A Model Predictive Control Approach on Dynamic Bike Reposition

Guilherme Saraiva
guilherme.saraiva@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2019

Abstract

Over the last few years, Bike Sharing Systems (BSS) have emerged as a solution to tackle the “last mile” problem associated with public transportation, as well as to improve the sustainability and decarbonization of cities. One major challenge associated with BSSs is the need to respond to the fluctuating demands by the users in order to provide a good customer service. The key solutions to this problem include a good prediction algorithm and an efficient rebalancing method.

With the introduction of the General Data Protection Regulation law in Europe, current datasets of European bike sharing systems are more restricted. With less data available, the existent prediction algorithms need adaptations and lose accuracy. Building on prior research, a Meteorology Similarity Weighted K-nearest-neighbors algorithm is adapted to be used to predict both the pick up and drop off demands, on a station level basis, considering the most similar days in terms of weather features.

We propose a convex Model Predictive Control approach to tackle the dynamic repositioning problem, which considers the station level demands and outputs a system level solution while trying to minimize the customer loss and at the same time reducing the reposition cost associated. The MPC reposition method was evaluated using a simulator to model the system dynamics with real-world datasets, where it was compared against existent reposition methods in order to evaluate its performance.

Keywords: Demand prediction, Rebalancing, Dynamic Reposition, Model Predictive Control, Bike Sharing System

1. Introduction

Cycling is one of the most sustainable urban transport mode. It causes virtually no environmental damage, promotes health through physical activity, takes up little space and is economical both in direct user costs and public infrastructures [13]. Bicycles address traffic congestion as they form a valid substitution for cars on short trips, have a low carbon footprint, and contribute to the use of public transport by providing an effective last-mile connectivity.

Bike sharing systems allow the inhabitants to use a bicycle without owning one nor having to carry it around. Most systems are composed of stations (docks) placed throughout the city, from which the users can rent a bike and return it in another station. The essence of an efficient bike sharing system relies on having a balanced number of bikes in every station at all times. An unbalanced system would have empty stations, from which the users cannot rent bikes, and full stations, which do not allow docking. Since there is an asymmetric pendular flow of users at different times of the day from residential areas to working areas and vice-versa, the stations tend to get jammed or empty and penalize the quality of the service provided to the user. This bicycle imbalance problem makes it necessary for bikeshare cities to employ costly bike redistribution, which is typically performed by trucks, moving bikes among stations. Furthermore, this problem is aggravated when there is real-time monitoring instead of predicting the user movements, as it takes too much time to reposition after an imbalance has occurred. Rebalancing is a challenging problem due to its intractability. In this thesis we seek to tackle this problem, by having a first approach to demand prediction and to the control of the system while minimizing the costs to the system operator and providing a good service to the user.

1.1. Objective

Our first goal is to find a global control algorithm that fulfils cyclists demand with a limited number of bicycles and a fixed infrastructure of stations. A very important constraint when dealing with mobility data is privacy. The General Data Protection Regulation law [1] imposes hard limits on per-
sonal data access, namely on digital traces of human movements. Due to a new look of privacy-preserving methods, our proposed methodology restricts the data model to a minimum: we seldom require samples in time of the number of bicycles on each station, with no knowledge of user trips.

We propose a datamodel which can be applied in both Citibike (New York) and GIRA (Lisbon), and attest the validity of it in the Citibike system.

2. Exploratory Data Analysis

For this Exploratory Data Analysis, we will analyze the datasets from 2 Bike Sharing Systems: Citibike, in New York and GIRA, in Lisbon. Our goal is to fit a data model which can be applied not only these two systems, but also any Bike Sharing System.

2.1. Concepts and Notation

**Pick Up (rent) and Drop Off Demands**

For a time slot \( t \), the pick-up frequency \( \psi_O \) is the number of bikes that were rented. The pick-up availability \( (\eta_O \in [0,1]) \) is a normalized value of the amount of time the station had bicycles for the users to rent, during time slot \( t \). The formula for the pick up demand \( O_i \), where the subscript indicates station \( i \), can be seen in Equation 1.

\[
O_i(t) = \frac{\psi_O(t)}{\eta_O(t)}
\]  

(1)

Similarly, the drop off frequency \( (\psi_r) \) is the number of bikes docked during \( t \), where \( \eta_r \) is the normalized time for which the station was not full. The formula for the drop off demand is:

\[
r_i(t) = \frac{\psi_r(t)}{\eta_r(t)}
\]  

(2)

**Netflow Demand**

The netflow demand, or total demand in station \( i \) corresponds to the total number of bikes being docked minus the number of bikes rented.

\[
y_i(t) = r_i(t) - O_i(t)
\]  

(3)

**Mean Absolute Error (MAE)**

MAE is the metric used to measure the performance of various algorithms. Its equation is:

\[
MAE = \frac{\sum_{t=1}^{m} |\hat{d}(t) - d(t)|}{m}
\]  

(4)

2.2. Meteorology Data Extraction

The weather reports from the airports are a source of free historical meteorology data. The METARs, short for Meteorological Terminal Air Report[2], are issued every 30 minutes. Due to the proximity of LaGuardia (NY) and Portela (Lisbon) airports to the city center, which are less than 9km) and 7km far from the city center, respectively, we will consider the airport weather reports to be similar to what is observed in the city center.

2.3. Bike Sharing Systems Dataset

Citibike provides monthly datasets on their website with all the trips performed by the users, with a duration of over 1 minute\(^1\) as well as a real-time ARCGIS feed with information regarding each station, including station ID, the number of bikes on the station and the station capacity.

GIRA system has an open data platform \(^2\), where they provide a real-time ARCGIS feed of station information with a sample time of 5 minutes, namely the station ID, the number of bikes on the station and the station capacity. Due to the General Data Protection Regulation [1], Lisbon open data platform is not allowed to share individual trip records as it does not follow privacy rules, having no information on the user movements, rendering our data model to be reduced to a minimum in which it only has access to inventory values. The individual trips record is important as it would allow us to build a network of stations[9, 8] and predict the demand on a system level, instead of each station individually.

After gathering all the data, we proceed to analyze the available common data in both Bike Sharing Systems as we want to design a model which works on both.

Based on previous work, we create a new variable, the Weather Conditions [9], which conveys weather phenomena clustered into four categories, according to their suitability for outdoor bicycling: \( U_1=\{\text{Cloudy, Sunny}\} \), \( U_2=\{\text{fog, mist, haze}\} \), \( U_3=\{\text{snow, rain, light snow, light rain}\} \), \( U_4=\{\text{heavy snow, heavy rain}\} \). The values corresponding to each category are: 0.25, 0.5, 0.75, 1 respectively.

In the Figures 1a-e, the relationship between the rent demand and the weather variables is visualized with histograms. In 1f, it is shown the average demand over a day. Each data point represents the hourly average bike rental for a specific weather variable. For example, the data point (30,3200) in Figure 1b, the value 3200 corresponds to the average hourly bike rentals in which the temperature was observed to be 30°C.

The relationship between each variable is attested and taken into consideration to build the prediction model in the next Section.

3. Model Predictive Control For Rebalancing

Taking into account the data analysis performed previously, in this Section we detail the implementation of Model Predictive Control (MPC) to solve the problem of dynamic rebalancing. We also com-

\(^1\)https://www.citibikenyc.com/system-data

\(^2\)https://emel.city-platform.com/opendata/
Figure 1: Influence of the various weather variables on the rent demand (a-c) and rent demand throughout the day of the whole system (f)

3.1. Data Model
As the available data of our system is the station inventory, our system state can be represented by the variable $x$

$$x_n = \text{number of bikes on station } n$$

The system is modeled linearly, where the difference in the number of bikes in the next time slot $t+1$ is the inflow of bikes, minus the outflow. The inflow and outflow functions account for the movements of the users and for the repositioned bikes. We can control the system through repositioning in order to set $x$ to an optimal value throughout the day.

Lacking the dataset corresponding to the user movements and violates the GDPR privacy law, we cannot build a station network considering the system changes. Instead there is only information on each individual station. We propose an implementation of a MPC controller, which provides a solution by attending to the needs of the system.

Our model must take into account predictions of the user movements. The MPC solves this problem very well as it considers the user movements as disturbances in the system.

Furthermore, we want our model to be subject to constraints on the vehicle and station capacity. A MPC is very effective at dealing with constrained control problems.

Since our bike repositioning system is modeled with linear dynamics, it simplifies the MPC control problem to a series of direct matrix algebra calculations that are fast and robust. This way, the MPC fits our problem very well.

3.2. Problem Statement and Notation
Besides the physical stations, we are considering a virtual station with index 0 (will be referred to as cloud), to account for the trips in progress at time slot $t$. Whenever a bike is rented from a certain physical station, it is placed in the cloud, for the duration of the trip. When it is returned to a physical station, it is removed from the cloud. The existence of the cloud forces the number of bikes in the system to be constant.

Accordingly, $x(t)$ has length $N+1$, where $N$ is the total number of physical stations, such that the index $n$ can take the values: $\{0, 1, ..., N\}$.

A bike station network is constructed where each node corresponds to a station, and the arcs between stations correspond to the repositioning of bikes between a pair of stations. Hence, we build a connected graph $G(t) = (N, A(t))$, where $N$ is the set of nodes and $A$ is the set of arcs. Each node with a positive node number represents a station, in which the node number 0 represents the virtual station. An example of the graph can be seen in Figure 2 for a system with small dimensions.

It is not a directed graph as it does not make sense to have repositions between stations that are too far. Therefore, there are only arcs between stations which have a travel time lower than a user-chosen value. The virtual station is connected to all the stations as any bike in use will go through it.

The dynamics of the system can be described by the following discrete-time equation:

$$x_n(t + 1) = x_n(t) + \sum_{m \to n} u_{m \to n}(t) - \sum_{n \to p} u_{n \to p}(t) + v_{0 \to n}(t) - v_{n \to 0}(t)$$

where $u$ is the control variable, which represents the bicycles to be moved by the bike sharing system operator; and $v$ is the disturbance variable, which represents the user movements. Bikes entering the station are positive flows and leaving are negative.

The MPC problem is formulated as follows:
Figure 2: Example of a connected geographical graph for a system with 4 stations, represented by the nodes. Station 0 is virtual and has the bikes which are currently in use by the user. The arcs define the number of bike repositions $u$.

\[
\text{minimize} \
\sum_{t=1}^{T} (x(t) - \hat{x}(t))^T Q(x(t) - \hat{x}(t)) + \sum_{t=0}^{T-1} u(t)^T Ru(t)
\]

s.t.
\[
x(t + 1) = x(t) + Au(t) + Bv(t), (x0 \text{ given}) (6a)
\]
\[
0 \leq u(t) \leq V_C, t = 0, ..., T - 1 (6b)
\]
\[
\xi \leq x(t) \leq \overline{x}, t = 1, ..., T (6c)
\]
\[
x(t) + Bu^+(t) \leq c, t = 0, ..., T - 1 (6d)
\]
\[
x(t) - Bu^-(t) \geq 0 , t = 0, ..., T - 1 (6f)
\]

where $c$ is the station capacity, $V_C$ is the vehicle capacity, $\hat{x}$ is the target state, $\xi$ and $\overline{x}$ define a minimum quality of service in which the stations should not have fewer bikes the defined lower limit, nor more bikes than the defined upper limit; $u^-(t)$ and $u^+(t)$ represent the minimum and maximum values of the control for each station, respectively.

In the objective function, we want to minimize the "distance" to the optimal state $\hat{x}$ while at the same time reducing the control movements. Matrix $Q$ defines the distance to target state penalty. Matrix $R$ defines the control movements penalty. Matrix $A$ selects each arc from $u_{n\rightarrow t}(t)$ and adds the value to each station. Matrix $B$ selects the user movements from $v(t)$ for each station where all the bikes coming from the cloud to a certain station $i$ count as positive flow and the bikes leaving station $i$ to the cloud count as negative flow.

Constraint 6c limits the control movements $u$ to the capacity of the repositioning truck, which cannot be negative either. Constraint 6d limits the state variable $x$ and represents the minimum service quality that we want to impose on the system. This constraint forces a minimum number and maximum of bikes in a station.

The MPC controller is admitting the rebalancing process happens immediately at the beginning of the time slot. However, there is a delay until repositions take effect and if there are multiple trucks assigned to a station, the trucks arrive at different times. Constraints 6e and 6f impose that the first trucks to arrive do not overflow (tries to insert bikes on a full station) the stations, or underflow (removes bikes from en empty station).

Our MPC problem is solved efficiently since it is a convex quadratic programming problem. Hence, if feasible, it will have an optimal global solution.

We are doing a relaxation of the problem since the number of bikes is an integer number. The output of the model will be rounded to the nearest integer. Alternatively, forcing the variables to be an integer number would turn the problem into a MILP problem, which is not convex, making it more challenging to solve and the solution is not guaranteed to be globally optimal.

3.3. User movements Prediction

The user movements represent the number of rents and returns at each station. Vector $v(t)$ (Equation 7) has length $2N$ and it is a concatenation of the vectors $v_{n\rightarrow 0}(t)$ and $v_{0\rightarrow n}(t)$ from Equation 5. Vector $v_{0\rightarrow n}(t)$ represents all the bikes leaving station $n$ to the cloud which has length. Similarly, $v_{n\rightarrow 0}(t)$ represents that arrive at station $n$ and leave the cloud.

\[
v(t) = \{v_{1\rightarrow 0}(t), ..., v_{N\rightarrow 0}(t), v_{0\rightarrow 1}(t), ..., v_{0\rightarrow N}(t)\}
\]

In this estimation process, we compare several methods of obtaining $\hat{v}(t)$: firstly using histograms to calculate historical mean values, from where we extract 3 different values, secondly a Weighted K-nearest-neighbors method for the most similar days inventory-wise. Thirdly, we use the Meteorology Similarity Weighted K Nearest Neighbors algorithm by Liu et al. [9] which considers weather features to the demand prediction.

Histograms

Historical data was used to compute monthly histograms. We calculated how many bikes were docked, or rented at each station for a given hour and the same day of every month, in Figure 3a and also the average hourly demand for a whole month, in Figure 3.

In total 3 estimated values were extracted from these histograms:

- **AV:** Average value, for the same day of every month

4
• **MP**: Most probable for the same day of every month

• **MHA**: Monthly Hourly Average for a certain target hour

![Figure 3: Monthly histograms of user movements.](image)

We can extract the mean value (AV) from the left Figure as well as the most probable (MP) number of bikes to leave the system. From the right picture, we can extract the demand for the target hour we want to estimate.

**Meteorology Similarity Weighted K Nearest Neighbors (MSWK)** [9]: The similarity is calculated based on a Gaussian Kernel function, in which the weather variables (temperature, weather conditions, wind speed, visibility, humidity) are compared.

A K-Nearest-Neighbors algorithm is applied to calculate a weighted average of the demand observed in the K-most similar days.

**Inventory K Nearest Neighbors (IKNN)**: This algorithm finds the most similar previous days regarding inventory occupation rate of the system and calculates a weighted average of the K-most similar days.

**Performance Comparison**

The MAE (Equation 4) of the demand prediction was measured for the testing set comprised of the month of January 2019 for both Citibike and GIRA systems. The MAE for each method can be seen in Figure 5. The K used for both K Nearest Neighbor methods was trained and it is shown the best one.

The MSWK algorithm produced the best results both in the pick up and drop off prediction with a low average error of 2 bikes per station in the pick up demand and 2.8 in the drop of demand.

3.4. Regularizing the estimated user movements

In order for the MPC to have at least one feasible solution, we design an algorithm which calculates \( v \) while taking into account the results produced by the prediction algorithms. For this, we consider the reposition non-existent (\( u = 0 \)) and the algorithm finds the feasible solution with \( v \) closer to \( \hat{v} \).

**3.4.1 Matrix B**

As matrix B selects user movements (\( v \)) to each station, it must have dimensions of \( N \times 2N \), which multiplied by vector \( v \) of dimensions \( 2N \times 1 \), outputs a vector of size \( N \), which corresponds to the "disturbance" at each node, the user movements at each station. The bikes leaving the station create a negative flow on the station, whereas the bikes entering create a positive flow. This way, matrix B will be composed of -1s and 1s for the selection of \( v_{n-0} > 0 \) and \( v_0 > n \) respectively and 0s for user movements that are not related to the station n.

3.5. Optimizing User Service

In the reposition process, we have two main problems: we want to minimize the customer loss, providing the best service, while at the same time do the least repositions possible. Hence, the objective function is divided into two components: the optimization according to the user service and the cost of repositioning.

Regarding the optimization to the user, we are minimizing the "distance" between the present a
estimated future states to the target state. This is done recurring to a norm Q, which can also be represented by $\| (x - \bar{x}) \|_Q$. For simplicity and respecting the fact that Q is a semi definite positive matrix, we assigned it to be an identity matrix of size N.

3.6. Minimizing repositions

Ideally, we want the system to be as close to the target state $\bar{x}(t)$ as possible, in order to provide the best service to the user. In order to do so, a lot of controls $u(t)$ have to be generated and this implies high repositioning costs for the system administrator. As such, we also need to minimize the number of non-zero controls.

Firstly, it does not make sense economically for a truck to travel a long distance for a reposition, unless the station is in a remote place. This way, we want to hard-limit the travel time of trucks by 15 minutes. We can not limit the control links by distance, since it does not account for traffic changes throughout the day.

For the traffic prediction, we are using Google Distance API, which gives us a distance and driving time for a certain time of the day. Instead of calculating the driving time for all pairs of stations, we can prune beforehand the ones which are too far. This is done by calculating the "Manhattan distance" (l1 norm) between 2 stations to approximate to New York streets and then query the driving time from Google Distance on all the pairs which are no farther than a parameter $D_{max}$, which is explained in Section 3.7.

Matrix R will be responsible for penalizing the control variables due to long distance between stations. It takes the dimensions $\text{size}(u) \times \text{size}(u)$ and it is a diagonal matrix, where each position of its diagonal is associated with a control $u$ and represents the driving time between stations.

Each pair of stations which are closer than $D_{max}$, have an arc added in the graph $G$, which means this pair of stations is concatenated to the vector $u$. This vector will have a total size of the number of arcs and the driving time queried from Google Distance on all the pairs are attached to the correspondent position in the diagonal of matrix R.

3.7. Graph size and frequency of control

The MPC computes a control every time slot $t$. The reposition process of all the trucks must be completed before the next reposition is generated, in order to keep the system coherent, instead of generating actions which are not considering the unfinished ones from previous time slot.

The total reposition time is given by:

$$t_r = t_{1\rightarrow 1} + t_{load} + t_{1\rightarrow 2} + t_{unload}$$

where $t_r$ is the total reposition time, $t_{1\rightarrow 1}$ is the driving time from the initial position of the trucks to the first station, $t_{load}$ is the loading time of the bikes onto the truck, $t_{1\rightarrow 2}$ is the driving time to the second station and $t_{unload}$ is the unloading time of the bikes from the truck.

The variable $t_{1\rightarrow 2}$ depends on the graph size and on the time of the day. If the reposition is done during rush hours, the driving time will increase due to traffic. As a result, the frequency of control of the MPC is dependent on the parameter $D_{max}$, which should be time variant in order to account for traffic.

A high $D_{max}$ will increase the graph size, which will allow reposition on stations which are farther away, than if $D_{max}$ is low. This way, the time slot chosen for the MPC should also be larger, in order for all the trucks to finish at the same time.

A small $D_{max}$ might make the graph unconnected, creating islands of station, which cannot reposition between them. Hence, if users have high flows between these islands, the MPC model will be unfeasible as it cannot perform controls.

4. Results

Reposition algorithms need a simulated environment where they can be compared. A simulator allows us to see how the dynamic system responds to the rebalancing changes. The efficiency of different rebalancing methods can be compared using different metrics, defined by the parameters outputted by the simulator.

In a first approach to the used simulator, PEBISS, the metrics will be compared with the repositioning methods available out of the box in PEBISS simulator, in order to analyze which metrics give us more information on repositioning.

With more insight on the important metrics, we will implement our MPC model and train it while trying to balance the customer loss and the repositioning costs.

To conclude, we will present final comments on the performance of our method.

4.1. Metrics

The simulator calculates several metrics during run time. These are calculated hourly and their formulas are as following:

- **Reposition:** Cost of the reposition. It is the only dynamic cost to the operator.
  
  $$R_C = C_t t_r$$

  where $C_t$ is the coefficient of the cost of reposition per hour per team and $t_r$ is the total time invested in a hour in reposition.

- **Bad service Penalty:** cost to the user. In case the users find an empty/full station at origin/destination and have to divert its original
course, they perceive the time lost to be more expensive, because it was wasted.

\[ B_S = \beta t_l \]

where \( \beta t_l \) is the coefficient of the cost of time lost per hour and \( t_l \) is the cumulative hourly lost access time.

**Death Penalty:** cost to the user, associated with the user leaving the system ("death"). The user can "die" due to the lack of bikes nearby, due to the destination being too far from the destination station, if the user cancels the trip beforehand (no nearby stations), if it is electric and the route is a climbing path.

\[ D_P = D_L L \]

where \( L \) is the coefficient of the cost of life and \( D_L \) is the total number of deaths in a hour.

The coefficients values are the ones proposed by Casado et al. [3]. These are an estimation of the real costs, but in real life they do not follow a linear function. As such, comparing them directly has no value since their values are meaningless. We can however compare the reposition methods within each metric.

### 4.2. Repositioning Functions

The author made available 2 types of dynamic repositioning methods to be selected in the simulator (Fixed Routes and Pool System), as well as no reposition method.

In the fixed routes method, the map is divided into a number of areas equal to the number of existent trucks. Every truck is assigned an area where a single route passing through all the station. Independently of the current state of the stations, the reposition truck will follow the tour for the whole simulation and at each stop, would load the excess of bikes if it had enough capacity, or unload all needed bikes if available. If the station is balanced, the truck will skip it and proceed to the next one. In the Pool system method, the stations with lack or excess of bikes will set an alarm and are introduced into the pool. Every time there is a pair of stations, in which a station is a provider of bikes and the other a consumer, a truck is assigned for repositioning.

### 4.3. MPC for Rebalancing

In this first approach to our MPC repositioning method, we are considering a time slot \( t \) of 30 minutes, which means it runs 2 times per hour. This means all the trucks must finish the repositioning meanwhile, before a new one is assigned. The parameter \( D_{\text{max}} \) (from section 3.3) defines the number of reposition controls between stations. A higher distance between stations leads to the graph having more arcs. We are assigning it a "Manhattan" distance of 800m.

For computational reasons, we are using a time horizon of \( T=9 \), which equals to 4.5 hours in advance when using a time slot of 30 mins.

Additionally, we are considering an infinite amount of trucks available, although we can use at most the number of trucks equal to the number of arcs between the stations in the graph.

Firstly, we tested the model from Equation 6, which minimizes the difference between the optimal inventory value and the R-norm of the control links \( ||u||_R \).

It is important to note that the baseline reposition methods used have a real-time monitoring to choose the next destination, whereas the MPC predicts the user movements. Similarly to a real life application, there are a lot of uncertainties in the demand prediction, as the simulator randomly creates users within the most popular areas and if the user dies, the simulator compensates by creating another user in a different place than what was predicted.

### 4.4. Metrics Evaluation

The metrics from Section 4.1 are used to compare the MPC with the different rebalancing methods designed by Casado [3], in the 2 systems: Citibike (New York), Bicing (Barcelona). Figures 9, 7, 8 represent each metric for each hour of the day, for the 4 reposition methods: MPC, Fixed Routes, Pool System, No reposition. Operations metric, however the difference is insignificant.

**Figure 6:** Hourly Reposition Metric, for each reposition method. The MPC has a much higher cost of reposition than the baseline methods.

In Figure 9, it is shown the Reposition metric, one of the most important metrics as it tells the cost of moving the trucks (Equation 9). The MPC has a high cost of repositioning, as it usually only moves small numbers of bikes and uses more trucks than the other methods. It is also important to note the repositioning cost of the Fixed Routes and Pool...
system is similar, although the Pool System shows better results overall in the other metrics.

Figure 7: Hourly Bad Service Metric, for each reposition method. The MPC obtains a similar value to the Pool System in New York, but in Barcelona the metric is almost null, proving the MPC provides the best customer service regarding re-routings by the users due to empty stations.

The bad service metric (Equation 10) is higher whenever there is a rerouting of the user due to empty/full stations. A lower value is a good indicator of the quality of the repositioning. For both Bicing and Citibike, the fixed routes method has very little advantage compared to no repositioning (Figure 7), whereas the pool system lowers the bad service metric by more than half. The MPC has the best results in Barcelona and New York, as they are bigger systems and benefit from the approach that the MPC has on minimizing the customer loss for the whole system.

4.5. MPC Reposition Cost Reduction

To achieve sparsity of the control variable $u$, $l_1$ is used by replacing the R-norm by the $l_1$ norm in the objective function:

$$
\sum_{t=1}^{T} (x(t) - x^*(t))^T Q (x(t) - x^*(t)) + \sum_{t=0}^{T-1} \lambda \|u(t)\|_1
$$

(12)

where the parameter $\lambda$ controls the trade-off between the repositioning cost and providing a better service for the user.

The weight of $\|u\|_1$ is modified for several test runs and the effect on each metric was analyzed.

Figure 8: Death Penalty Metric comparison. The MPC has the advantage as had the least number of users leaving the system due to having no bikes available

Figure 9: Reposition metric for different values of the parameter $\lambda$. A higher lambda make the MPC minimizing the number of controls, reducing the reposition costs

A lower $\lambda$ will force the controller to be more "nervous" and do more repositions, which in turn will cause an increase in the reposition metric. As the parameter increases, the reposition metric lowers since the controller will further minimize the control.

Analyzing Figure 8, both the Pool System and the MPC present the best results. Additionally, the Fixed Routes method has little advantage compared to no repositioning method.

Figure 10: Bad service metric for different values of the parameter $\lambda$. A higher lambda make the MPC minimizing the number of controls, reducing the reposition costs

With a decrease of the number of repositions with lambda, the service quality will also drop. The bad
Figure 11: Death metric for different values of the parameter $\lambda$. A higher lambda make the MPC minimizing the number of controls, reducing the reposition costs. Service metric in Figure 10 represents the detours by the user when the station becomes empty upon arrival, or full when the user wants to dock. Similar to the Bad Service Metric, the death metric increases with lambda, as the number of repositions decrease and more users die.

In the Figures below we compare all the reposition methods with the metrics related to the user (death and bad service metrics), against the reposition metric:

Figure 12: Trade-off between Reposition metric and Bad Service in New York. The blue points represent the MPC with the different $\lambda$ as the regularization parameter and also the MPC with $R$ norm minimization. The MPC has the advantage regarding the bad service metric, presenting less numbers of users who leave the system, or re-route from their original station due to empty/full stations. However, the MPC loses to the baseline methods in terms reposition costs.

As confirmed by Figures 9, 11 and 10, from values of $\lambda$ between 10 and 1000, there is a sudden rise in the repositions costs with $\lambda$ and a decrease in reposition costs. This causes the clustering of data points seen in the Figures 12 and 13.

Figure 13: Trade-off between Reposition metric and Bad Service in New York. The blue points represent the MPC with the different $\lambda$ as the regularization parameter and also the MPC with $R$ norm minimization. The MPC has the advantage in the Death metric, having less users dying due to empty stations. However, the MPC loses to the baseline methods in terms reposition costs. All the variations of the MPC are situated to the left of the baseline methods, meaning all provide a good service to the user.

5. Conclusion
In this work we presented a convex formulation to solve the dynamic repositioning problem, using Model Predictive Control. Our model takes into account the individual demands of the stations and produces a solution which considers the demands of the whole system, attenuating the lack of the data set of user trips. Using the simulator, we concluded the MPC obtained the best results in terms of providing the best service for the user. However, it came with a high cost of repositioning. Each method has its advantages and disadvantages regarding better customer service or lower reposition costs, thus there is no best method. The MPC can provide the best service compared to the real-time monitoring reposition, but in the end, it is up to the system administrator to decide if the trust of the customers and better mobility in the cities have more value than the money spent on repositions. Furthermore, a high death/bad service metric will lead to a decay of the system in the long term, as people will stop using the system and the bad publicity spreads fast.

6. Related Work
Demand Prediction

Several past works has studied several predicting models and compared their accuracy, such as Random forests, decision trees, gradient boosting machines and linear regression [12, 14]. Zhang et al. [15] applied Convolutional Neural Network (CNN)
for large-scale bike-sharing flow prediction. Liu et al. [9] consider a weighted multi similarity model using K-nearest Neighbours to predict pick up demand according to the most similar days according to weather variables. Drop off demand is calculated by designing a transition network between stations and attribute a destination based on the pick up demand.

There is extensive literature on predicting the user movements both on a station level basis, and on a region level, which is composed by a set of station close to each other and with similar patterns and urban functions. This clustering of stations reduces computation time significantly[6]. Additionally, these methods present better results in the demand prediction[10, 8, 7], but neglect the cost of the user to move to a station within the region, which may not the closest.

Rebalancing

Chemla et al. [4] proposed a heuristic method for the single-vehicle static rebalancing problem, where each station could be visited more than once and could be used as a buffer in which bicycles are stored for a later visit.

Regarding dynamic reposition, Lowalekar [11] extended the work provided by Ghosh et al. [5] with an online approach where they employ a Lagrangian decomposition problem (decouples the global problem into routing and repositioning slaves and employs a novel DP) and a greedy online anticipatory heuristic to solve large scale problems effectively. Ghosh et al. [6] further improved their previous work, creating an abstract formulation of the Mixed-Integer-Linear-Programming to speed up the computational time.

Li et al. [8] propose a Spatio-Temporal Reinforcement Learning model to maximize the long-term reward of a sequence of decisions, in which the reward to be given when transitioning to the next state is the negative costumer loss. The problem is formulated in a multi-agent way, such that a new reposition is generated to a trike without waiting for the completion of the others. For each cluster of stations, the model uses a deep Q learning network to generate the optimal policy.

References


