# SMARTER TOOLS FOR (CITI)BIKE SHARING

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

Eoin Daniel O'Mahony

August 2015

SMARTER TOOLS FOR (CITI)BIKE SHARING

Eoin Daniel O'Mahony, Ph.D.

Cornell University 2015

Bicycle-sharing systems provide a low-cost, environment-friendly urban transportation option. Efficient management of these systems poses a bicycle rebalancing problem comprising three questions: where do bikes need to be, when must they be there, and how can they get there? I apply operations research techniques to yield practical answers to these questions; my solutions optimize current operations at NYC (Citi)Bike.

## BIOGRAPHICAL SKETCH

Eoin Daniel O'Mahony grew up in Ireland where he completed a First Class Honors BSc. in Computer Science at University College Cork in 2010. Eoin then attended graduate school at Cornell University in Fall 2010.

This thesis is dedicated to my family, friends and Siobhan.

## ACKNOWLEDGEMENTS

First off I would like to thank my advisor Prof. David B. Shmoys for his advice and mentorship over this process. I would also like to thank the other members of my PhD committee, Prof. David Williamson and Prof. Carla Gomes for their insight and feedback. Throughout my PhD my classmates and friends have been an invaluable source of help and support, without them I would not have made it through my first few years at Cornell. My family have a constant source of support and encouragement over the years. I would also like to thank Vasu for her love and support on this journey. This research was supported in part by NSF grants CCF-0832782 and CCF-1017688.

## TABLE OF CONTENTS

## LIST OF FIGURES

CHAPTER 1

# INTRODUCTION

The way people move is changing. Smartphones, wearable technology and ubiquitous GPS tell us where we are with high accuracy. We already rely on this technology in our daily lives: if you are a smartphone user I challenge you to recall the last time you visited somewhere new without relying on your smartphone GPS. The sharing economy, wherein location-based services are exchanged between dispersed individuals and entities, has risen on the back of these technologies, and will continue to increase in importance as our cities grow. The sharing economy is changing how we use resources and particularly how we access transport; its success relies on access to real-time location and resource availability information. The availability of this information opens the door to whole categories of optimizations previously unthinkable. Operations Research must be at the forefront of optimization for this new economic paradigm. From ensuring there is somewhere to place your bike before a morning meeting, to computing a route to allow you to ride-share in a taxi, Operations Research has escaped the confines of the steel mill and emerged onto the streets. This work is an example of these new opportunities, where optimization, modeling and analytics are applied to improve the daily commutes of New Yorkers.

## 1.1 The Rise of Bike Sharing Systems

The rise of bike sharing is an example of this paradigm shift in transportation. A bike sharing system typically consists of stations located throughout a city, where bikes are stored. Subscribers to the system can take a bike from any station and return it to any other. An example of a user removing a bike from a station and an example of a station can be seen in Figure 1.1. The number of bike sharing systems has more than doubled


Figure 1.1: The image on the left shows a user using a key to remove a bike from a station, the image on the right is a bike share station located next to Grand Central New York.

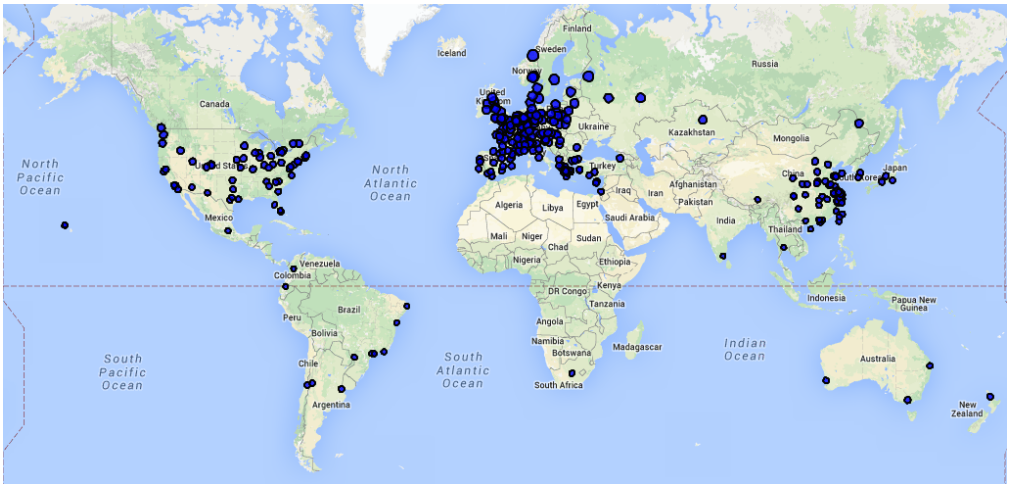


Figure 1.2: World map showing the locations of bike sharing systems.

since 2008, Larsen [16]. In 2008 there were 213 systems operating in 14 countries with 73,500 total bicycles; as of August 2014 there were over 500 systems with more than a half of a million bikes. A map showing the locations of current bike sharing systems is shown in Figure 1.2.

Bike sharing systems are a cost-effective way of promoting a sustainable lifestyle in urban areas. By offering a low-carbon alternative to commuters bike sharing promotes a healthy lifestyle. Studies of commuters show those who bike are happiest among other modes of transport [30]. Bike-share systems have also been found to have an overall

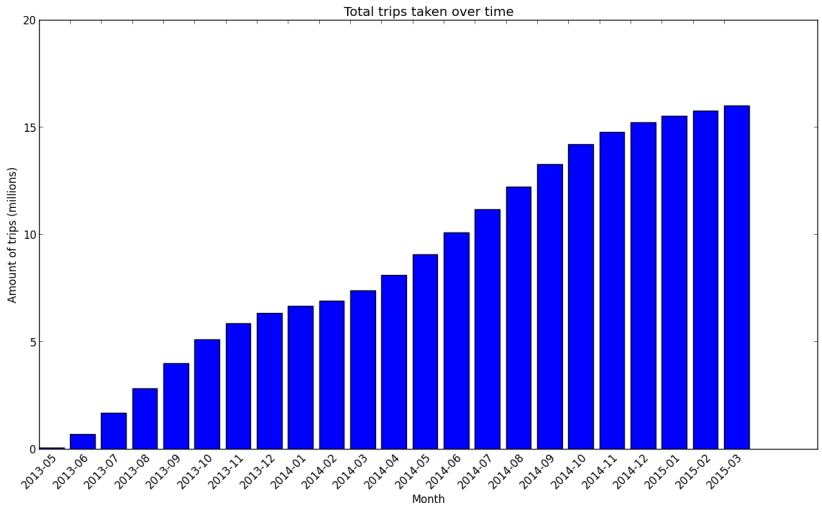


Figure 1.3: Total trips taken over time in the New York Citibike system.

positive health impact [34]. Increasing the number of cyclists improves road safety [13]; there has yet to be a recorded death in a US bike-share system in over 23 million rides [7].

New York City launched the largest bike-sharing system in North America, Citibike, in May 2013 with over 300 stations and 5000 bikes. The system has been a success with usage approaching 40000 trips per day, with a total of over 15 million trips. The total trips taken over time in New York are shown in Figure 1.3. The majority of trips taken by users of system occur during the morning and evening rush hours.

With the success of Citibike comes a set of management problems. Chief among these is the issue of system imbalance; bikes become clustered in certain geographic areas which leaves other areas devoid of bikes; for example, the traders wake up early in their East Village apartments, rapidly deplete the supply of bikes there, and then overwhelm the capacity for docks in the Wall Street area. Although the flow of bikes is largely symmetric across an entire day, the pernicious effect of commuters using Citibike to travel to work in the morning and to return home in the evening causes huge imbal-

ances during the day. Like all other bike-sharing systems, Citibike relocates bikes to maintain system balance. Typically this is achieved by using trucks to move between stations picking up and dropping off bikes. Trucks are used in New York but are also complemented by bike trailers; these are courier bikes with trailers attached that can move four or five Citibikes at a time and allow the relocation of bikes during periods of heavy traffic through the use of bike lanes. An example of one of these bike trailers can be seen in Figure 1.1 on the right.

## 1.2 Challenges in the Operations of Bike Sharing Systems

Rebalancing is one of the biggest challenges for the operators of bike sharing systems. It is expensive, resource intensive and difficult to implement successfully. Operators of bike sharing systems often have limited resources available to them, which constrains the extent to which rebalancing can occur. Hence, this domain is an exciting application for the field of computational sustainability. Based on a close collaboration with NYC Bike Share LLC, the operators of Citibike, we have formulated several optimization problems whose solutions are used to more effectively maintain the pool of bikes in NYC. There is an expanding literature on operations management issues related to bike sharing systems, which is covered in depth in Chapter 2. However, the problems addressed here are particularly suited to the complex blend of human and system constraints that are present for Citibike. For these problems, we shall present results that reflect a range of different approaches: stochastic modeling techniques to capture the uncertain nature of bike demand, theoretically-driven approximation algorithms that yield provably good solutions, integer programming formulations that can typically be solved (at scale) by off-the-shelf integer programming solvers, heuristic approaches that yield good solutions quickly as well as user-based rebalancing incentive schemes.

### 1.2.1 Planning Through Data

As mentioned earlier the majority of trips taken on Citibike occur during the morning and evening rush hours. We begin by tackling the problem of how best to use system data to plan for usage. That is, we want to place bikes at stations to handle the surge in usage experienced during rush-hours. This process is referred to as *prebalancing* by the operators of Citibike. Often due to issues of repair and theft, the number of bikes available to operators falls below the ideal level. When these conditions occur it is important that the available bikes are placed where they are needed most. To solve this problem we have built a series of models based on observed trip data that allow us to find the best placement of bikes before heavy rush-hour usage.

We use a number of approaches to tackle this problem; in Chapter 3 we explain in depth the series of models used as well as their relative strengths and weaknesses. Initially we used a model that clusters stations with similar usage patterns and we then hand label the clusters with appropriate fill levels. We break the stations up into those that accumulate bikes, those that lose bikes and stations that are self-balancing. We discuss the limitations of this approach and instead propose a method that infers true demand for trips and uses these amounts to better plan for system usage. Finally, we present a model that takes both demand for bikes and spots as well as the stochastic nature of usage into account. This is based on representing the station level over time as a continuous-time Markov chain and using this to base decisions of how many bikes are needed at each station. We also tackle the more basic planning problem of how many docks should be placed at stations to best facilitate usage; this is solved by a combination of continuous-time Markov chain modeling and integer programs. From these computations we know both when and where bikes are needed and progress to solving the problem of how do we get them there?

### 1.2.2 Optimizing Rebalancing

Moving bicycles around a city is expensive. The cost per bike moved can climb well over a dollar. As a consequence, operators of bike-sharing systems have limited resources available to them to carry out rebalancing. This shortage of available resources means it is crucial to use them as efficiently as possible. We propose a series of optimization models as well as an incentive scheme to rebalance the system. These are based on a combination of theoretical results as well as empirical work and result in a suite of tools to help direct rebalancing resources. These are covered in detail in Chapter 4.

We focus on two very different rebalancing problems. First, we tackle rebalancing the system during rush hour, developing novel methods for optimizing rebalancing resources. During rush hour, system usage is very high, rendering large truck routes unreliable since the system state may shift dramatically before the route is completed. Traffic is also at its peak during rush hour, which greatly limits the ability of trucks to move easily through the city; this motivates the use of bike trailers instead of large trucks. The nature of the mid-rush balancing requires a drastically different approach than the one used for the overnight problem. Our goal is not to maintain system balance but to ensure that users are never too far from either a bike or a dock. To achieve this, we formulate a clustering problem that yields stations in the city for which rebalancing resources can be targeted. We provide an IP formulation for this problem, from which we prove an approximation algorithm. This IP can also be solved by commercial solvers.

We then tackle the problem of moving bikes around the city overnight. Overnight the system experiences low usage and as a consequence stockage levels are relatively constant. Traffic is also at its lowest during these hours, resulting in trip times being both reliable and short. This allows a rebalancing plan spanning the overnight shift to be computed and executed without fear of users making it redundant or counterproduc-

tive. We formulate an optimization problem whose goal is to produce a series of truck routes to get the system as balanced as possible during the overnight shift. We provide both theoretical results as well as an empirical approach to this problem based on a (relatively) tractable integer programming formulation. The combination of these two approaches yields a fast method of obtaining high quality solutions that can be used in practice.

Finally, we propose an incentive scheme to complement rebalancing. This scheme is designed to provide incentives for users to shift their behavior to benefit the system as a whole. If you were to consider the demand for bikes and spots to be a topographical map with hills and valleys the aim of this scheme is to make the height of the hills less and the depth of the valleys less. It is based on a raffle scheme that has been shown in the work of [19] to be more effective than micro incentive based schemes. We design a way for users to obtain raffle tickets that are then used in raffles for cash (or prizes).

### 1.2.3 Discrete Event Simulation Validation

Evaluating the impact of interventions in a bike-sharing system can be challenging. Due to service level requirements imposed on operators by the cities and the risk of angering customers performing A-B testing where one option is to perform no rebalancing is a challenge. Also, operators would like too have more information on whether a proposed rebalancing approach may be effective before committing the resources to put it in place in the city.

To remedy this issue in Chapter 5 we present a discrete-event simulation framework. This framework can be used to test a range of modifications to the system from rebalancing approaches to the efficacy of other system interventions. We use this framework

to evaluate the bike placement and rebalancing actions we propose.

## 1.3 Contributions

The contributions of this work are as follows. We provide novel models for the optimization of bike rebalancing resources. These take the form of optimization models for the placement of bikes to prepare for rush-hour usage including a new approach to system planning and dock allocation. We develop a new approach for optimizing rebalancing resources during the busy rush hours which is dramatically different to all previous work. We provide efficient algorithms for this problem with provable quality guarantees. We also take a new approach to rebalancing a bike-share system overnight. We formulate an optimization problem, prove properties of the problem as well as developing empirical solution methods. As well as providing novel optimization models for rebalancing we present the first raffle-based incentive scheme for a bike-sharing system. These approaches are validated using a discrete-event simulator developed to act as a testbed for system intervention in New York.

CHAPTER 2

# RELATED WORK

Due to the increasing importance of bike-sharing programs, and the operational difficulties in managing them, a great deal of attention has been focused on a variety of problems that relate to bike-sharing. The majority of this work has focused on rebalancing bike-sharing systems. This work has applied techniques from integer programming, local search and other fields of combinatorial optimization to find routes specifying where trucks travel and how many bikes to move between stations. Another line of investigation taken by papers has been to analyze the usage data produced by bike-sharing systems to find patterns and different types of behavior. The insights gleaned from this analysis is then used to create stochastic models and other tools to help the operators of these systems.

## 2.1 Bike Share Rebalancing

Work focusing on (overnight) rebalancing includes [23, 24, 5, 26, 4, 27]. Raviv and Kolka [23] develops a model for determining the allocation of bikes across stations in preparation for a rush period, and is perhaps the closest work to ours. Raviv, Tzur and Forma [24] tackles the problem of finding truck routes and plans for how many bikes to move between stations. The paper minimizes an objective function tied to both the operating cost of the vehicles as well as penalty functions relating to station imbalance. The models are benchmarked on instances from both the Paris and Washington DC bike-sharing systems. Schuijbroek et al. [26] combines both finding service level requirements for stations with planning truck routes to keep stations rebalanced. They use a clustering heuristic for routing on data from Boston and Washington DC to produce truck routes. Rainer-Harbach et al. [22] use local search to find both routes for

trucks and the number of bikes to be collected or dropped off at each station. Contardo et al. [5] identify that a different rebalancing approach needs to be taken during rush hours. They formulate flow problems on space-time networks. Solutions are generated using a combination of Dantzig-Wolfe decomposition and the fast generation of upper and lower bounds. Chemla et al. [4] solves the static rebalancing problem, where a plan of where to move bikes is created. They provide a branch-and-cut algorithm for a problem realization and a tabu search to find heuristic solutions. Shu et al. [27] uses a time-flow formulation combined with stochastic modeling for rebalancing. These papers tackle rebalancing in a way that is similar to traditional inventory management and package routing problems, e.g., [1, 2, 10]. Perhaps the closet of these works to the approach taken in this thesis is the work of Schuijbroek, Hampshire and Van Hoeve [26] and, [23].

In the work of Schuijbroek etal. the authors firstly explore the correct numbers of bikes to be placed at stations. They produce *service levels* for each station that is then used to inform rebalancing operations. The service levels are effectively a minimum and maximum number of bikes to be placed at each station. They are chosen as to ensure the ratios of successful trips started and ended to the demand to start and end trips are above certain thresholds. The service levels for station $i$ are $s_i^{min}$ and $s_i^{max}$ where

$$s_i^{min} = \min\left[s \in \{0, \cdots, C\} : \frac{E[\text{Successful pickups given } s \text{ bikes initially}]}{E[\text{Total pickups}]} \geq \beta_i^-\right] \tag{2.1}$$

$$s_i^{max} = \min\left[s \in \{0, \cdots, C\} : \frac{E[\text{Successful drop-offs given } s \text{ bikes initially}]}{E[\text{Total drop-offs}]} \geq \beta_i^+\right] \tag{2.2}$$

$$\tag{2.3}$$

and $\beta_i^-, \beta_i^+ \in [0, 1]$ are constants fixed for each station. These ratios are calculated using an $M/M/1/K$ queueing model, where the flow of bikes in and out of stations is

assumed to be a Poisson process. This approach is similar to that employed in Chapter 3 where we use a similar model to decide where to place bikes. However, our approach moves beyond providing a binary score for each station (in the above case whether the station meets the requirements or not) and instead provides a measure of the *quality* of each possible fill level for the station. Our approach also avoids the problem of finding constants $\beta_i^-, \beta_i^+ \in [0, 1]$ that are feasible for each station, although the authors note that in data used from the Boston bike-sharing system, they did not run into problems of infeasability. However, $s_i^{min} > s_i^{max}$, in cities with much higher usage like New York, where this could easily become a problem. Even if the flow rates are such that it is possible to find service levels for each station, the realities of bike-sharing operations are such that often bikes are a scarce resource. Often in these systems, as a consequence of bikes needing repair and bike theft, the number of bikes available to place on the street is far from optimal.

Using the minimum and maximum fill levels for each station, Schuijbroek etal. then find rebalancing routes for trucks that will move bikes to have all stations within the required bounds. They use mixed integer programming models as well as some approximation algorithms to find these routes. Their mixed integer programming model is a time-indexed model where the binary decision variable $X_{i,j,t,v}$ represents whether truck $v$ moves from station $i$ to station $j$ at time $t$, and a corresponding decision variables $Y_{i,t,v}^+$ and $Y_{i,t,v}^-$ representing the number of bikes picked up and dropped off at station $i$ by truck $v$ at time $t$. The authors note that this formulation quickly becomes intractable for more than two trucks and more than fifty stations. To allow scaling to real-world problem instances they present Clustered Routing heuristics with the aim of finding high quality solutions in short amounts of time. The goal of clustered routing is to break the problem into a series of one-vehicle routing problems, which are far easier to solve in practice. The authors break the stations into a series of clusters where by just routing within a

cluster it is possible to meet the service levels. The heuristic first generates clusters using an approximate cost of intra-cluster routing as a guide. Once these clusters are generated single-vehicle routing problems are solved for each cluster to find the actual truck routes. Both mixed integer programming and constraint programming models are used to solve the routing problems; these perform well with a large number of stations but a small numbers of trucks.

The objective function in both of these models is to minimize the makespan, specifically the amount of time until all stations are within the required service levels. Again, this is different from our rebalancing optimization presented in Chapter 4, where we are in a setting which is impossible to fully rebalance a system with the resources available and instead we aim to rebalance as much as possible. From our close collaboration with the operators of Citibike, we feel this is a more realistic objective function as staff have a fixed overnight shift and rarely are there enough trucks or staff available to fully rebalance the system. The models presented in [26] are tested using data from Capital Bikeshare, Washington DC and Hubway, Boston. In the experiments, typically two or three trucks are used, with only one experiment using five trucks (a typical number for a large system).

The work of Raviv and Kolka [23] also uses continuous-time Markov chains to represent station behavior over time. They focus on the inventory management aspect of the problem, specifically where to locate bikes in the system. This problem is an interesting variant of traditional inventory management problems as placing inventory at a location may harm the overall state of the system since by placing bikes empty racks are taken. In contrast to the work of Schuijbroek etal. and our work they use non-stationary Poisson processes to model station behavior. The rates for stations are dependent on time; this better captures the shift in usage throughout the day. However with this flexibility

in modeling there is a trade off in the ease of computing the transition matrix of the Markov chain; this requires the authors to approximate these amounts. In contrast we are able to quickly compute the transition matrix of the Markov chain. The authors also prove a similar result on the convexity of this Markov chain based cost function, $f(s)$, to that presented in Chapter 3.

The follow up work of Raviv, Tzur and Forma [24] uses this cost function to direct vehicle routing. They produce routes that are a schedule of pick-ups and drop-offs at stations. The objective function for this model is a combination of minimizing the station cost function but also the time taken to rebalance.

$$\min \sum_{i \in N} f_i(s_i) + \alpha \sum_{i \in N} \sum_{j \in N} \sum_{v \in V} t_{ij} x_{ijv} \tag{2.4}$$

where $N$ is the set of stations, $V$ the set of available vehicles, $x_{ijv}$ the binary variable representing whether vehicle $v$ moved between stations $i$ and $j$, $s_i$ the final state of station $i$ and $t_{ij}$ the time cost of moving between stations $i$ and $j$. The objective function is linearized in a manner similar to that used in Chapter 3. The authors then present a time-indexed model where time is discretized; this approach removes some restrictions of the original model and allows stations to-be visited multiple times. These models are solved to optimality using heuristic methods as well as problem size reduction techniques. To help solve the large mixed integer programs quickly some integrality constraints are dropped; the resulting problem is still a mixed integer, but the authors claim it is easier to solve. The integral solution produced is then fixed and the original integrality constraints are added back in. Also employed to speed up the computation of solutions is some problem-size reduction. Specifically, the graph between stations is sparsified by taking the metric closure of the graph and removing edges that pass through another station location. These models are tested on data from Velib in Paris and Capital Bikeshare in Washington DC. However, the authors restrict themselves to a subset of the Paris sta-

tions. In experimental analysis, the number of trucks is at most two and the number of stations thirty, sixty or the 104 stations of Capital Bikeshare.

## 2.2 Approximation Algorithms

In our work we approach the problem in a manner closer to orienteering problems [31, 3]. Our work handles full-size instances, both in terms of the number of stations and the number of trucks. (Much of the previous work has focused on instances for which two or three trucks are available.) Furthermore, our work builds on an understanding of the practicalities of running Citibike. For example, it is much simpler operationally to have trucks go to an overloaded station, fill the truck with bikes, and then deposit all of them at a (sufficiently) depleted station.

Our work for mid-rush rebalancing is closely related to existing work on bottleneck optimization, specifically the $k$-center problem. Hochbaum and Shmoys [11] provide a 2-approximation algorithm for this problem and show that no better approximation is possible unless $P = NP$. Furthermore, Hochbaum and Shmoys [12] provide a framework for tackling bottleneck optimization problems, providing lower bounds and approximation algorithms for variants of the $k$-center problem. Our work also uses clustering approaches similar to that of Khuller [15].

## 2.3 Data Analysis and bike-sharing

Although not immediately relevant to our contributions, other work on bike-sharing includes Nair, Miller-Hooks, Hampshire, and Bušić [20], Vogel and Mattfeld [32] and

Kaltenbrunner, Meza, Grivolla, Codina and Banchs [14, 17]. This work focuses on modeling how users will impact the system, detecting usage patterns from behavior. This insight into usage patterns is used to create stochastic models representing system usage. Vogel and Mattfeld [32] classifies stations based on their usage patterns, identifying stations used by commuters, tourists, etc. Other work of Garca-Palomares, Gutirrez and Latorre [6], Martinez, Caetano, Eir and Cruz [18] and Romero, Ibeas, Moura, Benavente and Alonso [25] aims to optimize the placement of stations in bike-sharing systems.

## 2.4 The Design of Incentive Schemes

The work of Merugu, Prabhakar and Rama [19] used an incentive scheme to attempt to have people shift their commute time to reduce road congestion. They worked with Infosys Technologies, Bangalore to incentivize some of the 14,000 commuters there to change the time of their commute. They use a raffle-based incentive similar to that we propose in Chapter 4. This incentivizes people with a large random reward as opposed to a smaller guaranteed amount. This system experienced success in altering peoples commuting patterns and thus reducing road congestion.

Singla, Santoni, Bartok, Mukerji, Meenen and Krause [29] present what we believe is the first implemented work on incentive schemes in bike-sharing systems. They create an incentive scheme that rewards users to alter their behavior with micro-payments. A user enters the trip they want to take and the system offers a nearby alternative and a price they are prepared to pay the user to change their trip. Machine learning and mechanism design feature in the incentive scheme to learn the optimal price to offers users for rebalancing. The system was implemented in Mainz, Germany where users

collect rewards for rebalancing the system. They also tested the system on data from Boston.

## 2.5 Different Approaches Taken

Our work provides fundamentally different models for bike rebalancing compared to previous approaches: for near-term recommendations for mid-rush-hour rebalancing, we employ a covering-problem viewpoint, closely tied to the very small number of pairs of stations that can be rebalanced by bike trailers; for overnight rebalancing in the near term we focus on full truck-load routes that give rise to a problem of covering a bipartite graph with sufficiently short alternating paths. We also explore models that are built from continuous-time Markov chain modeling and Poisson process modeling together with coupling theory to establish structural results that open the door to efficient large-scale optimization. All of our modeling is driven by observations and experience obtained through our collaboration with Citibike.

CHAPTER 3

# WHERE DO BIKES NEED TO BE?

Our earliest goal for the collaboration with NYC Bike Share LLC was to make their planning and decision making *data driven*. Specifically, it is crucial to use the data available to understand how the system is being used and where usage is putting stress on the system. The first problem we tackled was the problem of rush-hour planning. Weekday rush-hours (6am-10am and 4pm-8pm) account for the majority of bike trips taken in New York. Although mostly symmetric over the course of a day, each rush-hour period in isolation is highly asymmetric. In fact, we observe many extremes of behavior with some areas of the city having a large outflow of bikes and other areas having a large inflow of bikes during a given rush-hour.

The initial plan for the system was for all stations to be 50% full at the start of each rush-hour. We quickly realized that this is both an unachievable and undesirable goal. It is unachievable as the volume of bikes required to maintain these levels requires far more re-balancing resources than are available. It is undesirable as stations that will see a large outflow are, in a sense, wasting half of their docks and in the reverse, stations that will fill up have half their spaces taken up by bikes that will not be used during the rush-hour. To remedy this, we aimed to answer the following question. Suppose we could click out fingers and have the system in any state before the morning and evening rush hours, what would that state be? To answer this question we used a number of methods, each of which raises deeper questions about the analysis of bike-share usage.

## 3.1 Planning for Rush-Hour Usage: An Initial Approach

To plan for a rush hour we need to know where we expect bikes to be taken from and areas where we expect they will accumulate. We also need to identify stations that are *self-balancing*, specifically their flow of bikes in is roughly equal to their flow of bikes out thus requiring no rebalancing actions. Out first approach to discover the ideal system state before the morning and evening rush-hours relied on clustering stations based on their observed usage. The intuition is that stations that experience similar behavior during rush-hours will belong to the same cluster. We can then analyze the type of behavior typical of each cluster and label the cluster with a desired fill level.

To attempt to identify classes of stations we first represent each station by the curve showing the average net flow of bikes throughout a given time period. For example, when analyzing the morning rush hour for a given station we may compute the average flow of bikes for each minute over a month of weekdays. Taking this average we then compute the net flow of bikes for each minute, the value at minute $t$, $acc_t = acc_{t-1} + avg_t$. We use the net flow as opposed to the vector of bikes in or out over time to ensure that stations that end up lacking in bikes will be similar even if they have heavy load at different times during the rush-hour. In essence, the norm of two vectors ranging over time might not be a good reflection of similarity and by taking the net we smooth out these vectors to some degree.

We then cluster stations based on these curves. Using a *k-means* clustering algorithm [9] three distinct clusters emerge from the stations; stations that accumulate bikes, stations that lose bikes and stations that stay close to zero net flow; these can be seen in Figure 3.2. Analysis of membership within these clusters yields a labeling for each cluster. The clusters correspond to stations that *produce* bikes, stations that *consume* bikes

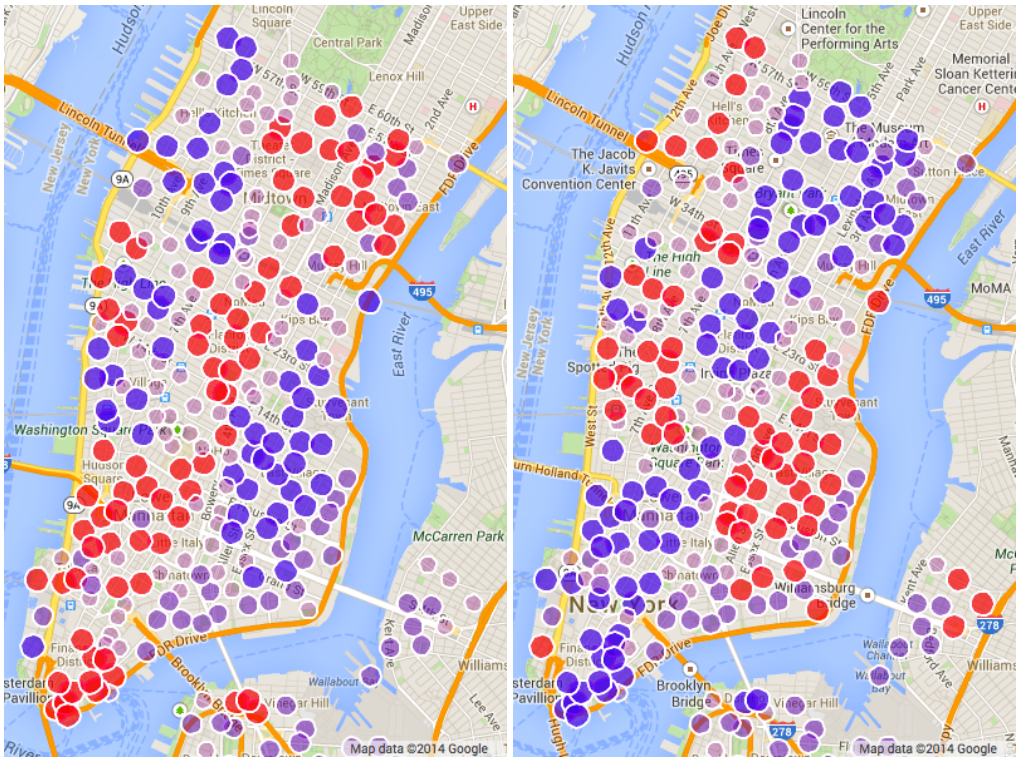


Figure 3.1: Levels assigned based on cluster membership for morning (left) and evening (right). Blue corresponds to consumers, red to producers and purple to a self-balancing.

and stations that are *self-balancing*. The result of this clustering approach on real data from the New York system is shown in Figure 3.1. In this figure, the clusters from analyzing both the morning and evening rush hours are shown, these are heavily correlated to residential versus business districts.

The results of this clustering approach are used to inform the operators of Citibike where bikes need to be placed to anticipate user demand. Accordingly, we assign desired fill levels to each of the clusters; this is the number of bikes we would want to have at that station before the rush hour period in an ideal world. These levels are 90%, 50% and 10% for consumer, self-balancing and producer stations accordingly. To allow more bikes to be placed at particularly heavy usage stations, the self-balancing stations were split into two groups, ones with high usage and ones with low usage. The low usage

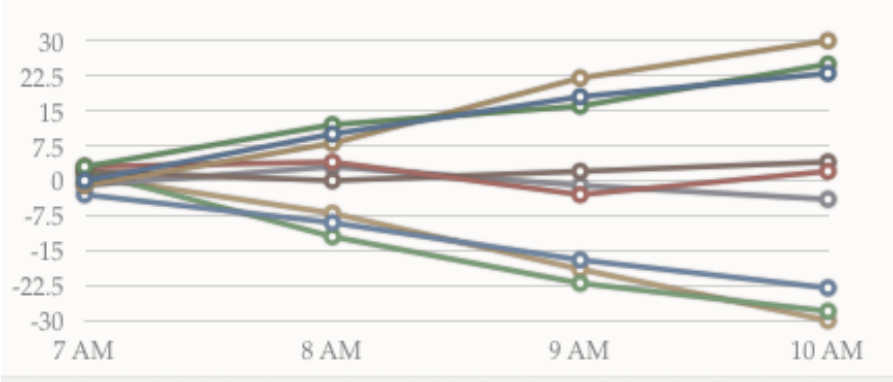


Figure 3.2: Figure showing the three clusters that emerge from a $k - means$ clustering on the net bike accumulation curve for each station. Stations that accumulate and lose bikes can be see as well as those that stay balanced.

self-balancing stations were assigned a fill level of 30%.

This approach has been highly successful; by using the fill levels generated by this computation the operations team at NYC Bike Share have been able to tailor their rebalancing operations to best serve the heavy rush-hour usage. However, although successful, this method has a number of limitations. Specifically, there is a weakness related to exploitation versus exploration. This method is slow to react to shifts in usage; to illustrate this we will consider an example that happened at a station in Brooklyn. The station was initially self-balancing with low usage; thus it was assigned a 30% fill level. However, over time the behavior of this station shifted to become a consumer of bikes. This change was not reflected when the computation was run every month as there were never enough bikes placed there for the usage necessary to change the cluster this station was getting assigned to. This station was identified by chance and led us to believe that there may be others in a similar situation. This caused us to consider the question, assuming we could keep each station stocked with bikes and spaces, what would a typical day's usage look like? This question is, in essence, what is the true demand for bikes in the system?

## 3.2 True Demand for Bikes

Although system data gives us information on each ride taken by users in New York, these data may not be a true reflection of the actual demand for a system. Consider a bike-share station at Penn Station in the evening rush hour. A huge number of commuters want to return bikes and take a train from the station. However, if we were to not rebalance this station it would quickly fill up and we may observe a fraction of the trips that could have happened. This motivates the computation of the underlying demand. Knowing the true demand for bikes and docks in the system allows us to plan more effectively for usage.

Observed trip data differs from the true demand due to censoring; that is, stations that are empty or full preventing users from taking from or returning bikes to the station. For many days we may observe zero trips for a time period but perhaps this is related to the station being empty/full. These outage windows are highly consistent as the morning and evening rush-hour behavior patterns are similar from day to day, meaning that the same stations are empty at the same time almost every day. However due to rebalancing operations we have days where we managed to replenish these stations with bikes or remove excess bikes at certain intervals.

$$O = \begin{bmatrix} 0 & 1 & 10 & 12 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 14 & 8 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 9 & 11 & 0 & 10 & 0 & 0 & 0 \\ 0 & 1 & 10 & 12 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 9 & 6 & 0 & 0 & 1 \\ 1 & 1 & 7 & 12 & 0 & 12 & 2 & 0 & 0 \\ 0 & 0 & 9 & 12 & 2 & 15 & 0 & 1 & 0 \\ 0 & 2 & 11 & 12 & 2 & 9 & 12 & 0 & 0 \end{bmatrix}, L = \begin{bmatrix} 23 & 22 & 12 & 0 & 0 & 0 & 20 & 20 & 20 \\ 25 & 22 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 22 & 20 & 11 & 0 & 10 & 0 & 0 & 19 & 19 \\ 23 & 22 & 12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 8 & 0 & 16 & 6 & 10 & 1 & 1 & 0 \\ 20 & 19 & 12 & 0 & 14 & 2 & 0 & 0 & 0 \\ 0 & 23 & 14 & 2 & 16 & 1 & 1 & 0 & 0 \\ 30 & 28 & 17 & 5 & 25 & 16 & 4 & 4 & 4 \end{bmatrix}$$

Our aim is to rely on rebalancing operations having had sufficient impact to give us data for most stations. Consider a matrix of observations, $O$, for bikes taken out for a specific station. Each row represents a day and each column represents some time period and an entry is the number of bikes taken out from that station. We also have a level matrix,

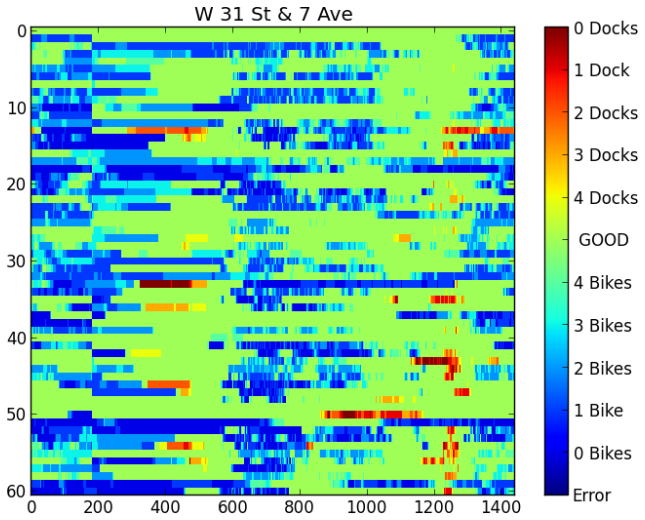


Figure 3.3: Example mask matrix for a Penn Station bike-share station over 60 days.

$L$, which for the same time period represents the number of bikes in the station at the end of the period. Our goal is to use the average number of trips for each time window as a representation of the true demand. However we need to take censoring of demand into account; to do this we observe the levels at the station. We mask the observed trip matrices, removing elements where the corresponding level element is at zero. This, in essence, ensures that zeroes that occur in the observed matrix are actual zeroes and not just zeroes due to the station state. Below you can see the masking operation using the above two matrices.

$$O = \begin{bmatrix} 0 & 1 & 10 & 12 & \boxed{0} & \boxed{0} & \boxed{0} & 0 & 0 \\ 0 & 3 & 14 & 8 & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} \\ 1 & 2 & 9 & 11 & \boxed{0} & 10 & \boxed{0} & 0 & 0 \\ 0 & 1 & 10 & 12 & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} \\ 0 & 0 & 8 & \boxed{0} & 9 & 6 & 0 & 0 & 1 \\ 1 & 1 & 7 & 12 & \boxed{0} & 12 & 2 & \boxed{0} & \boxed{0} \\ \boxed{0} & 0 & 9 & 12 & 2 & 15 & 0 & 1 & \boxed{0} \\ 0 & 2 & 11 & 12 & 2 & 9 & 12 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 10 & 12 & & & & 0 & 0 \\ 0 & 3 & 14 & 8 & & & & & \\ 1 & 2 & 9 & 11 & & 10 & & 0 & 0 \\ 0 & 1 & 10 & 12 & & & & & \\ 0 & 0 & 8 & & 9 & 6 & 0 & 0 & 1 \\ 1 & 1 & 7 & 12 & & 12 & 2 & & \\ & 0 & 9 & 12 & 2 & 15 & 0 & 1 & \\ 0 & 2 & 11 & 12 & 2 & 9 & 12 & 0 & 0 \end{bmatrix}$$

In practice, due to broken bikes and broken docking points we use a soft outage

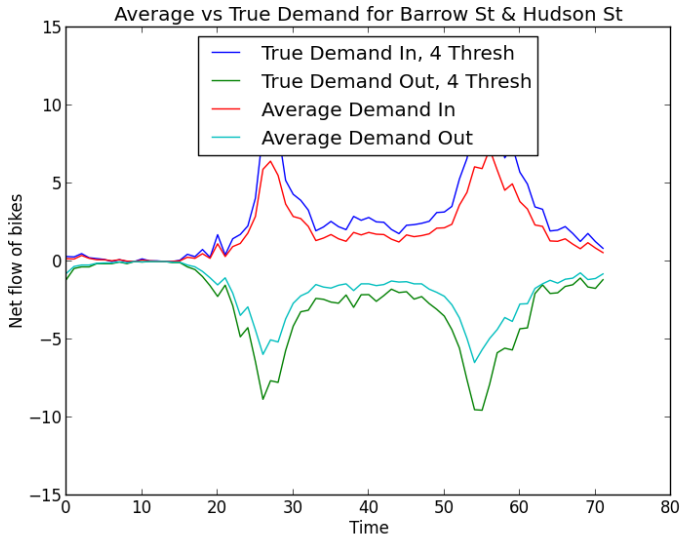


Figure 3.4: Example true demand for a West Village station.

number; that is, we consider a station out if the number of bikes or docks drops below a given threshold. In practice, this threshold is between two and five. An example of such a mask matrix for one of the Penn Station locations is shown in Figure 3.3. To pick the correct threshold level to use for the true demand computation, we searched over a number of possibilities and picked the levels that yielded the largest true demand trip numbers. This level was to consider anything less than 4 bikes or docks as an outage.

Another consideration is the many external factors that impact the usage of the system, particularly weather. Rain or snow can cause a large reduction in ridership; when computing the true demand it is important to filter out days where we believe external factors reduced the ridership. This prevents us from seeing low trip numbers and low outages for days where there was little actual demand for bikes. Taking all of this into account, the true demand net flow curves for a West Village station is shown in Figure 3.4. From this Figure we can see that the true demand is larger in both magnitude and duration throughout the rush-hours.

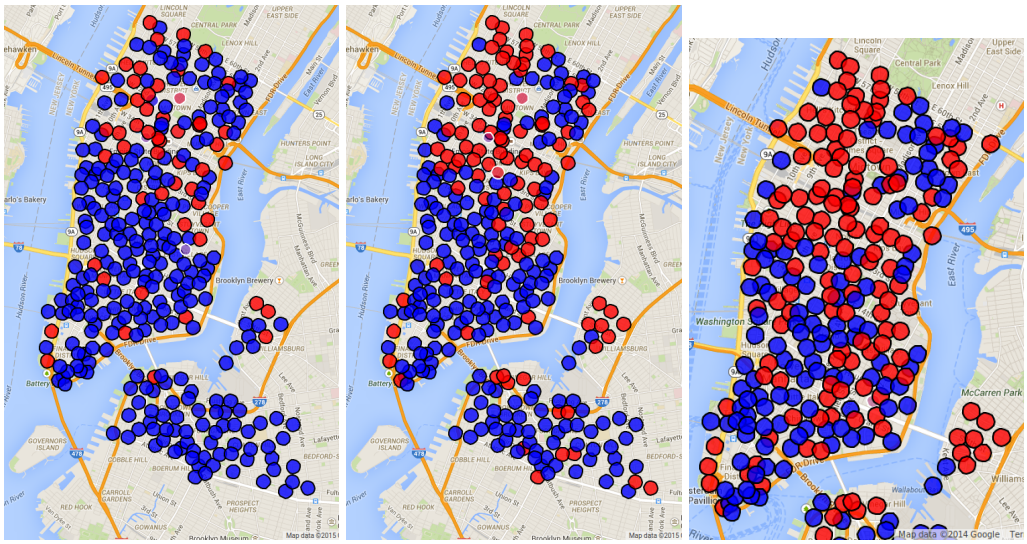

Figure 3.5: Morning levels assigned based on cost function that minimizes the maximum gap. This is from left to right, two thousand bikes, four thousand bikes and six thousand bikes in the system.

## 3.3 Optimizing Resource Allocation

The decision of how many bikes to place at a station as well as how large a station should be, are crucial decisions in the management of a bike sharing system. Our goal is to use data on ridership to evaluate the quality of these decisions. We will use a number of different objective functions to best capture where bikes are needed. In contrast to the first approach mentioned, where fill levels are assigned via labeling clusters based on usage patterns, in this approach we optimize with a given budget of bikes. Although the static fill levels may work well, often due to theft, repairs or other factors, the fleet of bicycles available varies. We need to ensure that if bikes become a scarce resource we assign them where they are most needed.

Under the assumption that the true demand curves we have computed are a good proxy for the underlying user demand, optimization of management decisions for the system becomes possible. Previously, we had clustered stations based on their net flow

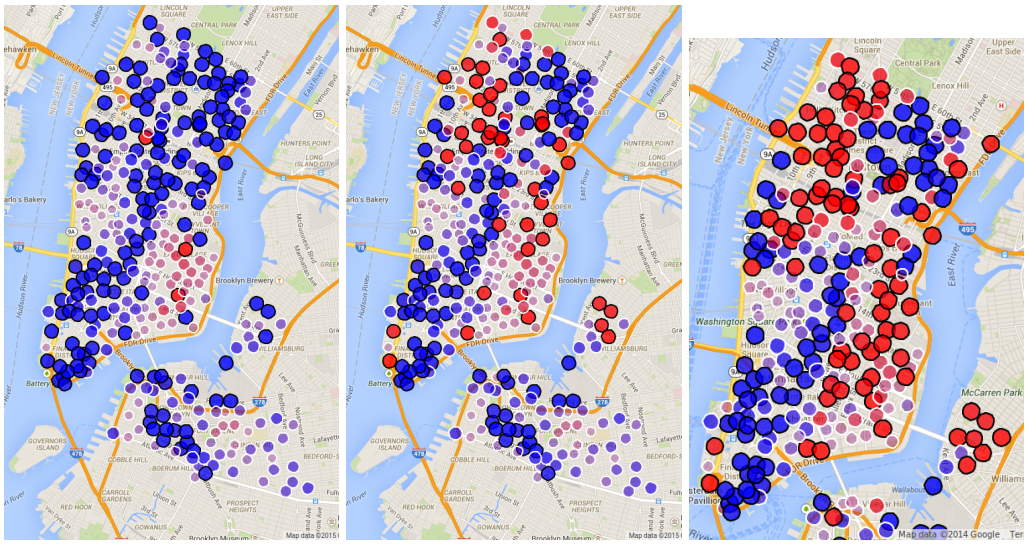

Figure 3.6: Morning levels assigned based on cost functions that is minimizing the sum of the gaps. This is from left to right, two thousand bikes, four thousand bikes and six thousand bikes in the system.

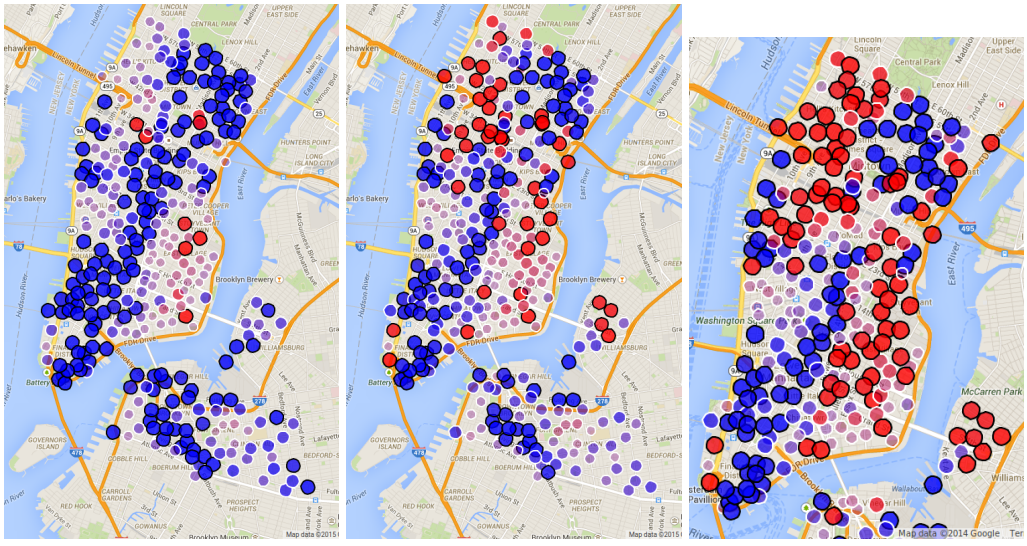

Figure 3.7: Morning levels assigned based on cost function that is minimizing the sum of the gaps squared. This is from left to right, two thousand bikes, four thousand bikes and six thousand bikes in the system.

of bikes and manually assigned levels to them. However, we can frame the question of how many bikes to place at a station before the rush hour as an optimization problem. Given, for each station, the expected number of bikes in and out of the station for each minute of the rush hour, the station capacities, the number of bikes that can be deployed into the system and a cost function, $J$, mapping the initial level at stations to a value related to missed bike rides we solve the following optimization problem where $X_s$ is the number of bikes to be placed at station $s$ at the beginning of a rush-hour, $c(s)$ is the capacity of station $s$ and $B$ the available number of bikes.

$$\begin{aligned} & & \min J(X) \\ s.t. & & \sum_s X_s \leq B \\ \forall s & & 0 \leq X_s \leq c(s) \end{aligned}$$

The structure of $J(X)$, the cost function, impacts the quality of the solution to the problem. There is a tradeoff between the ease of optimizing $J(X)$ and the expressive power of the function. One initial candidate is to compute the net flow from the true demand for each station and look at the smallest and largest values for this curve over the rush hour. These values are the maximum imbalance the flow for the station will create. We can then penalize a station's level for being too far under or over these values.

How we penalize stations for being too far under or over these values can take a number of forms. Given for each station the maximum over imbalance $o_s$ and maximum under imbalance $u_s$ throughout the rush hour we can construct a series of optimization models. Initially, we penalize the maximum imbalance that is not served by bikes or spots being at the station. For convenience, we let $R_s$ denote the number of spots left

empty at station $s$:

$$\begin{aligned} & & \min C \\ s.t. & & \sum_s X_s \leq B \\ \forall s & & C \geq C_s \\ \forall s & & C_s \geq o_s - R_s \\ \forall s & & C_s \geq u_s - X_s \\ \forall s & & 0 \leq X_s + R_s = c(s) \end{aligned}$$

The resulting fill levels of the above optimization for two, four and six thousand available bikes, $B$, are shown in Figure 3.5. Alternate objective functions are also possible. For example, minimizing the sum of the differences

$$\min \sum_s C_s,$$

the resulting fill levels from this objective function are shown in Figure 3.6. This is a much more desirable solution where we have a more balanced fill level across the system. Finally we tested minimizing the sum of the errors squared

$$\min \sum_s C_s^2$$

to ensure that we were not focusing on some stations at the expense of others. The results from this objective function are shown in Figure 3.7. Although these levels match operator intuition gained from observing the system, these optimizations are focused on net flow of bikes throughout a rush-hour.

However, solely taking the largest imbalances of the net flow curves does not capture all patterns of usage. Consider a station that experiences very high usage but has balanced in and out flow. This station will have a very small imbalance but may require

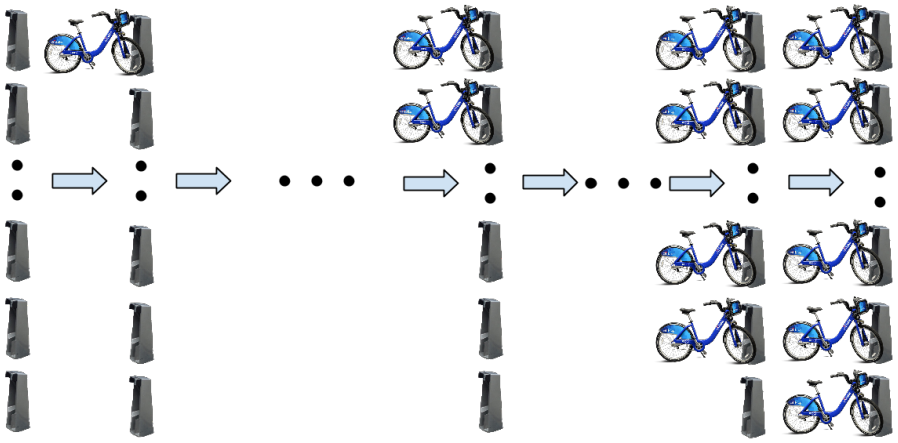

Figure 3.8: Example of the states of the continuous time Markov chain.

a large number of bikes to handle the large in/out flow. If balanced in/out flow is considered to be the result of some stochastic process we want to ensure that we have enough bikes at the station to avoid the tails of this stochastic process causing outages.

## 3.4 Using Continuous Time Markov Chains to Capture Stochasticity

To capture the stochastic nature of bikes arriving and leaving we use a continuous-time Markov chain to estimate the time the station would be full or empty given the decisions made.

We want to be compute the expected amount a station will be full or empty given an initial number of bikes and a capacity. We assume that the flows of bikes in and out of the station are both Poisson processes that are independent. Using these assumptions we model the behavior of a station over time as a continuous-time Markov chain. The

states of the chain represent the level of the station. We let

$$X(t) \in \{0, 1, \cdots, C-1, C\}$$

be the state of the chain at time $t$, specifically the number of bikes at the station at time $t$. An example showing the different states of the Markov chain can be seen in Figures 3.8.

For each station, $i$, we model the flow of bikes out of the station as a stationary Poisson process with mean $\lambda_i$. To model where these bikes are going for each station $i$ and end up at station $j$ we have a multinomial distribution $P_i$, we let $P_{ij}$ be the probability a bike taken from station $i$. Given these, the arrival rate of bikes at station $j$ is

$$\mu_j = \sum_{i \neq j} \lambda_i P_{ij}$$

### 3.4.1 Computing the CTMC

We first need to define a *rate matrix*, $A$ for a station $i$ with capacity $C_i$. We assume that station $i$ has a flow in of bikes that is a Poisson process of rate $\mu$ and a flow out of bikes that is a Poisson process of rate $\lambda$. We introduce the rate matrix for the CTMC, $A$. $A$ is a $C \times C$ square matrix. The element $A_{ij,i \neq j}$ represents the rate departing from state $i$

and arriving in state $j$. $A$ has the following tri-diagonal form:

$$A = \begin{pmatrix} -\mu & \mu & & & & & & \\ \lambda & -(\lambda+\mu) & \mu & & & & & \\ & \lambda & -(\lambda+\mu) & \mu & & & & \\ & & . & & & & & \\ & & & . & & & & \\ & & & & \lambda & -(\lambda+\mu) & \mu & \\ & & & & & \lambda & -\lambda & \end{pmatrix}$$

We now define another matrix $M^t = \left[M^t_{ij}\right]$ that depends on $t$ where the entry

$$M^t_{ij} = \mathrm{E}\left[\int_0^t \mathrm{I}(X(s) = j)|x(0) = i)ds\right]$$

This is the expected amount of time the station was at $j$ bikes in the time interval $[0, t]$ given that the station started with $i$ bikes. We now define the *time-s transition matrix* $P(s)$, where $P(s)_{ij}$ is equal to the probability of being in state $j$ at time $s$ given you started with $i$ bikes at the station. There is a relationship between $P(s)$ and the stationary distribution of the Markiv chain. First we let $\pi$ be the stationary distribution such that

$$\pi^\intercal = [\pi_0, \pi_1, \cdots, \pi_c]$$

If $\pi$ is the stationary distribution of the continuous time Markov chain with generator $A$, then

$$\pi^\intercal A = 0$$
$$\sum_{i=0}^{C} \pi_i = 1$$

If we let the matrix $\Pi$ be

$$\Pi = \begin{bmatrix} \pi^\intercal \\ \pi^\intercal \\ \vdots \\ \pi^\intercal \end{bmatrix}$$

we can observe the following properties of $P(s)$, namely

$$P(0) = I$$

$$P(\infty) = \Pi$$

The probability of being in any state at time 0 is 1 if and only if the chain started in that state. After an infinite amount of time the probability of being in any state is equal to its value in the stationary distribution no matter the start state of the chain. Given $P(s)$, then

$$M^t = \int_0^t P(s)ds$$

To compute $P(t)$ we use the matrix exponential

$$e^{At} = \sum_{k=0}^{\infty} \frac{t^k A^k}{k!}$$

For a given time value $t$, $P(t) = e^{At}$, this is a result of a derivation based on the Kolmogorov forward equation. This can be written as the following matrix differential equation

$$P'(t) = AP(t);$$

when solved this yields $P(t) = e^{At}$. Is it worth noting that as we are solely concerned with the amount of time the station is either empty or full we only care about the first and last columns of $M$. Taking the above into account, we can compute the matrix $M$

$$M = \int_0^t P(s)ds = \int_0^t (P(s)ds + \Pi - \Pi) =$$

$$\Pi t + \int_0^t (P(s) - \Pi)\, ds = \Pi t + \int_0^\infty (P(s) - \Pi)\, ds - \int_t^\infty (P(s) - \Pi)\, ds \qquad (3.1)$$

To begin to simplify the above, we make the following observations

$$P(s) \to \Pi \qquad \text{as } s \to \infty \qquad (3.2)$$

This is clear to see as we had stated earlier that $P(\infty) = \Pi$.

$$\forall t \qquad P(t)\Pi = \Pi \qquad (3.3)$$

Note that $P(t)$ has row sum 1 and each column of $\Pi$ has the same value.

$$\text{for } s > t \qquad P(s) = P(t)P(s-t) \qquad (3.4)$$

Using the above observations we use the following simplification.

$$\begin{aligned}
\int_t^\infty (P(s) - \Pi)\, ds = & \\
\int_t^\infty (P(t)P(s-t) - \Pi)\, ds = & \qquad \text{using (3.4)} \\
\int_t^\infty (P(t)P(s-t) - P(t)\Pi)\, ds = & \qquad \text{using (3.3)} \\
P(t)\left(\int_t^\infty (P(s-t) - \Pi)\, ds\right) = & \\
P(t)\left(\int_0^\infty (P(s) - \Pi)\, ds\right) &
\end{aligned}$$

Plugging this into equation (3.1) we get

$$\begin{aligned}
\text{Letting } Z = \int_0^\infty (P(s) - \Pi)\, ds & \\
M = \Pi t + Z - P(t)Z = & \\
\Pi t + (I - P(t))Z &
\end{aligned}$$

In the above equation we can compute all the components, apart from $Z$. To compute $Z$, we will show properties of $Z$ that yield a system of equations with only one unique solution, allowing us to solve for $Z$. The system needed is:

$$AZ = \Pi - I$$

$$\Pi Z = 0$$

To show that the above statements are valid first we take

$$\Pi Z = \Pi \int_0^\infty (P(s) - \Pi)\, ds =$$
$$\int_0^\infty (\Pi P(s) - \Pi\Pi)\, ds =$$
$$\int_0^\infty (\Pi - \Pi)\, ds =$$
$$\int_0^\infty (0)\, ds = 0$$

We then show that

$$AZ = A \int_0^\infty (P(s) - \Pi)\, ds =$$
$$\int_0^\infty (AP(s) - A\Pi)\, ds =$$
$$\int_0^\infty (AP(s) - 0)\, ds$$

Note that $A\Pi = 0$ as $A$ has a row sum of $0$.

$$AZ = \int_0^\infty (AP(s))\, ds;$$

via the Kolmogorov forward equation we know that $P'(t) = AP(t)$, and thus

$$AZ = \int_0^\infty (P'(s))\, ds =$$
$$[P(s)]_0^\infty = P(\infty) - P(0) =$$
$$\Pi - I$$

Via the result in [] these equations are sufficient to guarantee a unique value for $Z$, allowing us to solve them to obtain a usable value.

**Algorithm for computing $k^{th}$ column of M**

- *Step 1* Compute $P(t) = e^{At}$
- *Step 2* Solve the linear system

$$\pi^{\intercal} A = 0$$

$$\sum_i \pi_i = 1$$

- *Step 3* Compute $z_k$ the $k^{th}$ column of $Z$ by solving

$$A z_k = \pi_k \underset{\sim}{e} - \underset{\sim}{e_k}$$

$$\pi^{\intercal} z_k = 0$$

where

$$\underset{\sim}{e} = \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix} \qquad \underset{\sim}{e_k} = \; k^{th} \text{ entry} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

- *Step 4* Compute $\underset{\sim}{m_k}$

$$\underset{\sim}{m_k} = \pi_k t \underset{\sim}{e} + (I - P(t)) z_k$$

### 3.4.2 Convexity of CTMC based Cost Function

The described computation allows us to know the expected amount of time each station is empty or full given a starting allocation of bikes. This approach better captures the stochastic nature of the usage of bikes and ensures that our optimizations are focused on outage reduction and thus the number of upset customers. To allow ease of optimizing across the expected number of upset customers we prove a convexity result on the following function. We define a function for each bike share station $f(b)$ to be the expected number of upset people, by either not being able to get a bike or not being able to return a bike they have already taken out, given that we started a rush-hour period with $b$ bikes at the station. Formally we define

$$f(b) = \lambda E_b\left[T(t)\right] + \mu E\left[S(t)\right]$$

where

$$T(t) = \int_0^t I(X(s) = 0)ds,$$

$$S(t) = \int_0^t I(X(s) = C)ds,$$

$$E_b[\cdot] = E\left[\cdot | X(0) = b\right], \text{ and so}$$

$$f(b) = \lambda E_b\left[\left(\int_0^t I(X(s) = 0)ds\right) | X(0) = b\right] + \mu E_b\left[\left(\int_0^t I(X(s) = C)ds\right) | X(0) = b\right]$$

is the expected number of upset people over the time period $[0, t]$ given a starting condition of $b$ bikes.

**Lemma 3.4.1** *$f$ is convex in $b$.*

**Proof** To prove convexity, we use a coupling argument. First, we redefine

$$f(b) = \lambda E_b\left[T(t)\right];$$

a similar argument suffices for $\mu E_b[S(t)]$ our goal is now to prove convexity for this redefined $f(b)$.

We let $N_\lambda(t)$ be a Poisson process of rate $\lambda$. We let $N_\mu(t)$ be a Poisson process of rate $\mu$, independent of $N_\lambda$. Set $X_i(t)$ to be the continuous time Markov chain started in state $i$ with net input process $N_\mu(t) - N_\lambda(t)$ but restricted to the values $\{0, 1, 2, \cdots, C-1, C\}$. So

$$X_i(0) = i$$
$$i \in \{0, 1, 2, \cdots, C-1, C\}$$

We show that $f(i)$ is decreasing and convex in $i$ by showing that

$$E[T_i(t) - T_{i+1}(t)] \geq 0 \qquad i \geq 0 \qquad \text{it is decreasing} \tag{3.5}$$
$$E[T_i(t) - T_{i+1}(t)] \leq E[T_{i-1}(t) - T_i(t)] \qquad i \geq 1 \qquad \text{it is convex} \tag{3.6}$$

**Proving** $E[T_i(t) - T_{i+1}(t)] \geq 0$ To show equation 3.5, we first note that

$$X_{i+1}(t) \geq X_i(t) \qquad \forall t,$$

Specifically, the chain started at $i+1$ will always be in a higher or equal state than the chain started at $i$ with the same trace. This gives us

$$I(X_{i+1}(t) = 0) \leq I(X_i(t) = 0) \qquad \forall t,$$

so

$$T_{i+1}(t) \leq T_i(t) \qquad \forall t,$$

If we take the expectations of the above, we have that

$$E\left[T_{i+1}(t)\right] \leq E\left[T_i(t)\right]$$

$$E\left[T_{i+1}(t)\right] - E\left[T_i(t)\right] \leq 0$$

$$E\left[T_i(t)\right] - E\left[T_{i+1}(t)\right] \geq 0$$

$$E\left[T_i(t) - T_{i+1}(t)\right] \geq 0$$

**Proving** $E\left[T_i(t) - T_{i+1}(t)\right] \leq E\left[T_{i-1}(t) - T_i(t)\right]$ : To show equation 3.6 we first note that

$$g_i(t) = E\left[T_i(t) - T_{i+1}(t)\right]$$

is increasing in $t$ for each fixed $i = 0, 1, \cdots, C$. We note that

$$T_i(t) = T_{i+1}(t) = 0$$

until the first time $X_i(t)$ hits 0. After this $T_i(t) - T_{i+1}(t)$ increases at some rate, while $X_t(t) = 0$ and $X_{i+1}(t) = 1$ or stays constant if $X_i(t) \geq 1$ until $X_i(\cdot)$ and $X_{i+1}(\cdot)$ couple at which point it remains constant. Thus $T_i(t) - T_{i+1}(t)$ is path-wise increasing in $t$; taking expectations we have that $g_i(t)$ is increasing in $t$.

We now introduce the function

$$\tau_i(j) = \inf\{s : X_i(s) = j\}$$

specifically, $\tau_i(j)$ is the first time that $X_i(\cdot)$ hits $j \neq i$, for $i \in \{1, 2, \cdots, C\}$ and $j \in \{0, 1, \cdots, i-1, i+1, \cdots, C\}$. To prove convexity, we will perform a case analysis on how $\tau_i(j)$ behaves. First we observe that

$$g_i(t) = E\left[T_i(t) - T_{i+1}(t)\right] =$$

$$E\left[\int_0^t \left(I(X_i(s) = 0) - I(X_{i+1}(s) = 0)\right) ds\right] =$$

$$E\left[\int_{min(\tau_i(i-1),t)}^t \left(I(X_i(s) = 0) - I(X_{i+1}(s) = 0)\right) ds\right] \qquad \text{for } i \geq 1$$

This follows since

$$X_{i+1}(s) \geq X_i(s) \qquad \forall s \leq \tau_i(i-1)$$

We now perform a case analysis on how $\tau_i(j)$ behaves.

**Case 1:** $\tau_i(i-1) \geq t$ . In this case, we never enter the state $i-1$ until the very end of the time interval (at the earliest) thus

$$E\left[\int_{min(\tau_i(i-1),t)}^{t} (I(X_i(s)=0) - I(X_{i+1}(s)=0))\, ds\right] = E\left[\int_{t}^{t} (I(X_i(s)=0) - I(X_{i_1}(s)=0))\, ds\right] = 0$$

and equation 3.6 holds.

**Case 2: Coupled by time** $\tau_i(j)$ . This happens if $\tau_i(C), \tau_i(i-1)$ and results in

$$X_{i+1}(s) = X_i(s) \qquad \forall s \geq \tau_i(i-1)$$

Thus

$$E\left[\int_{min(\tau_i(i-1),t)}^{t} (I(X_i(s)=0) - I(X_{i+1}(s)=0))\, ds\right] = 0$$

and equation 3.6 holds.

**Case 3: Uncoupled at time** $\tau_i(i-1)$ . In this case, we have that at time $\tau_i(i-1)$, $X_i(\tau_i(i-1) = i-1$ and $X_{i+1}(\tau_i(i-1) = i$. We also know that $\tau_i(i-1) < t$, otherwise we would be in Case 1 and $\tau_i(i-1) < \tau_i(C)$; otherwise we would have already coupled. Using this we show that

$$E\left[\int_{min(\tau_i(i-1),t)}^{t} (I(X_i(s)=0) - I(X_{i+1}(s)=0))\, ds\right] = E\left[\int_{\tau_i(i-1)}^{t} (I(X_i(s)=0) - I(X_{i+1}(s)=0))\, ds\right].$$

Since $X_i(\tau_i(i-1) = i-1$ and $X_{i+1}(\tau_i(i-1) = i$ and we have the Markovian property the above is exactly the value of $g_{i-1}(t - \tau_i i - 1)$ where

$$g_i(t) = E\left[\int_{\tau_i(i-1)}^{t} (I(X_i(s) = 0) - I(X_{i+1}(s) = 0))\, ds\right] =$$

$$g_{i-1}(t - \tau_i i - 1) = E\left[\int_{0}^{t-\tau_i i-1} (I(X_{i-1}(s) = 0) - I(X_i(s) = 0))\, ds\right]$$

$$\leq g_{i-1}(t)$$

since $g_{i-1}(s)$ is increasing in s and Equation 3.6 holds. By the above proof, we have shown that the function $f(i)$ is both decreasing in $i$ and convex in $i$.

A similar proof success to show that the function

$$f(b) = \mu E_b\left[S(t)\right]$$

is convex and thus we have that the cost function

$$f(b) = \lambda E_b\left[T(t)\right] + \mu E_b\left[S(t)\right]$$

is a convex function. □

### 3.4.3 Optimization of bike placement

The properties of the cost function allow us to easily optimize the placement of bikes in stations before the beginning of a rush-hour. Like previous approaches to this optimization problem we are given a budget of available bikes, $B$, and we want to place them at stations to minimize a total cost function.

$$\min J(X)$$

$$s.t. \qquad \sum_s X_s \leq B$$

$$\forall s \qquad 0 \leq X_s \leq c(s)$$

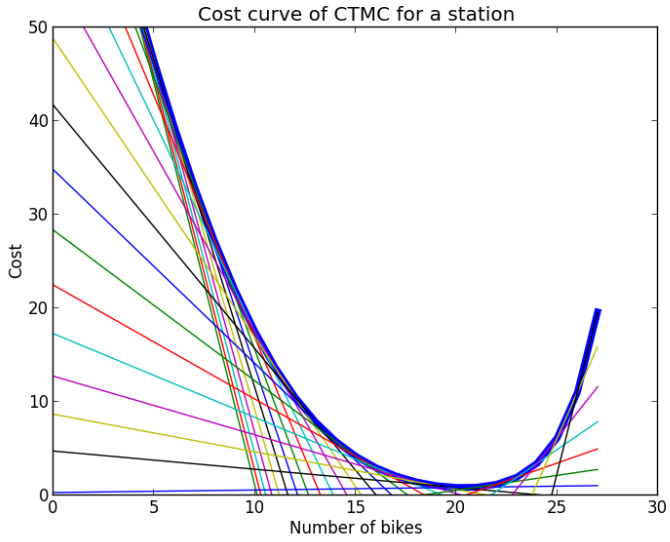


Figure 3.9: Example of the CTMC cost curve and the representation as series of linear constraints for one station.

In this case, the specific optimization problem we are solving is

$$\begin{aligned} & & \min \sum_s f_s(X_s) \\ s.t. & & \sum_s X_s \leq B \\ \forall s & & 0 \leq X_s \leq c(s) \end{aligned}$$

Where $f_s$ is the cost function over the interval $[0, t]$ for station $s$. We notice that this is a convex integer program. However, we will show that this can be solved by a linear program whose optimal solution lies at an integral point.

We construct the linear program in the following way. For each station, $s$, we define a set of $C - 1$ constraints where each constraint $i$ can be thought of as the line through the two points $f_s(i)$ and $f_s(i + 1)$. We then introduce a cost variable that must lie above each of these lines. An example of the constraints added in this manner can be seen in

Figure 3.9. The resulting linear program is

$$\begin{aligned}
& & & \min \sum_s y_s \\
s.t. & & & \sum_s X_s \leq B \\
\forall s & \qquad \forall i = 0, \cdots, C-1 & \qquad & y_s \geq (i - X_s)\left(f_s(i+1) - f_s(i)\right) \\
\forall s & & & 0 \leq X_s \leq c(s)
\end{aligned}$$

Due to the convex nature of the cost function we can show that the optimal solution to this linear program lies at an integer point and this we can find the best placement of bikes quickly.

**Lemma 3.4.2** *The optimal solution of the generated linear program lies at an integral point.*

**Proof** Consider a fractional solution the above linear program, we have at least two fractional $X_s$ variables, $X_{s_1}$ and $X_{s_2}$. For each station there can only be one constraint that is tight as the value $y_{s_1}$ lies between the two points $f(\lfloor X_{s_1} \rfloor)$ and $f(\lceil X_{s_1} \rceil)$ and similarly for $y_{s_2}$. Our goal will to be increase one of $X_{s_1}, X_{s_2}$ and decrease the other. We look at the tight constraint for each of these stations and the slope of that constraint. One of them has slope less than or equal to the other, thus by decreasing one value of $X_s$ and increasing the other we decrease the reduction in cost for one station is greater than or equal to the increase in cost of the other station. We increase or decrease the values in this manner until at least one of them becomes integral. If there are remaining fractional values we continue the process. Thus, by simply moving variables towards vertices of the polytope and making them integral we have not increased the cost of the solution, giving us the result that the optimal solution of the linear program lies at an integral point. □

### 3.4.4 Running Time of CTMC Optimization

Although the ability of the linear program to produce optimal integer solutions allows fast optimization for bike allocation in practice the above approach is not polynomial. Instead, we have a *pseudo-polynomial* as the number of constraints depends on the value of a input number, specifically the station capacity. It is conceivable that an ellipsoid approach would result in a polynomial-time algorithm for solving the linear program; however, to do this we would need to query cells of the matrix $M$ in polynomial-time. The current approach to the computation of this matrix does not allow this; future work may attempt to approximate the value of $M$ while avoiding the computation of a pseudo-polynomialy sized matrix. However, we will show experimental results that demonstrate that in practice this can be solved in seconds.

### 3.4.5 Experimental Results

In this section we will present a series of experimental results related to the computation of the continuous time Markov chains and the optimization problems solved using their output.

In Figure 3.10 the time taken to compute the columns of the matrix $M$ that are needed for different station capacities. This graph shows that for stations under one hundred docks less than one second of computation is needed. Since there is no station in New York even close to one hundred docks, this method is more than capable of handling the computation for the New York system.

The results of optimizing across the CTMC-based cost function are shown in Figure 3.11. From intuition gained from time spent working with staff in New York and months

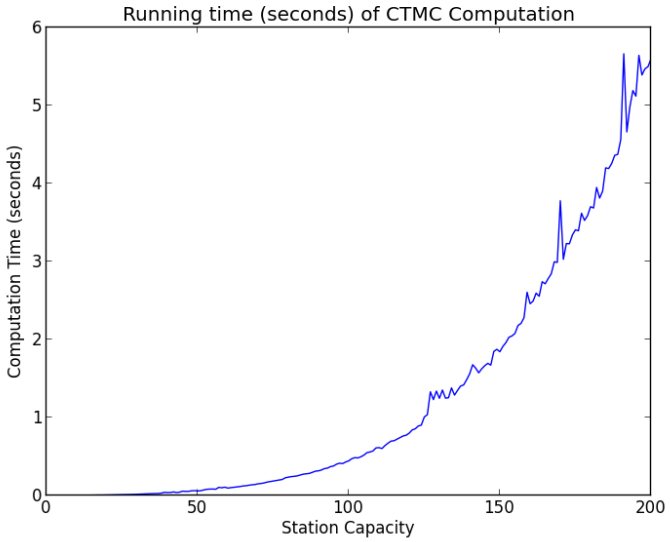


Figure 3.10: Graph showing the average time taken to compute the matrix $M$ of continuous time Markov chains of different size taken over ten randomly generated flow rates.

of staring at maps of system usage, this is a very natural allocation of bikes for the morning rush-hour. We also note a difference from the other models presented earlier in that this solution has far fewer completely empty stations.

### 3.4.6 Optimizing Fleet Size

Another use for the approaches presented in this chapter is answering the question of what is the optimal fleet size of bicycles in New York. The answer to this question is particularly salient when considering the impact of expansion on the system.

To answer the question of how many bikes are needed for the Citibike system we aimed to answer the question of how many bikes are needed at the beginning of the morning rush hour. Rush-hour usage of the system is the predominate factor driving ridership. We chose the morning rush hour in particular as overnight rebalancing allows

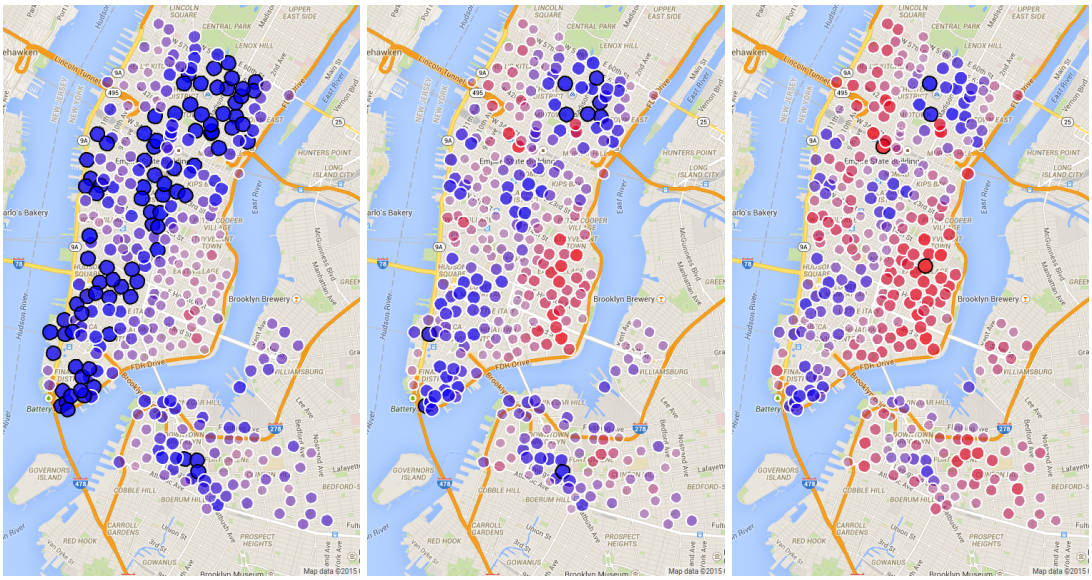

Figure 3.11: Morning levels assigned based on a CTMC cost functions. From left to right, two thousand bikes available, four thousand bikes available and six thousand bikes available. Blue stations are emptier and red fuller.

rebalancing staff to get the system into a near ideal state. We want to know: how many bikes are needed to achieve that ideal state? That number can be computed by finding the smallest value of $B$ for the linear programs described above such that solving with $B + 1$ bikes does not improve the objective function.

## 3.5 Allocating Docks Using Continuous Time Markov Chains

So far we have solely looked at using continuous-time Markov chains to direct an optimal allocation of bikes to stations. Another planning problem and one that arguably has a more profound impact on the success of a bike-sharing system, is that of where to place racks. Specifically, what is the capacity that should be located at each station in the system if we have a limited number of total racks. We can use a similar modeling approach to that used successfully to locate bikes at stations before a rush-hour based

on continuous-time Markov Chains. Where previously we optimized over the function $f_s(x) \to \mathbb{R}^+$ with $x$ being the number of bikes placed at station $s$, we switch and optimize over the function $f_s(x, r) \to \mathbb{R}^+$. This maps the number of bikes and racks placed at a station to a similar pain score as that used earlier. We prove similar convexity results on this new function as well as other properties needed to optimize across the two dimensional integer lattice.

First, we introduce a different function. When using the function $f(x, r)$ we need to ensure that $x \leq r$ to ensure the bikes have somewhere to be placed, to avoid this constraint we introduce the function $g(i, j)$ where

$$g(i, j) = f(i, j + i)$$

$g(i, j)$ can be thought of the cost of placing $i$ bikes in racks at the stations and $j$ empty racks.

### 3.5.1 Properties of $g(i, j)$

To facilitate optimization over the function $g(i, j)$ we first prove some properties. We prove that $g(i, j)$ is convex in each dimension and also a super-modularity result on adding racks to a station. We also prove two properties about the original $f(i, j)$ function.

**Lemma 3.5.1** *If we fix $i$ then $g(i, j)$ is convex in $j$*

$$g(i, j) - g(i, j - 1) \leq g(i, j + 1) - g(i, j), \forall i, j \geq 1$$

**Lemma 3.5.2** *If we fix $j$ then $g(i,j)$ is convex in $i$*

$$g(i,j) - g(i-1,j) \leq g(i+1,j) - g(i,j), \forall i,j \geq 1$$

**Lemma 3.5.3** *Adding racks to a station is super modular in the presence of extra bikes in racks*

$$g(i,j+1) - g(i,j) \leq g(i+1,j+1) - g(i+1,j), \forall i,j \geq 0$$

**Lemma 3.5.4** *For the original function $f$*

$$f(x-1,y) - f(x-1,y-1) \geq f(x,y) - f(x,y-1), \forall x,y \geq 1$$

**Lemma 3.5.5** *For the original function $f$*

$$f(x,y-1) - f(x-1,y-1) \leq f(x+1,y) - f(x,y), \forall x,y \geq 1$$

**Proof** To prove the above lemmas we first observe that the functions f and g are the sum of two terms. The first being the a cost of the station being empty and the second being a cost of the station being full. We will prove that the properties hold for each of the terms individually and this have that the properties hold for a linear combination of the two terms.

To prove the properties for the two terms we will prove for one term and show how *flipping* the rates of arrival and departure for the other term shows that is has the same properties. Consider proving the properties for the term related to the station being empty, to show for the other process we construct a modified but equivalent continuous time Markov chain where instead of bikes arriving at a station spaces do and spaces leave the station, now a station being empty (what we are focused on the first term) is equivalent to the station being full. Thus proving these properties for the empty station term yields us the same properties for the full station term and linear combinations of these with positive coefficients.

Thus we now redefine the function $g$ letting

$$g(i,j) = E\left[\int_0^t I(X(s)=0)ds\right]$$

**Decreasing in $j$** First we let the time $t > 0$, and integers $i, j \geq 0$ and observe that $g(i,j)$ is decreasing in $j$ for a fixed $i$. Consider any path of the CTMC, by allowing another rack we cannot spend more time in the 0 state. If two processes, one representing $g(i,j)$ and one with $g(i,j+1)$, start at the same level they stay coupled until the level of bikes hits the upper limit of the chain with $j$ and the processes decouple. They can only couple again by both hitting the empty state where the process with $g(i,j)$ will spend strictly more time in the empty state. Thus giving us

$$g(i,j) \geq g(i,j+1)$$

**Convexity** We want to show that

$$g(i,j+1) - g(i,j) \leq g(i,j+2) - g(i,j+1)$$

or that

$$f(i,k+1) - f(i,k) \leq f(i,k+2) - f(i,k+1)$$

in the original function, $f$m where $k = i + j$. We have that

$$f(i,k+1) - f(i,k) = E\left[\int_0^t \left(I(X_1(s) = 0) - I(X_0(s) = 0)\right) ds\right]$$

where $X_0$ and $X_1$ are sample paths of processes started with a bike level of $i$ and capacities of $k$ and $k+1$ respectively. We note that $0 \leq X_1(s) - X_0(s) \leq 1$ for all $s \geq 0$ and this the integration in the above equation never increases and only decreases when $X_0$ is in state $0$ and $X_1$ is in state $1$. We now create two additional processes $X_2$ and $X_3$ in such a way that

$$f(i,k+2) - f(i,k+1) = E\left[\int_0^t \left(I(X_3(s) = 0) - I(X_2(s) = 0)\right) ds\right]$$

The processes $X_2$ and $X_3$ are almost identical to those of $X_1$ and $X_0$. They are identical apart from when the processes $X_2$ and $X_3$ exceed states $k+1, k+2$. If you were to take the trace of the process and remove everything in or above these states and then join up the remainder you would have an identical trace to that of $X_0$ and $X_1$. We note that the states required for the above equation to decrease are exactly that of the states required for $X_2$ and $X_3$ to decrease. Thus we have

$$E\left[\int_0^t \left(I(X_1(s) = 0) - I(X_0(s) = 0)\right) ds\right] \leq E\left[\int_0^t \left(I(X_3(s) = 0) - I(X_2(s) = 0)\right) ds\right]$$

taking expectations we have that

$$f(i,k+1) - f(i,k) \leq f(i,k+2) - f(i,k+1)$$

this gives us a proof of Lemma 3.5.1. The proof of Lemma 3.5.2 is almost identical. We want to show that

$$f(i+1,k+1) - f(i,k) \leq f(i+2,k+2) - f(i+1,k+1)$$

We associate processes $X_1$ with $f(i+1,k+1)$ (on the left-hand side of the inequality), $X_0$ with $f(i,k)$, $X_3$ with $f(i+2,k+2)$ and $X_2$ with $f(i+1,k+1)$ (on the right-hand side of the inequality). We then initiate the processes $X_{0,1}$ and $X_{2,3}$ in states $(i,i+1)$ and $(i+1,i+2)$ respectively. Again the processes $X_2$ and $X_3$ are almost identical to that of $X_0$ and $X_1$.

**Super-Modularity** To prove Lemma 3.5.3 we first write in the original form, we want to show

$$f(i,k+1) - f(i,k) \leq f(i+1,k+1) - f(i+1,k)$$

This proof is almost identical. We associate process $X_1$ with $f(i,k+1)$, $X_0$ with $f(i,k)$, $X_3$ with $f(i+1,k+1)$ and $X_2$ with $f(i+1,k)$. The proof is then just as it was before with processes $X_0$ and $X_1$ started at state $i$ and processes $X_2$ and $X_3$ started in state $i+1$.

**Local Properties** First we write it in the original form

$$f(i,k-1) - f(i,k) \geq f(i-1,k-1) - f(i,k) \qquad i,k \geq 1$$

We then create processes $X_0(s)$ and $X_1(s)$ where $s \geq 0$. $X_0$ and $X_0(0) = X_1(0) = i-1$. $X_0$ and $X_1$ move up and down together and $0 \leq X_1(s) - X_0(s) \leq 1$, we give $X_0$ capacity $k-1$ and $X_1$ capacity $k$. This represents the right hand side of the equality. The processes first separate at time $T_{0,1}$, the first time $s$ that $X_1(s) = X_0(s) = k-1$ For the left hand side of the inequality we create processes $X_2$ and $X_3$ with $X_2(0) = X_3(0) = i$ and we give $X_2$ capacity $k-1$ and $X_1$ capacity $k$. Then the two pairs of processes couple at time $T_{0,1}$ and we have that $X_0(s) = X_1(s)$ for $s \leq T_{0,1}$ which is not necessarily true for $X_2$ and $X_3$. Thus the inequality holds.

To prove the final local property again we write it in the original form.

$$f(i, j-1) - f(i-1, j-1) \leq f(i+1, j) - f(i, j)$$

$$f(i, j-1) - f(i+1, j) \leq f(i-1, j-1) - f(i, j)$$

We again create two processes $X_0(0) = i - 1$ and $X_1(0) = i$ assigning capacities $j - 1$ to process $X_0$ and $j$ to process $X_1$. We also create the processes $X_2(0) = i$ and $X_3(0) = i+1$ with capacities $j-1$ and $j$ respectively. We generate the pairs of processes independently up to the first time, $\tau$ that $(X_2(s), X_2(s)) = (x-1, x)$ which is where we started the processes $X_0, X_1$ at. From time $\tau$ onwards we have that $X_2, X_3$ are delayed versions of $X_0, X_1$. From this we can see that the left hand side can only gain after the right hand side has gained and thus the inequality holds.

Thus we have proved the required properties to allow optimization over the function $g$.

### 3.5.2 Optimizing over $g(i, j)$

To optimize the function $g(i, j)$ across all stations with a budget of bikes and racks available we solve the following optimization problem

$$\begin{aligned} & \min \sum_s g_s(b_s, r_j) \\ s.t. \quad & \sum_s b_s \leq B \\ & \sum_s b_s + r_s \leq R \\ b_s, r_s \geq 0 & \end{aligned}$$

To optimize this, we will use a mixed integer linear program. To represent the function $g(i, j)$ we will use a series of planes generated from points on the integer lattice.

Each point $(i, j)$ will touch at least one plane. We show that the planes are valid for all points, specifically each plane generated is a lower bound on all points. These two properties allow us to optimize over this function.

**What lies beneath: generating valid planes from $g(i, j)$** We generate one plane for each integer point on the integer lattice. This ensure that at least one plane touches each integer point. To ensure that the planes we generate are valid we need to show that they are lower bounds for each point. We prove the following lemma.

**Lemma 3.5.6** *Planes generated from each point $(i, j)$ of the form*

$$p(x, y) = g(i, j) + (x - i)\,[g(i + 1, j) - g(i, j)] + (y - j)\,[g(i, j + 1) - g(i, j)]$$

*are valid for all points on the integer lattice.*

**Proof** We will prove this validity by looking at the four different quadrants, upper right, upper left, lower right and lower left individually. For each of these quadrants different properties of the function $g(i, j)$ will be used to show that the planes are valid. We can represent the value of a point as the value of a different point plus the sum of the value differences of a path between the points in the integer lattice. The proofs for each quadrant will use observations about the path taken from the point generating the plane to all others.

**Upper Right Quadrant** In this case we show a plane generated from the point $(i, j)$ is valid for all points $(x, y)$ where $x \geq i$ and $y \geq j$. Such a case is shown in Figure 3.12. For notational convenience in all cases we let

$$a_0 = g(i + 1, j) - g(i, j)$$

$$b_0 = g(i, j + 1) - g(i, j)$$

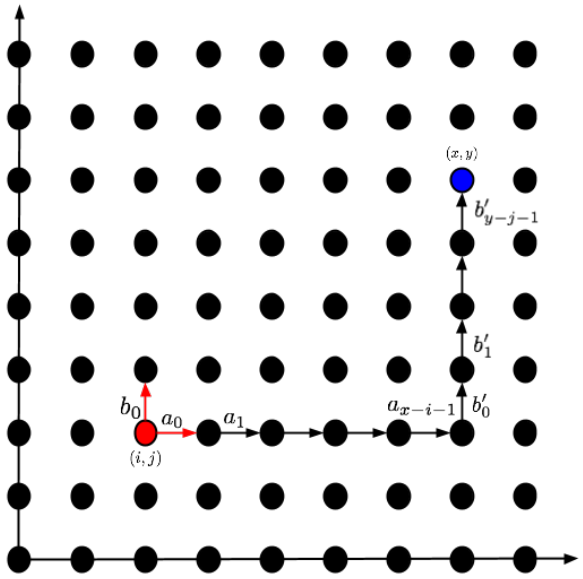


Figure 3.12: Example of a lattice with the point the plane has been generated from and the point we are showing is above the plane that is in the upper right quadrant.

For the plane to be valid we need that

$$p(x, y) = g(i, j) + (x - i)a_0 + (y - j)b_0 \leq g(x, y).$$

We can represent $g(x, y)$ as

$$g(x, y) = g(i, j) + a_0 + a_1 + \cdots + a_{x-i-1} + b'_0 + b'_1 + \cdots + b'_{y-j-1};$$

this is the value at point $(i, j)$ and the sum of the path along the horizontal axis and then up the vertical axis as shown in Figure 3.12. Thus we need that

$$\begin{aligned} g(i, j) + (x - i)a_0 + (y - j)b_0 \leq \\ g(i, j) + a_0 + a_1 + \cdots + a_{x-i-1} + b'_0 + b'_1 + \cdots + b'_{y-j-1} \\ (x - i)a_0 + (y - j)b_0 \leq \\ a_0 + a_1 + \cdots + a_{x-i-1} + b'_0 + b'_1 + \cdots + b'_{y-j-1}. \end{aligned}$$

We observe that

$$
\begin{aligned}
a_0 \leq a_1 \leq \cdots \leq a_{x-i-1} \qquad & \text{via Lemma 3.5.2} \\
b'_0 \leq b'_1 \leq \cdots \leq b'_{y-j-1} \qquad & \text{via Lemma 3.5.1} \\
b_0 \leq b'_0 \qquad & \text{via Lemma 3.5.3} \\
\implies b_0 \leq b'_0 \leq b'_1 \cdots \leq b'_{y-j-1}; &
\end{aligned}
$$

the combination of these give us the needed inequality and the validity for all points in the upper right quadrant.

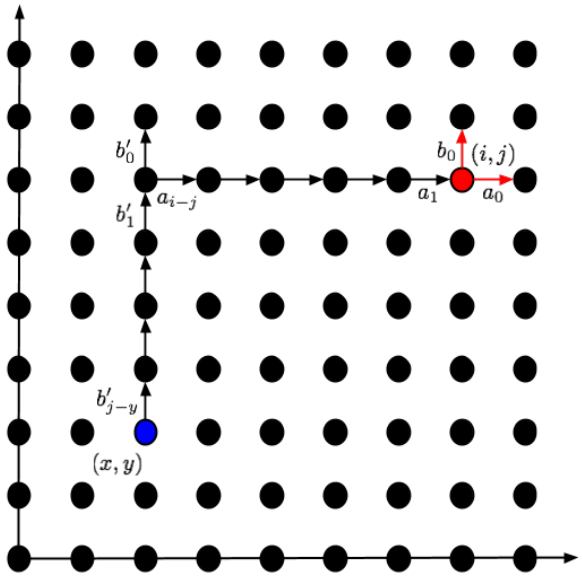


Figure 3.13: Example of a lattice with the point the plane has been generated from and the point we are showing is above the plane that is in the lower left quadrant.

**Lower Left Quadrant** In this case we show a plane generated from the point $(i, j)$ is valid for all points $(x, y)$ where $x < i$ and $y < j$. Again we look at the sum of the differences along a path in the integer lattice, the path taken in this case is shown in

Figure 3.13. The plane at this point is

$$p(x,y) = g(i,j) + (x-i)a_0 + (y-j)b_0 = \\ g(i,j) - (i-x)a_0 - (j-y)b_0;$$

we need

$$g(i,j) - (i-x)a_0 - (j-y)b_0 \leq g(x,y)$$

$$g(i,j) - (i-x)a_0 - (j-y)b_0 \leq g(i,j) - a_1 - a_2 - \cdots - a_{i-x} - b'_1 - b'_2 - \cdots - b'_{j-y}$$

$$-(i-x)a_0 - (j-y)b_0 \leq -a_1 - a_2 - \cdots - a_{i-j} - b'_1 - b'_2 - \cdots - b'_{j-y}$$

via convexity and super-modularity we have that

$$a_{i-x} \leq \cdots \leq a_1 \leq a_0 \qquad \text{via Lemma 3.5.2}$$

$$b'_{j-y} \leq \cdots \leq b'_1 \leq b'_0 \qquad \text{via Lemma 3.5.1}$$

$$b'_0 \leq b_0 \qquad \text{via Lemma 3.5.3}$$

$$\implies b'_{j-y} \leq \cdots \leq b'_1 \leq b'_0 \leq b_0;$$

thus we have that the inequality is valid for the lower left quadrant.

**Upper Left Quadrant** In this case we show a plane generated from the point $(i,j)$ is valid for all points $(x,y)$ where $x < i$ and $y > j$. Again we look at the sum of the differences along a path in the integer lattice, the path taken in this case is shown in Figure 3.14. The path taken in this case differs as we do not travel directly along each axis but instead take a path that alternates between axes for as long as possible. The plane in this case is

$$p(x,y) = g(i,j) + (x-i)a_0 + (y-j)b_0 = \\ g(i,j) - (i-x)a_0 + (y0j)b_0;$$

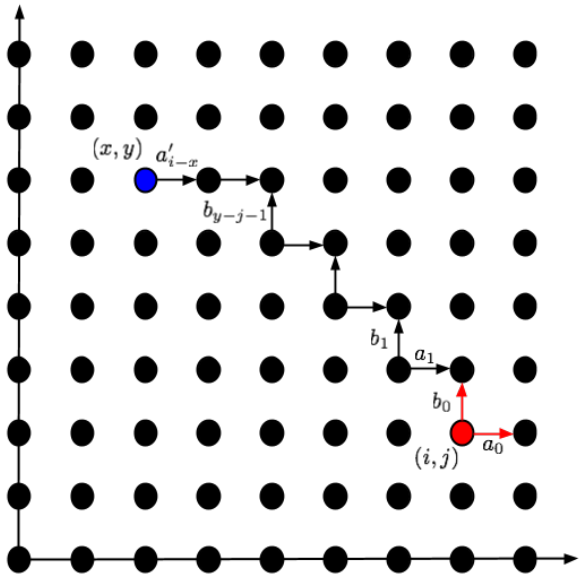


Figure 3.14: Example of a lattice with the point the plane has been generated from and the point we are showing is above the plane that is in the upper left quadrant.

we need the inequality

$$g(i,j) - (i-x)a_0 + (y0j)b_0 \leq g(x,y) =$$

$$g(i,j) - a_1 - \cdots - a_{i-x-1} - a'_(i-x) + b_0 + b_1 \cdots + b_{y-j-1}$$

$$-(i-x)a_0 + (y0j)b_0 \leq -a_1 - \cdots - a_{i-x-1} - a'_(i-x) + b_0 + b_1 \cdots + b_{y-j-1}.$$

To show that this inequality is valid we the following two lemmas.

**Lemma 3.5.7** *$b_i \geq b_{i-1}$ where $b_i = g(i, j+1) - g(i,j)$ and $b_{i-1} = g(i+1, j) - g(i+1, j-1)$*

**Proof** The proof follows from Lemma 3.5.4. We need that

$$g(i, j+1) - g(i, j) \geq g(i+1, j) - g(i+1, j-1)$$
$$f(i, i+j+1) - f(i, i+j) \geq f(i+1, j+i+1) - f(i+1, i+j);$$

letting $i + j = y$ and $i = x$ we have

$$f(x, y+1) - f(x, y) \geq f(x+1, y+1) - f(x+1, y)$$
$$f(x-1, y) - f(x-1, y-1) \geq f(x, y) - f(x, y-1),$$

which is true via Lemma 3.5.4.

**Lemma 3.5.8** *$a_i \geq a_{i-1}$ where $a_{i-1} = g(i+1, j) - g(i, j)$ and $a_i = g(i, j+1) - g(i-1, j+1)$*

**Proof** The proof follows from Lemma 3.5.5. We need

$$g(i, j+1) - g(i-1, j+1) \leq g(i+1, j) - g(i, j)$$
$$f(i, i+j+1) - f(i-1, i+j) \leq f(i+1, i+j+1) - f(i, i+j).$$

Letting $i = x$ and $i + j = y$,

$$f(x, y+1) - f(x-1, y) \leq f(x+1, y+1) - f(x, y)$$
$$f(x, y) - f(x-1, y) \leq f(x+1, y+1) - f(x, y+1)$$
$$f(x, y-1) - f(x-1, y-1) \leq f(x+1, y) - f(x, y),$$

which is exactly Lemma 3.5.5.

Using these lemmas, we have that

$$b_0 \leq b_1 \leq \cdots b_{y-j-1}$$

$$a_0 \geq a_1 \geq \cdots a_{i-x-1} \geq a'_{i-x};$$

since these are all negative quantities we have that the inequality is valid.

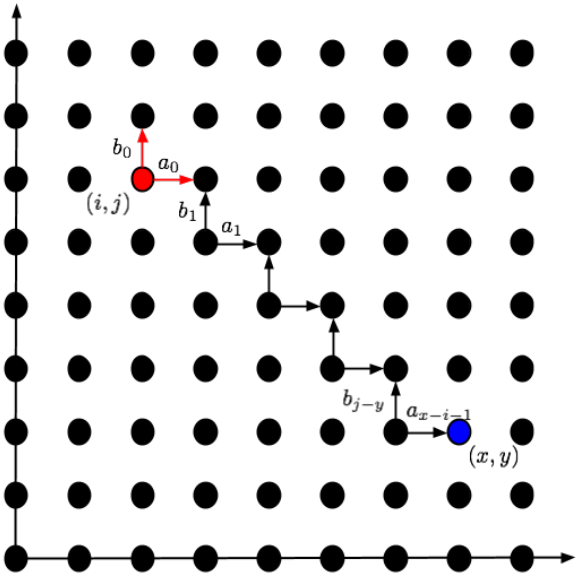


Figure 3.15: Example of a lattice with the point the plane has been generated from and the point we are showing is above the plane that is in the lower right quadrant.

**Lower Right Quadrant** In this case we show a plane generated from the point $(i, j)$ is valid for all points $(x, y)$ where $x > i$ and $y < j$. Again, we look at the sum of the differences along a path in the integer lattice; the path taken in this case is shown in Figure 3.15. Similar to the upper left quadrant, the path taken in this case differs as we do not travel directly along each axis but instead take a path that alternates between axes for as long as possible.

The plane in this case is

$$p(x,y) = g(i,j) + (x-i)a_0 + (y-j)b_0 =$$
$$g(i,j) + (x-i)a_0 - (j-y)b_0.$$

We need the inequality

$$g(i,j) + (i-x)a_0 - (j-y)b_0 \leq g(x,y) =$$
$$g(i,j) + a_0 + a_1 + \cdots a_{x-i-1} - b_1 - \cdots - b_{j-y}$$
$$(i-x)a_0 - (j-y)b_0 \leq a_0 + a_1 + \cdots a_{x-i-1} - b_1 - \cdots - b_{j-y}.$$

Using the lemmas used for the upper left quadrant, we have that

$$a_0 \leq a_1 \leq \cdots \leq a_{x-i-1}$$
$$b_0 \geq b_1 \geq \cdots \geq b_{j-y}$$

These inequalities along with the negativity of all the amounts give us the inequality.

By showing that the planes generated are valid for all quadrants we have that the planes are valid for all points. This concludes the proof that the planes are valid.

As well as a formal proof we also generated the function $g(i,j)$ for all stations, the surface generated for one station can be seen in Figure 3.16. We then generated all the plane and checked them against all points. No planes violated the required properties.

### 3.5.3 Mixed Integer Programming Formulation

To optimize the placement of bikes and racks at stations we use an Integer Programming formulation based on the generation of valid planes discussed earlier. The MIP is as

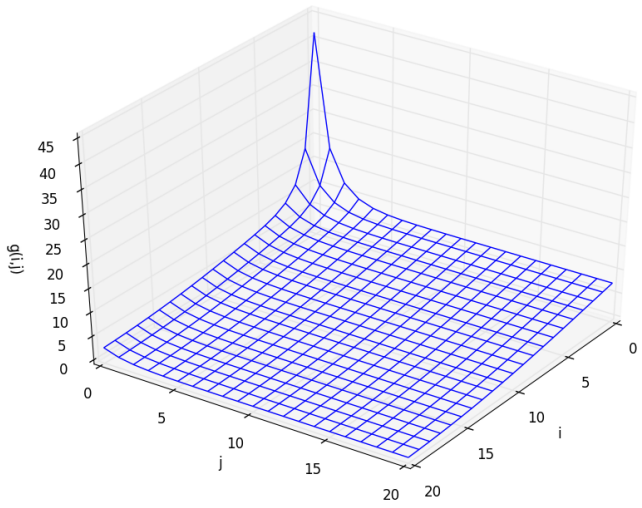


Figure 3.16: Example of the function for a station $g(i, j)$ plotted.

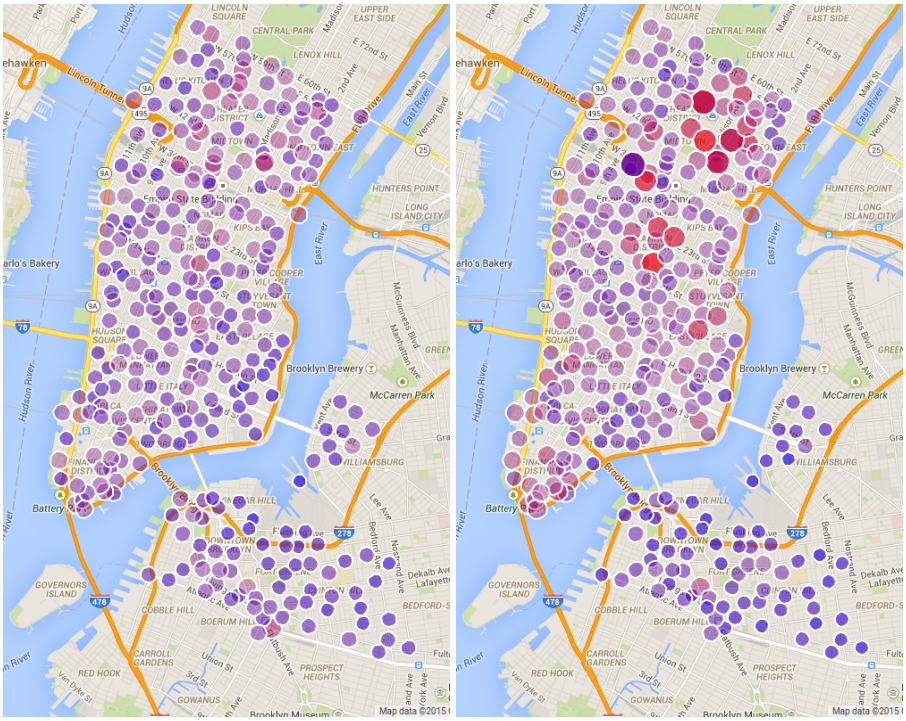


Figure 3.17: Two maps of station capacity, the map of the left shows the current allocation while the map shows the optimal allocation of the docks across the city. Red points correspond to high capacity stations and blue to low capacity stations.

follows:

$$\min \sum_s c_s$$

$$s.t. \quad \sum_s x_s \leq B$$

$$\sum_s y_s + x_s \leq R$$

$$\forall s \forall (i,j)$$

$$c_s \geq g(i,j) + (x_s - i)\left[g(i+1,j) - g(i,j)\right] + (y_s - j)\left[g(i,j+1) - g(i,j)\right]$$

$$\forall s \quad x_s + y_s \leq SMAX$$

$$x_s \in \{0, \cdots, SMAX\}$$

$$y_s \in \{0, \cdots, SMAX\}$$

where $SMAX$ is the maximum size of a station. This MIP provides a valid solution as each point touches at least one plane and all planes are valid for all other points.

Solving the above optimization problem for the New York system yields the solutions in Figure 3.17. In this Figure the MIP is solved for the current station locations in New York. The total number of docks available is allocated optimally amongst stations. The map on the left is the current allocation while the map on the right shows the optimal allocation. The station capacity in midtown as well as around transport hubs is far higher in the optimal allocation.

This approach can be of use for system planning as well as expansion and alteration. Coupled with an approach such as that of Singhvi etal. [28], where a prediction for flow rates during rush hours for candidate station locations can be generated, this method allows optimization of the placement of available racks at stations giving useful information to system planners.

This chapter has focused on using data from the system as well as observations

about patterns in usage to know where to plan optimally for periods of heavy usage. The following chapter deals with the problem of how to achieve these desired system states through rebalancing intervention?

CHAPTER 4

# HOW TO GET BIKES THERE?

The methods of Chapter 3 provide high-level guidance in how to allocate dock capacity and bikes between bike stations. They do not, however, provide a blueprint for how to achieve those target bike levels in practice. Moreover, practical realities may limit the amount of rebalancing we can achieve. In such a setting, one might look for rebalancing actions that use the resources available as efficiently as possible. We present two novel approaches to rebalancing, one to prepare for rush-hours and one to handle the surge of ridership during a rush-hour. Both of these are motivated through a close collaboration with system operators, taking the unique mix of human and logistic constraints found in New York into account, and takes a fundamentally different approach to previous work. In the remainder of this chapter we discuss the practical realities of rebalancing, indicate some of the very practical models we have developed and their performance in practice.

## 4.1 Mid-Rush Rebalancing

The morning and evening weekday rush-hours account for the majority of trips taken on New York's bike share system. Usage is extremely high during these periods and asymmetric; that is, the net flow of bikes out of many stations is largely positive or largely negative. From observation of trip data, the net flow of stations can be computed. The behavior of stations from one day's rush-hour to the next is very consistent, allowing us to classify them as either producers, consumers or self-balancing stations (which we need not worry about).

Our goal during the rush hour is to ensure that users of the system are not too far

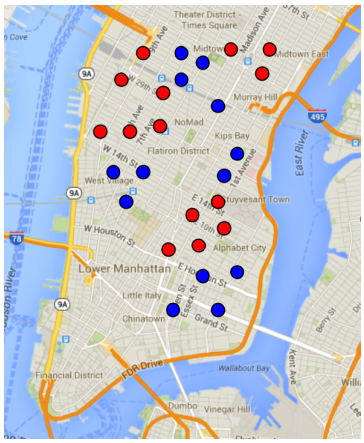


Figure 4.1: An example instance of the Mid-Rush Pairing Problem.

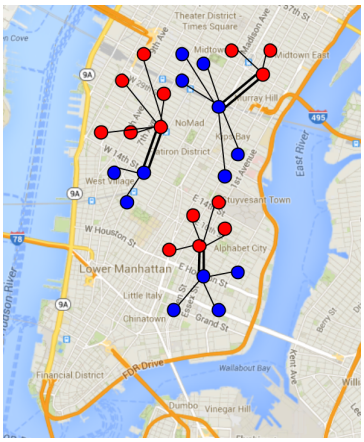


Figure 4.2: Example solution to the Mid-Rush Pairing Problem, black edges show the closest rebalanced station. Double black edges show the routes that will be run to move bikes.

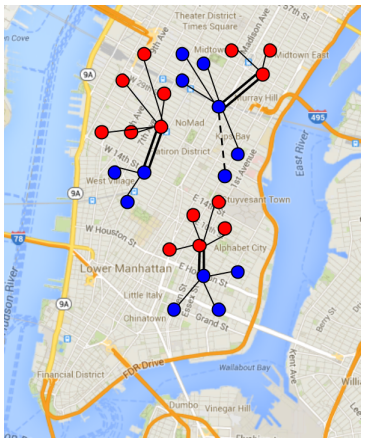


Figure 4.3: Highlighting the edge that contributes all of the objective function with a dashed black line.

Figure 4.4: Example of a Mid-Rush Pairing, its solution and the cost of the solution.

from either a bike or a dock. A criterion close to this is contained within the contract that requires the operator of Citibike to maintain a specified level of quality of service in a range of aspects; fines are imposed for failing to meet certain levels. We will focus rebalancing resources on covering the critical areas of the city and will only be able to rebalance a small subset of stations. Using historical data that indicate which stations accumulate bikes (producers) and which lose bikes (consumers), we want to ensure that each producer station is close to a producer station that will be rebalanced, and that each consumer station is close to some rebalanced consumer station. To rebalance a consumer station, bikes must be delivered to it, ideally from a producer station where bikes need to be removed for rebalancing. We select producer and consumer stations to rebalance, pairing them up so that rebalancing is achieved for both producer and consumer simultaneously.

In NYC, most of the mid-rush rebalancing is done by special bicycles outfitted with

trailers that can hold a few Citibikes, typically three; due to the nature of this resource, the most effective plan is to designate certain pairs of stations (i.e., with one producer and one consumer) to be targeted for mutual rebalancing. These pairs must be sufficiently close, so that the bicycles can be moved effectively within the narrow timeframe of a quickly transpiring rush hour. These considerations gave rise to the following optimization model:

**Definition** *Mid-Rush Pairing Problem* We are given a complete undirected graph $G = (P \cup C, E)$ with a nonnegative metric distance function $d(e), \forall e \in E$, as well as an integer $k$, and an integer $T$; the goal is to select two subsets, $P' \subseteq P$ and $C' \subseteq C$ such that $|P'| = |C'| \leq k$, and a perfect matching such that there exists a perfect matching in $\{e \in E : d(e) \leq T\}$ between $P'$ and $C'$ which minimizes $\max\left(\max_{p \in P} d(p, P'), \max_{c \in C} d(c, C')\right)$ We define $d(p, P')$ to be the distance between $p$ and the closest point in $P'$ to it.

Figure 4.4 shows a sample input to this problem. On the left, an instance of the problem can be seen, where producer stations are marked in red and consumer stations on blue. In this example we have $k = 3$. The middle figure shows a feasible solution with double black edges indicating the pairs of stations to be rebalanced; the black edges show, for each producer or consumer, the closest rebalanced station of its type. The objective function is determined by exactly one edge length; in other words, this is a *bottleneck optimization problem*. In the final image, the dashed black edge highlights the edge whose distance is equal to

$$\max\left(\max_{p \in P} d(p, P'), \max_{c \in C} d(c, C')\right)$$

**Lemma 4.1.1** *If, for any $\epsilon > 0$, there exists a $(2 - \epsilon)$-approximation algorithm for the Mid-Rush Pairing Problem, then $P = NP$.*

**Proof** The same claim holds for the $k$-center problem [11]; in the $k$-center problem, we are given a complete undirected graph $G = (V, E)$ with a nonnegative metric distance function $d(e)$, for each $e \in E$ and an integer $k$; the aim is to select $V' \subset V$ where $|V'| \leq k$, so that $\max_{v \in V} d(v, V')$ is minimized. We show that it is simple to give an approximation-preserving reduction from the $k$-center problem to the mid rush pairing problem. Given a $k$-center input, we simply set $P = V$ and let $C$ denote a set of $k$ new nodes. The distance between each pair of nodes in $P$ is given by the $k$-center metric $d$. Each pair of nodes in $C$ is distance 0 apart, and finally, $d(p, c) = T$ for each pair of nodes $p \in P$, $c \in C$, where $T = \max_{(u,v) \in V \times V} d(u, v)$. These distances form a metric on $P \cup C$, and there is a 1-1 mapping between feasible solutions for the input of the $k$-center problem and the input for the mid rush pairing problem, with equal objective function values.

### 4.1.1 Approximation Algorithm

We present a 3-approximation algorithm for the Mid-Rush Pairing Problem, which is based on rounding a feasible solution to a linear programming relaxation of the problem. Similar to the case of the $k$-center problem, Hochbaum and Shmoys [11], there are only a polynomial number of possible optimal values - $|P|^2 + |C|^2$ - since one term in the objective function corresponds to an inter-producer distance, and the other corresponds to an inter-consumer distance. For each potential optimal value, we either verify its infeasibility (by showing that a linear programming relaxation is infeasible), or otherwise, we show how to round the feasible fractional solution to an integer one with objective function value at most 3 times the desired target.

**The Linear Programming Relaxation**

$$\sum_{u \in V} x_u \leq k, \qquad \forall V \in \{P, C\}; \tag{4.1}$$

$$\sum_{u \in V:(u,v) \in E_V} y_{(u,v)} = 1, \qquad \forall v \in V, \qquad \forall V \in \{P, C\}; \tag{4.2}$$

$$y_{(u,v)} \leq x_u, \qquad \forall (u, v) \in E_V, \qquad \forall V \in \{P, C\}; \tag{4.3}$$

$$\sum_{c \in C:(p,c) \in E_M} m_{(p,c)} = x_p, \qquad \forall p \in P; \tag{4.4}$$

$$\sum_{p \in P:(p,c) \in E_M} m_{(p,c)} = x_c, \qquad \forall c \in C; \tag{4.5}$$

$$x_u \in [0, 1], \qquad \forall u \in V; \tag{4.6}$$

$$y_{(u,v)} \in [0, 1], \qquad \forall (u, v) \in E_P \cup E_C; \tag{4.7}$$

$$m_{(p,c)} \in [0, 1] \qquad \forall (p, c) \in E_M. \tag{4.8}$$

In the integer linear programming formulation of the Mid-Rush Pairing Decision Problem, we wish to decide if there is a feasible solution in which each producer is served within a distance of $d_P$ and each consumer within $d_C$, and that the paired nodes are within input threshold $T$; we let $E_P$ be those pairs $(u, v) \in P \times P$ for which $d(u, v) \leq d_P$, $E_C$ be those pairs $(u, v) \in C \times C$ for which $d(u, v) \leq d_C$, and let $E_M$ be those pairs $(u, v)$ such that $d(u, v) \leq T$. In the IP formulation, there exists a 0-1 decision variable $x_u$ for each $u \in P \cup C$ to indicate whether (or not) that node is selected as one of the $2k$ paired stations; there is a 0-1 decision variable $y_{(u,v)}$ for each $(u, v) \in E_P \cup E_C$ to indicate whether node $v$ is served by node $u$ in the pairing. Finally, we have a 0-1 decision variable $m_{(p,c)}$ for each $(p, c) \in E_M$ to indicate whether producer $p$ and consumer $c$ are paired. Hence, we get the linear programming relaxation as shown above.

### 4.1.2 LP Rounding

The rounding algorithm has the following overall structure. We use the support of the feasible solution (those variables with positive value) to define a graph $G_V = (V, E_V^+)$, for each $V \in \{P, C\}$; that is, if $(x^*, y^*, m^*)$ is the feasible fractional solution, then $E_V^+ = \{(u, v) \in E_V : y^*_{(u,v)} > 0\}$. For each graph $G_V$, we partition the nodes of the graph into at most $k$ clusters. Then, we use this clustering to construct a circulation network with integral upper and lower flow bounds, for which the LP fractional solution yields a feasible solution. Hence, there exists an integral flow; this yields a feasible pairing for which it can be shown that each node $u \in V$ is within $3d_V$ of its nearest paired station, for each $V \in \{P, C\}$.

The clustering algorithm for $G_V$ works as follows. Initally, each node in $V$ is unmarked, and the algorithm proceeds until every node becomes marked. In each iteration, select any unmarked node $v$, and consider the set of nodes $u$ such that $(u, v) \in E_V^+$; this is a cluster $\mathscr{C}$. Mark each node in $\mathscr{C}$, as well each node $w$ for which there is an edge $(u, w) \in E_V^+$ for some node $u \in \mathscr{C}$.

The clustering algorithm has the invariant that for each unmarked node $v \in V$, none of the nodes $u$ for which $y^*(u, v) > 0$ is contained in any cluster formed thus far, and hence constraint (4.2) still holds, even if we restrict the sum to those nodes not contained in any previously formed cluster. To see this, suppose that the invariant is false, and consider the first moment that this happens in the course of executing the algorithm. This means that there is a currently unmarked node $v$, for which there exists a node $u$ that was included in the cluster formed by the previous iteration; however, in that case, $v$ must have been marked in that iteration as well, which is a contradiction. One immediate consequence of this invariant is that each node is placed in at most one cluster.

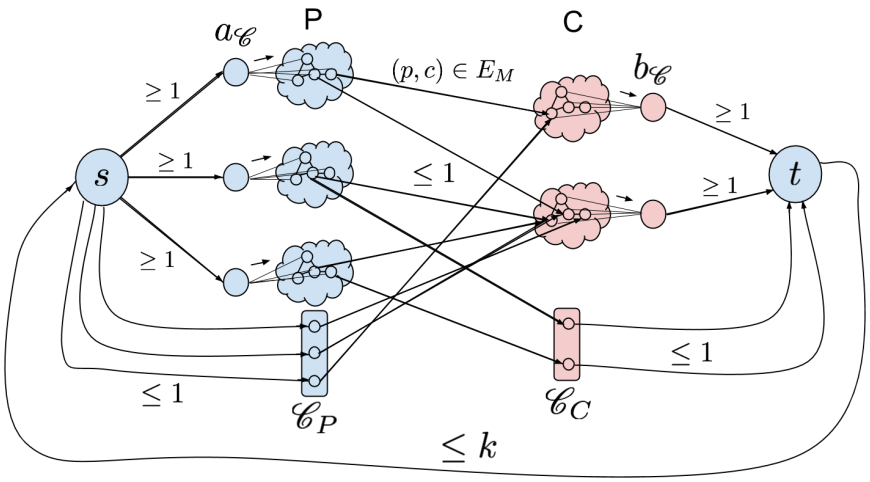


Figure 4.5: Example of the circulation network used to round the linear program solution.

**Lemma 4.1.2** *The clustering algorithm produces at most $k$ clusters.*

**Proof** Consider one of the clusters $\mathscr{C}$, and suppose it was formed by the selected unmarked node $v$. By the feasibility of $(x^*, y^*, m^*)$, we have that $\sum_{u \in V:(u,v) \in E_V^+} y^*_{(u,v)} = 1$, and hence, by construction, $\sum_{u \in \mathscr{C}} y^*_{(u,v)} = 1$. However, for each $u \in \mathscr{C}$, we also have that $x^*_u \geq y^*_{(u,v)}$; hence, for each cluster $\mathscr{C}$, we have that

$$\sum_{u \in \mathscr{C}} x^*_u \geq 1. \tag{4.9}$$

By the fact that $\sum_{u \in V} x^*_u = k$ and that the clusters contain disjoint sets of nodes in $V$, the claim follows.

Note that not all nodes are included in some cluster; that is, there might be nodes that were marked, but not included in any cluster; call those nodes $\mathscr{C}_P$ and $\mathscr{C}_C$, for $G_P$ and $G_C$, respectively.

Next we construct the circulation network as follows (see Figure 4.5). In addition to the nodes in $P \cup C$, we introduce a node $a_{\mathscr{C}}$ for each cluster $\mathscr{C}$ formed from $G_P$ (but

there does not exist a node corresponding to $\mathscr{C}_P$), a node $b_{\mathscr{C}}$ for each cluster formed from $G_C$ (again, not including $\mathscr{C}_C$), and two final nodes $s$ and $t$. For each $(p, c) \in E_M$, we introduce the corresponding arc $(p, c)$ of a capacity 1. For each "true" cluster $\mathscr{C}$ in $G_P$, we introduce an arc $(s, a_{\mathscr{C}})$ with a lower bound requirement of 1, and for each node $u \in \mathscr{C}$, we introduce an arc $(a_{\mathscr{C}}, u)$ of capacity 1; for the leftover cluster $\mathscr{C}_P$, we introduce an arc $(s, p)$ for each $p \in \mathscr{C}_P$ with (upper bound) capacity of 1. Similarly, for each "true" cluster $\mathscr{C}$ in $G_C$, we introduce an arc $(b_{\mathscr{C}}, t)$ with a lower bound requirement of 1, and for each node $v \in \mathscr{C}$, we introduce an arc $(v, b_{\mathscr{C}})$ of capacity 1; again, for the leftover cluster $\mathscr{C}_C$, we introduce the arcs $(c, t)$ for each $c \in \mathscr{C}_C$, again with a capacity of 1. We introduce a "back" arc $(t, s)$ of capacity $k$.

We can construct a flow by first assigning a flow of value $m^*_{(p,c)}$ on each arc $(p, c) \in P \times C$ in the network, and then simply extending this by enforcing flow conservation requirements so that the net flow into each node is equal to the net flow out. The feasibility of the linear programming solution means, in particular, that constraints (4.4) are satisfied, and so the flow on each arc $(a_{\mathscr{C}}, u)$ is $x_u$; similarly, by (4.5), the flow on each arc $(v, b_{\mathscr{C}})$ is $x_v$. But then, by the properties argued about the clustering algorithm, and in particular (4.9), we have that flow constructed is feasible.

Since there is a feasible fractional flow, there must also be an integral feasible flow, and this yields the desired matching. It remains only to show that each unmatched node can be supplied within the claimed distance. By construction, the flow selects at least one node from each cluster. Consider when a cluster is formed. By the definition of $E_V$, $V \in \{P, C\}$, each adjacency considered in forming that cluster requires that the pair of nodes be at most $d_V$ apart. Hence, for each node in the cluster, each node marked by forming this cluster is within a distance at most $3d_V$ of each node in the cluster. Hence, by considering each value $\delta$ that is equal to $d(u, v)$ for either both $u, v$ in $P$ or both in

$C$, and then setting $d_P = d_C = \delta$, and then determining the minimum $\delta$ for which the linear programming relaxation is feasible, we have the following theorem.

**Theorem 4.1.3** *There is a polynomial-time algorithm for the Mid-Rush Pairing Problem that finds a solution of objective value at most three times the optimum.*

A natural generalization of the problem Mid-Rush Pairing Problem would include an additional constraint on the total length of the pairing (in additional to requiring that the distance between paired nodes is at most $T$); let $D$ be the bound limiting the total length of the matching. We shall call this the *Weighted Min-Rush Pairing Problem*. This can be added to the integer programming formulation by the constraint that

$$\sum_{(p,c)\in E_M} d_{(p,c)} m_{(p,c)} \leq D.$$

It is completely trivial to generalize the previous proof to yield a 3-approximation algorithm for this more general optimization model. The only change needed is to include the additional constraint in the linear programming relaxation, and then to consider a minimum-cost circulation network, instead of the one above that just required finding a feasible circulation. The cost of each arc is 0, except for each arc $(p, c) \in P \times C$, which has cost $d_{(p,c)}$. The construction above ensures the existence of feasible circulation of cost at most $D$, and hence there must be an integral one as well.

**Corollary 4.1.4** *There is a polynomial-time algorithm for the Weighted Mid-Rush Pairing Problem that finds a solution of objective value at most three times the optimum.*

### 4.1.3 Empirical Analysis

The simplicity of the approximation algorithm lends itself to implementation. To compute routes for the New York system to operate we implemented the stated algorithm. In comparison to the LP lower bound over 60 instances generated from system data with different values of $k$ and different distance limits the algorithm yielded an average factor of $1.427$. For the same set of instances a MIP model, where the problem is solved to optimality achieves the value of the LP in all but one instance, where it is a factor of $1.00979$ off.

## 4.2 Overnight Rebalancing

The majority of rebalancing operations occur overnight. From our analysis of system data and underlying dammed we have computed the desired state of the system for start of the morning rush-hour. Previous work has attempted to compute routes for rebalancing trucks that allow a fully general pattern of pickups and drop-offs, and specify the number of bikes to be collected from and dropped to each station. We take a different approach that is motivated by working closely with the operators of the New York bike-sharing system. We restrict ourselves to moving truckloads of bikes. With this restriction, we formulate a model to optimize the use of a given-size fleet of rebalancing trucks. We derive an IP formulation that is reasonably tractable for fleets with a small number of trucks, and then provide a heuristic approach that takes advantage of the fact that the IP finds high quality solutions for the 1-truck special case. Routes generated by the approaches described in this section are in daily use by the operators of Citibike.

During the overnight rebalancing shift, the goal is to get the system ready for the

morning rush hour. We aim to have all stations at their desired fill level as specified by a balancing plan. Often this is an unrealistic demand as the resources available for rebalancing overnight are inadequate to achieve this. This motivates the problem of getting the system as close as possible to this state with the resources available. To achieve this, we compute a set of routes for rebalancing trucks that optimize the number of stations rebalanced. We limit these routes to only move full truckloads of bikes. Previous approaches have focused on the number of bikes to move between stations. From analyzing system state at the beginning of the overnight shift we observed that it is desirable to only move full trucks of bikes. A full truck of bikes is, in most cases, enough to bring a station to the required level. Also, the travel time in Manhattan can dominate the loading time of a truck, making it desirable from an operational standpoint to only move full truck loads of bikes. Finally, the simplicity of the instructions needed to implement full truckload routes is an important element in the practicality of this approach. Using these observations, we formulate the problem as trying to find a set of truck routes that rebalances as much of the system as possible in the time available. We route trucks in a bipartite graph, where one node set consists of stations with a surplus of bikes and the other of stations with a deficit of bikes. We now formally define the *Overnight Rebalancing Problem*. The intuition for this model is that we want to have routes for trucks that alternate between surplus and deficit stations. The distance between a surplus and a deficit station takes into account both the travel time and loading/unloading time. The time limit is determined by the length of a shift operated by rebalancing staff.

**Definition** *Overnight Rebalancing Problem* We are given a complete bipartite graph $G = (P \cup M, E)$, a number of trucks $T$, a non-negative metric distance function $d(e)$ for each $e \in E$, and a distance limit $L$, and the aim is to find $T$ vertex-disjoint paths $\mathscr{P}$, each starting in $P$ and ending in $M$, such that $\forall p \in \mathscr{P}, \sum_{e \in p} d(e) \leq L$, so that the total number of vertices visited by at least one path is maximized.

### 4.2.1 Mixed Integer Programming Solution

Given the importance of having the bike share system in a good state before the morning rush hour it is crucial to be able to quickly produce high quality routes for the overnight rebalancing shift. To achieve this we tackled the Overnight Rebalancing Problem from an empirical perspective.

$$\max \sum_{t\in\{1,\ldots,T\}} \sum_{v\in P\cup M} z_v^t \qquad \text{subject to} \tag{4.10}$$

$$\sum_{p\in P} r^t_{(d_{start},p)} = 1 \qquad \forall t \in \{1,\ldots,T\} \tag{4.11}$$

$$\sum_{m\in M} r^t_{(m,d_{end})} = 1 \qquad \forall t \in \{1,\ldots,T\} \tag{4.12}$$

$$\sum_{u\in\mathcal{N}(v)} r^t_{(u,v)} = \sum_{u\in\mathcal{N}(v)} r^t_{(v,u)} \qquad \forall v \in P\cup M \quad \forall t \in \{1,\ldots,T\} \tag{4.13}$$

$$\sum_{e\in E} r^t_e \cdot d(e) \le L \qquad \forall t \in \{1,\ldots,T\} \tag{4.14}$$

$$\sum_{t\in\{1,\ldots,T\}} z_v^t \le 1 \qquad \forall v \in P\cup M \tag{4.15}$$

$$\sum_{u\in P\cup M} r^t_{(u,v)} = z_v^t \qquad \forall v \in P\cup M \quad \forall t \in \{1,\ldots,T\} \tag{4.16}$$

$$f^t_e \le r^t_e \cdot |P\cup M| \qquad \forall e \in \bar{E} \quad \forall t \in \{1,\ldots,T\} \tag{4.17}$$

$$\sum_{u\in\mathcal{N}(v)} f^t_{(u,v)} = \sum_{u\in\mathcal{N}(v)} f^t_{(v,u)} + z_v^t \qquad \forall v \in P\cup M \quad \forall t \in \{1,\ldots,T\} \tag{4.18}$$

$$r^t_{(u,v)} \in \{0,1\} \qquad \forall (u,v) \in \bar{E} \quad \forall t \in \{1,\ldots,T\} \tag{4.19}$$

$$f^t_{(u,v)} \in \{0,\ldots,|P\cup M|\} \qquad \forall (u,v) \in \bar{E} : v \notin \{d_{end}\} \quad \forall t \in \{1,\ldots,T\} \tag{4.20}$$

$$z_v^t \in \{0,1\} \qquad \forall v \in P\cup M \quad \forall t \in \{1,\ldots,T\} \tag{4.21}$$

One approach to solving the Overnight Rebalancing Problem is by formulating it as an integer programming problem; we give a formulation for which standard IP soft-

ware typically computes high-quality solutions for modest-sized inputs within reasonable time bounds. Given the input graph $G = (P \cup M, E)$, we construct an augmented (directed) graph $\bar{G}$: we start with the input bipartite graph, bidirecting its edges, and add a start *depot* vertex $d_{start}$ as well as a finish *depot* $d_{end}$. In addition to two directed copies of each edge $(u, v) \in E$, there is an edge from $d_{start}$ to each vertex in $P$ and an edge from each vertex in $M$ to $d_{end}$. Let $\bar{E}$ denote the augmented set of edges, and for each $u \in P \cup M$, we let $\mathcal{N}(u)$ denote the set of vertices $v$ for which there exists an edge $(u, v) \in \bar{E}$. The optimization model is to maximize the number of stations rebalanced by a fleet of $T$ trucks, where each truck is alotted a total distance of $L$ that may be traversed from its first (positive) node in $P$ to its last (minus) node in $M$, where the route alternate between nodes in $P$ and $M$. In constructing an integer programming formulation, there will be three type of integer variables. First, there are 0-1 variables $z_v^t$ that indicate whether the truck $t \in \{1, \ldots, T\}$ rebalances node $v$. We also have a set of 0-1 variables $r_{(u,v)}^t$ that indicate whether the edge $(u, v)$ is on the route traversed by truck $t$, $t = 1, \ldots, T$. Hence, it is natural to have flow conservation constraints (4.11), (4.12), and (4.13), which ensure, respectively, that the path starts at a node in $P$, ends at a node in $M$, and that whenever the path enters a node, it must exit that node as well. Similarly, it is natural to have the length constraint (4.14) to upper bound the length of the path, and disjointness constraint (4.15) to ensure that at most one truck rebalances a given node $v$. Finally, it is clear that the node $v$ is visited by $t$, if there is some edge $e = (u, v)$ for which $r_e^t = 1$; hence, we get the constraints (4.16).

However, if one considers the feasible solutions to just these constraints, then it is easy to forget that a feasible 0-1 solution for $r^t$ might not correspond simply to a path, but to a path plus a collection of cycles. The role of the final set of variables is to enforce that the only nodes that are serviced by $t$ are those nodes on the path indicated by $r$. For each edge $e = (u, v) \in \bar{E}$, where $u \in d_{start} \cup P \cup M$, and $v \in P \cup M$, the integer

variable $f_e^t$ counts the number of nodes in $P \cup M$ yet to be traversed in its path (and so, for example, if the path for truck 1 rebalances 8 nodes, starting node $u \in P$, then $f^1_{(d_{start},u)} = 8$). First, each variable $f_e^t$ is positive (and of course is at most $|P \cup M|$) only when $r_e^t$ is 1; this is enforced by constraints (4.17). Finally, if the count entering node $v$ is $\ell$, then the count exiting it $\ell - 1$ (provided the truck traverses node $v$); this is exactly captured by the constraint (4.18). And notice the effect of this constraint on a potential cycle selected by the variables $r$; its corresponding $f$ value must decrease by 1 for each edge traversed in the cycle, but clearly this is impossible. Hence, this additional set of flow variables and constraints preclude the possibility of selecting cycles.

### 4.2.2 Heuristic Approach

As the number of trucks increases, the time it takes to solve the IP increases. From experimental results shown below, one truck is solvable by the IP, whereas larger number of trucks require more than the modest time limit given to the solver. This leads us to investigate a heuristic approach to the problem, specifically a greedy algorithm. In this greedy algorithm, we repeatedly solve the IP for one truck, and then remove those vertices from the graph and solve again. Removing the vertices covered by the route is valid, since the bipartite graph is complete, and once chosen no other truck will be able to use the removed vertices.

Framing this optimization problem as a covering problem allows us to analyze properties of the greedy heuristic. In this case we are choosing a subset from a ground set of all possible truck routes, paths starting in $P$, ending in $M$ of distance at most $L$, to cover another set, the set of all vertices. Given a subset of truck routes, the number of vertices covered is equal to twice the number of $(p, m)$ pairs covered by trucks, where each $p$

and each $m$ can appear in at most one pair. To compute this, consider ordering the truck routes and take all pairs defined by the first route. For all subsequent routes, if a vertex on the route has already been visited we shortcut the pair it belongs to in the route and continue. We observe that this objective function is submodular (Lemma 4.2.1). This property allows a greedy approach, where at each step the best possible route for a truck is taken, to yield a $\left(1 - \frac{1}{e}\right)$ solution, as shown in Nemhauser, Wolsey and Fisher [21].

Typically, when maximizing a submodular function over a finite ground set, at each stage the element from the set that increases the objective function the most is added. However, in this case, the ground set is exponential in size requiring a different approach than iterating through the elements of the ground set. Given a problem instance, it is clear that solving the problem for one truck yields the best route from the ground set. For the following iterations, by removing routes we have already taken, the best route for one truck is equal to the best element from the ground set to add. Thus solving the 1-truck IP to find the best route at each stage gives us a $(1 - \frac{1}{e})$ approximation to the original problem (though albeit without any guarantee on how efficient the algorithm is).

**Lemma 4.2.1** *The function mapping a set of paths of length at most $L$ that start in $P$ and end in $M$ to the number of $(p, c)$ pairs (where each $p$ and each $m$ can appear in at most one pair) visited by at least one path is monotone submodular.*

**Proof** Consider adding a path to the set of paths and re-evaluating the objective function. There can be no decrease in the objective function as we are not removing any paths. Given two sets of paths $A, B$ with $A \subseteq B$ we can not gain more by adding another path to $B$ than by adding it to $A$ as any pairs visited by $A$ are visited in $B$ and thus and shortcutting that must take place in A must take place in B.

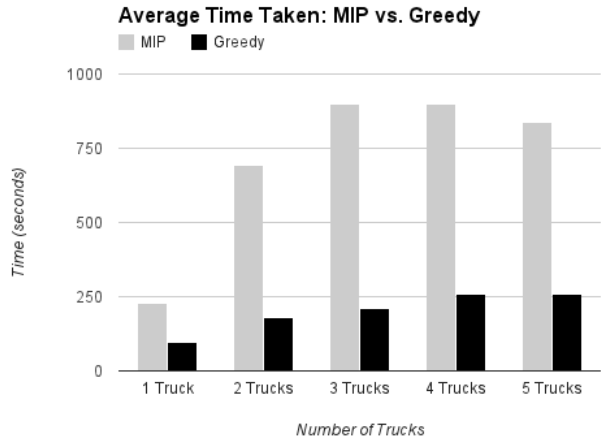


Figure 4.6: Average time taken by the IP and greedy approaches for different numbers of trucks.

### 4.2.3 Experimental Results

We tested the integer-programming approaches on real-world instances gathered from actual system data. We implemented the IP in Gurobi and carried out a number of experiments. To generate the real world instances we took a series of system snapshots of the system state at the 8pm start of the overnight shift during June 2013. We used a standard plan for the system state at the start of the morning rush hour to compute the stations that make up the bipartite graph, specifying the surplus and deficit nodes $P$ and $M$. With station location GPS information, we can compute an estimated travel distance between the stations. For each instance we vary the number of trucks available for rebalancing and analyze both the runtime taken by the solver as well as solution quality; in total, the data set contained 50 instances. We ran the MIP with a 900 second cutoff and restricted the greedy to use only 300 seconds for each greedy call to the IP. The results can be seen in Figures 4.6 and 4.7. From this it is clear that as the number of trucks increases, the greedy approach produces higher quality solutions in less time than the IP. In Figure 4.7 we compare the solutions returned by both methods to the

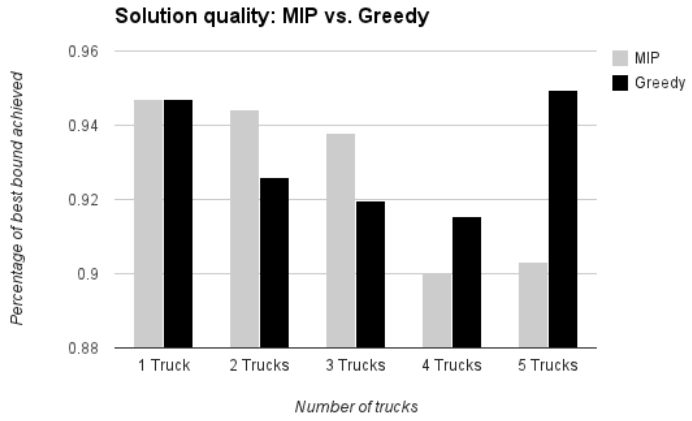


Figure 4.7: Solution quality found by the IP and greedy approaches for different numbers of trucks.

best bound found by the solver in 900 seconds. It is interesting to note that although one can solve the 1-truck inputs to optimality (say, with an hour of computation time), it is typically the case that this only shows the incumbent solution to be optimal. The performance of the greedy solution, both in terms of time taken and solution quality allows it to be used in practice.

We also conducted experiments on randomly generated instances where we fix the number of trucks at $5$ (a realistic number of available trucks) and we vary the number of $P$ and $M$ vertices. The results of this experiment are shown in Figures 4.8 and 4.9. Again the greedy approach outperforms the MIP. The performance of the greedy solution, both in terms of time taken and solution quality allows it to be used in practice.

## 4.3 Incentive-Based Rebalancing

Rebalancing through the use of trucks and bike trailers is resource intensive, requiring a large number of staff as well as a fleet of dedicated vehicles. Rebalancing itself is akin

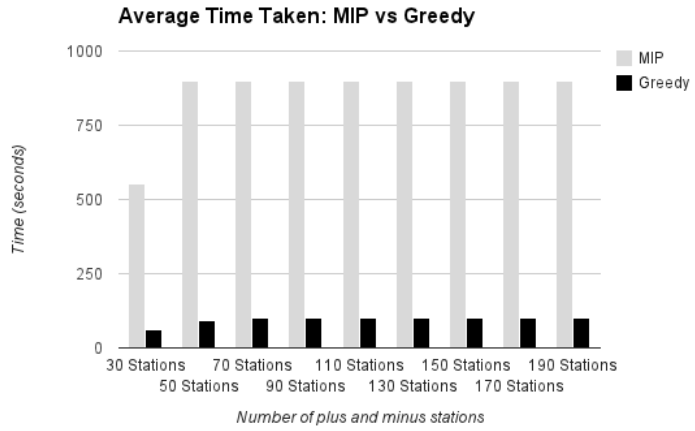


Figure 4.8: Average time taken by the IP and greedy approaches for different instance sizes in random instances.

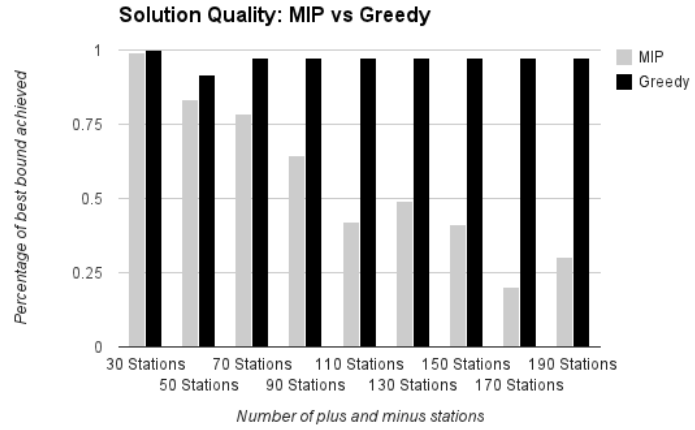


Figure 4.9: Solution quality found by the IP and greedy approaches for different instance sizes in random instances.

to attempting to hold back the tide, if somehow enough resources are available to ensure the system is balanced, the availability of bikes and docks will drive an increase in usage and thus require more rebalancing. This vicious cycle is prohibitively expensive, in fact the cost per bike move can run into amounts greater than a dollar.

As an alternative to vehicle based rebalancing incentivizing users to change their behavior may be far more scalable. Offering users a monetary (or otherwise) incentive to shift their behavior slightly could have a markable impact on system imbalance. This is as a result of a number of areas in New York that, although geographically close experience completely opposite usage patterns. Consider, for example, the stations surrounding Penn station, during the evening rush-hour these stations will experience a huge influx of bikes and shortages of space to put bikes are common, only two blocks away on 6th Avenue as commuters leave their office buildings stations are chronically short of bikes. Our goal is to design an incentive system that will encourage riders to shift their behavior slightly to benefit the system, if a commuter has ten minutes to spare on their way to Penn station we want to reward them for walking two blocks.

Due to the *all you can eat* subscription plan in place in New York where subscribers pay a yearly fee for unlimited access to the system if is difficult to use price as a lever to shift behavior. If trips were priced on a per trip basis a congestion pricing scheme akin to Uber surge pricing could be developed to alter behavior. However, with a flat subscription model, increasing prices on a trip by trip basis is impossible, this leaves us needing to give users a reward or create an alternate currency

### 4.3.1 Raffle Based Incentive Scheme

Following from the work of [19], which shows raffle based incentive schemes are far more effective than micro-incentive based systems, we designed a raffle based incentive system to Citibike. The high level concept is that by taking rides that benefit the system users earn raffle tickets, these can be seen as an alternate currency. Users can use the tickets they earn to enter daily raffles for cash prizes as well as using the tickets to buy Citibike merchandise in a system similar to that used by air mile rewards programs.

When designing the incentive scheme we focused on a series of simple goals:

- *Ease of understanding:* The most important aspect of the incentive scheme is that is it both transparent and easy to understand for users. We must ensure that if a user has taken the decision to take a good ride that firstly they will be rewarded and secondly it is easy for them to determine if a proposed trip benefits the system.
- *Difficult to Game:* An incentive scheme also needs to be resistant to attempts to game it. It is important to ensure that when paying out rewards for good trips that these trips were indeed good for the system.
- *Reward small shifts in behavior:* Citibike experiences massive usage during rush-hour periods. We want to alleviate outages by encouraging people to shift their behavior slightly as opposed to encouraging people to take trips they otherwise would not have.

As a first approach, we developed a system based on breaking the city up into different zones. We operate the incentive scheme solely during the rush-hours with each rush-hour having its set of zones . Each of these zones is labeled as either a shortage zone, a surplus zone or a neutral zone. Shortage zones are areas of the city where stations experience chronic shortages of bikes during the rush-hour, the opposite is true for

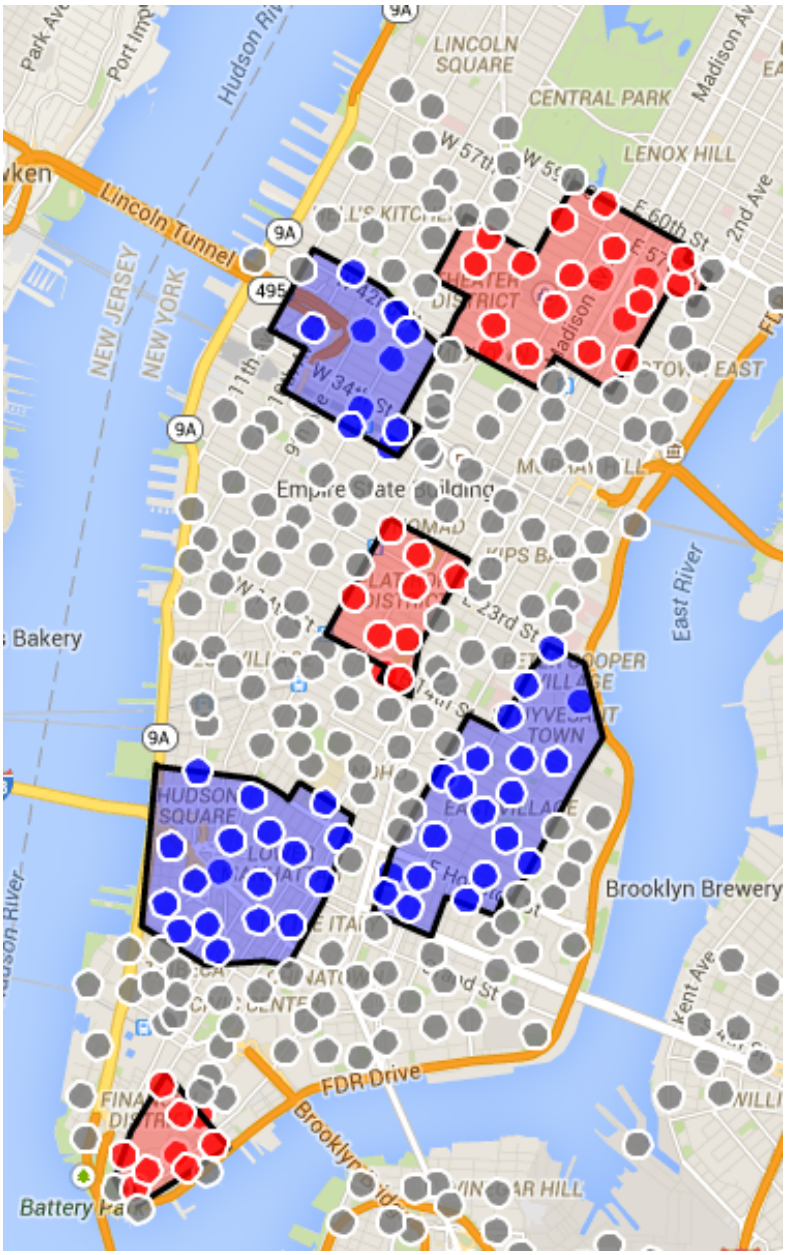


Figure 4.10: A map showing the different zones used for an incentive scheme. This map is for the evening rush hour where blue zones experience a surplus and red zones a shortage.

| ORIGIN \ DESTINATION | Shortage | Surplus | Neutral |
|---|---|---|---|
| Shortage | 1 | 0 | 0 |
| Surplus | **2** | 1 | 1 |
| Neutral | 1 | 0 | 0 |

Figure 4.11: Table showing the number of raffle tickets rewarded for trips based on their start and end zones.

surplus zones. Neutral zones are areas of the city, typically the borders between shortage and surplus zones that have self-balancing stations. An example of the zones used for the evening rush-hour is shown in Figure 4.10.

The quality of a trip is based on the type of zones the start and end stations were located in. We only reward users for trips that start in one zone and finish in another. The most desirable kind of trip is one starting in a surplus zone and ending in a shortage zone, however we also want to reward the trader leaving the financial district (a shortage zone) who has time to spare and might return the bike at another shortage zone (midtown) instead of the surplus zone at Penn station. We allocate a number of tickets depending on the quality of the trip. The matrix of start and end station zones and the number of tickets allocated to each kind of trip can be seen in Figure 4.11.

To prevent gaming of the system we restrict users to be rewarded for only trip during each rush-hour period. Although this might seem restrictive at first and rewards based on net impact an attractive alternative consider the following attack. A user owns two keys, they will find two stations that are geographically close and for which a trip between them yields raffle tickets. They then use the *good* key to take the trips that yield

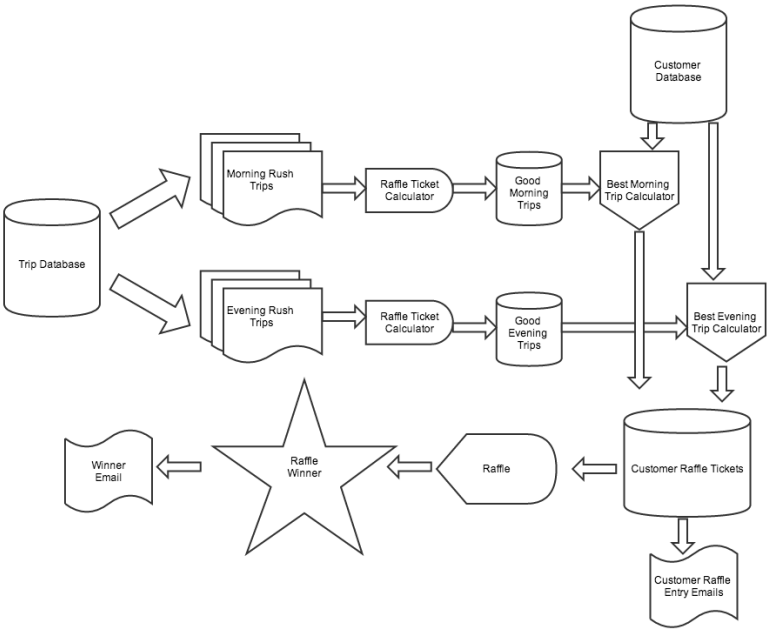


Figure 4.12: Flow diagram showing the operation of the raffle scheme.

raffle tickets upon returning the bike they then use their *bad* key to take the same bike out returning to the original system. This attack can be carried out repeatedly during one rush hour and the account associated with the good key will accumulate a large amount of raffle tickets. Such attacks are prevented by only rewarding users for one good trip during a rush-hour, this also adheres to the principal of rewarding small shifts in behavior.

Once a day after the end of the evening rush-hour the database will be scraped and all good trips calculated. A raffle is then carried out for a large prize (cash or otherwise) and users know each day if they have won a prize due to their rebalancing efforts. A flow diagram showing the operation of the system can be seen in Figure 4.12.

The appeal of a raffle system with a cash prize is that is hides the expected cost per bike moved by an incentive scheme. If the incentive scheme were to simply pay a small amount of cash per bike moved, users might react poorly to a reduction in the rate. In a

raffle based system however the size of the pool of entrants is not known to the entrants themselves and the cost can be easily reduced by decreasing the frequency of the raffle but increasing the amount less. For example, instead of running a raffle every day for five hundred dollars run the raffle every three days but for a prize of only one thousand dollars. To ensure that users are not discouraged by not winning the main raffle we propose a marketplace where raffle tickets can be exchanged for Citibike merchandise. This is a model similar to that used with air-miles rewards schemes. This also reinforces the idea of using rebalancing to create an alternative currency.

CHAPTER 5

# SIMULATION TOOLS FOR EVALUATION

Validating the impacts of our work with Citibike is not a straightforward problem. Since we started our collaboration right as the system launched in May 2013, we have continuously refined the approaches we use and the models we employ. As our work has matured, Citibike staff have gained experience and learned lessons on running bike share systems. Simply put, as we have gotten better, so have they. This means that it is difficult for us to get a true sense of the impact of some interventions. Due to political considerations, it is not feasibly to cease rebalancing work for a length of time to perform an A-B test. This difficulty in evaluating the impact of our contributions also arises when considering new rebalancing strategies. For example, real estate in New York is expensive and the operators of Citibike would like to know if getting access to storage space for bikes in a high demand area of the city would be worthwhile before signing a contract and committing resources. To attempt to answer these questions, we built a discrete-event simulator to simulate a typical weekday in New York.

## 5.1 Simulation Framework

We have built a discrete-event simulator to model the Citibike system. In the simulation, the goal is to model the impact on station outage levels and numbers of successful trips that different rebalancing actions have. We model on the station level and simulate events on a minute by minute basis. At each minute every station uses a randomized process to generate trips. These trips are then attempted in the system; if a trip is not possible, due to the station being empty it is recorded as a failed trip. For successful trips, we use another randomized procedure to estimate their duration. Finally, the successful trips are started in the system. When a trip is finished, we attempt to return the bike to the

station; if this is successful, the trip is closed and recorded as a successful trip. However, if there is not an available dock for the bike we trigger logic to handle a *different end* trip.

### 5.1.1 Generating Trips

We use the following procedure to generate trips in the simulation. For every station we have a random variable corresponding to the number of trips taken at a given minute, we represent this a Poisson random variable

$$O_i^t. \tag{5.1}$$

This is the number of trips starting at station $i$ at minute $t$. We use data from the Citibike system of weekday rides during the peak season to infer the means of these distributions. From observation of the raw data, we believe it is a reasonable assumption that the trips out per minute is Poisson.

To find the destinations of each of these trips, we sample from the multinomial distribution representing the probability of the trip ending at each of the other stations. If $P_{ij}^t$ is the probability that a trip starting at station $i$ at minute $t$ ends at station $j$ we have

$$\sum_j P_{ij}^t = 1 \tag{5.2}$$

We let

$$E_i^t \tag{5.3}$$

be the random variable representing the end point of a trip started at station $i$ at minute $t$. To ensure that the distribution of end points is not biased to a small number of stations

we build it from a twenty minute time-window around minute $t$. The last remaining property of a trip we are concerned with is the trip duration. Again, this is sampled from a distribution generated from observed trip data. For point-to-point locations we look at the distribution of trip times between these stations. Again, from observation this assumption to model this process with a Poisson distribution is reasonable; thus we have the random variable

$$D_{ij}^{t} \tag{5.4}$$

representing the duration of a trip between stations $i$ and $j$ started at time $t$.

### 5.1.2 Failed Trip End Logic

When a trip reaches the end location and there are no empty docks available, we first record this occurrence. We then aim to simulate the user checking their smartphone to find a nearby station that has an open dock. We find the nearest station that has an open space and send the trip there, to find the travel time of this trip we use the random variable $D_{ij}^{t}$. We also maintain a list of stations previously visited while trying to find a space for a bike and never attempt to dock at a station we have already failed at. If the list of stations visited grows too long, we simulate the user abandoning the bike and record this event.

### 5.1.3 Rebalancing

Evaluating the impact of rebalancing actions is far more of a challenge than evaluating the impact of different fill levels. This is due to the need to incorporate vehicle-based rebalancing into the simulation and also due to limitations of the simulation framework.

To incorporate vehicles into the simulation we need to know the routes they are taking, where they are picking up and dropping off bikes and the expected time it will take to travel and load the vehicle. Also needed is logic for when the vehicle can load and unload bikes; for trailers that are rebalancing during rush-hours we ensure that by rebalancing we do not cause outages in the system, leaving a minimum number of bikes and spots at each station.

However a difficulty in evaluating the impact of rebalancing is in the effect it has on customer behavior. Customer behavior can shift due to rebalancing at least two ways. The first being an immediate change in behavior due to the availability of bikes or spots at a customer's favored station. Customers have access to station level information through smartphone apps and can use this information to decide where to start and end trips. The simulation framework detailed here makes assumptions about the independence of station levels and trip likelihood that restrict its ability to capture this shift in behavior.

Another, more longterm, impact that rebalancing has in ridership is that of shifting usage patterns. As rebalancing increases and customers can more reliably take and return bikes at stations ridership increases as Citibike becomes more reliable as a mode of transport. For example if rebalancing is focused on only one of the stations surrounding Penn Station over time users will identify this trend and finish their trips at that station, reducing the inflow to the other surrounding stations. These shifts in user behavior are not supported by our simulation framework. To properly capture these behaviors, a more comprehensive user decision model is needed.

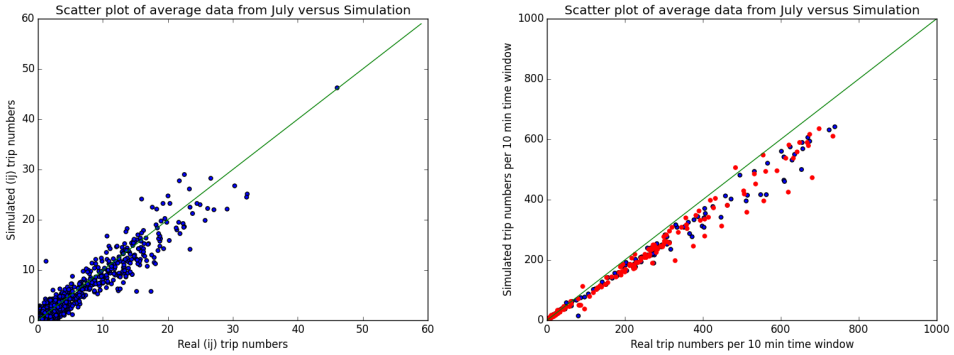


Figure 5.1: Scatter plot comparing trips between pairs of stations averaged across July 2014 compared to an average over runs of the simulation on the left. The right scatter plot shows trips per hour both in and out, red and blue respectively.

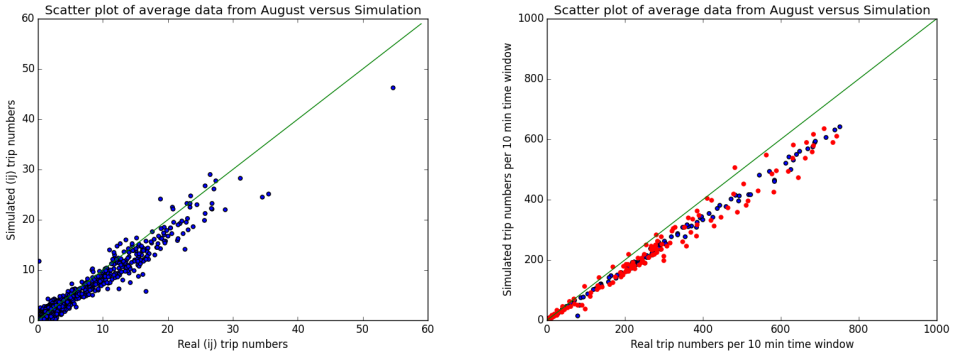


Figure 5.2: Scatter plot comparing trips between pairs of stations averaged across August 2014 compared to an average over runs of the simulation on the left. The right scatter plot shows trips per hour both in and out, red and blue respectively.

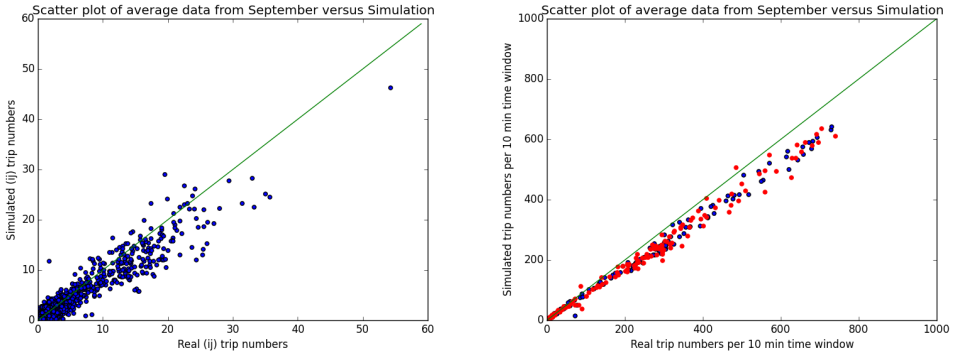


Figure 5.3: Scatter plot comparing trips between pairs of stations averaged across September 2014 compared to an average over runs of the simulation on the left. The right scatter plot shows trips per hour both in and out, red and blue respectively.

## 5.2 Validation of Simulation With Real Data

To ensure that the simulation framework used properly captures the dynamics of usage in New York we ran a number of tests against real data. In this test we created the distributions used to generate trips and run the simulation from trip data in August 2014. We then ran a number of simulations and gathered average trip data. We then compared this data to observed trip data from July, August, September and October 2014. For each month we produced two scatter plots, one comparing the amount of trips between all pairs of stations, the other comparing the number of trips started and ended in ten minute windows. These can be seen for July (Figure 5.1), August (Figure 5.2), September (Figure 5.3) and October (Figure 5.4). In these figures the simulation performs well with all scatter plots looking approximately linear. We believe these results justify the simulation framework as a means of testing interventions in New York.

As we have validated the performance of the simulation framework, we can now use it to evaluate different intervention strategies in New York. Throughout the rest of this chapter, we outline the evaluation used and test the approaches presented in this work.

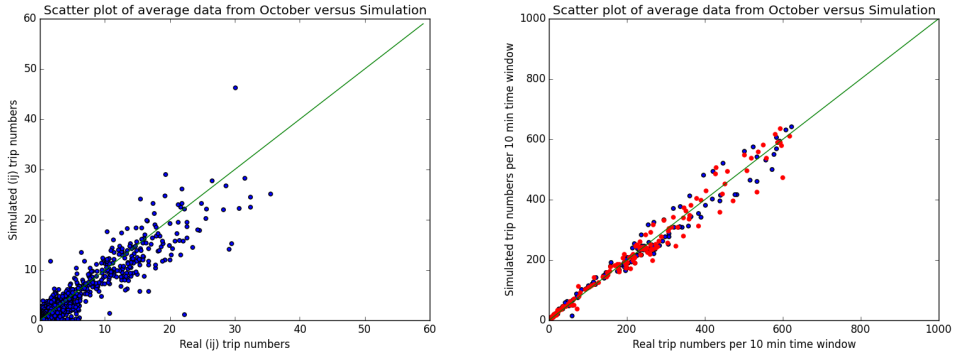


Figure 5.4: Scatter plot comparing trips between pairs of stations averaged across October 2014 compared to an average over runs of the simulation on the left. The right scatter plot shows trips per hour both in and out, red and blue respectively.

Although we focus on simulating weekday usage, by changing the data used to create the distributions discussed earlier this framework can be used to simulate weekend usage. In fact by changing both the distributions and the station information the framework can adapt quickly to different cities, as long as the same distributional assumptions made about New York usage are reasonable.

## 5.3 Evaluating Approaches in Simulation

To evaluate different rebalancing strategies or other interventions in New York we examine a number of factors. Firstly we set up experiments to compare different approaches averaged over a series of runs of the simulation. For each run we simulate usage from midnight to midnight, we then record statistics and average them over the number of runs.

To evaluate the different approaches we use the following metrics:

- *Successful Trips*: Number of trips successfully completed in the system, a bike

was available at the desired start location and also at the desired end location.

- *Completed Trips*: Number of trips finished in the system, this included trips where an alternative end location was used.
- *Total Trips*: The Total number of trips attempted by the simulation.
- *Different Ends*: The number of trips where an alternative end point was needed to complete the trip
- *Failed Ends*: The number of trips where the bike was *abandoned* due to too many alternative end points being attempted.
- *Failed Starts*: The number of trips that could not happen because of no available bike.
- *Outage Minutes*: The total number of outage minutes (minutes a station was either empty or full) of all stations,
- *Full Minutes*: The total number of minutes stations were full during the simulation.
- *Empty Minutes*: The total number of minutes stations were empty during the simulation.

Given the size and complexity of a bike-sharing system it is difficult to know when one approach outperforms another. To simplify this we use some basic observations, first being that failed starts are far better than different ends of trips. Specifically it is much worse for a customer to have a bike and have trouble finding a place for it than the customer having to find an alternative form of transport (our simulation currently does not model users choosing an alternate start point for their trip). Even more problematic is the number of failed ends in the system; these cause unsatisfied users and may also

precipitate the abandoning and loss of bikes, a very expensive occurrence for the system. In addition to problematic trips, we look at the amount of station outage minutes occurred, both full and empty amounts.

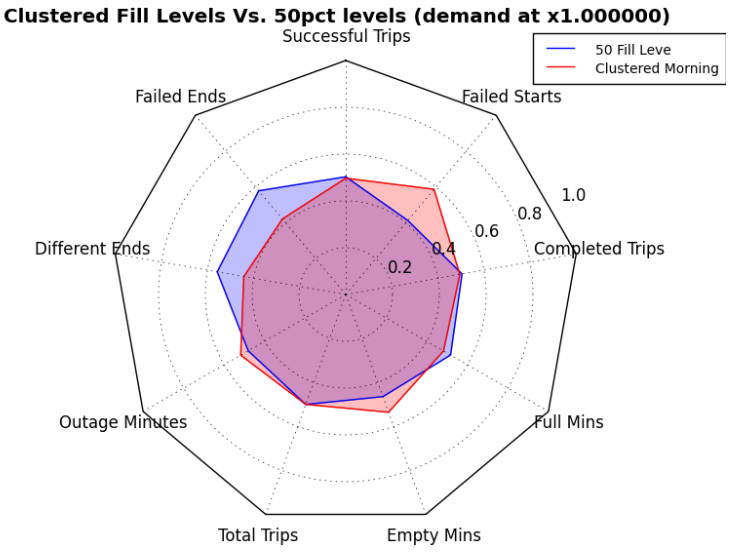


Figure 5.5: Radar chart comparing average simulation runs of a day in New York starting with a 50% fill level and with levels defined by the clustering approach presented in Chapter 3.

To present these different metrics and allow easy comparison between approaches we use radar plots. An example of one comparing a uniform $50\%$ fill level to the fill levels derived from clustering stations based on their usage can be seen in Figure 5.5. Since the attributes compared have vastly different scales, we compare the approaches by making a convex combination of each pair of attributes. This ensures that each value is in the interval $[0, 1]$, and this allows each comparison between the attribute values. The relative shapes of the two approaches compared are a quick guide as to which approach is superior. In Figure 5.5, although the clustered approach yields more failed starts, it registers far fewer failed ends and different ends than the uniform $50\%$ fill level and has a similar number of both successful and completed trips despite registering slightly more outage minutes. These plots will be used for the remainder of the chapter to allow

quick comparison of different approaches.

## 5.4 Validating Pre-Rush Hour Fill Levels

This section will compare the different methods of allocating bikes before the morning rush-hour as described in Chapter 3. Each different method will be compared to the baseline of filling each station to a 50% fill level before the days use.

### 5.4.1 Clustering-based fill levels

The first method of finding pre-rush fill levels is that presented in Chapter 3. Stations are clustered according to their usage and each cluster is assigned a percentage fill level. The results of the comparison with a baseline of 50% fill levels everywhere is shown in Figure 5.6. In the figure different multipliers are put on the demand to simulate increased ridership and compensate for censored demand. In this figure, as demand increases, it is clear that the clustered levels approach far outperforms the baseline. This is particularly true when looking at the number of failed and different ends, which are situations that operators certainly want to avoid.

### 5.4.2 Minimizing Functions of Net Difference

The next series of plots shows the results of testing the different objective functions of net difference presented in Chapter 3. The first objective function tested is that of minimizing the maximum between what is placed at a station and the net difference of the flows. The results of this test are show in Figure 5.7. From these plots it is clear that

this is, in fact, far worse than filling every station to a 50% level, even when demand is increased.

The next objective function tested is that of minimizing the sum of the differences. The results of this test are shown in Figure 5.8. In these results, minimizing the sum of the costs outperforms the baseline in terms of failed ends and different ends but is far worse when looking at outage minutes and failed starts. There is also a slight reduction in the number of successful trips. Although the approach performs better as the demand is increased, it is still disappointing. The same is true for the final objective function based on the net difference, minimizing the square of the net difference. This test is shown in Figure 5.9, where the performance is very similar to that of minimizing the sum of the costs.

### 5.4.3 Continuous Time Markov Chain Optimized Levels

The final approach tested are the fill levels generated from the continuous time Markov chains as presented in Chapter 3. The results of this can be seen in Figure 5.10. Although the CTMC approach has more failed trip starts, it far outperforms the baseline in terms of failed ends and different ends while keeping the outages minutes very close. This performance improves as the demand is increased, leading us to conclude that this is the correct approach to generating fill levels.

This approach uses far fewer bikes in racks than filling the stations to 50%. Often the number of bikes available to the operators of Citibike fall far below this number. Using system data, we observed that often there are approximately four thousand bikes available to be placed at stations. Factoring this into account we re-optimized the placement using the CTMC approach with a new budget of four thousand bikes and tested

this against an approach where bikes are distributed equally between each station (such that each station has the same fill percentage). The results of this test are shown in Figure 5.11. Again, in this test the CTMC approach far outperforms the baseline with all demand multipliers.

These tests show that using continuous-time Markov chains to optimize bike placement is highly effective. It reduces outages when compared to other approaches and also reduces the crucial metrics of different ends and failed ends. These levels are in use in New York to direct rebalancing efforts; they are integrated into a web app that dispatchers use to direct rebalancing efforts and quickly view the system state.

## 5.5 Evaluating Rebalancing Effectiveness

To test the effectiveness of rebalancing actions we took the routes run in New York by bike trailers throughout rush-hours and added them to the simulation. This set of 14 pairs of stations correspond to heavily used stations for both rush-hours. The result of adding these stations to the simulation can be seen in Figure 5.12. These results show almost no difference between the two approaches and further our observation that the current simulation framework is insufficient to properly capture the effects of rebalancing.


Figure 5.6: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with a 50% fill level and with levels defined by the clustering approach presented in Chapter 3. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$

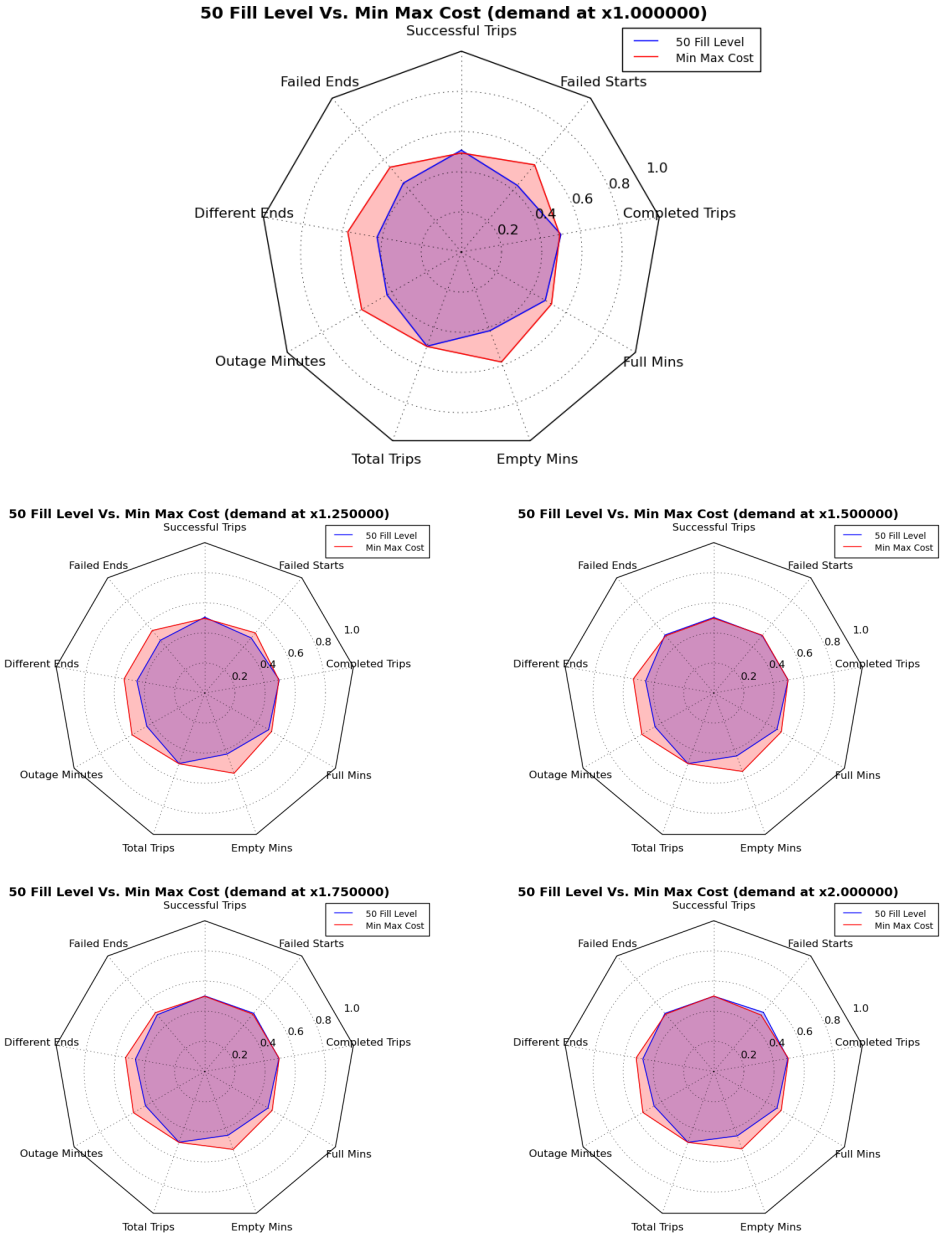


Figure 5.7: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with a 50% fill level and with levels defined by optimizing the objective function of $\min\max$ difference, presented in Chapter 3. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$

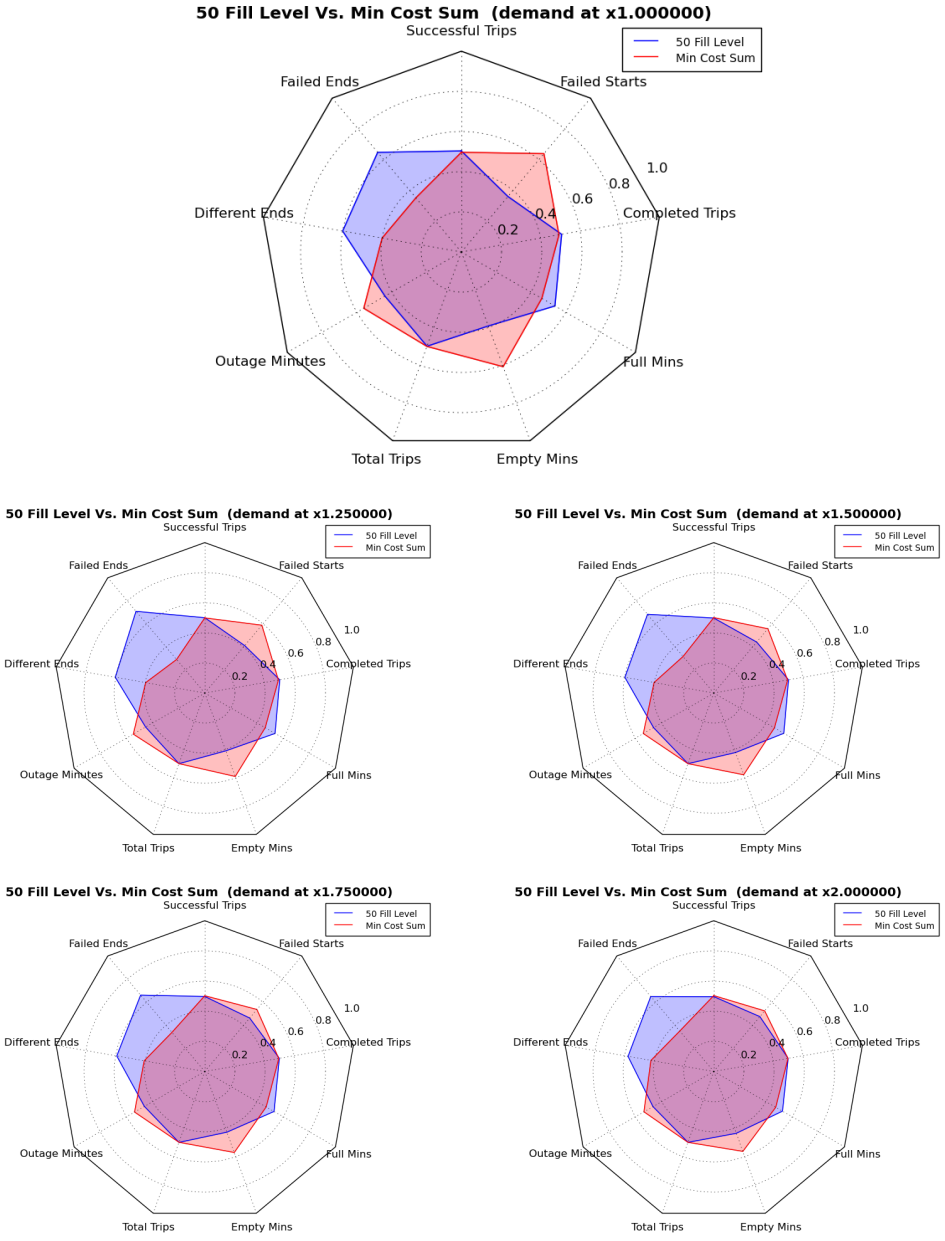


Figure 5.8: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with a 50% fill level and with levels defined by optimizing the objective function of $\min \sum$ difference, presented in Chapter 3. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$


Figure 5.9: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with a 50% fill level and with levels defined by optimizing the objective function of $\min \sum$ difference squared, presented in Chapter 3. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$

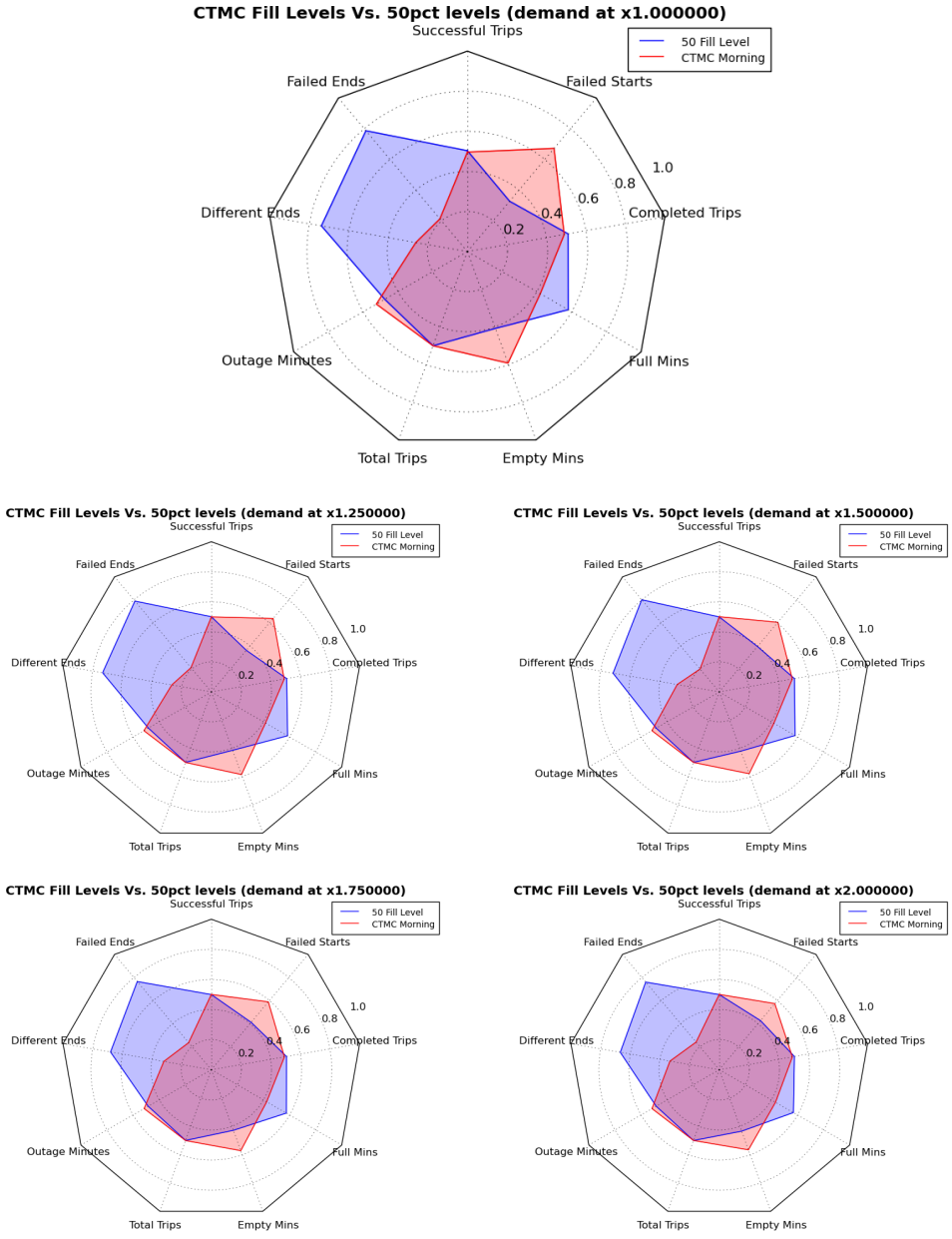


Figure 5.10: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with a 50% fill level and with levels defined by optimizing placement based on the continuous time Markov chain presented in Chapter 3. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$


Figure 5.11: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with an equal allocation of four thousand bikes and with levels defined by optimizing placement based on the continuous time Markov chain presented in Chapter 3. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$


Figure 5.12: Radar plots comparing average simulation runs of a day in New York with increased demand, starting with an equal allocation of four thousand bikes and with levels defined the continuous time Markov chain presented in Chapter 3. Two two runs being compared are one with trailer rebalancing between busy stations and one with no rebalancing actions. The demand factors are, from the top, $1, 1.35, 1.5, 1.75, 2$

# CHAPTER 6

# CONCLUSION

In this thesis, we provide a novel way of thinking about the management of bike-share systems. For this unique blend of human, resource and logistical factors we take a data-driven approach to optimization. Like many new application of optimization and analytics that have arisen from the explosion of available data, we use observations about data to inform modeling choices as well as using the data to direct our objective functions. These approaches allow us to take operations research from the steel mill and onto the streets.

In optimizing the planning of bike-sharing systems we employ methods from queueing theory and stochastic modeling to optimize the placement of both bikes and racks. We expand on the work of [23] with a model to place capacity in stations as well as bikes. The new results proved about properties of these functions allow their efficient optimization. This is an invaluable tool to planners when expanding, retrofitting or even creating systems. Even if not retrofitting the system, these models inform operators where resources are best allocated. For example, to simulate adding capacity staff in NYC *valet* stations where they chain bikes together on a sidewalk and ensure bikes and racks are available. Another potential intervention is the use of *depots* to store bikes to be introduced or removed from the system.

For rebalancing, our models are motivated by operational constraints and observations gleaned from a close collaboration with bike-share operators. We provide solution methods for our models that are sufficient to provide high quality, usable solutions to real world instances from New York City as well as providing provable guarantees on solution quality. Solutions to the mid-rush rebalancing problem are already in use in New York and the tools developed to solve the Overnight Rebalancing Problem are cur-

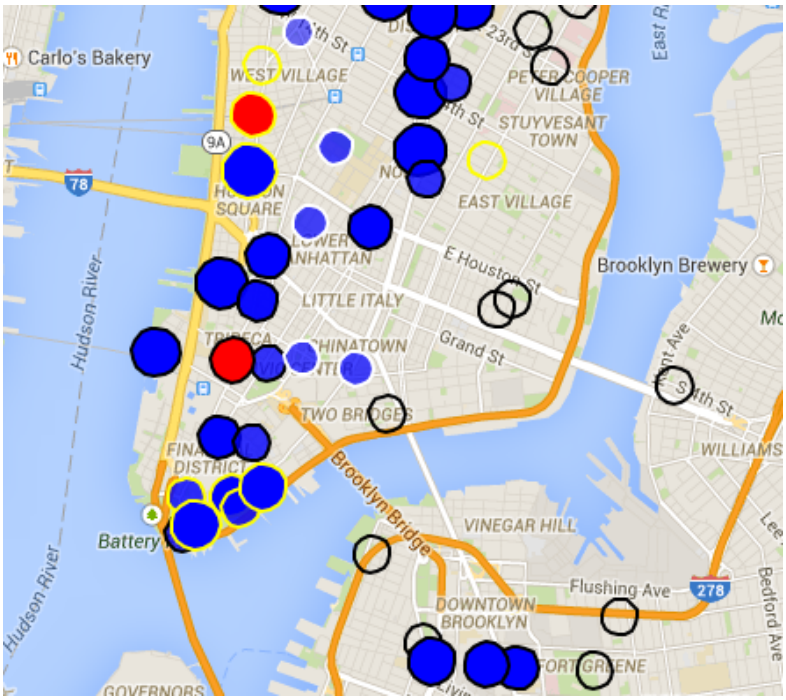


Figure 6.1: Example of the map used by dispatchers to direct rebalancing efforts to prepare for rush-hours.

rently being integrated into the truck dispatching system. Our approaches provide ”the overarching vision for how we like our system to look,” according to then director of operations Michael Pellegrino [33].

## 6.1 Integration Within Citibike

As well as novel models and approaches to planning and rebalancing bike-sharing systems, the contributions of this work include its implementation and impact on New York City. Implementing these ideas and integrating them within the workflow of Citibike was a key part of our collaboration with the operators of Citibike. This first required a cultural shift in the organization to make decision-making data driven. Integrating our work meant analyzing the workflow within Citbike and creating the necessary decision support tools.

The first modification made to the operation of Citibike was to shift overnight rebalancing to prepare for the morning rush-hour. We use the optimal fill levels generated in Chapter 3 to direct these rebalancing efforts. Rebalancing trucks are directed between stations by dispatch staff. These staff use maps of the system and a rebalancing plan to tell trucks where they are needed. To allow quick decision making by dispatch staff we created a web based interface that allows easy planning for rush hours. Stations that are far from their desired fill levels are highlighted on the map allowing dispatchers to quickly get a sense for the state of the system. A screenshot of the maps produced by this decision support tool is shown in Figure 6.1.

The tools developed for planning capacity in bike-sharing systems allow us to evaluate potential plans for expansion and retrofit of the existing system. By taking potential station locations and using predictive models such as [28], we can provide advice on where to locate capacity to best handle heavy usage. By developing predictive models for different cities these tools can also inform where to locate capacity when building new systems, a crucial tool to getting the most of new systems.

Optimizing rebalancing is currently being added to the decision support tools used at Citibike. Although some routes generated for bike trailers during rush-hours are in use in the city, a future goal is the integration of this optimization into a dispatch framework. This would allow both bike trailers and trucks to be routed automatically to where they are most needed, freeing up dispatch staff to focus on other issues such as broken bikes and required repairs. We have also implemented a proof of concept of the incentive scheme described in Chapter 4. This was done to show that the flow diagram described was feasible without extra requirements of the database.

The collaboration with New York Bike-Share LLC. is ongoing: a wealth of other operational challenges remain to be tackled, such as optimizing battery replacement for

stations and routing trucks to collect broken bikes. Furthermore, we are continuing to improve the solution methods for the models presented in this paper, refining both the computational results as well as improving the models through feedback from people in the field.

## 6.2 Open Questions and Future Directions

Bike-sharing and the wider domain of vehicle sharing present a vast number of interesting, relevant and worthwhile problems. The problems tackled in this work and the solutions we generated raised a number of interesting research questions.

Knowledge of how customers make decisions about how to use the system is crucial to truly optimize and validate rebalancing. Modeling user decision-making as a Markov decision process and using the data available on how users have interacted with the system to infer the individual MDPs is a fascinating question. The resulting MDPs, as well as illuminating the underlying decision procedures users make, are a generative model that would allow simulation frameworks to model individual behavior and yield a more accurate validation of interventions.

When implemented in New York the incentive scheme will provide a fascinating source of data. The extend to which users participate in the system as well as its impact on rebalancing in New York will give valuable information on the design of more advanced incentive schemes and potential alternative pricing mechanisms in other cities.

There are a number of open problems related to the models presented in this paper. From a theoretical perspective, it would be nice to have matching upper and lower bounds on performance guarantees for the Mid-Rush Pairing Problem. For the

Overnight Rebalancing Problem, a corollary of Lemma 4.2.1, in Chapter 4 is that an approximation algorithm for the one truck case of the Overnight Rebalancing Problem with a $(\alpha \cdot OPT)$ guarantee yields a $(1 - \frac{1}{e^{-\alpha}}) \cdot OPT$ solution when used in a greedy framework via the result of Goundan and Schulz [8]. Therefore providing a constant-factor approximation for the 1-truck case would yield a constant-factor approximation for the general case. Similarly we conjecture that the linear relaxation of the capacity placement integer program presented in Chapter 3 is integral. We believe that this is a consequence of the convexity and validity of the generated planes as well as empirical evidence.

This work is an exciting mix of theory, practice, data and implementation. The transition from data to conceptual model to implementation in New York was key to this work's success. The interplay of methodological tools, with the statistical models providing guidance for long-term issues, stochastic models providing structural insights, and discrete optimization models providing detailed implementation results, is likely to be a recurrent phenomenon in this new age of such data-driven decision-making environments.

## BIBLIOGRAPHY


[1] Shoshana Anily and Julien Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics (NRL)*, 46(6):654–670, 1999.

[2] Claudia Archetti, Luca Bertazzi, Gilbert Laporte, and Maria Grazia Speranza. A branch-and-cut algorithm for a vendor-managed inventory-routing problem. *Transportation Science*, 41(3):382–391, 2007.

[3] Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3):23, 2012.

[4] Daniel Chemla, Frdric Meunier, and Roberto Wolfler Calvo. Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120 – 146, 2013.

[5] Claudio Contardo, Catherine Morency, and Louis-Martin Rousseau. *Balancing a dynamic public bike-sharing system*. 2012.

[6] Juan Carlos Garca-Palomares, Javier Gutirrez, and Marta Latorre. Optimizing the location of stations in bike-sharing programs: A GIS approach. *Applied Geography*, 35(1?2):235 – 246, 2012.

[7] Barbara Goldberg. After 23 million rides, no deaths in u.s. bike share programs. *Reuters*, August 2014.

[8] P. R. Goundan and A. S. Schulz. Revisiting the greedy approach to submodular set function maximization. *Tech. rep*, 2007.

[9] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[10] Hiplito Hernndez-Prez and Juan Jos Salazar Gonzlez. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks*, 50(4):258–272, 2007.

[11] Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[12] Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986.

[13] P. L. Jacobsen. Safety in numbers: more walkers and bicyclists, safer walking and bicycling. *Injury prevention : journal of the International Society for Child and Adolescent Injury Prevention*, 9(3):205–209, September 2003.

[14] Andreas Kaltenbrunner, Rodrigo Meza, Jens Grivolla, Joan Codina, and Rafael Banchs. Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system. *Pervasive and Mobile Computing*, 6(4):455 – 466, 2010. Human Behavior in Ubiquitous Environments: Modeling of Human Mobility Patterns.

[15] Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. In *In Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136*, pages 152–166. Springer, 1996.

[16] Janet Larsen. Bike-sharing programs hit the streets in over 500 cities worldwide. *Earth Policy Institute*, 2013.

[17] Jenn-Rong Lin and Ta-Hui Yang. Strategic design of public bicycle sharing systems with service level constraints. *Transportation Research Part E: Logistics and Transportation Review*, 47(2):284–294, 2011.

[18] Luis M. Martinez, Lus Caetano, Toms Eir, and Francisco Cruz. An optimisation algorithm to establish the location of stations of a mixed fleet biking system: An application to the city of lisbon. *Procedia - Social and Behavioral Sciences*, 54(0):513 – 524, 2012. Proceedings of {EWGT2012} - 15th Meeting of the {EURO} Working Group on Transportation, September 2012, Paris.

[19] Deepak Merugu, Balaji S. Prabhakar, and N. S. Rama. An incentive mechanism for decongesting the roads: A pilot program in bangalore. 2009.

[20] Rahul Nair, Elise Miller-Hooks, Robert C. Hampshire, and Ana Bušić. Large-scale vehicle sharing systems: Analysis of Velib. *International Journal of Sustainable Transportation*, 7(1):85–106, 2013.

[21] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

[22] Marian Rainer-Harbach, Petrina Papazek, Bin Hu, and Gnther R. Raidl. Balancing bicycle sharing systems: A variable neighborhood search approach. In Martin Middendorf and Christian Blum, editors, *EvoCOP*, volume 7832 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2013.

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

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

[25] Juan P. Romero, Angel Ibeas, Jose L. Moura, Juan Benavente, and Borja Alonso. A simulation-optimization approach to design efficient systems of bike-sharing. *Procedia - Social and Behavioral Sciences*, 54(0):646 – 655, 2012. Proceedings of {EWGT2012} - 15th Meeting of the {EURO} Working Group on Transportation, September 2012, Paris.

[26] Jasper Schuijbroek, Robert Hampshire, and Willem-Jan van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. *Tepper School of Business*, Paper 1491, 2013.

[27] Jia Shu, Mabel C. Chou, Qizhang Liu, Chung-Piaw Teo, and I-Lin Wang. Models for effective deployment and redistribution of bicycles within public bicycle-sharing systems. *Operations Research*, 61(6):1346–1359, 2013.

[28] Divya Singhvi, Somya Singhvi, Peter I Frazier, Shane G Henderson, Eoin OMahony, David B Shmoys, and Dawn B Woodard. Predicting bike usage for new york citys bike sharing system. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[29] Adish Singla, Marco Santoni, Gabor Bartok, Pratik Mukerji, Moritz Meenen, and Andreas Krause. Incentivizing users for balancing bike sharing systems. In *Proc. Conference on Artificial Intelligence (AAAI)*, 2015.

[30] Oliver Smith. Commute well-being among bicycle, transit, and car users in portland, oregon. *92th Annual Meeting of the Transportation 36 Research Board*, 2013.

[31] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011.

[32] Patrick Vogel and Dirk C. Mattfeld. Strategic and operational planning of bike-sharing systems by data mining - a case study. In Jrgen Bse, Hao Hu, Carlos Jahn, Xiaoning Shi, Robert Stahlbock, and Stefan Vo, editors, *ICCL*, pages 127–141. Springer, 2011.

[33] Chelsea Wald. Wheels when you need them. *Science*, 345(6199):862–863, 2014.

[34] James Woodcock, Marko Tainio, James Cheshire, Oliver O'Brien, and Anna Goodman. Health effects of the london bicycle sharing system: health impact modelling study. *BMJ*, 348, 2014.