a request is rejected in one batch, it will be infeasible to find a trip to serve them in later batches as well.

The underlying passenger assignment framework used in this work is based on the trip-oriented formulation of [3], which solves the assignment problem by decomposing it into separate subproblems. The online arrival of requests is split into batches of uniform length and computations are performed once for every batch. For any vehicle $v \in V$ and any set of requests $r_b \subseteq \mathcal{R}_b$ all in the same batch, a trip $t = (v, r_b)$ is a pairing of the vehicle and the requests. For any given trip, an optimal route for the vehicle and associated cost c_t can be found by solving a constrained

¹deterministic travel times and no dropouts

traveling salesman problem (CTSP). The method used in [3] for trip generation is very general and allows for the insertion of additional constraints, something we will take advantage of as we extend the model for the agencies.

The formulation of [3] is presented as an optimization formulation that first assigns a large penalty for requests that are not assigned and second penalizes the cost of serving the assigned trips. They use a binary variable ϵ_r to track which requests are rejected and then apply a large penalty, M . For the secondary objective, the cost c_t is left unchanged. The optimization problem remains general, however, since one could choose $M = 0$ and then use an arbitrary function to assign cost c_t , possibly negative, for each trip. The operator's assignment problem in a non-EV ridepool system would thus be the solution of

$$\begin{aligned}
& \min_x \sum_{t \in Trips} c_t x_t + M \sum_r \epsilon_r \\
& \text{subject to} \quad \sum_{\substack{t \in Trips: \\ v \in t}} x_t \leq 1 & \quad \forall v \in V \\
& \quad \quad \quad \sum_{\substack{t \in Trips: \\ r \in t}} x_t + \epsilon_r = 1 & \quad \forall r \in \mathcal{R}_b \\
& \quad \quad \quad x \in \{0, 1\}^{|Trips|}, \epsilon \in \{0, 1\}^{|\mathcal{R}_b|}
\end{aligned}$$

where here $Trips$ is the set of feasible trips and the solution is constrained to assign at most one trip to each vehicle and choose at most one trip containing each request. Notice that this relies on full enumeration of all feasible trips. The authors of [3] observe that when the quality of service requirements are tight, as is typical in ridepooling applications, the set of feasible trips is small enough to approximately

enumerate in real-time.

Trips are found using a shareability graph, a concept introduced in [60]. A shareability graph, as extended by [3], is an undirected graph $G = (N, E)$ in which the node set $N = V \cup \mathcal{R}$ is comprised of nodes for each vehicle and request. The edge set E contains all pairs (v, r) , $v \in V, r \in \mathcal{R}$ if vehicle v can serve r and all its current passengers while satisfying the quality of service constraints. E also contains all pairs $r_1, r_2 \in \mathcal{R}$ if it is possible for an ideal hypothetical vehicle to serve both requests (for example, if the requests are five minute apart and the maximum waiting time is two minutes even an ideally located vehicle cannot serve both).

Every feasible trip induces a clique in the shareability graph containing exactly one vehicle node. This is the case since the edge set was constructed from necessary conditions for the feasibility of a trip. While the general problem of finding cliques is hard, [3] proposes a method for searching for the cliques in the shareability graph that is exact and, when heuristics are used such as restricting requests to be paired with the nearest 30 vehicles, can be solved in real time.

It may be the case that after the assignment process some requests are not served and some vehicles are left unassigned. If this is the case, a minimum cost matching based on distance is performed between the vehicles and the rejected requests. Those vehicles are then routed to the locations of the rejected requests as a heuristic for rebalancing to areas that require additional coverage. This is only one strategy for rebalancing; more sophisticated rebalancing strategies have also been proposed [40, 77].

5.4.2 Modification for Microtransit Systems

While from a descriptive standpoint there are many differences between ridepool systems in general and first/last mile transit systems, from an algorithmic point of view the primary difference between such systems is in the definition of constraints. This can lead to two problems: 1) computing routes and feasibility for potential trips and 2) handling the possible exponential increase in the number of feasible trips caused by the relaxation of many constraints, which in the general setting had been used to prune down the size of the list of feasible trips.

First we address the issue of computing routes for potential trips. A key advantage of the trip based framework is that the evaluation of potential trips allows the algorithm to be separated into modular components, whereby adding new constraints to routes that are served does nothing other than change the black-box function that is used to evaluate them. There are several constraints that we might consider in the first/last mile setting that are different from what was proposed in [3], and some the might be considered in either that were not addressed. This includes

1. Separate maximum wait times for first-mile origins vs last-mile origins
2. Longer boarding times for requests that need a lift to board/alight
3. Constraints for first-mile trips to meet specific transit connections
4. Dynamic fleet sizing, vehicles entering/exiting service
5. Heterogeneous request population, such as some requiring lift deployments to board

Each of these constraints can be naturally incorporated into the trip evaluation process through a simple construct-and-reject method. This method can be implemented through a depth-first search that builds paths by sequentially considering every possible next step the path could follow. Each possible next step is evaluated to see whether adding it will cause one of the constraints to be violated. If so, that particular decision is rejected and other choices are tried. If there is a path that can serve all of the requests while meeting the constraints, at least one of these searches will add every single request to a complete path. In that case, the solution is the complete path that has the best objective value, however we choose to define it. If there is no choice that yields a solution, then the trip is infeasible. It is important that the path evaluation process use the same mechanics of motion as the environment simulator. If they have different behavior, it is possible that a trip that is thought to be feasible ends up violating a constraint when executed.

The second issue is handling the large number of potential trips. In the case of dense, city-scale ridepooling, we have the ability to impose strict QoS constraints that effectively limit the number of feasible trips. Even when these constraints are somewhat loose, simple timeout heuristics can still produce candidate trips sets that yield sufficiently high performance while being manageable in size. However, in first/last mile transit settings a significant portion of the requests may be able to share the same vehicle with each other. In this case, the number of feasible trips can quickly grow to be too large to handle.

One way to handle this is by reducing the problem into an easier one. The overall

solution may not be optimal, though this is expected since the exact solution is too expensive to compute. The way we simplify the problem, when necessary, is to only generate cliques of size one — meaning pairing a vehicle and a request. All previous assignments to the vehicle are preserved implicitly. We then run the assignment ILP to match as many of the requests as possible.

5.5 Results

This section describes the experiences and results of the iterative design process with the two agencies.

5.5.1 Initial sites in King County.

Work began first on the King County location with an objective of choosing a site for the pilot to run. Initially, there had been multiple candidate locations in the county that were considered. Among the candidates, the area around Kent was chosen based on favorable estimates for demand. The purpose of the first round of experiments was to introduce the agency to the simulation tools and to solicit feedback on refining the service design.

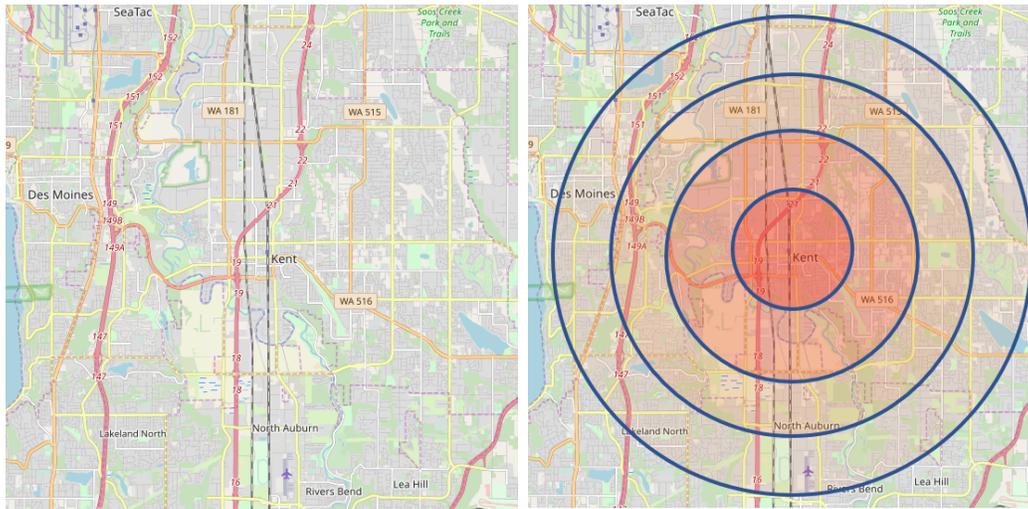
The initial concept was a first/last mile service coming in and out of Kent railway station. As a first attempt at a concrete siting for the pilot, little was known about how the agency wanted to operate the system. The many unknowns included the

size of the service area, requirements for standards of service, vehicle counts, whether booking in advance would be allowed, as well as information about user demographics and density. Rather than being an all day service, the first set of experiments considered only a first-mile service for the Sounder commuter train, which provides access to jobs in downtown Seattle.

To get the temporal distribution of demand, boarding data for the sounder trains was used to estimate the intensity of demand in 30-minute buckets, spanning from 5:30am until 8:30am. The bucket from 8:00 to 8:30 was discarded since it would not be possible to request a ride in that interval while still catching a train by 8:30. When we needed to generate a request, we first identified the bucket and then uniformly selected a time within the bucket.

The spatial distribution was complicated by the fact that we did not know how big a service area would be desired. We defined four service zones by creating four concentric rings around Kent railway station, each one, two, three, and four miles in radius respectively, as shown in Figure 5.2, then creating zones by nesting the rings. Within each one-mile thick ring we proposed a density of demand considering both distance from the station and the density of roadways in the ring. We then defined four groups of demand profiles for a service in a circular area with each radius.

Finally, since we did not know what kind of demand intensity to expect we generously created two cases: 270 requests per day and 450 requests per day. These numbers were chosen by estimating that between 20% and 33% of inbound rides on the Sounder rail could be converted to first mile rides. We explored serving



(a) Initial Kent service area. (b) Demand spread in concentric rings.

Figure 5.2: The initial service area was drawn from OpenStreetMap. The service centered on Kent railway station. A simple technique to create demand regions was chosen by forming concentric rings around the station and defining a demand intensity within each one.

the demand using between three and five vehicles of capacity eight and added the constraint that the waiting time plus the onboard delay for passengers that were served could be at most 30 minutes.

The results showed that service could be very promising in the one and two mile radius areas, but the larger four mile area had an unacceptably low percentage of requests being served, as shown in Table 5.2. Analysis at the request-by-request level revealed that in the large service areas, requests that were made further away from the station were significantly more likely to be rejected. This suggested that the large service area was also unrealistic since users in the peripheral areas would be less likely to be repeat customers.

Table 5.1: Service rates.

Vehicles	3	4	5
1 mile	99.71	99.71	99.71
2 miles	52.55	64.41	74.62
3 miles	35.21	44.6	53.44
4 miles	27.59	35.37	42.41

Table 5.2: Service rates (percent of requests served) for the nested service zones with radius of one, two, three, and four miles. While having additional vehicles is better, it is a struggle to achieve good service rates in the largest service area.

While somewhat naive, these experiments were valuable in familiarizing both groups with the service area and the simulation technology. As a result of the experiments, a dialog began with regards to the boundaries of the service area. It was agreed that the four mile service area was too large and the agency gave a general introduction to the Kent area, describing certain areas as being residential, commercial, and industrial. This is when it became clear that the agency’s objective was to serve passengers in low-income areas where the fixed-route service did not provide adequate coverage. Despite this, questions were still left unanswered about details of exact service area boundaries, many parameter settings, and questions of fleet size. The possibility of night-ahead booking was still not eliminated, though it was not part of the experiments presented.

5.5.2 Site revision.

In the second iteration of work, headway was made exploring ways to make a selection for the service area. Following information from the agency, it turned out that

population density was not evenly distributed over the large area defined by the four-mile circular proposal. Instead, a new boundary was considered directly east of Kent railway station. The goal of this modification was to focus on residential areas and to gain efficiency by reducing the size of the boundaries to get higher densities of requests.

The new proposal on the east Kent service area never made it through the complete experimentation process since the transit agency made a decision to revive a proposed service area in the area of Kent just north of the station. The new service area had two distinct zones - a residential area and an industrial area, with the residential zone being defined with a larger and smaller option available, as shown in Figure 5.3. In addition to the refined area, the new service area also came with new proposed hours of 5am until 7pm.

Despite the progress on the service area choice, the challenge with using experimentation to continue developing the service was the sensitivity of the results to the levels of demand, customer behavior, and the lack of decisions being made on parameters that were key to defining the service, everything from those defining trip constraints to higher level choices such as on-demand versus book-ahead style service. The research team used an algorithm that had specific constraints, such as maximum waiting and delay times, that had significant impact of the performance of the system since these constraints drove the decision between accepting and rejecting rides. On the other hand, the agency was used to defining constraints in terms of average performance and deferred these decisions to other members of the project.

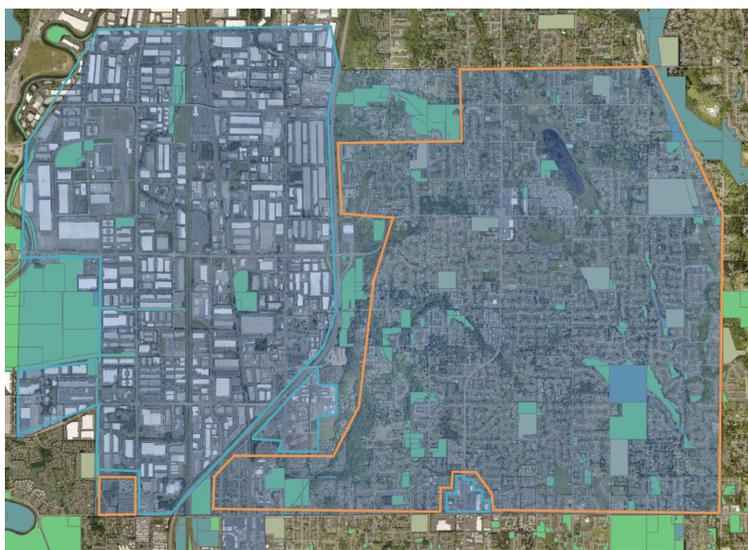


Figure 5.3: Service area north of Kent. The region can be partitioned into residential areas, highlighted in orange, and industrial areas, highlighted in blue. The railway station is near the bottom left. Credit Andrea Broaddus.

This created a difficulty since it became hard for the research team to be sure that agency understood the questions that needed to be addressed to continue meaningful development on the project.

A benefit of having a well-defined service area is that work could begin on developing higher-fidelity models for demand generation. Three issues considered were demand estimation, mode-share estimates, and generating demand profiles for experiments. Demand level estimates were produced using a combination of two models. For first mile trips, generation models were used that captured demand from residential customers using data from known sources including employment locations and transit ridership levels. For last mile trips, gravity models were used for employment and shopping areas. This number is compared with the mode-share estimates,

which are used to produce the final estimated ridership level. This process yielded an estimate of 350 riders per weekday, which represented a significant reduction in demand density compared to the first round of experiments which only covered the morning rush hour.

The meeting with the agency had a focused purpose of soliciting decisions by presenting the model in the most concrete way possible, hoping this would lead to a discussion from the agency that would give insights into their thinking. To this end, the demand generation methodology above was presented along with a description of the service in terms of specific parameters, vehicle counts, and a proposal that night-ahead booking be dropped from consideration. They agreed with the decision to drop night-ahead booking and thought that the demand generation model seemed reasonable. The times of service were changed to 6am-7pm. The service area was fixed, but they brought up that they might also consider an option where only a smaller portion of the residential area was included in the pilot.

5.5.3 Demand modeling for North Kent.

The next step for the research team was to develop demand profiles that matched the proposed demand model. Of concern was the reliability of the demand estimates. Not only was the actual demand hard to estimate due to its ultimate entanglement with the quality of the pilot service design and the unpredictable changes in customer behavior due to the COVID-19 pandemic, but the results from simulation showed

sensitivity to the demand levels. Therefore, the research team moved away from specific estimates of demand to instead showing the various performance metrics for a range of possible demand levels, allowing for consideration of design decisions under the full range of possible demands that could actually occur when the service launched. See Figure 5.8 for an example.

The first big challenge was implementing the new demand model. Using the model to generate demands consisted of two steps: 1) determining the balance between residential and industrial traffic and 2) estimating the temporal spread of the demand in each direction. This was done by analyzing boarding and alighting data for buses that traveled through the respective zones.

The temporal distribution of demand was formed by dividing the day into three parts: morning from 6am-9am, midday from 9am-4pm, and evening from 4pm-7pm. Results showed that demand intensity was higher in the morning and evening compared to midday, as shown in Figure 5.4. In general, more travel seemed to occur in the evening.

By comparing the number of riders on the buses used in the data, it was estimated that in general 44% of demand would come from residential areas and 56% of demand would be to the industrial area, as shown in Figure 5.5. Under the option with smaller bounds in the residential areas 16% of demand would come from the residential area and 84% would come from the industrial area, a value computed by comparing the ratio of sizes of the residential areas in each option and assuming an even spread of demand.

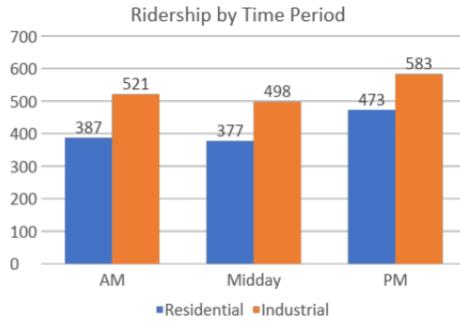


Figure 5.4: Number of transit riders by time of day, split into residential demand (blue) and industrial demand (orange). Note that the AM and PM peaks are each three hours long while the midday period is seven hours long. This means that demand intensity during midday is significantly lower than during peak hours.

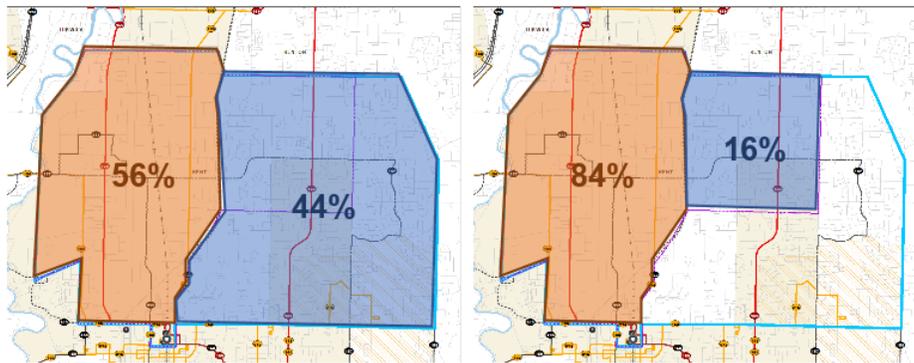


Figure 5.5: Break down of relative demand by region, comparing full residential option (left) and the smaller residential option (right).

Finally, the bus data was used to estimate the ratio of first-mile riders that would be traveling into Kent railway station to last-mile riders heading out. As shown in Figure 5.6, the residential area showed strong directional trends over the course of the day, suggesting a pattern of commuting to work in the morning and returning in the evening. The industrial area is more balanced, suggesting steady employment work load in the area.



Figure 5.6: Directional trend of demand by time of day. Residential demand shown in blue, industrial demand shown in orange.

In this round of experiments, the research team introduced the first breakdown of the results into separate scenarios so that direct comparisons could be drawn between different parameter choices. The baseline scenario used three vehicles of capacity eight and a maximum delay time of 30 minutes. The second scenario reduced the fleet size to two vehicles and the third scenario explored using two vehicle with an extra third vehicle during peak hours 6am-9am and 4pm-7pm. The fourth and fifth scenarios explored raising the fleet size to four and five, respectively. The sixth scenario increased the capacity of the vehicles to fourteen to see whether investment in larger vehicle should be considered. Finally, the seventh and either scenario replace the 30 minute maximum delay time with a total-journey limit of 2x Door-to-Door time and Door-to-Door time plus 15 minutes, respectively. The simulation tool itself had some modest improvements, now allowing us to model the time it takes for passengers to board and alight the vehicles, something we refer to as dwell time. Initially we made the simple assumption that the dwell time is 30 seconds in all cases.

In general, we found that adding more vehicles increased the percent of demand that could be served, but it did not significantly reduce the total delay times for

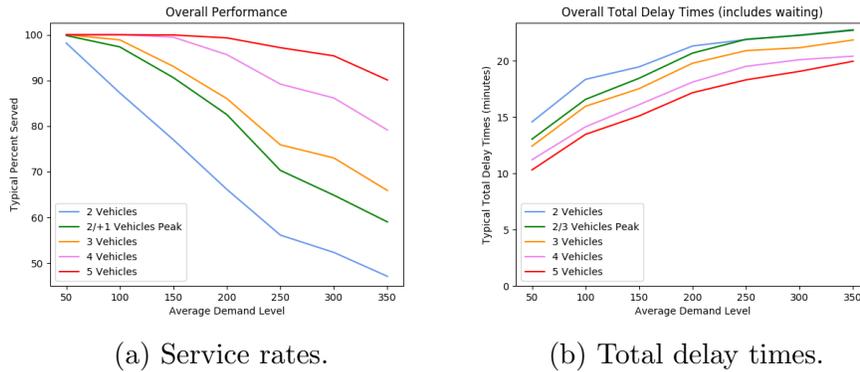


Figure 5.7: While increasing the number of vehicle makes a considerable improvement in service rate (left), it has little effect on the total delay time for passenger (right).

riders. This is illustrated in Figure 5.7. The reason the delay time did not decrease had to do with the way the extra capacity was being used. Since the primary objective is serving as many people as possible, the algorithm was using the extra capacity to serve requests that Although the delay time may have decreased for individual passengers, many passengers who were rejected since their trips would have added too much delay time were now able to be served. Since the primary objective of the assignment process is to serve as many people as possible, the ability to serve the extra demand dampened any benefit seen to average delay times. If there had been enough vehicles to serve all of the requests then it might be expected that service qualities, such as delay time, might decrease.

We found that performance was strongly affected by the choice of delay constraint. Allowing for 30 minutes of total delay performed best while 2x door-to-door time performed worst, as shown in 5.8. Roughly, this is due to the magnitude of time that each strategy allowed for. In addition, 2x door-to-door time also suffered from

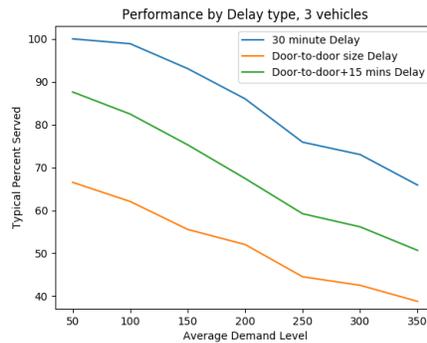


Figure 5.8: Comparison of delay constraints. 30 minute delay performed best, followed by door-to-door + 15 minutes delay and door-to-door delay. The ordering is not surprising. The constraints that effectively allow more delay time perform better.

rejecting passengers closer to Kent railway station at a higher rate. This is because the passengers in this area had a short direct time and therefore could only be picked up by vehicles that were very close. Figure 5.9 shows rejections in the service area under the best and worst delay limits. Careful inspection reveals a higher density of rejections near the bottom, where the railway station is located, in the 2x door-to-door experiment compared to the 30 minute max delay experiment. We recommended that the default of 30 minute max delay be kept to maximize the number of people served, but consideration may also be made to customer experience and preference.

In addition, we found that using three vehicles all day instead of two vehicles with an extra at peak hours made little improvement in the service rate. Unsurprisingly, the smaller residential zone led to more people served than the larger one since demand was spread out over a smaller area. Finally, we found that increasing the capacity of the vehicles beyond eight did not provide a measurable benefit. This is

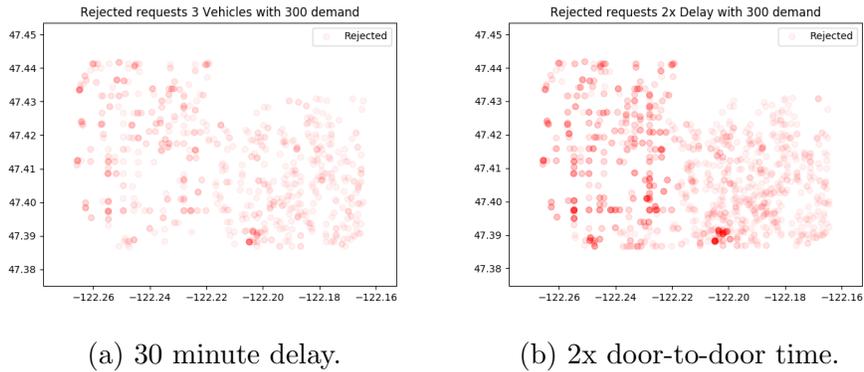


Figure 5.9: Rejection patterns for two delay constraints. While the distribution of rejections is fairly uniform in the 30 minute delay model (left), in the case of 2x door-to-door time (right) the density of rejections is higher in the lower portion of service area, corresponding with locality to the hub. In many ways this is expected since the delay constraints would require very fast service in those areas.

likely because the constraints on service quality have a larger impact on rejections than the capacity of the vehicle.

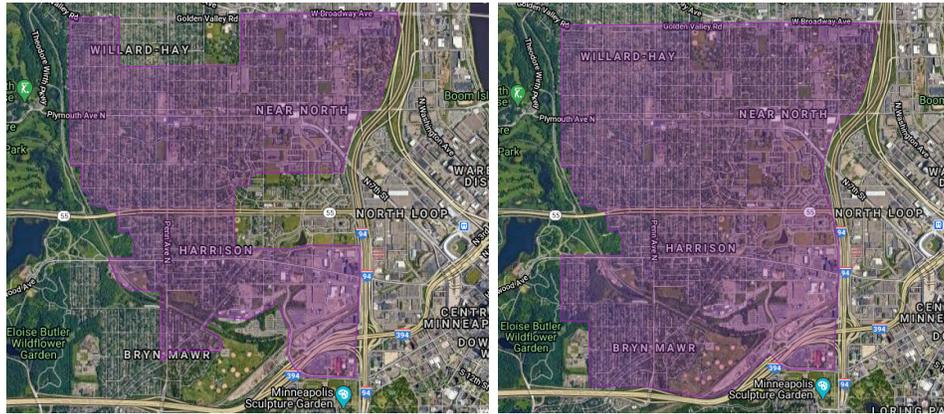
5.5.4 Initial experiments for Minneapolis

The initial experiments were conducted after the general site had already been chosen, which had been made prior to the research team starting significant work in the area. An advantage of starting work in Minneapolis after having some prior work already done was the maturity of the simulation tools and our better positioning to solicit useful information early on in the process. At the request of the agency, the simulator now included the new ability to accommodate riders that needed lift deployments in order to use the service. This meant that a lift request would be

treated as requiring a longer amount of time to board and alight the vehicle, distinct from the time that is normally required for other passengers. We were concerned that the extra dwell time needed for lift request passengers would cause them to be rejected at a higher rate than other passengers, an issue that could raise equity concerns. To address this, we made the assumption that the extra time required by a lift request passenger would not count against the maximum dwell time constraints for passengers on the vehicles since they would be understanding. The dwell times for other passengers remained, with values updated to be one minutes for boarding and 30 seconds for alighting.

The first question to be studied was the choice between a larger and smaller service boundary, as shown in Figure 5.10. Then, we wanted to know how important it was to focus on acquiring vehicles with larger capacity. The default capacity for vehicles during the COVID-19 pandemic was assumed to be three. However, we wanted to identify if spending more to get vehicles of capacity eight would be a beneficial tradeoff. Third, we wanted to know the sensitivity of the service rate to the number of vehicles. The agency had suggested budget for three vehicles, but we were interested in seeing the impact of adding an extra vehicle. The experiment was to focus on running from 5am until 2am.

Finally, we wanted to address a more difficult problem: the choice of delay constraints. The choice impacts the system, but the nuances between them and how they change the customer experience can be difficult to understand for those unfamiliar with the difference in definitions. We explored four options for the delay



(a) Smaller service option.

(b) Larger service option.

Figure 5.10: Two possible service area options proposed for the Minneapolis pilot after initial site selection. The smaller service option (left) is contained within the larger service option (right).

constraint.

1. 2x Door-to-Door time (2x Detour) for total in-vehicle time.
2. Fixed Detour, 10 minutes or less for excess riding time.
3. Hybrid option, any trip okay if it satisfies either of the previous two constraints.
4. Fixed Delay, passengers want the sum of detour and waiting time and excess riding time to be less than 10 minutes.

Being the first round of experiments, there were limitations in making demand profiles because we had only received preliminary data on the service area. We used data on the C-Line BRT to determine the distribution of demand across 30 minute increments, as it was presented in the data provided. We placed an even weight on first-mile and last-mile trips since the data did not distinguish directions. Spatially,

we spread the demand uniformly across the service regions. Each trip was either first or last mile to the bus station at Olson & Penn. In addition, not having data on lift deployments, we made the assumption that 2% of requests would need to use a lift.

We presented the results of our experiments to the agency using the service rate, waiting times, and detour times, though here we illustrate using only the service rate metric. In general, we found that the performance was significantly higher than what we had seen in King County. We attribute this to the smaller general size of the candidate service areas.

As for the first question about the design of the service area, we found that the service rate tended to not be sensitive to which area we used. As shown in Figure 5.11, either area performed better in some experiments at different demand levels. This suggests that the difference is mostly notable at higher demand levels where the smaller area yields better performance since the average travel distance required is slightly smaller.

For the second and third question we had asked, we found that the number of vehicles was significantly more important than the capacity of the vehicles. As shown in Figure 5.12, the service rate for high demand levels climbs sharply when increasing from three to five vehicles. However, increasing the capacity from three to eight makes negligible difference.

Finally, we found that the best detour constraint was the “Fixed Detour” option, which was similar to the one we had recommended in King County. In essence, this

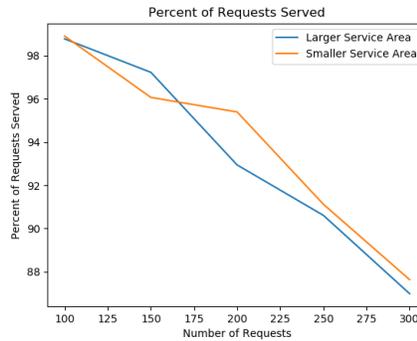
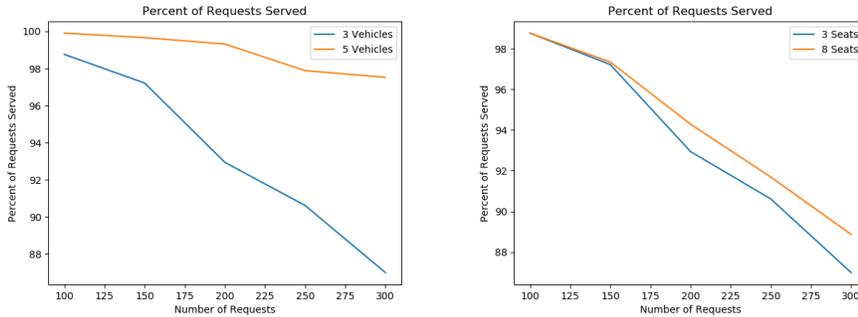


Figure 5.11: Comparing the larger (blue) and smaller (orange) service area option. The smaller service area may be performing better at high demand levels since the shorter average distances make matches between requests and vehicle more likely.



(a) Varying fleet size.

(b) Varying capacity.

Figure 5.12: Evaluations of capacity vs fleet size indicate that fleet size is more important. In the left figure, the 5-vehicle fleet (orange) does not suffer from degraded service as demand increases anywhere near as much as the 3-vehicle fleet (blue) does. In the case of vehicle capacity, shown in the right figure, a capacity-6 vehicle (orange) offers only a small benefit over a capacity-3 vehicle (blue) option.

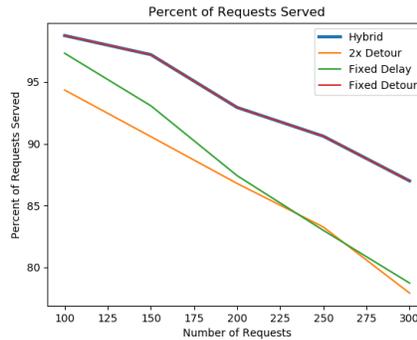


Figure 5.13: Service rate of the system under various delay constraints. The best was the Fixed Detour option (red) which overlapped the Hybrid option (blue). The Hybrid option turned out to be equivalent to the Fixed Detour option.

boils down to this version of the constraint being the most generous and therefore also the least likely to require rejecting passengers. As shown in Figure 5.13, the Hybrid constraint also performed as well as the Fixed Detour constraint. This is not surprising, since it allowed any trip permitted under either Fixed Detour or 2x Door-to-Door time (2x Detour) constraints, and the Fixed Detour constraint was always the most generous.

As for next steps, the agency agreed to provide ramp deployment data for fixed route buses to help make estimates for the actual percentage of riders that would require lift deployments. They also pointed the research team towards data that could be used to derive directional trends in demand. They also decided to move forward with the larger service area to cover more demand since the performance was expected to be similar for both possible service area sizes.

5.5.5 Second round experiments for Minneapolis

The second round of experiments introduced new functionality. First, it introduced the ability to consider a dynamic fleet where vehicles can enter and exit service at predefined times during the day. Second, it added realism to the simulation by differentiating different street corners for C line stations, which was relevant since they were located on divided roadways. And third, it allowed maximum waiting time constraints to be defined separately for first-mile trips vs last-mile trips.

The first question we wanted to answer was whether it would be beneficial to add a second bus station to the service area at Plymouth & Penn, as shown in Figure 5.14. Second, we wanted to know if relaxations of the constraints for passengers needing a lift allowed them to be served with the same general rate as all customers as a whole. Third, we also wanted to know whether the dynamic fleet allocation plan proposed by the agency performed differently from using a static number of vehicle all day, with the same total number of vehicle hours. Fourth, we wanted to know how the maximum waiting time for customers being picked up at the bus station impacted the service, varying it separately from waiting times at first mile origins. This feature is important since weather in the area is expected to cause lower waiting tolerances for last mile riders. Finally, we wanted to know what impact our assumptions on boarding and alighting times for customers had on the performance results we were getting. Namely, we wanted to try increasing them from the one minute boarding/30 second alighting used in the previous round since we considered it possible that actual boarding times may be less optimistic than assumed in our

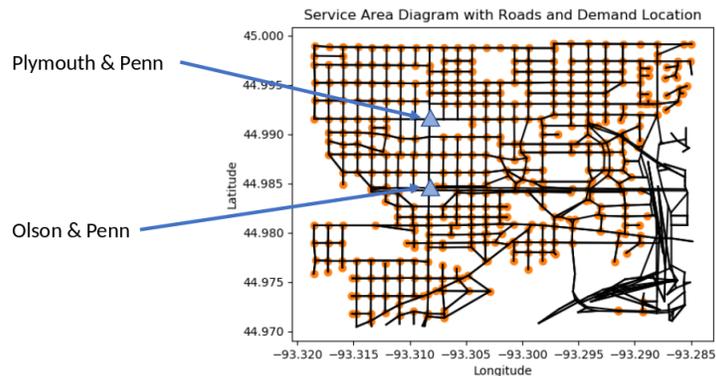


Figure 5.14: Illustration of the Minneapolis service area, with orange dots indicating intersections requests can be made from. Two blue triangles show the location of the two fixed route stations considered to be hubs.

previous rounds.

The baseline we used included the two bus stations, dynamic fleet sizing, and constraints for fixed detour and maximum waiting set to ten minutes. The next scenario we tried considered replacing the dynamic fleet schedule with a static schedule that used the same number of vehicle hours per day. The third scenario varied the maximum waiting time tolerance for last-mile customers. The fourth scenario explored removing the new station at Plymouth & Penn, only leaving the station at Olson & Penn, to see whether the extra station improved performance. Finally, the fifth scenario looked at the sensitivity of the simulations to our assumption of boarding and alighting times.

We found that dynamic fleet sizing outperformed the static 3-vehicle fleet in our experiments, even though the number of vehicle hours was the same. We attribute the slight improvement in service to the way the dynamic fleet sizing plan given by

agency coincided with patterns of demand.

Reducing the maximum waiting time for last-mile trips from ten to five minutes reduced the average waiting time by about 1.5 minutes at the cost of serving 3% fewer requests. While serving more requests is usually considered better, the demand for lower waiting times comes from passengers desiring less time spent waiting outside for a ride during the cold winter months. Since both longer waiting and higher rejection rates are considered to be undesirable, we left it to the agency to make a decision on the tradeoff.

While adding an extra bus station did not make significant improvement in the percentage of requests that were served, we found that waiting times and in-vehicle times were significantly reduced. We attribute this gain to the reduction in average distance required to serve requests since they can now choose stations that are closer to their first/last mile location. On the other hand, the lack of change in percentage served likely can from the two stations reducing sharing opportunities.

On a less positive note, we found that performance showed sensitivity to dwell times when it took more than two minutes for passengers to board and 60-seconds to alight. These times were explored since it is possible that first-mile customers may not already be at the street side when the vehicle arrives at their location. As long as these times remain low, decent performance may be expected to be maintained.

As before, the results overall appeared very optimistic since the service area overall is fairly small.

5.5.6 Final round experiments for Minneapolis

The final round of experiments for Minneapolis did not involve as many updates to the simulator itself. Rather, having established the technical needs from previous iterations this round mostly focused on using the simulation tool to compare additional design alternatives.

At the request of the agency, we considered the impact of adding a third station at Plymouth & Fremont/Emerson, as shown in Figure 5.15. This station was not on the C-line and required us to produce new demand profiles based on new bus data provided by the agency. From the data, we estimated there would be twelve vehicle lift deployments per day on the fixed route services and scaled that amount in each microtransit scenario according to the demand level. For illustrative purposes, we also tested the system performance under two types of “no denials” settings. First, we do this by increasing the number of vehicles until no one is rejected. Second, we do this by removing all waiting and delay constraints. This was an important test as it would provide the agency with an understanding of the impact of the constraints they were requesting.

Similar to the addition of the second station, we found that adding the third bus station did not significantly change the percentage of requests that were served but it did reduce the waiting and in-vehicle time of the passengers, as shown in Figure 5.16. An exception to this is that the average time that last-mile trips waited to be picked up at the bus station made a small increase.

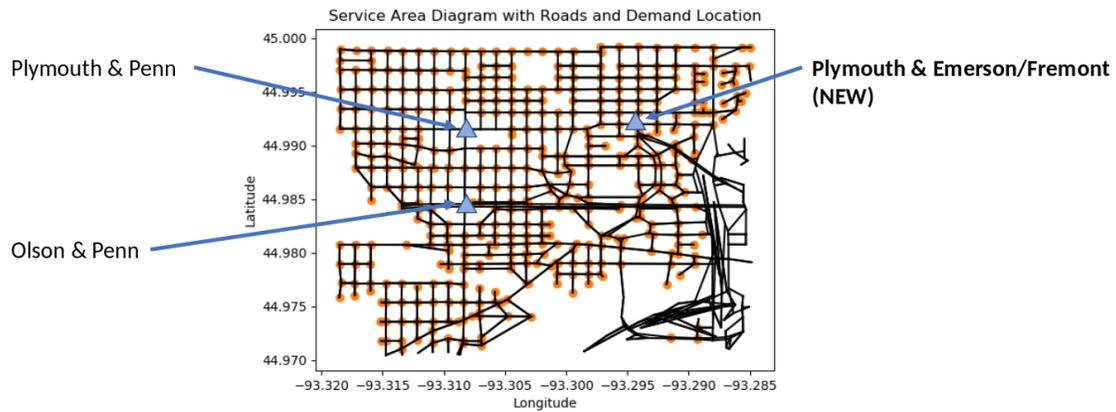


Figure 5.15: Map of Minneapolis service area, showing roads and intersections. Three blue triangles indicate the location of three fixed-route stations under consideration for the final round of experiments.

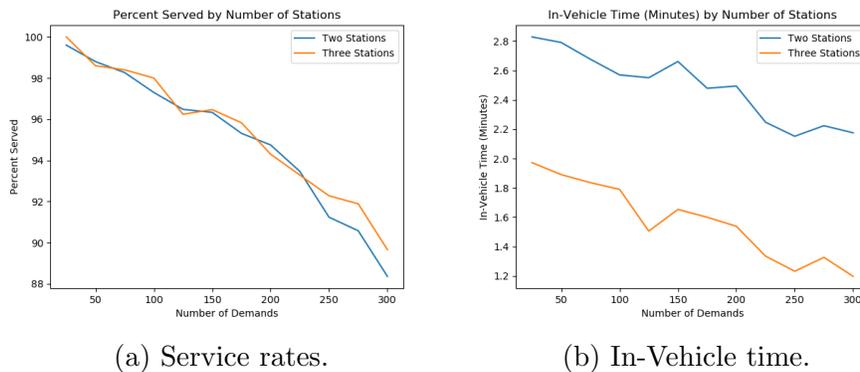


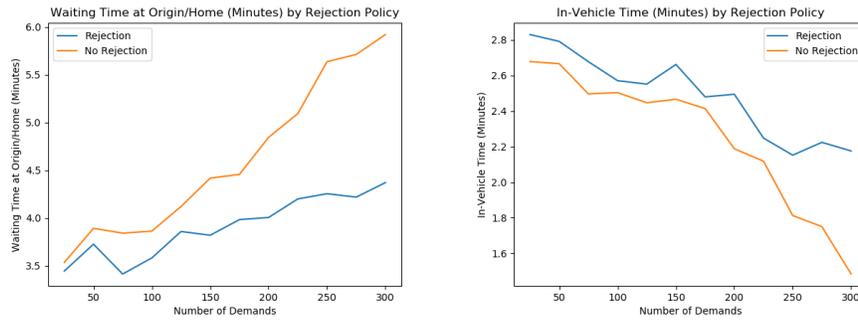
Figure 5.16: Evaluation of adding a third station. While the service rate (left) does not noticeably change, the in-vehicle time (right) is reduced for the three station option. This is likely since the average distance between hub and destination is lower when there are more hubs to choose from.

In the no-rejection scenarios, we found that increasing the fleet size to eight vehicles could serve 100% of the daily demand. Increasing beyond eight gave steady improvements in waiting time and in-vehicle time, but the marginal benefits decreased considerably.

In the second no-rejection scenario where all of the constraints are relaxed, we found that the typical waiting time for passengers increased though the in-vehicle time decreased. We attribute these seemingly opposite changes to an increase in efficiency in moving the passengers. By removing nearly all of the constraints and focusing instead on throughput, we are able to batch passengers together into efficient groups that move many people at the same time. This means people must wait longer for a ride, but once on board the assignments are effective at getting passengers to the station quickly. This agrees with the objective we use in the simulation solver to minimize vehicle miles traveled: when all passengers are served, this metric is minimized when each passenger on average spends the least possible amount of time on board. In practice, a no-rejection policy would likely not be attractive to an agency since the wait times for a small number of individuals can be very high.

5.5.7 Final round of experiments for King County.

Having paused to spend time on work for Minneapolis, we now returned to King County to do a final round of experiments, taking advantage of the developments we had made to our simulation tools and understanding of several transit agency goals



(a) First-mile waiting time.

(b) In-Vehicle time.

Figure 5.17: Effect of rejection constraints. The average first-mile waiting time grows substantially when there are no rejections (orange) compared to rejections (blue). The in-vehicle time decreases when there are no rejections, attributable to more efficient assignments.

to help answer lingering questions about the Kent pilot.

The final round of experiments included a new baseline. The service was to run from 6am until 7pm on weekdays and 7am till 7pm on weekends. There would be two vehicles, plus a third vehicle during peak hours on weekdays each with a capacity of six passengers. The dwell time assumed in simulation for picking up passengers would be two minutes, the drop off time 30 seconds, and groups of passengers could board/alight at the same location and only use the dwell time once. As before, lift deployments were five minutes and 2% of requests were assumed to require a lift.

Similar to the case in Minneapolis, the agency was interested in adding a second hub to provide more options for riders. In this case, they wanted to locate the new station in the middle of the industrial portion of the service zone. In general, they anticipated 80% of demand would want to travel to the railway station and 20%

would want to travel to the new station in the industrial area. In addition to the new station, they also wanted to know what system performance would look like under a possible “COVID” scenario where the vehicle capacity was reduced from six to three and the split of demand between the railway station and the new station became 50-50. They also wanted to know the impact of reducing the maximum waiting time from twenty minutes to ten minutes.

A new feature we introduced to the simulator was the ability to exactly match a request with a particular fixed route bus. For this experiment, we generated demand profiles where customers request service approximately 15-25 minutes before the bus they want to catch, uniformly spread, if they are first mile trips. If we cannot find a vehicle route that will get the passengers to the station by that time their bus departs we are required to reject them.

The difficulty with making demand profiles that specify a target bus for the requests is that customer behavior affects the service rate, as illustrated in Figure 5.18. Meaning, if customers in the data set request rides well in advance then a larger percentage of the customers will be served and brought to their bus on time. However, if they request at the last minute then we will see most of them being rejected. It is important to study this aspect of the problem separately since otherwise we cannot know how any other results we derive from these data sets may have been influenced by the particular choice of behavior.

As shown in Figure 5.19, as customers make their requests earlier and earlier the percentage of requests that we can serve steadily increases. At the low end,

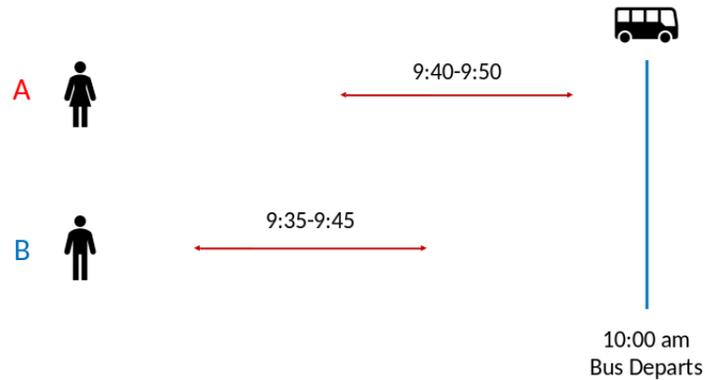


Figure 5.18: Customer behavior. Alice and Bob want to connect to a 10am bus departure. If Alice requests 10-20 minutes in advance of the bus departure she is less likely to be served than Bob, who requests 15-25 minutes in advance, since there are fewer options to serve tighter connections than looser ones. When generating demand profiles, we must consider whether customers are more like Alice or more like Bob.

requesting 5-15 minutes in advance typically leads to less than 50% of demand being served while in cases where passengers request 20-30 minutes in advance around 70-80% is served. As a representative choice for our experiments, we decided to use 15-25 in advance as the default option, trading off a slightly better expected customer experience with a slightly lower service rate.

Overall, we found that using two stations instead of one did not noticeably increase the percentage of requests served or reduce the waiting time for passengers, but it did decrease the amount of time passengers spent on board the vehicle. In addition, the average waiting time for last-mile trips was slightly longer. We interpret this to mean that last mile passengers waited longer since they were split into two locations, and a vehicle might be close to one of the locations but not the one the request comes from. Similarly, waiting time for first-mile trips did not significantly

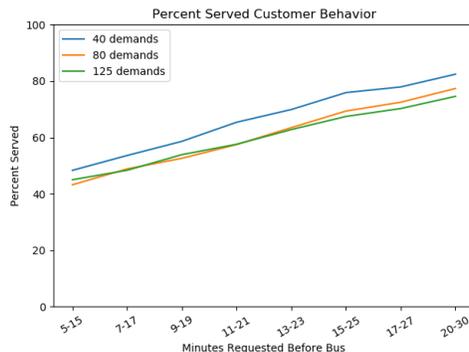


Figure 5.19: Service rate by how far in advance of a connection requests are made. Generally, booking earlier is better.

change since having two stations does not change how far these customers are from the typical vehicle location at the time they submit their request. Finally, on-board time decreased since the typical customer had a shorter average trip distance between their origin and destination, as the new station was in a more central location. This is similar to our findings in the Minneapolis experiments.

In a similar vein, we found that the COVID scenario had similar results to the two station scenario, though on board time was further decreased. We attribute these results to the fact that the COVID scenario was defined for two stations and more weight was placed on the central station in the industrial zone rather than the railway station, which is at the extreme southern end of the service area.

Reducing the maximum waiting time from twenty minutes to ten minutes made a 3-5% reduction in service rate, as well as nearly 16% decrease in waiting time for first-mile trips. Last-mile trips were less affected and in-vehicle time increased modestly.

5.6 Conclusions

The use of a simulation tool in planning proposed microtransit pilots aided several agency decisions. Besides being a focal point for envisioning the system as it would look at deployment, the metrics it produced also informed important decisions, sometimes in expected and sometimes less expected ways. Some results reinforced what the agencies already suspected, such as the early finding that extremely large service areas are hard to operate with short waiting and delay times. Other results were pleasant surprises, such as the value added from increasing the number of hubs the FMLM service connected to. It would have been difficult to predict that on-board times would be impacted more than service rate without the aid of a simulation tool. Some results led to recommendations that agencies ultimately ignored. For example, results showed that there was little marginal benefit to using higher capacity vehicles. However, this did not stop one of the agencies from pursuing that route. They believed that since buses bring in groups of requests at the same time, higher capacity vehicles could take advantage of batching in a way our results were not showing. Since the simulation demand profiles we used assumed a somewhat uniform spread of demand across time, they reasoned it could be failing to capture that effect.

It is important to highlight the key ways simulation tools add value to the planning process. First, these tools give agencies the ability to compare service design elements such as service area, hub placement, and vehicle capacity in nuanced ways by comparing output statistics from simulation experiments that test a wide variety of scenarios. Second, the output statistics from the simulation allow agencies to un-

derstand the customer experience in a way that is not possible with a simple high level analysis. Metrics such as waiting time for first mile rides, total in-vehicle time, and the percent of rides that are shared are all sensitive to the scenario and the various parameters at a level of detail that cannot be made without a simulation tool. Finally, the simulation tool also served an algorithm “proof-of-concept”. This means that when a simulation experiment is run that implements standard constraints such as maximum waiting time limits as well as custom constraints such as first-mile request bus matching, it serves as its own assertion that an algorithm can be run in real-time to make assignments that meet these constraints.

Nonetheless, simulation tools do have limitations. The simulation tool used in this work had limitations unique to its academic nature. Since the primary focus of the research simulator was the study of the assignment algorithm process, the tool was very developed in terms of performance of trip route selection and generation. However, this came at the cost of other features that may be of interest in real-world settings. One example of this is the deterministic travel times used in the road network. Any real-world implementation of the service would need to account for and model uncertainty in travel times and should also give a nuanced consideration to the way that route elements such as left turns and traffic lights can impact travel times. Another example is the way the simulation tool only considers vehicle boarding and alighting at road intersections. This restriction again comes from the simplicity of the travel time calculations — the internal representation of the road network only stores the travel times between intersections. None of these restrictions are theoretical in nature, rather they are artifacts of representing lower priority goals in a university

simulation tool where exact travel time models don't contribute to algorithm research questions being asked.

In addition, using a simulation tool does not help with predicting features that might be regarded as input. For example, the simulation tool does not give a novel way to predict demand. In fact, the demand profiles used in this study were fairly simple in nature. Rather, since the tool takes demand profiles as an input it is clear that the problem of improved demand modeling still remains open. It also cannot teach us about customer behavior. A reasonable research question might be how a service would perform over a long period of time considering the way that customers' use of the system depends on their own past experiences. While a future simulation tool could include elements of behavioral feedback, such a system would still require the user to choose the behavior model for the customers and the sensitivities of the results to those assumptions would need to be carefully studied.

Going forward, we highly recommend the use of a flexible simulation tool to aid the design of microtransit systems. Not only do these tools give transit agencies transparency in how performance metric forecasts and design recommendations are made, it allows close collaboration between technology experts and transit agency field experts. At this time much work needs to be done to have such high quality tools readily available for agencies and potential partners to study future services. However, with a growing number of dedicated organizations appearing that specialize in microtransit services it is possible that such tools could be developed in the near future.

CHAPTER 6

SEIR-CAMPUS: MODELING INFECTIOUS DISEASES ON UNIVERSITY CAMPUSES

6.1 Introduction

SEIR-Campus is a Python package designed to aid the modeling and study of infectious diseases spreading in communities with a focus on fast computations for large university campuses. It uses an agent based framework with interactions derived from individual movement patterns and interactions. For example, in the university setting using course registration data and models of student social dynamics to simulate day-by-day spread of infections in discrete time. Its features include:

- An epidemiological model based on the SEIR model
- Distinction between symptomatic and asymptomatic cases
- Modeling interactions (e.g., course contacts and social interactions) that change day-by-day
- Specifying stochastic model parameters (e.g., the infectious period, recovery time etc. can be a stochastic process)
- Highly customized testing protocols and quarantine procedures for individuals
- Integration of contact tracing
- Ability to add custom social networks

- Fast computation of large problem instances (e.g., 20,000 individuals for a semester in less than five seconds)

The package is designed to be easy to use out of the box, at a minimum only requiring formatting an input file identifying a list of individuals and a list of gatherings they attend. The package is also designed to be highly customizable for a variety of circumstances of interest to users. For example, the in-built infection testing model can be extended to allow for any custom function to decide who to test and what failure rates may occur. Also, a simple default example of contact tracing is included, but any custom contact tracing algorithm can be inserted. How customizations work, and more, is explained in the examples Section 6.4 to allow for simple extensions. We encourage users with implementation experience to explore the source code themselves.

Our work is motivated by the emerging literature on studying disease spread in university settings. One such example, [24], creates a model that estimates the dynamics and interactions at a university by randomly generating interactions among agents for each day, including randomization for constructing student course schedules. Another related study [6], considers interactions in terms of agents meeting at physical locations. This is done via a graph modeling spaces on campus with agents' itineraries randomly generated as movements in the graph on an hour-by-hour basis. As an alternative to largely randomized models, [78] uses real university transcript data to study the interaction graph of students based on historical course enrollment. Building from these ideas, our simulator uses an agent based model that allows for

dynamic interaction graph construction by integrating interaction data (e.g., from transcript data) with information/models of other social gatherings, both in a deterministic and randomized manner. While the accuracy of the specific predictions about actual disease propagation are dependent upon the data and parameters selected by the user and other factors, the main intended purpose of this tool is to help understand the relative changes in the dynamics of the system under different assumptions.

Related to our package, Epidemics on Networks (EoN) [49] is a Python package that simulates infections in Susceptible-Infected-Susceptible (SIS) and Susceptible-Infected-Recovered (SIR) disease models both over networkx graphs and differential equation models, also including a variety of visualizations. In contrast to EoN, our package is specifically targeted at the context of communities (e.g. university campuses) with discrete day-by-day dynamics and includes features designed to be convenient when information such as transcript data, demographic information for students, and campus group identifications are available.

One limitation of our model compared to EoN is that we only consider single, simple contagion to which agents recover with immunity. EoN has the ability to process other simple contagion, such as infections that do not produce lasting immunity, competing diseases which cause partial cross immunity, cooperative diseases where one helps the other spread, and complex contagion where infection rates may be non-linear in the number of infectious individuals agents are exposed to. EoN also models diseases where susceptible individuals become vaccinated at a given rate,

something that could be added to our model with small modifications. While EoN models run very fast, it is hard to compare the running time efficiency of EoN with SEIR-Campus directly due to differences in our model, as we allow daily changing interaction graphs, which means infectious periods for agents must be computed across multiple iterations to account for temporal changes in contacts.

Our package gains its computational speed by compressing all interactions within each day as an unordered collection of events. Since the infections we are studying do not cause individuals to become contagious on the same day they are exposed, we need only to compute the groups of individuals who would interact each day and the amount of time they are exposed to each other, possibly different every day of the semester. Events, such as individuals being quarantined or recovering from infection and no longer being susceptible or contagious, can be handled efficiently during simulation runtime. While some stochastic elements such as infection duration, whether exposures occur, and whether exposed individuals develop symptoms are efficiently handled during runtime, other randomized elements such as changes to the daily interaction networks are preprocessed and have computation time highly dependent on the customized procedures used to create the changes.

The applications of this tool are not limited to simulations of an entire university campus; it can be used in any context in which a community of individuals whose intra-community interactions need to be modeled explicitly. For example, professional sports leagues are keeping athletes in special “bubbles” that are intended to isolate them from the outside world so that there are no infections internally, a

context well suited for this tool. Similarly, as many universities are moving to online-only instruction, some are considering offering special programs to keep athletes on campus. In practice it is prudent to consider that external infections may reach such groups and simulations tools that help plan policies that are robust against the spread of infection may prove invaluable in getting the best results.

This document introduces the modeling framework we use and illustrates the software's usage through sets of examples. The modular implementation of the simulator allows for custom choices of quarantine policy, contact tracing policies, and testing procedures, all of which are explained and included in the examples file in the package. We will use the example of student interactions in a university campus throughout the rest of the discussion, but note again that the applicability of the package is not limited to this use case.

6.2 Epidemiological Model

This packages assumes a standard agent based Susceptible-Exposed-Infectious-Removed (SEIR) model that considers each student as an agent and models student interactions through meetings (e.g., course contacts and social interactions). Within each meeting, all individuals have equal exposure to all other individuals for the full duration of the meeting (though we discuss generalizations in 6.2.1). SEIR models give each agent one of four states: susceptible (has not been infected with disease), exposed (has the disease but is not yet infectious), infectious (can spread disease to

others, may or may not have symptoms), and removed (either recovered or otherwise unable to spread/receive disease). An initial state is given to each agent and the purpose of the model is simulate how infections cause agents' states to change over the duration of simulation period. While traditional agent-based SEIR models place agents as nodes in a static social network graph, either as a time-free model or with Markovian transmissions over time, this software implements the ability to have a transmission graph that changes each day, and whose weights are determined by the durations and intensities of the respective interactions, the product of which we refer to as the effective exposure time.

The key building block of our tool is meeting events, which act as an efficient shortcut for interaction graph generation by defining gatherings that recur on a regular or semiregular basis. Meetings can be created in many ways, e.g., by social interactions or by class interactions. Meetings have three fundamental properties: a set of individuals, meeting dates, and meeting duration. Classes, a key focus of our tool, can be defined in this way automatically through transcript data. Transcripts can tell us which classes students are enrolled in, which days the classes meet, and how long they last each day. Models, perhaps aided by available data, may also be able to create meetings that capture expected social interactions by students outside of class. Of course, students will typically have many other interactions both with each other and the community at large that are hard to capture by direct modeling. We model these interactions via an exogenous infections rate parameter that can be calibrated appropriately. While admittedly this is a somewhat crude approximation, our focus is on understanding the implications of the intra-campus

interactions which we have the data to model (e.g., course network and specific social gatherings), and the impact of intervention policies that alter these interactions (e.g., course size restrictions, hybrid courses, limits on social gatherings).

The SEIR model propagates infection between infectious individuals and susceptible individuals when they share a meeting, with effective minutes of exposure time as a weight for whether the disease successfully spreads each day. This means that two individuals who share a longer class will have a higher probability of spreading an infection to each other than two individuals who share a shorter class. While the realities of infection dynamics depend on many factors, infection rate is approximated in our model through the flexible choice of exposure duration. For example, if students mingle before or after class, the effective duration of the class might be considered to be longer than its schedule time. If a class has long duration but involves significant social distancing, such as some lab classes, their effective duration can be reduced to represent the duration of close contact expected during the class. In other words, the effective duration (a scaled version of the actual duration) can be used as a tool to calibrate the infectious risk of individual meetings.

We use a SEIR model with the following standard extensions. First, we differentiate symptomatic infectious individuals from asymptomatic infectious individuals by creating two separate infectious states. This distinction allows us to capture differences in infection spread rate and agent actions, such as not attending meetings after noticing the development of symptoms. It also allows us to model compliance by treating a fraction of those who are symptomatic as asymptomatic agents, cap-

turing behavior where agents choose to have close contacts despite their symptoms. Second, we consider the possibility that individuals will be placed in quarantine. The two principle distinctions with quarantine is whether the individual is actually infected, as uninfected individuals leave quarantine as susceptible individuals while infected individuals only leave upon recovery. For the purposes of statistics, and the possibility of features that take advantage of the extra information, we create a quarantine state to mirror each of the states other than the removed state.

6.2.1 Generalizations and possible future extensions

In the SEIR model we adopt, individuals fully interact with each other for the full duration of the meeting. However, there are generalizations of this that might be considered for more realistic scenarios that could easily extend our model and implementation without much work.

One such extension would consider that not all individuals in meeting spend the same amount of time in class. For example, if a small number of individuals leave early this reduction in exposure could be captured by means of a rejection process: if the person is identified as exposed in the simulation, a coin could be flipped that with some probability rejects the infection. If multiple people have differing arrival and departure schedules this method would involve more effort to implement as it would not be compatible with the assumption that meetings cannot be subdivided, which was used to optimize transmission computation performance in the SEIR-Campus

package. In this case, within each meeting each infectious individual's transmissions are computed separately and each susceptible individual's rejection probability can be selected based on the amount of overlap between each of their schedules.

Another possible extension would consider that individuals in different part of the room have unequal chances of transmitting based on features such as physical distance. Similar to the previous extension, this would not be compatible with the currently implemented computation optimizations since it requires infectious individuals to be considered separately from each other. Similar to the previous case, the solution to implementing this would be to run separate computations for each infectious individual and calculate transmissions to others by coin flips weighted by a function of their physical separation based on the seating arrangement.

6.3 Model Details

Here we give a precise, detailed overview of the simulation model. We begin by formally defining the states individuals can be in, describe the transition mechanics that govern when individuals change state, and then describe the algorithm for efficiently implementing the defined model.

6.3.1 States

Each agent in the system is in one of nine states. The states are extensions of the four typical SEIR states, with two separate infectious states and a quarantine state to mirror each state other than removed.

1. Susceptible (S)
2. Exposed (E)
3. Infectious, asymptomatic (I_a)
4. Infectious, symptomatic (I_s)
5. Quarantined while Susceptible (Q)
6. Quarantined while Exposed (Q_e)
7. Quarantined while infectious, asymptomatic (Q_a)
8. Quarantined while infectious, symptomatic (Q_s)
9. Removed/Recovered (R)

6.3.2 Actions

Agents in the model change state through the follow set of transition rules. Some of the rules may not apply to all simulations, such as those that do not use testing, contact tracing, or quarantine, while other happen automatically if not interrupted,

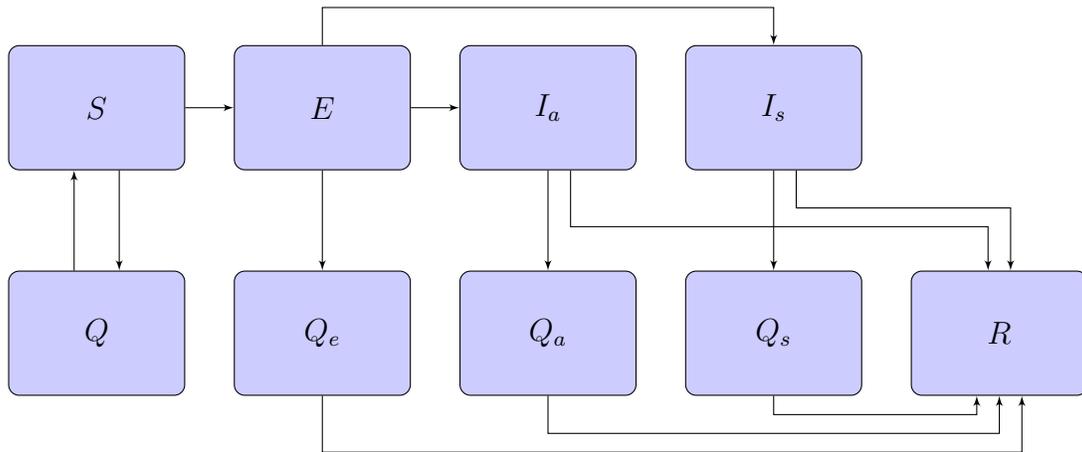


Figure 6.1: State flow chart. Susceptible individuals (S) may be quarantined (Q), but then return to being susceptible afterwards. When infected, susceptible individuals become exposed (E). An exposed individual eventually becomes infectious, either asymptomatic (I_a) or symptomatic (I_s). Infectious individuals, if not subject to any interventions, recover and enter the removed state (R). Exposed (E) or infectious (I_a , I_s) individuals may be quarantined in (Q_e), (Q_a), or (Q_s) respectively. Since these individuals are infected before they enter quarantine, they only leave once they have recovered and are moved to the removed state (R).

such as exposed agents becoming infectious. Figure 6.1 gives a visual representation of agent states and possible transitions.

1. **Transmission.** Each day, infectious individuals from I_a and I_s can transmit infection to individuals in S at a rate determined by effective exposure minutes between pairs of individuals. This causes the state transition $S \rightarrow E$.
2. **Spontaneous exposure.** Each day, individuals from S become exposed spontaneously (due to infections exogenous to the interaction model) with a certain probability. This causes them to transition $S \rightarrow E$.
3. **Incubation.** Individuals in E transition to either I_a or I_s with a given prob-

ability and transition at a rate dependent on which state they transition to.

4. **Testing.** Tests happen both at regular intervals and at scheduled times due to an individual being selected by contact tracers. Testing latency causes results to be returned after a delay from when the test is performed, by default the next day. Tests have certain probabilities of type 1 and type 2 errors. A test is correctly positive when an individual is any of states I_a , I_s , Q_a , or Q_s . When a test yields a positive result, contact tracing is invoked on the the individual who tested positive. A positive test result causes state transitions $S \rightarrow Q$, $E \rightarrow R$, $I_a \rightarrow R$, and $I_s \rightarrow R$.
5. **Contact tracing.** Contact tracing on an individual identifies a group of peers that have been near the individual for an extended period of time, though the precise method used to do this is not part of the model specification. Each individual identified in contact tracing is moved into quarantine for a specified number of days. It causes the state transitions $S \rightarrow Q$, $E \rightarrow Q_e$, I_a to Q_a , and $I_s \rightarrow Q_s$. Each individual quarantined may be tested
6. **Release from Quarantine.** It is assumed that exposed individuals in quarantine are detected at some point during the infection and never leave until they are recovered. Thus, they leave quarantine at the later of their recovery time and the expiration of their Quarantine. This results in state transitions $Q_e \rightarrow R$, $Q_a \rightarrow R$, and $Q_s \rightarrow R$. Additionally, unexposed individuals return to the susceptible state with the transition $Q \rightarrow S$.
7. **Recovery.** On an individual's recovery date, they are no longer infectious and move to the removed state, through the transition $I_a \rightarrow R$ and $I_s \rightarrow R$.

6.3.3 Internal Structure

The model is executed over a set of days D , which includes weekends and holidays from the beginning of the semester. The model contains a bipartite graph between individuals and meetings M . Since there are courses at the university that have irregular meeting days and times, there is a function that gives the duration the course meets each day $Dur : (D, M) \rightarrow \mathbb{R}_+$. There are sets corresponding to each of the 9 states each individual can be in. In addition, each member of E , I_a , I_s , Q , Q_a , and Q_s are associated with a date at which they will automatically transition to a subsequent state if there are no interventions. These transition times are stored in associative array data structures.

The only two types of actions that are not scheduled are contact tracing and testing. While contact tracing is typically scheduled on the same day that a positive test result is returned and therefore is not scheduled in advance, testing can happen under a variety of conditions that may require prior scheduling. For example, weekly infection testing may occur or someone who is quarantined for any reason may be scheduled to receive a test result the following day. In general, an individual may be tested multiple times if they are tested while susceptible or exposed but not yet contagious. However, only in the case of a susceptible individual does leaving quarantine mean returning to the general population. Therefore, in effect only susceptible individuals can ever test positive more than once and only susceptible individuals can invoke contact tracing multiple times.

6.3.4 Order of Operations

The simulation processes time in units of days¹. For each simulated day, events are processed in the following order:

1. Scheduled transitions due that day (such as an individual recovering) are processed and sets are updated.
2. Testing results come in. A list is made for contact tracing and quarantines are updated.
3. Contact tracing is performed and quarantines are updated.
4. Daily infection spread is processed.

6.3.5 Processing Daily Infections

In traditional SEIR models on static interaction graphs, all of the outgoing edges from an infectious individual to neighboring susceptible individuals are considered and a weighted coin is flipped to determine whether the infection spreads to each susceptible individual. The coin flips can even be preprocessed on an edge-by-edge basis. However, this approach is not possible for models where the interaction graphs change dynamically. Rather, we use a procedure that aims to efficiently compute transmission by considering each day sequentially.

¹Using units of days is sufficient and without loss when infections cause agents to change state on the order of multiple days.

In simulating the model each day, infection computations are done on a meeting-by-meeting basis. Each of the meetings is processed sequentially to determine that day's transmissions. First, the set of infectious individuals are identified in each meeting. Then, since infectious individuals transmit infections to susceptible individuals at a fixed rate per effective minute of exposure time, and since we assume that meetings are fully connected interactions, we next determine the total number of exposure minutes experienced by susceptible individuals in the meeting that day. Exposure per susceptible individual is equal to the total number of infectious individuals multiplied by the transmission rate and effective duration of the meeting. To determine which susceptible individuals become infected, we use one of two procedures depending on which involves fewer calculations.

If the effective exposure per susceptible individual is less than one, then the expected number of transmissions during that class is less than the number of susceptible individuals. In this case we use a Poisson process to determine the total number of infections that occur and then uniformly, with replacement, draw that number of individuals from the susceptible population. On the other hand, if the number of expected transmissions is greater than the number of susceptible individuals we instead use a binomial random variable for each susceptible individual using the probability per individual of transmission.

6.3.6 Time Considerations

The running time of each repetition of the model is divided into two portions: preprocessing, which creates the set of nominal interactions, and runtime, which processes infection spread and agent states. While the runtime of the preprocessing step cannot be stated since it allows an arbitrary number of customizations, the largest driving factors of runtime for the simulation are the number of individuals who eventually become infected and the number of distinct meetings in the simulation. The influence of these factors comes directly from the model: each day, the spread of infection is event driven and carries out a computation based on the set of meetings that have at least one infected individual.

It is worth mentioning some alternative approaches to processing infection spread that we rejected either due to runtime considerations or compatibility problems with our dynamic model. Since classical SEIR models compute the spread of infection by flipping coins on weighted edges between infectious and susceptible individuals, a natural first idea is to compute a large static graph of contacts for each infectious individual that covers all susceptible individuals they will contact during their infectious period, complete with the appropriate weights based on the the sum of the contact times from each days' meetings. This approach fails, however, to allow generalizations with testing and contact tracing procedures which may ultimately limit the number of days the infectious or susceptible individuals have contact with others. We decided instead to compute the spread of infection on a day-by-day basis. Computationally, this is just a reordering of events since in either case we

must determine the total contact minutes between pairs of individuals on each day's interaction graph. Further, this can sometimes lead to simplifications since some individuals may become infected or quarantined on later days, reducing the number of computations needed.

Another key note is the choice to have two procedures to compute transmission based on the expected number of individuals to infect. Initially we anticipated that a very small number of people would become infected in each meeting each day, suggesting that the most efficient manner to compute infections would be to compute the number of susceptible individuals that would become infected using a Poisson process and then drawing, with replacement, that number of people from the susceptible population and label them as infected. In scenarios where few people become infected this procedure can be implemented very fast in Python since it can be performed by external libraries, such as Numpy's `random.choice` function, which offer significant speed advantages over explicitly written `for`-loops. However, in scenarios where many individuals are infectious it becomes slow. In these cases, the expected number of infections in a meeting may exceed the number of susceptible individuals, which can lead the `choice` function to redundant work since we draw with replacement. Therefore, in these situations, we instead use an alternate procedure of flipping coins for each susceptible individual using an explicit `for`-loop, which ensures no redundant work is done.

6.4 Usages Examples

The best way illustrate the flexibility of the model is through a series of demonstrations. This section explains the basics for loading data into the model, and then discusses ways the model can be expanded and customized. Since the SEIR-Campus model is implemented in Python, each of the examples below is also written in Python and is included in the `Examples` Jupyter notebook from the git repository. To get the most value, the reader is encouraged to play along with these examples as they read.

6.4.1 Basic Usage

To begin, we will create and load a simple example to demonstrate how the simulation works. We will describe the input file format, how the data is loaded, and how repetitions of the simulation are performed. This model will assume that individuals only interact through course contacts and that no testing or contact tracing programs are in place.

Input file format.

A course description file has three types of entries: student entries (demographic information), meeting entries (used for classes or precomputed social gatherings), and group entries (used for constructing custom social groups). Each line of the file

contains a single entry and is in the form of a Python dictionary, whose key is the entry type and whose value contains a dictionary of the appropriate information for each entry type. For example, a stem student enrolled as an undergraduate might have an entry:

```
1 {'student': {'id': 12345, 'demographics': {'gender': 'm', 'is_grad':  
      'n', 'field': 'STEM'}}}
```

The key `student` indicates that this entry is for a student, rather than a meeting or group. The value is a dictionary which defines the student `id` and a demographics dictionary. The demographics dictionary has no fixed format; it is a convenience for use in building custom groups and meetings by user code.

Meetings, used to build classes or other fixed social gatherings, are defined with the key `meeting`. As an example, a class might be created using

```
1 {'meeting': {'id': '5166', 'info': {'name': 'CLASS 5166'}, 'meets':  
      [('9/2/2020', '11/13/2020', 'MWF', 60)], 'members': [1452, 1633,  
      1960, 2305, 2417, 2456, 2535, 2635, 2682, 2738, 2767, 2832, 2873,  
      2945, 2975, 3235, 3250, 3332, 3334, 3644, 3812, 3835, 4036,  
      4053, 4059, 4154, 4220, 4288, 4423, 4582, 4591, 4848, 4925, 4932,  
      4976, 5033, 5143, 5230, 5251, 5258, 5261, 5274, 5365, 5379,  
      5422, 5455, 5463, 5474, 5514, 5516, 5523, 5600, 5634, 5644, 5663,  
      5777, 5802, 5887, 5915, 5944, 5957, 6026, 6061, 6071, 6095,  
      6145, 6401, 6446, 6624, 6706, 6734, 6786, 6873, 6898, 6978, 7092,  
      7201, 7408, 7448, 7462, 7595, 7601, 7666, 7751, 7771, 7772,  
      7809, 7818, 7832, 7890, 7987, 7998, 8049, 8102, 8114, 8125, 8300,  
      8316, 8339, 8371, 8376, 8386, 8399, 8446, 8449, 8473, 8496,
```

```
8520, 8545, 8613, 8616, 8620, 8630, 8631, 8645, 8725, 8726, 8739,
8804, 8826, 8854, 8875, 8943, 8956, 8966, 8971, 8972, 9000,
9034, 9036, 9062, 9141, 9212, 9221, 9288, 9310, 9311, 9342, 9492,
9500, 9535, 9727, 9860, 9880, 9975, 10108, 10139, 10147, 10208,
10209, 10304, 10311, 10321, 10362, 10413, 10437, 10545, 10618,
10667, 10716, 10791, 10806, 10853, 10895, 11054, 11236, 11260,
11402, 11435, 11553, 11594, 11604, 11653, 11663, 12346, 12511,
12933, 13193, 13287, 13449, 13572, 13713, 13970, 14041, 14243,
14287, 15000, 15110, 15419, 15522, 15657]}}
```

The meeting is defined by an `id`, a list of `members` comprised of student ids, and an `info` dictionary. The `info` dictionary contains a list of `names` the class is referred to (typically one but classes are sometimes cross listed) and `meets` information which lists tuples of the form (`start date`, `end date`, `weekday pattern`, `duration`). Dates are written using Month/Day/Year convention and weekday patterns can contain any combination of the letters MTWRFSU. Many classes need only one such tuple, but some classes with custom schedules may require multiple.

Finally, there are group definitions. These are not required, but provide an easy way to load information about groups into the system for later manipulation into social meetings. Groups are not the same as classes or social meetings, per se. The intention is that they are for creating meetings dynamically, such as the meeting patterns of members of a club which may be random or meetings that the user wants to create in custom ways. This is in contrast to typical meeting definitions which are static and computed at the time of the creation of the data set.

A group only defines a name and a list of members. An example entry might be,

```
1 {'group': {'name': "Women's Varsity Ice Hockey", 'members': [311,
    351, 811, 1151, 1251, 1331, 1391, 1411, 1471, 1571, 1631, 1851,
    1871, 1931, 1951, 1991, 2031, 2051, 2071, 2111, 2131, 2151, 2171,
    2191, 2211]}}
```

Production of sample data

The data included in the repository for this package is produced from a combination of publicly available data and a randomized model for courses and students. The underlying course network is from the network used by Weeden and Cornwell [78] in the supplemental materials [79]. From this data, student demographic information is assigned by labeling all even student id numbers as female and all odd student id numbers as male, something that will be used to create varsity teams later on. Graduate/undergraduate status and field of study are directly from the Weeden and Cornwell data.

Since the data has been anonymized and therefore does not contain any information about the identifies of the courses, we must make artificial choices about meetings times and dates. Despite the wide variety in class meeting times in reality, we assume all classes in the data set meet for 60 minutes. We determine the weekly meeting pattern randomly, as in [24], by assuming 40% of courses meet Monday/Wednesday/Friday, 20% meet Monday/Wednesday, and 40% meeting Tuesday/Thursday. All starting and end dates are selected to be September 2, 2020 through November

13, 2020, in order to align with the Cornell University Fall 2020 academic calendar. We also note that October 14, 2020 is the only holiday during the semester, though holiday information is added during runtime and is not part of the data set.

For varsity data, we synthetically create the teams as follows. First we chose four team names and roster sizes as:

- Women’s varsity Ice Hockey, 25 students
- Men’s varsity Squash, 15 students
- Varsity Football, 100 students
- Varsity Gymnastics, 20 students

Then we select students, without replacement, from the list of students provided in the Weeden and Cornwell data. We do this by starting with student index 10 for women and 11 for men, and incrementing the indices 10 at a time, selecting a student subject to not being labeled as a graduate student. After populating a team, the starting index for populating the next team resumes where the previous index left off.

The resulting list of students, courses, and varsity teams is then formatted appropriately as an input file for the simulator. Remember, this data set is not intended to be a realistic scenario; rather, it is just a random data set to show the functionality of the SEIR-Campus model.

Loading the data.

The basic object of the simulation is the `Semester` object. This object stores all fields described in the input file, expands the meeting schedules into daily meeting lists, and accepts modifications during runtime. To create the `Semester` object, we pass the data filename and a list of holidays that classes will not occur on to the object constructor. The `Semester` object will automatically not expand class meetings on those days.

```
1 from datetime import datetime
2
3 holiday_list = [(2020,10,14),]
4 holidays = set(datetime(*h) for h in holiday_list)
5 semester_full = Semester('publicdata.data', holidays)
```

Running the simulation

Creating a simulation is now very simple. We create a `Parameters` object using the default settings, but changing the number of repetitions to 100 (note that repetitions is the only, and optional, argument to the `Parameters` object; however we will discuss all of its members as we go through the examples). Then we use the convenience function `run_repetitions` to execute multiple instances of the simulator.

```
1 parameters = Parameters(reps = 100) # <-- Can change from default
   number of repetitions
2 run_repetitions(semester, parameters)
```

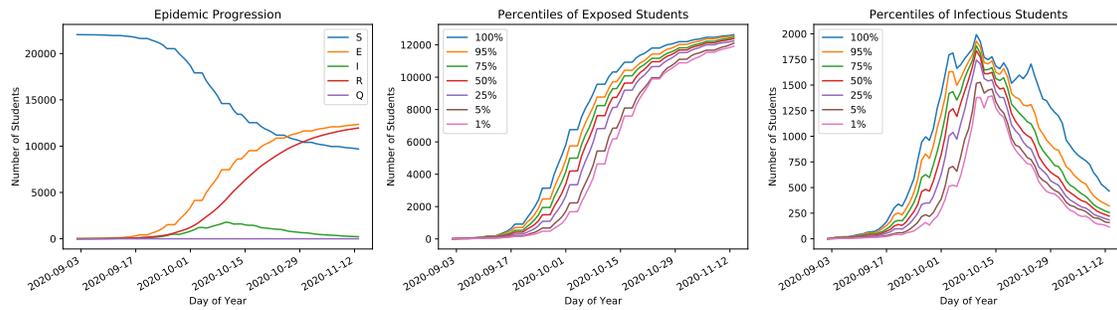


Figure 6.2: Results from running a basic simulation. Each simulation outputs three charts. The first shows the number of susceptible, exposed, infectious, recovered, and quarantined individuals by date. The second shows the percentiles for the number of exposed individuals by date across all repetitions. The final chart shows the percentiles for the number of infectious individuals by date across all repetitions.

Refer to Figure 6.2 for sample results.

6.4.2 Asymptomatic and Symptomatic Individuals

SEIR-Campus allows you to make distinctions between individuals who are asymptomatic and those who are symptomatic. The primary distinction we make is that symptomatic individuals are 1) more contagious than asymptomatic individuals and 2) will self report to take a test when they notice symptoms developing. Control over which individuals are symptomatic and how long infections last can be fully customized, though some convenient defaults are also provided. For full control, you must make a class that provides the following function call:

```

1 def duration(self, simulation, student, date):
2     # (Custom code goes here)

```

```
3     return infectious_start_date, infectious_end_date,
        is_symptomatic
```

The function is called by the simulator with a reference to itself, as well as the student's id and the date of infection. The return value is the day the individual will become infectious, the day the individual will no longer be infectious (possibly because they isolate themselves after recognizing symptoms), and a boolean indicating whether the individual will develop symptoms. For users who don't need as much flexibility, there are two built-in classes ready to use. If you want control over the infection duration without considering symptomatic individuals, you can use the `BasicInfectionDuration` class. It takes two arguments: the rate at which individuals become contagious (per day) and the rate at which contagious individuals recover (in days). It then draws actual durations from a geometric distribution with the given parameters.

```
1 duration_computer = BasicInfectionDuration(1 / 3.5, 1 / 4.5)
```

This gives a mean duration of 3.5 days until an infected individual becomes infectious, and an average of 4.5 days subsequently until they recover.

If symptomatic individuals are of concern, you can use the class `VariedResponse`. This class is initialized with the rate at which individuals become contagious, the rate at which individuals recover, the rate at which individuals develop symptoms (if they will be symptomatic), and the percentage of individuals that never develop symptoms. The default used by the simulator is:

```
1 duration_computer = VariedResponse(1 / 3.5, 1 / 4.5, 1 / 2, 0.75)
```

This gives the same number of days, on average, as before until an individual becomes infectious and then recovers, but also provides that an average of 2 days pass after becoming infectious before the individual starts showing symptoms and that 75% of the individuals are asymptomatic. This is also the default method used by the simulator to compute infection durations if you do not provide an alternative.

The infection duration method of your choice can be passed to the simulator by placing it in the `Parameters` object. Here is an example where all individuals are asymptomatic. Note that the code for functions that are already implemented can be seen in the code repository and can serve as a template for creating new ones.

```
1 parameters = Parameters(reps = 100)
2 parameters.infection_duration = BasicInfectionDuration(1 / 3.5, 1 /
   4.5)
3 run_repetitions(semester, parameters)
```

Sample results for this code are shown in Figure 6.3.

6.4.3 Testing

The first intervention strategy we will look at is testing, which identifies whether individuals are infected. In simulation, determining whether an individual is infected comes from inspection of the individual's state. E , I_a , and I_s could all be states

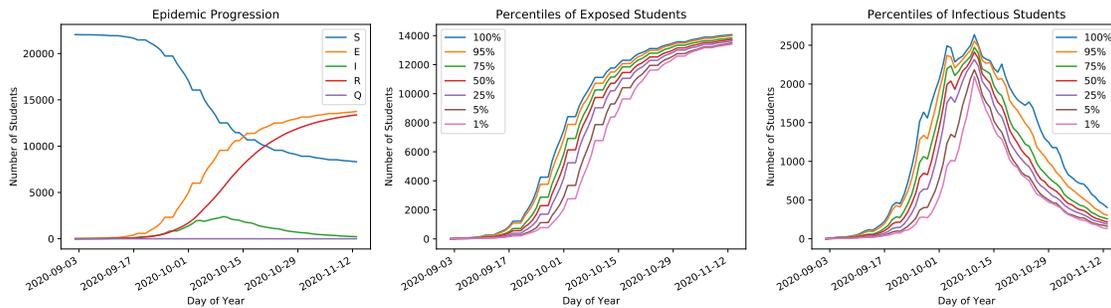


Figure 6.3: The three output charts for the basic simulation with the modification that none of the individuals report/have symptoms.

that might cause a positive test. Some infections may only be detectable while the individual is in infectious, in states I_a and I_s . The goal of a testing policy is to report individuals who are infected so that they can quarantine to prevent further spread of infection and possibly help with contact tracing efforts. While the default setting in the simulation is to not use any testing, there are multiple interfaces one can use.

The `Simulator` object tracks several quantities across the duration of the simulation to help facilitate testing policies, even though the `Simulator` itself leaves testing to outside functions. The most useful quantity for knowing who to test is a dictionary called `test_requests`, indexed by date. This dictionary contains a list for each day of individuals who have been recommended for testing on that day. Individuals may have been added to this list for several reasons, for example individuals who show symptoms that may wish to be tested or individuals reported by contact tracing for testing. Of course, individuals not on the `test_request` list might also be tested as well, especially in infection surveillance programs that purposely test individuals that might not know they need to be tested. Other information tracked

by the `Simulator` is also directly relevant to determining test results, such as state information that tracks which individuals are currently infectious.

The `Simulator` object offers a very general way to implement testing policies by placing the responsibilities for testing in a callable object given by the user and placed in the `Parameters` object. The goal of the object is to report to the `Simulator` which individuals have taken a test that will return a positive result. The object is called by the simulator on each simulated day. At a more detailed level, the object should be an instance of a class that provides the function `testing` whose arguments take a reference to the `Simulator` and the current date as arguments. The `testing` function can determine who to test and who will get a positive result using any of the resources mentioned in the previous paragraph, possibly even including random elements if false positive and false negatives are to be modeled. Positive results are reported through a function exposed by the `Simulator` object called `schedule_positive_test_result`, which takes as an argument the id of the individual who tests positive. It will automatically notify the individual of their test result after the default test return latency (1 day).

As an example, consider the following testing policy, which is included in the `SEIR-Campus` package.

```
1 class IpGeneralTesting:
2     def __init__(self, test_groups):
3         self.test_groups = test_groups
4     def testing(self, simulation, date):
5         to_test = set(self.test_groups[date.weekday()])
```

```

6         to_test.update(simulation.test_requests[date])
7         for s in to_test:
8             if s in simulation.state.Ia or s in simulation.state.Is
or \
9                 s in simulation.state.Qa or s in simulation.
state.Qs:
10                simulation.schedule_positive_test_result(s)

```

This testing policy tests a fixed group of individuals each day of the week, as well as any who have been identified by contact tracing. The class initializes with a dictionary mapping the weekday (as a number 0-6). When the `testing` function is called, it gets the set of individuals that regularly test on that day of the week and adds to that the set of individuals who need to be tested because of contact tracing or showing symptoms. For each individual identified, if they are infectious or in quarantine and infectious it reports they have tested positive.

The flexibility offered by creating a testing class allows potential for other considerations, such as testing that does not conform to weekly patterns, false negative and false positive test results, and failure to participate for individuals.

SEIR-Campus comes with some convenient ready-made testing policies to explore. Below are descriptions and how to create them.

- `IpWeeklyTesting(semester, weekday = 0)` Test all individuals on a given day of the week (default is Monday).

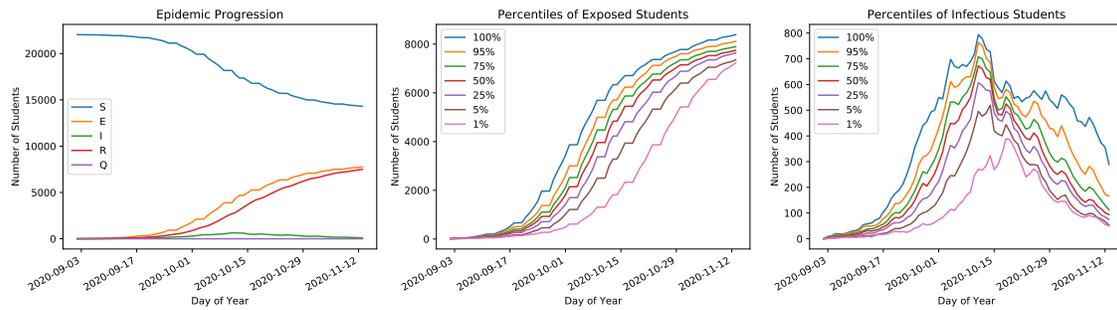


Figure 6.4: The three output charts for the basic simulation with the modification that each weekday, 1/5 of the student body is tested for infection. Those that test positive are placed in quarantine the next day when the result comes back.

- `IpWeekdayTesting(semester)` Test individuals once per week, with individuals evenly assigned to be tested on Monday - Friday.
- `IpDailyTesting(semester)` Test individuals once per week, with individuals evenly assigned to be tested each of the seven days of the week.
- `IpRollingTesting(semester, days = 5)` Test all individuals on a 5 day cycle, independent of alignment with the week. Individuals evenly assigned to test each day of the rolling cycle.

Here we run an example of testing, with sample results shown in Figure 6.4.

```

1 parameters = Parameters(reps = 100)
2 parameters.intervention_policy = IpWeekdayTesting(semester)
3 run_repetitions(semester, parameters)

```

6.4.4 Contact Tracing

Contact tracing is an essential tool in public health. Contact tracing allows those who may have been exposed to an infectious individual to either be tested or quarantined before they would normally be tested or develop symptoms. The default setting in SEIR-Campus is to not perform any contact tracing, though the package comes with some simple policies that can be used as well as the ability to make custom policies. A custom contact tracing policy can be made by producing a class that provides the function `trace`, taking a reference to the `Simulator` object and the date as arguments. To illustrate this, let's look at the implementation of `BasicContactTracing` in SEIR-Campus.

```
1 class BasicContactTracing:
2     def __init__(self, quarantine_length, trace_length = 3):
3         self.quarantine_length = quarantine_length
4         self.trace_length = trace_length
5     def trace(self, simulation, date):
6         ''' For each student to trace, find all of their peers from
7         the past
8             3 days. Then, quarantine all those peers for some days
9         and
10            schedule a test for them. '''
11         trace_dates = [date - timedelta(days = d + 1) for d in range
12            (self.trace_length)]
13         for s in simulation.contact_trace_request[date]:
14             recent_meets = set()
15             for m in simulation.semester.student_enrollment[s]:
```

```

13         if simulation.semester.meeting_type[m] != MeetType.
UNTRACEABLE:
14             for d in trace_dates:
15                 if m in simulation.semester.meeting_dates[d
]:
16                     recent_meets.add(m)
17                     break
18             peers = set()
19             for m in recent_meets:
20                 peers.update(simulation.semester.meeting_enrollment[
m])
21             if len(peers):
22                 peers.remove(s)
23                 for peer in peers:
24                     simulation.initiate_quarantine(peer, date, self.
quarantine_length)

```

The `BasicContactTracing` class gets the contact tracing list from the `Simulator` object's dictionary member `contact_trace_request`, indexed by date. By default, this policy looks back three days at all meetings for the individual that are traceable and sends all members of those groups to quarantine. It does this using the `Simulator` object's member function `initiate_quarantine`, which requires arguments of the student's id, the date the quarantine starts, and optionally the length of the quarantine. If no length is given, the default specified in the `Parameters` object is used.

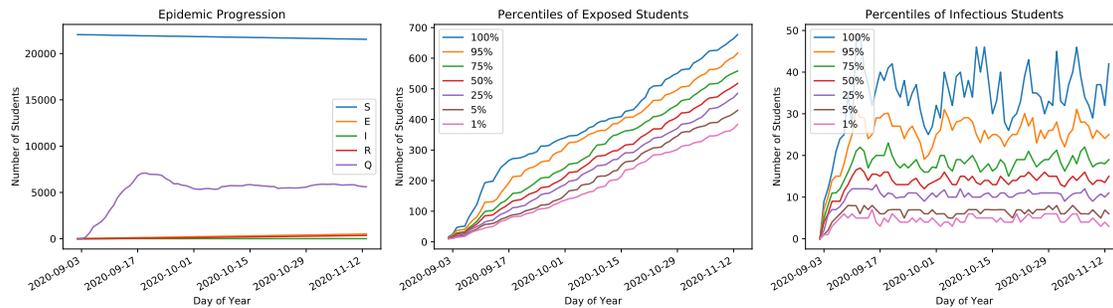


Figure 6.5: The three output charts for the basic simulation with the modification that when individuals show symptoms and take a test with a positive result, everyone they have been near in the past three days is placed in quarantine.

Here we demonstrate the usage of the basic form of contact tracing.

```

1 parameters = Parameters(reps = 100)
2 parameters.contact_tracing = BasicContactTracing(14)
3 run_repetitions(semester, parameters)

```

Sample results from this code are shown in Figure 6.5.

6.4.5 Exploring Alternative Course Designs

In preparing a policy for operation, course design can have a big impact since it is a primary contributor of social interactions under the control of the university. The `Semester` object in SEIR-Campus comes with function to help explore modifications to the course meeting design by adding and removing individual courses. There are also several built in functions that can modify a semester in specific ways.

To add a course, the `Semester` object provides a function `add_meeting`. It takes

as arguments

1. **name** The id the course will be referenced by in the simulation.
2. **info** A dictionary of information about the meeting.
3. **meets** Can be given in one of two forms. The first form is a dictionary, indexed by date, that gives the duration of the class meeting on the days it meets. The second form is a list of tuples. Each tuple contains a start date, an end date, a weekday meeting pattern, and a duration.
4. **members** A list or set of student ids corresponding to individuals enrolled in the course.
5. **meet_type** One of the following: `MeetType.COURSE`, `MeetType.SOCIAL`, or `MeetType.UNTRACEABLE`.
6. **holidays** If giving the **meets** argument as a list, this is a list of dates to exclude from class meetings.

To remove a course, you only need to know the name (id) of the course that should be removed, using the function `remove_meeting`, taking the name as the only argument. There is also a function, `clean_student_list`, that removes any individuals that are not part of any meeting.

The usage of these function can be shown through the following example. Suppose we want to move all classes with capacity higher than 50 online, meaning they no longer represent in-person contact. The following built-in function accomplishes this:

```

1 def make_alternate_smallclasses(semester_base, max_size = 50):
2     ''' No large course: only classes with at most 50 students. '''
3     semester = copy.deepcopy(semester_base)
4     canceled_meetings = [m for m, ss in semester.meeting_enrollment.
5                           items()
6                           if len(ss) > max_size
7                           and semester.meeting_type[m] == MeetType.
8                           COURSE]
9     for m in canceled_meetings:
10         semester.remove_meeting(m)
11     return semester
12 semester_alt_small = make_alternate_smallclasses(semester, max_size
13         = 50)

```

First this class makes a clean copy of the `Semester` object. Then it makes a list of all courses with capacity greater than 50. Each class in this set is then removed from the meeting list through the `remove_meeting` function provided by the `Semester` object.

SEIR-Campus also comes with two other functions to modify schedules. The first is `make_alternate_hybrid`, which randomly separates courses into two groups and then, alternating each week, schedules one group to be in-person and the other to be online. Online education does not involve contact, so the class is effectively canceled on that week. The second is `make_alternate_splitclass`. This function splits all courses in half (effectively into two classes each with half of the students) and then

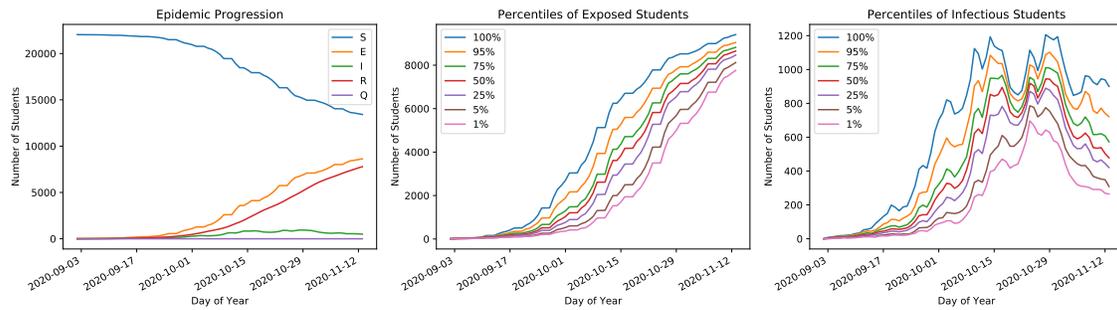


Figure 6.6: The three output charts for the basic simulation with the modification that courses are split into two groups, each groups alternating each week between meeting in-person and meeting online.

alternately schedules, week by week, the two splits. As an example, the following code creates and simulates a hybrid schedule starting from the loaded data.

```

1 semester_alt_hybrid = make_alternate_hybrid(semester)
2 parameters = Parameters(reps = 100)
3 run_repetitions(semester_alt_hybrid, parameters)

```

Sample results are shown in Figure 6.6.

6.4.6 Social Groups

While social groups can have a large impact on spread of infectious diseases, unlike courses the precise interaction network is not known. The SEIR-Campus `meeting` objects provide a framework to add social interactions by modeling these interactions as courses with specific meeting durations and dates. For example, a group of friends that meets every weekend can be modeled as a class that meets for an hour or

two on Saturdays. In practice, social networks are often not known exactly and instead generative models are used to create interaction networks. The social network features in SEIR-Campus come with a convenient type of object, called a `Cluster`, that is specifically intended for use in producing social networks. A `Cluster` is like a simplified form of course, only containing a list of members and a dictionary, indexed by date, of meeting durations. The sample social-network generators all return lists of `Clusters`. Once all desired social interactions have been produced, the list of `Clusters` is passed to the function `make_from_clusters` which automatically expands each `Cluster` into a social meeting in the `Semester` object.

Randomized social meetings.

The simplest way to produce social meetings is to produce them randomly from the population. The built-in function `make_randomized_clusters` provides an easy way to do this, given the population and a `ClusterSettings` object. This object is specifically tailored for use in the built-in functions for making clusters but is not generally necessary if you wish to make your own custom cluster generators. The built-in functions work on the following principle: from the population, draw a certain number of clusters of a certain size (to the extent possible). Then, each individual interacts with everyone in their cluster for a predefined amount of time, given that the day is either a weekday or a weekend. To enable this, the `ClusterSettings` object contains the following settings:

- `start_date, end_date` Time range for the set clusters

- `weekend_group_count`, `weekday_group_count` Number of individuals per group, by weekend or weekday.
- `weekend_group_size`, `weekday_group_size` Number of individuals to draw from each group each day to form a cluster, by weekend or weekday.
- `weekend_group_time`, `weekday_group_time` Amount of time individuals in each cluster spend with each other.

A simple way to produce a social network for a `Semester` object is illustrated below. In this example, each weekday 280 groups of 10 random students meet for 120 minutes and each day of the weekend 210 groups of 20 random students meet for 180 minutes.

```

1 settings = ClusterSettings(
2     start_date = min(semester.meeting_dates), end_date = max(
3     semester.meeting_dates),
4     weekday_group_count = 280, weekday_group_size = 10,
5     weekday_group_time = 120,
6     weekend_group_count = 210, weekend_group_size = 20,
7     weekend_group_time = 180)
8 clusters = make_randomized_clusters(semester.students, settings)
9 final_semester = make_from_clusters(semester, clusters)

```

As shown above, first we create a `settings` object, create the clusters using the `make_randomized_clusters` function, and then apply them to the `Semester` object using the `make_from_clusters` function.

Another built in function is `make_social_groups_pairs`. This function, which as an argument takes a `Semester` object, a number `fraction_paired` for the percentage of individuals that are paired, a number `interaction_time` for the number of minutes each pair interacts per day, and an optional set `excluded` of students ids of individuals that should not be considered in the pairing process.

Many of these function utilize randomization. In order to ensure that a new randomized network is used with each simulation repetition, there is a special option in the `Parameters` object to set a function `preprocess`, which as argument takes a `Semester` object and returns a new `Semester` object. This function is called to create a new `Semester` object prior to each repetition. Its usage is demonstrated in the following example, which uses the same definition of `ClusterSettings` as above, but instead of creating a single `Semester` object it creates a new version each time.

```
1 def groups_random(semester):
2     clusters = make_randomized_clusters(semester.students,
3     settings)
4     return make_from_clusters(semester, clusters)
5 parameters = Parameters(reps = 100)
6 parameters.preprocess = groups_random
7 run_repetitions(semester, parameters)
```

Sample results are shown in Figure 6.7.

Similarly, we can use the `preprocess` capability to create new sets of pairs of individuals in each repetition.

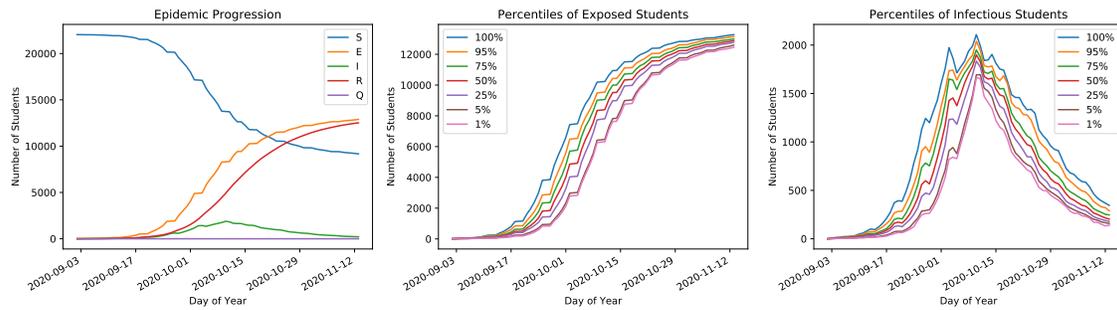


Figure 6.7: The three output charts from the basic simulation with the modification that social groups have been added via random social clusters. These groups meet for 2 hours on weekdays and 3 hours on weekends.

```

1 def pairing(semester):
2     clusters, _ = make_social_groups_pairs(semester, 0.25,
3     interaction_time = 1200, weighted=False)
4
5     return make_from_clusters(semester, clusters)
6
7 parameters = Parameters(reps = 10)
8 parameters.preprocess = pairing
9 run_repetitions(semester, parameters)

```

Sample results are shown in Figure 6.8.

6.4.7 Social meetings from groups.

The final example we will give shows how to utilize group information loaded in the input file. Recall that group information in the input file is optional but can be useful for creating custom groups. Here we will make social groups corresponding

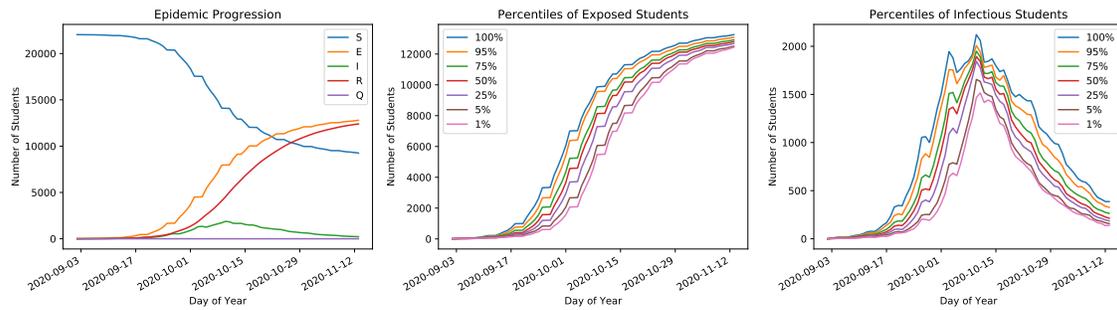


Figure 6.8: The three output charts for the basic simulation with the modification that 25% of individuals have been randomly assigned to be pairs that meet for the equivalent of 20 hours each day, which accounts for hypothesized lack of social distance and not wearing face masks.

to varsity athletic teams. Within each of the varsity sports, each day we will split the group into clusters of six (to the extent possible) under the assumption that six is the maximum group size allowed to interact. Individuals are assumed to interact with all other individuals within the cluster of six for 180 minutes per day. Included in the data set that we loaded are entries like the following

```
1 {'group': {'name': "Men's Varsity Squash", 'members': [170, 610,
    630, 950, 970, 1150, 1350, 1530, 1590, 1630, 1770, 1850, 1870,
    1890, 1910]}}
```

with we have created to mimic the composition of of each varsity sports team.

The built-in function we use to process these teams and create clusters is shown below.

```
1 def make_social_groups_varsity(semester,
2                               weekday_size, weekday_time,
3                               weekend_size, weekend_time):
```

```

4     varsity = {name : members for name, members in semester.groups.
items()
5
6         if 'Varsity' in name}
7     max_size = max([len(x) for x in varsity.values()])
8     cluster_settings = ClusterSettings(min(semester.meeting_dates),
max(semester.meeting_dates),
9
10         max_size, weekday_size,
11         weekday_time,
12         max_size, weekend_size,
13         weekend_time)
14     clusters = []
15     processed_students = set()
16     for sport, roster in varsity.items():
17         clusters.extend(make_randomized_clusters(roster,
18         cluster_settings))
19         processed_students.update(roster)
20
21     return clusters, processed_students

```

First the function reads through the list of groups in the `Semester` object and saves those that have “Varsity” in the name. Since the behavior of the `ClusterSettings` object is to make only a limited number of groups from the total population, we set its parameters `weekday_group_count` and `weekend_group_count` to a sufficiently large number (`max_size`) so that the group count constraint never causes anyone on the roster to be omitted from placement in a cluster. Then, for each sports team we use the `ClusterSettings` object to make the clusters based on the team’s roster. We

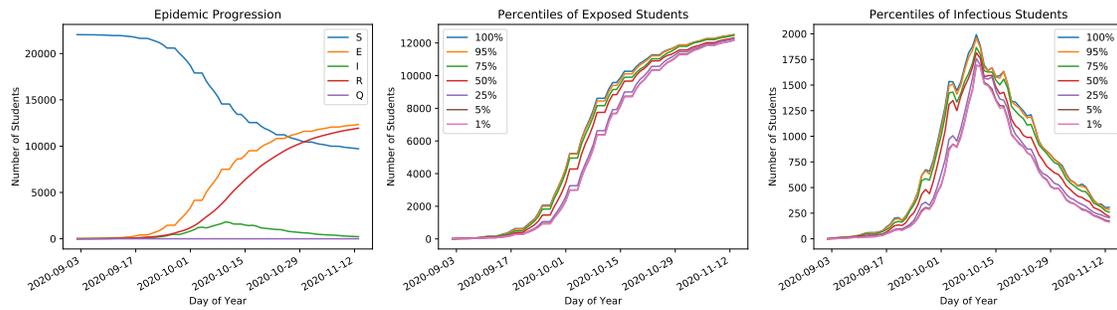


Figure 6.9: The three output charts for the basic simulation with the modification that members of varsity teams spend time with each other each day, with clusters changing within each team each day.

return the set of clusters as well as a set called `processed_students` which let's the user know which individuals are present in clusters. This may be useful if you don't want individuals assigned a group by one function to be assigned another group in a another function. The following code runs the scenario with varsity student social groups.

```

1 def groups_varsity(semester):
2     clusters, processed = make_social_groups_varsity(semester,
3     6, 240, 6, 240)
4
5     return make_from_clusters(semester, clusters)
6
7 parameters = Parameters(reps = 10)
8 parameters.preprocess = groups_varsity
9 run_repetitions(semester, parameters)

```

Sample results are shown in Figure 6.9.

Under some social models, the remixing of groups into new clusters each

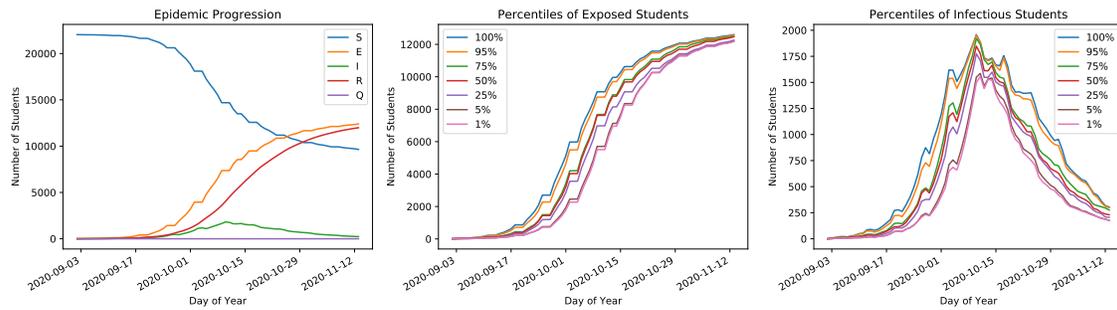


Figure 6.10: The three output charts for the basic simulation with the modification that members of varsity teams spend time with each other each day. In this version, clusters do not change over the course of the semester.

day may seem excessive. In this case, we can substitute the function `make_randomized_cluster` with `make_randomized_static_clusters`, which takes the same arguments but uses the same clusters each day, defaulting to weekday settings for each day. Demonstrating this alternative of the previous scenario, the following code produces the result shown in Figure 6.10.

```

1 def groups_static_varsity(semester):
2     clusters, processed = make_social_groups_varsity_static(
3         semester, 6, 240)
4
5     return make_from_clusters(semester, clusters)
6
7 parameters = Parameters(reps = 10)
8 parameters.preprocess = groups_static_varsity
9 run_repetitions(semester, parameters)

```

6.4.8 Overview of other Parameters settings

Not all of the options available in the `Parameters` object were described in the examples above. This section lists and summarizes the members and how to use them.

- `verbose` - Output day-by-day statistics of infection states, as well as computation time.
- `rate` - Probability that an infectious individual spreads their infection to a susceptible individual per effective minute of contact time.
- `daily_spontaneous_prob` - Probability that a susceptible individual becomes infected each day from an external source that is not part of the simulation. This models infections coming from outside the community.
- `contact_tracing` - Python object that has member function `trace` that takes as arguments `Simulator` and a date with no return value. It performs contact tracing as described in previous sections.
- `intervention_policy` - Python object that has member function `testing` that takes as arguments `Simulator` and a date with no return value. It performs interventions, such as testing, as described in earlier sections.
- `infection_duration` - Python object that has member function `duration` with arguments `Simulator`, student id, and date. It then returns the date the individual will become infectious, the day they will stop being infectious, and a Boolean indicating if this is because the individual will become symptomatic.

- `quarantine_length` - Default number of days individuals sent to quarantine spend before being returned to the general population, if they are in fact not infected.
- `preclass_interaction_time` - Extra time individuals spend near each other at the beginning and end of class that adds extra interaction time due to entering and leaving the classroom. Only applies to meetings of type `MeetType.COURSE`.
- `initial_exposure` - Either a list of individuals who are initially exposed or an integer for a number of individuals to draw at random at the beginning of the simulation that are exposed.
- `preprocess` - Function that takes `Semester` object as input and output that performs any preparation, including randomization, prior to each repetition.
- `repetitions` - Number of simulation repetitions to perform.
- `start_date` - Python `Datetime` object representing the first day to simulate. The default value is September 2, 2020.
- `end_date` - Python `Datetime` object representing the last day to simulate. The default value is November 13, 2020, the day before Thanks Giving break.

6.5 Conclusions

We encourage everyone to visit our Git repository at <https://github.com/MAS-Research/SEIR-Campus>. It includes a copy of the SEIR-Campus package as

well as a Jupyter notebook containing all of the examples included in this chapter. We hope that the package will be useful and remind users that the intention of the package is to be customized to fit your needs. We encourage user comments and suggestions for additional features that may be useful for us to include in the future.

APPENDIX A
APPENDIX FOR CHAPTER 1

A.1 Battery Section

An important part of running any system with electric vehicles is to understand how the batteries behave. In this section we briefly review some of the literature on batteries used in electric vehicles and then demonstrate that simplified models can be justified for typical everyday use in the context of operational optimization of ridepooling fleets.

A.1.1 Charging.

While it may be natural to assume that batteries gain charge linearly over time when they are plugged into charge stations, this turns out to not be the case [54]. Many batteries charge using a procedure called Constant-Current Constant-Voltage. Under this system, the voltage in the charger slowly increases to maintain a constant current into the battery. However, batteries have a maximum safe voltage they can operate at. Once the voltage in the charger reaches that threshold, the voltage is then held constant at the maximum value and the current into the battery steadily drops off. This explains why batteries can charge to mid-high charge values, such as 75%, quickly though getting to 100% can take significantly longer.

The differential equation that governs the state of charge in the battery in the constant voltage regime is given by

$$\frac{d}{dt}\text{SOC}(t) = \frac{V_{CV} - V_{OC}(\text{SOC}(t))}{R \cdot 3600 \cdot Q}$$

where SOC is the state of charge, V_{CV} is the voltage in the charger at the constant voltage level, V_{OC} is the open circuit voltage, R is the resistance in the charger, and Q is the maximum battery capacity in ampere-hours. Simplifying the form of the differential equation, we get

$$\frac{d}{dt}\text{SOC}(t) = C_1 - C_2 V_{OC}(\text{SOC}(t)).$$

If we make the rough approximation that V_{OC} is a linear function of the state of charge, then the solution to the differential equation is

$$\text{SOC}(t) = \frac{C_1}{C_2} + ke^{-C_2 t}.$$

Since under this model the battery can never theoretically be fully charged, we make the following assumption so that charging can have a well defined endpoint.

Assumption A.1.1. *The manufacturer specifies that “100%” charge is some fixed level below this asymptotic maximum. Meaning, “100%” is not truly at full charge.*

As an illustration of the model, combining the solution for $\text{SOC}(t)$ for the Constant-Voltage regime and a linear function for the Constant-Current, let τ denote the time to fully charge an empty battery to the manufacturer’s full capacity (less than true theoretical capacity), let ρ denote the number of minutes after which

the charging rate of an initially empty battery begins to decrease and let Q be the amount of charge in the battery at time ρ . Figure (A.1) shows a typical charging curve.

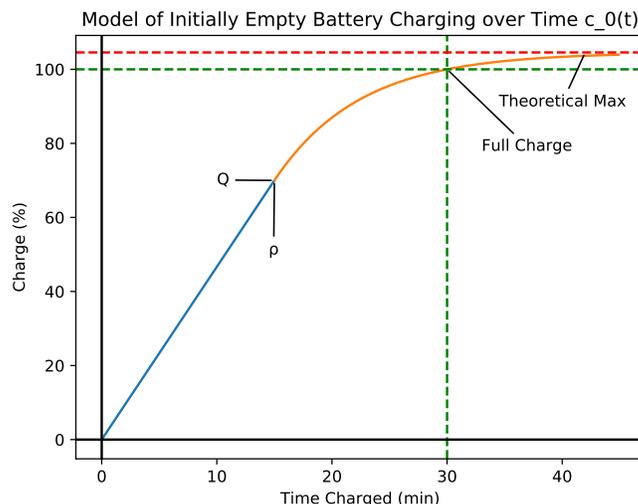


Figure A.1: Simple model of charge in an initially empty battery as a function of charging time. The curve has two components: in blue, the charge in the battery initially increases linearly until time ρ , at which point the battery has charge Q and charging continues as the orange curve which asymptotically approaches the battery's theoretical maximum charge at an exponentially decreasing rate. Denoted in green are the lines $x = \tau$ and $y = 1$. The asymptotic limit is shown as a red line. In this example, linear charging lasts for the first $\rho = 15$ minutes at which point the battery has charge $Q = 0.7$ and it takes $T = 30$ minutes for a full charge.

Due to assumption A.1.1, the charge in the initially empty battery as a function of time is given by the concave function

$$c_0(t) = \begin{cases} \frac{Qt}{\rho} & 0 \leq t \leq \rho \\ 1 - \frac{Q}{\rho\beta} \left(\frac{1}{e^{\beta(\tau-\rho)}} \right) (e^{\beta(\tau-t)} - 1) & \rho < t \end{cases}$$

where the scaling parameter β is given by

$$\beta = \frac{Q}{\rho(1-Q)} + \frac{1}{\tau-\rho}W(ze^z), \quad z = \frac{\rho-\tau}{\rho} \frac{Q}{1-Q},$$

where W denotes the Lambert W function and $\frac{Q}{\rho} > \frac{1}{\tau}$.

Now consider a battery that is not initially empty. If the vehicle requires t minutes to completely charge, the amount by which the battery was initially depleted must have been $c(t) = 1 - \mathbf{c}_0(\tau - t)$. Similarly, given a battery with state of charge q the time to fully charge is given by $f(q) = c^{-1}(q)$.

Since charging is less time efficient as the battery nears full charge, it is natural from an economic standpoint that there may be a policy that vehicles should only charge up to some level $q \in [Q, 1]$ instead of fully charging due to the opportunity costs associated with charging, such as inability to carry passengers during charging. Additionally, since charging is most efficient in the linear charging region with state of charge in $[0, Q]$ we make the following assumption about when vehicles begin charging.

Assumption A.1.2. *Vehicles always deplete their battery to a state of charge less than or equal to Q before they begin charging.*

Consider the following scenario. Suppose an operator allows the batteries in the fleet's vehicles to discharge down to zero at which point the operator orders the vehicles to charge their batteries back up to state of charge q' .

For simplicity, suppose that q_{est} is the discharge rate while vehicles operate and

that vehicles have to travel empty for d minutes to reach a charge station. If the cost per minute of unusable vehicles is c , then the average operating cost is

$$k(q') = \frac{d + f(q')}{d + f(q') + q'/q_{est}} \cdot c.$$

The optimal value for q' can be quickly found via binary search. Figure A.2 shows examples of choosing a charge up-to policy of minimum cost. The first shows a general example and the second shows the effect of having no lost time ($d = 0$) traveling with no passengers while they travel to charging stations.

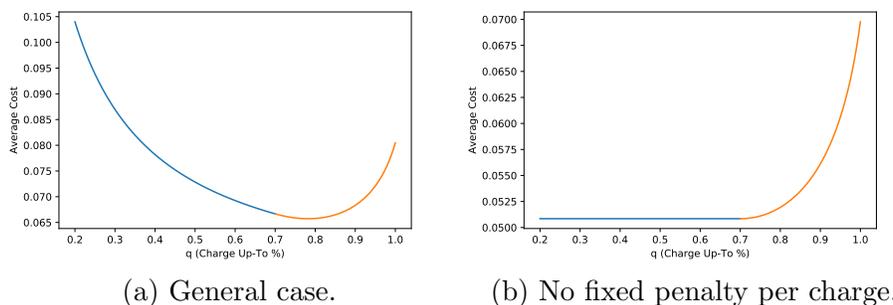


Figure A.2: Assume a vehicle takes 30 minutes to fully charge and receives 70% of its charge from empty in the first 15 minutes. Suppose that on average the vehicle wastes 5 minutes of time traveling to the charge station while not holding any passengers, a full battery lasts 400 minutes, and that offline time costs the operator \$1 per minute. Figure (A.2a) plots $k(q')$ with $d = 5$ and Figure (A.2b) plots $k(q')$ with $d = 0$. Optimal value in general case is $q = 78.2\%$ and optimal value when $d = 0$ is any value less than Q .

In the general case, see Figure (A.2a), the optimal behavior is to allow the vehicles to charge slightly beyond the constant-voltage portion of the charging curve. Referencing the corresponding charging curve in Figure (A.1), charging just beyond

the constant voltage point, Q , can be approximately modeled as a linear function with little loss of accuracy.

Thus, while studies such as [23] have policies that assume full recharging and find that the non-linear portion of the charging curve is important to model accuracy, under optimal operating policies we can assume that the battery charging functions nearly linear and therefore model them as linear functions.

APPENDIX B
APPENDIX FOR CHAPTER 3

B.1 CTSP Heuristics

This appendix briefly describes how the CTSP can be exactly solved. It then uses the same machinery to describe two heuristics.

In the case of exact enumeration, the easiest way to find the optimal route is via a recursive, depth first search of the order in which the requests are to be served. For each request that has not yet boarded the vehicle, we create a `NodeStop` for their origin and destination. A `NodeStop` contains four pieces of data: a reference to the request it is associated with, a physical location it is associated with, whether it represents a pickup or a drop off, and the latest time the `NodeStop` can be visited. The last piece of information is an invaluable way to check that QoS constraints are satisfied as the path is being built. For requests onboard a vehicle, a `NodeStop` only needs to be made for the drop off.

To make the recursive search more efficient, and to generalize in a way that is useful to our heuristics, we create a new structure called a `MetaNodeStop`. A `MetaNodeStop` contains two pieces of data: a `NodeStop`, and a (possibly empty) list of follower `MetaNodeStops` that can only be accessed subsequent to visiting the `NodeStop`. This information is used in the recursive search procedure by means of a list of available `MetaNodeStops`: the next `NodeStop` on the optimal path can

only be selected from among those contained in the available MetaNodeStops. Once a NodeStop is selected as the next candidate node in the path search process, all follower MetaNodeStops are added to the available list in the recursive call to find the optimal remainder of the path.

The simplest application of this is precedence constraints for pickup and drop off. Suppose that the MetaNodeStop associated with each pickup NodeStop contained the associated drop off MetaNodeStop as the only follower. If the recursive path function is initially given an availability list that only includes MetaNodeStops associated with pickups, with the exception that we include MetaNodeStops for drop offs for onboard requests, then any path produced by the recursive search function will automatically satisfy the required precedence constraints.

Using this terminology, we now discuss two heuristics below and comment on their stability. Refer to Definition 3.5.1 for the definition of a stable heuristic.

B.1.1 Fixing order of on-board passengers

The On-board Order Fixing (OOF) heuristic was used in [3], though we use a variation here. In its original presentation, this heuristic is only used in place of exact search when the number of passengers plus the number of requests yet-to-board exceeds a threshold, which they choose to be four. It can be checked that this is a stable threshold policy.

The OOF heuristic works by fixing the drop off order of all passengers onboard the vehicle, but any order can be used for requests that have yet to board. In terms of MetaNodeStops, each MetaNodeStop associated with a drop off has as its only follower the MetaNodeStop associated with the next passenger drop off, in the order determined by the route assigned to the vehicle in the previous assignment iteration. The set of initially available MetaNodeStops is given by all MetaNodeStops associated with the pickup of new requests, and the singular MetaNodeStop associated with the drop off of the passenger onboard (if any exist) that was scheduled to be dropped off first in the previous iteration's assignment.

Lemma B.1.1. *The OOF heuristic is stable.*

B.1.2 Limit and recall prefix

The Limit and Recall Prefix (LRP) heuristic is a generalization of the OOF heuristic that allows for larger trip sizes. The idea behind LRP is that if a vehicle was previously assigned many requests, then adding new requests in a future iteration is more likely to change the portions of the path that are further in the future than the portions that will happen more imminently.

The LRP heuristic works by choosing a threshold for the maximum number of NodeStop that can be ordered through complete enumeration. A passenger onboard a vehicle would count as one NodeStop while a request that had not yet boarded would be two NodeStop, one for pick up and one for drop off. If the number of NodeStop

exceeds the threshold, then we start by recalling the vehicle's path assignment from the previous iteration. We remove from the previous path all NodeStop that do not represent a NodeStop in the current problem. Then, among those that are left we choose the smallest prefix of that path such that number of NodeStop we need to consider that are not part of the prefix is at most the threshold. If no prefix can satisfy this condition, we can either reject the trip or just use the maximum possible prefix size. We then search for the optimal path with the constraint that the all NodeStop in the prefix occur in the same order as in the previous iteration, though NodeStop not in the prefix may interweave.

Lemma B.1.2. *The LRP heuristic is stable.*

BIBLIOGRAPHY

- [1] Ravindra K Ahuja, James B Orlin, and Dushyant Sharma. Very large-scale neighborhood search. *International Transactions in Operational Research*, 7(4-5):301–317, 2000.
- [2] Lina Al-Kanj, Juliana Nascimento, and Warren B Powell. Approximate dynamic programming for planning a ride-hailing system using autonomous fleets of electric vehicles. *European Journal of Operational Research*, 2020.
- [3] Javier Alonso-Mora, Samitha Samaranyake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.
- [4] Javier Alonso-Mora, Alex Wallar, and Daniela Rus. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3583–3590. IEEE, 2017.
- [5] Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge weighted online windowed matching. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 729–742, 2019.
- [6] Riti Bahl, Nicole Eikmeier, Alexandra Fraser, Matthew Junge, Felicia Keesing, Kukai Nakahata, and Lily Z Wang. Modeling covid-19 spread in small colleges. *arXiv preprint arXiv:2008.09597*, 2020.
- [7] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.
- [8] Gordon S Bauer, Amol Phadke, Jeffery B Greenblatt, and Deepak Rajagopal. Electrifying urban ridesourcing fleets at no added cost through efficient use of charging infrastructure. *Transportation Research Part C: Emerging Technologies*, 105:385–404, 2019.

- [9] C Bongiovanni, Mor Kaspi, and Nikolas Geroliminis. A two phase heuristic approach for the dynamic electric autonomous dial-a-ride problem. In *hEART Conference*, 2018.
- [10] Claudia Bongiovanni, Mor Kaspi, and Nikolas Geroliminis. The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological*, 122:436–456, 2019.
- [11] T Donna Chen, Kara M Kockelman, and Josiah P Hanna. Operations of a shared, autonomous, electric vehicle fleet: Implications of vehicle & charging infrastructure decisions. *Transportation Research Part A: Policy and Practice*, 94:243–254, 2016.
- [12] TC Edwin Cheng and Qing Ding. Single machine scheduling with deadlines and increasing rates of processing times. *Acta Informatica*, 36(9-10):673–692, 2000.
- [13] Regina R Clewlow and Gouri S Mishra. Disruptive transportation: The adoption, utilization, and impacts of ride-hailing in the united states. 2017.
- [14] Terra Curtis, Meg Merritt, Carmen Chen, David Perlmutter, Dan Berez, and Buffy Ellis. Partnerships between transit agencies and transportation network companies (TNCs). *TCRP Research Report*, 204, 2019. ISBN: 9780309480581 Number: Project J-11/Task 26.
- [15] [dataset] New York City Taxi & Limousine Commission. Trip record data, Accessed March 2019.
- [16] Camille Démarre. Algorithms for emerging transportation systems, internship report. Technical report, École Polytechnique, 2020.
- [17] Jan Dornoff, Uwe Tietge, and Peter Mock. On the way to” real-world” co2 values: The european passenger car market in its first year after introducing the wltp. Technical report, International Council on Clean Transportation Europe, 2020.
- [18] Gregory D Erhardt, Sneha Roy, Drew Cooper, Bhargava Sana, Mei Chen, and

- Joe Castiglione. Do transportation network companies decrease or increase congestion? *Science advances*, 5(5):eaau2670, 2019.
- [19] Stephen Errity. New electric vans 2020 and beyond. <https://www.drivingelectric.com/news/622/new-electric-vans-2020-and-beyond>, 2020.
- [20] H Farhan and T Donna Chen. Impact of ridesharing on operational efficiency of shared autonomous electric vehicle fleet. *Transportation Research Part C: Emerging Technologies*, 93:310–321, 2018.
- [21] Fabian Fehn, Florian Noack, and Fritz Busch. Modeling of mobility on-demand fleet operations based on dynamic electricity pricing. In *2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 1–6. IEEE, 2019.
- [22] Chiara Fiori, Kyoungcho Ahn, and Hesham A Rakha. Power-based electric vehicle energy consumption model: Model development and validation. *Applied Energy*, 168:257–268, 2016.
- [23] Aurélien Froger, Jorge E Mendoza, Gilbert Laporte, and Ola Jabali. New formulations for the electric vehicle routing problem with nonlinear charging functions. *Centre interuniversitaire de recherche sur les reseaux d’entreprise, la logistique et le transport (CIRRELT)*, 2017.
- [24] Philip T Gressman and Jennifer R Peck. Simulating covid-19 in a university environment. *preprint arXiv:2006.03175*, 2020.
- [25] Xiaotong Guo, Yang Liu, and Samitha Samaranayake. Solving the school bus routing problem at scale via a compressed shareability network. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 1900–1907. IEEE, 2018.
- [26] Nils Haglund, Miloš N. Mladenović, Rainer Kujala, Christoffer Weckström, and Jari Saramäki. Where did kutsuplus drive us? ex post evaluation of on-demand micro-transit pilot in the helsinki capital region. *Research in Transportation Business & Management*, 32:100390, 2019.

- [27] Shuo Han, Ufuk Topcu, and George J Pappas. Quantification on the efficiency gain of automated ridesharing services. In *2017 American Control Conference (ACC)*, pages 3560–3566. IEEE, 2017.
- [28] Hadi Hosni, Joe Naoum-Sawaya, and Hassan Artail. The shared-taxi problem: Formulation and solution methods. *Transportation Research Part B: Methodological*, 70:303–318, 2014.
- [29] Riccardo Iacobucci, Benjamin McLellan, and Tetsuo Tezuka. Optimization of shared autonomous electric vehicles operations with charge scheduling and vehicle-to-grid. *Transportation Research Part C: Emerging Technologies*, 100:34–52, 2019.
- [30] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. *arXiv preprint arXiv:2007.01409*, 2020.
- [31] Michal Koháni, Peter Czimmermann, Michal Váňa, Matej Cebecauer, and L’uboš Buzna. Location-scheduling optimization problem to design private charging infrastructure for electric vehicles. In *International Conference on Operations Research and Enterprise Systems*, pages 151–169. Springer, 2017.
- [32] Antoon WJ Kolen, Jan Karel Lenstra, Christos H Papadimitriou, and Frits CR Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.
- [33] Nicholas Kullman, Martin Cousineau, Justin Goodson, and Jorge Mendoza. Dynamic ridehailing with electric vehicles. 2020.
- [34] Antti Lajunen. Evaluation of energy consumption and carbon dioxide emissions for electric vehicles in nordic climate conditions. In *2018 Thirteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*, pages 1–7. IEEE, 2018.
- [35] Albert YS Lam, Yiu-Wing Leung, and Xiaowen Chu. Electric vehicle charging station placement: Formulation, complexity, and solutions. *IEEE Transactions on Smart Grid*, 5(6):2846–2856, 2014.

- [36] Ungki Lee, Namwoo Kang, and Ikjin Lee. Shared autonomous electric vehicle design and operations under uncertainties: a reliability-based design optimization approach. *Structural and Multidisciplinary Optimization*, pages 1–17, 2019.
- [37] Li Li, DianChao Lin, Theodoros Pantelidis, Joseph Chow, and Saif Eddin Jabari. An agent-based simulation for shared automated electric vehicles with vehicle relocation. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3308–3313. IEEE, 2019.
- [38] Jane Lin, Wei Zhou, and Ouri Wolfson. Electric vehicle routing problem. *Transportation Research Procedia*, 12:508–521, 2016.
- [39] Yeqian Lin, Wenquan Li, Feng Qiu, and He Xu. Research on optimization of vehicle routing problem for ride-sharing taxi. *Procedia-Social and Behavioral Sciences*, 43:494–502, 2012.
- [40] Yang Liu and Samitha Samaranayake. Proactive rebalancing and speed-up techniques for on-demand high capacity vehicle pooling. *arXiv preprint arXiv:1902.03374*, 2019.
- [41] Benjamin Loeb, Kara M Kockelman, and Jun Liu. Shared autonomous electric vehicle (SAEV) operations across the Austin, Texas network with charging infrastructure decisions. *Transportation Research Part C: Emerging Technologies*, 89:222–233, 2018.
- [42] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. Zac: A zone path construction approach for effective real-time ridesharing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 528–538, 2019.
- [43] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. Zone path construction (zac) based approaches for effective real-time ridesharing. *Journal of Artificial Intelligence Research*, 70:119–167, 2021.
- [44] Emma Lucken, Karen Trapenberg Frick, and Susan A. Shaheen. “three ps in a MOD:” role for mobility on demand (MOD) public-private partnerships in

- public transit provision. *Research in Transportation Business & Management*, 32:100433, 2019.
- [45] Shuo Ma, Yu Zheng, and Ouri Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 410–421. IEEE, 2013.
- [46] Shuo Ma, Yu Zheng, and Ouri Wolfson. Real-time city-scale taxi ridesharing. *IEEE Transactions on Knowledge and Data Engineering*, 27(7):1782–1795, 2014.
- [47] Madhav V Marathe, Ramamoorthi Ravi, Ravi Sundaram, SS Ravi, Daniel J Rosenkrantz, and Harry B Hunt III. Bicriteria network design problems. *Journal of algorithms*, 28(1):142–171, 1998.
- [48] J Carlos Martínez Mori and Samitha Samaranyake. On the request-trip-vehicle assignment problem. *arXiv e-prints*, pages arXiv–2011, 2020.
- [49] Joel C Miller and Tony Ting. Eon (epidemics on networks): a fast, flexible python package for simulation, analytic approximation, and analysis of epidemics on networks. *reprint arXiv:2001.02436*, 2020.
- [50] P Mock, U Tietge, V Franco, J German, A Bandivadekar, NE Ligterink, U Lambrecht, J Kuhlwein, and I Riemersma. From laboratory to road. a 2014 update of official and real-world fuel consumption and co2 values for passenger cars in europe. Technical report, International Council on Clean Transportation Europe, 2014.
- [51] Evdokia Nikolova. Approximation algorithms for reliable stochastic combinatorial optimization. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 338–351. Springer, 2010.
- [52] Nuri Cihat Onat, Murat Kucukvar, and Omer Tatari. Conventional, hybrid, plug-in hybrid or electric vehicles? state-based comparative carbon and energy footprint analysis in the united states. *Applied Energy*, 150:36–49, 2015.
- [53] Masayo Ota, Huy Vo, Claudio Silva, and Juliana Freire. Stars: Simulating taxi ride sharing at scale. *IEEE Transactions on Big Data*, 3(3):349–361, 2016.

- [54] Samuel Pelletier, Ola Jabali, and Gilbert Laporte. Charge scheduling for electric freight vehicles. *Transportation Research Part B: Methodological*, 115:246–269, 2018.
- [55] Supun Perera, Chinh Ho, and David Hensher. *Resurgence of Demand Responsive Transit services – Insights from BRIDJ trials in Inner West of Sydney, Australia*, 2020. Accepted: 2020-04-08 ISSN: 1832-570X.
- [56] Jacob F Pettit, Ruben Glatt, Jonathan R Donadee, and Brenden K Petersen. Increasing performance of electric vehicles in ride-hailing services using deep reinforcement learning. *arXiv preprint arXiv:1912.03408*, 2019.
- [57] J Restrepo, J Rosero, and S Tellez. Performance testing of electric vehicles on operating conditions in bogotá dc, colombia. In *2014 IEEE PES Transmission & Distribution Conference and Exposition-Latin America (PES T&D-LA)*, pages 1–8. IEEE, 2014.
- [58] Connor Riley, Antoine Legrain, and Pascal Van Hentenryck. Column generation for real-time ride-sharing operations. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 472–487. Springer, 2019.
- [59] Caroline Rodier. The effects of ride hailing services on travel and associated greenhouse gas emissions. 2018.
- [60] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–13294, 2014.
- [61] Douglas Oliveira Santos and Eduardo Candido Xavier. Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.
- [62] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. Neural approximate dynamic programming for on-demand ride-pooling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 507–515, 2020.

- [63] Susan Shaheen and Nelson Chan. Mobility and the sharing economy: Potential to facilitate the first- and last-mile public transit connections. *Built Environment*, 42(4):573–588, 2016.
- [64] Susan Shaheen, Adam Cohen, Nelson Chan, and Apaar Bansal. Sharing strategies: carsharing, shared micromobility (bikesharing and scooter sharing), transportation network companies, microtransit, and other innovative mobility modes. In Elizabeth Deakin, editor, *Transportation, Land Use, and Environmental Planning*, pages 237–262. Elsevier, 2020.
- [65] Susan A. Shaheen, Adam Cohen, Balaji Yelchuru, and Sara Sarkhili. *Mobility on Demand Operational Concept Report*, 2017.
- [66] Ma Shaohan and WD Wallis. Maximal-clique partitions of interval graphs. *Journal of the Australian Mathematical Society*, 45(2):227–232, 1988.
- [67] Bilong Shen, Bo Cao, Ying Zhao, Haojia Zuo, Weimin Zheng, and Yan Huang. Roo: Route planning algorithm for ride sharing systems on large-scale road networks. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–8. IEEE, 2019.
- [68] Jie Shi, Yuanqi Gao, Wei Wang, Nanpeng Yu, and Petros A Ioannou. Operating electric vehicle fleet for ride-hailing services with reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2019.
- [69] Jie Shi, Yuanqi Gao, and Nanpeng Yu. Routing electric vehicle fleet for ride-sharing. In *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–6. IEEE, 2018.
- [70] Andrea Simonetto, Julien Monteil, and Claudio Gambella. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101:208–232, 2019.
- [71] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment*, 11(11):1633–1646, 2018.

- [72] JM Van den Akker, Cor AJ Hurkens, and Martin WP Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.
- [73] Sacha Varone and Vytenis Janilionis. Insertion heuristic for a dynamic dial-a-ride problem using geographical maps. In *MOSIM 2014, 10eme Conference Francophone de Modélisation, Optimisation et Simulation, Nancy, France*, 2014.
- [74] José Viegas, Luis Martinez, Philippe Crist, and Sharon Masterson. Shared mobility: Innovation for liveable cities. In *International Transport Forum’s Corporate Partnership Board*, pages 1–56, 2016.
- [75] Joel Volinski, Transit Cooperative Research Program, Transportation Research Board, and National Academies of Sciences, Engineering, and Medicine. *Microtransit or General Public Demand-Response Transit Services: State of the Practice*. Transportation Research Board, 2019. Pages: 25414.
- [76] Alex Wallar, Javier Alonso-Mora, and Daniela Rus. Optimizing vehicle distributions and fleet sizes for shared mobility-on-demand. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3853–3859. IEEE, 2019.
- [77] Alex Wallar, Menno Van Der Zee, Javier Alonso-Mora, and Daniela Rus. Vehicle rebalancing for mobility-on-demand systems with ride-sharing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4539–4546. IEEE, 2018.
- [78] Kim A Weeden and Ben Cornwell. The small-world network of college classes: implications for epidemic spread on a university campus. *Sociological Science*, 7:222–241, 2020.
- [79] Kim A Weeden and Ben Cornwell. The small world network of college classes: Implications for epidemic spread on a university campus, supplemental materials, 2020.
- [80] Marla Westervelt, Emma Huang, Joshua Schank, Nolan Borgman, Tamar Fuhrer, Colin Peppard, and Rani Narula-Woods. *UpRouted: Exploring Microtransit in the United States*. Eno Center for Transportation, 2018.

- [81] Chak Fai Yuen, Abhishek Pratap Singh, Sagar Goyal, Sayan Ranu, and Amitabha Bagchi. Beyond shortest paths: Route recommendations for ride-sharing. In *The World Wide Web Conference*, pages 2258–2269, 2019.

- [82] Matthew Zalesak, Andrea Broaddus, Melissa Ruhl, and Samitha Samaranayake. Exploring first-mile last-mile microtransit service design: A simulation case study from seattle. Technical report, 2020.

- [83] Hongcai Zhang, Colin JR Sheppard, Timothy E Lipman, Teng Zeng, and Scott J Moura. Charging infrastructure demands of shared-use autonomous electric vehicles in urban areas. *Transportation Research Part D: Transport and Environment*, 78:102210, 2020.