# Optimizing the Crowdsourcing-based Bike Station Rebalancing Scheme

Yubin Duan and Jie Wu
Department of Computer and Information Sciences, Temple University, USA
Email: {yubin.duan, jiewu}@temple.edu

***Abstract*—User dynamics in both spatial and temporal domains bring uncertainty to bike-sharing systems (BSSs) and usually lead to bike imbalance. This may generate out-of-service events due to bike underflow or overflow, at a bike station. In this paper, we recruit workers through crowdsourcing to rebalance loads among bike stations. We assume that workers have their individual sources and destinations, and assign them to move bikes from overflow stations to underflow stations. We partition the complex spatial and temporal problem into a sequence of slices with a fixed duration in the temporal domain. In each slice, we focus on the spatial domain and allocate a pair of overflow/underflow stations to a worker such that the summation of detour cost among workers is minimized. The hardness of finding the min-cost allocation is shown by a reduction from a 3-dimensional matching problem (i.e., matching among workers, overflow stations, and underflow stations). We propose a 3-approximation algorithm for the problem when the detour cost is proportional to the detour distances. Then, the configuration dynamic in the sequence of slices is captured by carefully determining the rebalancing frequency and target for each rebalancing operation. We investigate heuristic approaches to decide rebalancing frequencies and targets over a sequence of slices in order to minimize the total number of bike movements (i.e., the total number of workers), and hence to derive the average total detours per slice. We simulate our algorithms on both real-world and synthetic datasets based on different time-slice granularities. The experiment results show that our approaches can reduce the average total detour per slice.**

***Index Terms*—Bike sharing system, bike rebalancing scheme, matching problem, urban computing.**

## I. Introduction

Nowadays, bike-sharing systems (BSSs) are earning increasing attention and have been widely adopted in major cities [1–3]. Recent research has shown that deploying BSSs in cities not only benefits the environment but also increases the welfare of the economy. Accessibility and affordability of BSSs greatly motivates users to ditch their cars for bikes. Besides, the social benefits brought by the development of BSSs also include transport flexibility, reduced congestion and fuel consumption, as well as financial savings for individuals. Although the benefits brought by BSSs are great and attractive, maintaining the sustainability of the system is challenging.

The development of BSSs heavily depends on the sustainability of the system. However, the dynamics of user mobility often lead to an unbalanced bike distribution among stations. Specifically, *overflow* or *underflow* stations may occur under an unbalanced distribution, i.e., users cannot rent bikes from underflow stations and cannot return bikes to overflow stations.

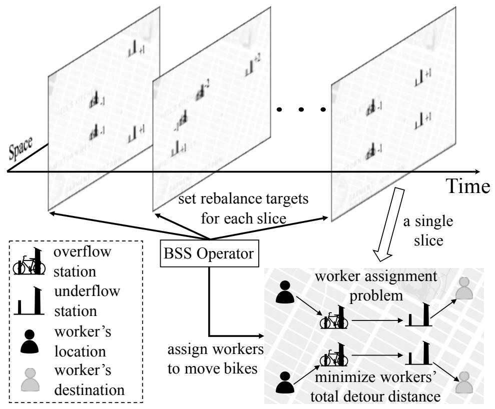

Space
Time
set rebalance targets for each slice
a single slice
BSS Operator
overflow station
underflow station
worker's location
worker's destination
worker assignment problem
assign workers to move bikes
minimize workers' total detour distance

Fig. 1. A crowdsourcing-based rebalancing scheme.

Potential users can also be missed due to such *Out-of-Service* (OoS) events. Therefore, it is necessary for the BSS operator to rebalance the bike distribution among these overflow and underflow stations to maintain good user experience.

We propose a crowdsourcing-based rebalancing scheme that rebalances the BSS across both spatial and temporal domains. When overflow or underflow occurs, the BSS operator can determine the *rebalancing target* (i.e. the number of bikes to move in/out) for each station and recruit workers to rent/return bikes from overflow stations to underflow stations and re-balance the system. Notice that the workers have their own source and destination locations. Recruiting workers to ride bikes between certain stations can not only rebalance bikes in the BSS but also give workers opportunities to earn monetary rewards (e.g., a free ride) with a slight detour.

Spatial and temporal domains are decoupled into a sequence of slices with a fixed duration in the temporal domain (as shown in Fig. 1). In each slice, we focus on recruiting workers to meet the rebalancing targets in the spatial domain in a particular time slice such that the overall detour cost is minimized. In the temporal domain over a sequence of slices, we decide on the rebalancing frequency (i.e. the time interval between rebalancing operations) and its corresponding target to minimize bike movements in rebalancing (and hence the number of workers needed) in an effort to minimize the total detour over multiple slides.

The *Worker Assignment Problem* (WAP) is proposed for each slice. Given the rebalancing target as well as the sources and destinations of workers, we aim to assign a pair of stations, one from overflow stations and one from underflow

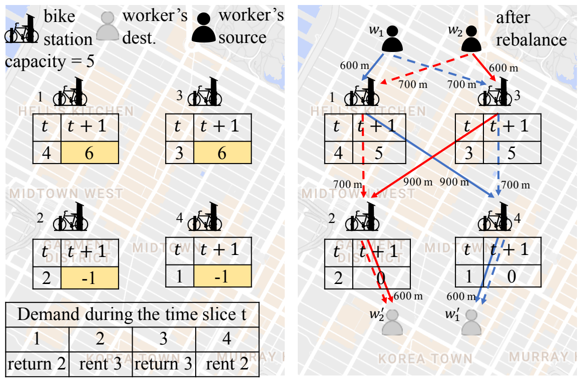


Fig. 2. A user assignment problem scenario in time slice $t$.

stations, to each worker to minimize the detour cost of all workers. Besides, the rebalance frequencies and targets for multiple slices should be determined by the system operator. We propose to minimize the number of workers needed for rebalancing. Also, the targets for each slice should be self-balanced, i.e., the total number of bikes moved in and out should be the same.

Designing such a scheme is still challenging after we decouple the original problem in spatial and temporal domains. The challenge in the *spatial* domain is minimizing the overall detour cost. Assigning workers to overflow/underflow stations is a 3D matching problem (i.e., matching among workers, rent stations, and return stations) which is NP-hard. Determining the rebalance frequency and targets for different slices that minimize bike movements across multiple slices poses challenges in the *temporal* domain. Even if the future user demand can be precisely predicted, it is hard to decide the number of time slices that the system should look ahead.

We use a toy example in Fig. 2 to illustrate these challenges. The left sub-figure shows the OoS events and the right one shows a rebalancing plan to avoid such OoS events. The capacity of each bike station is 5. At time slice $t$, the *states* (i.e. inventory levels) of stations are 4, 2, 3, and 1. Following the demand during slice $t$ (i.e. time duration $[t, t+1)$) listed in the figure, OoS events occur at every station. Among them, stations 1 and 3 suffer from overflow. No more bikes can be returned to these stations. In contrast, stations 2 and 4 face underflow, and users cannot rent any more bikes. Recruiting workers to move bikes from overflow stations to underflow stations before time $t$ can help the BSS avoid these OoS events.

In the spatial domain, suppose two workers are located at $w_1$ and $w_2$ with the corresponding destination $w_1'$ and $w_2'$. The BSS operator hopes that the states of the stations at time $t+1$ are $(5, 0, 5, 0)$, as shown in the right sub-figure, in order to avoid having overflow and underflow stations. Then, the rebalance target is to move one bike each from stations 1 and 3 to stations 2 and 4. Assume that the detour cost is proportional to the detour distance. If we simply assign workers to rent bikes from the station nearest to their source and return at the station nearest to their destination (as following the solid line in Fig. 2), their total moving distance is $4,200\text{m}$. The optimal value is $4,000\text{m}$ and can be achieved by following the dashed line in Fig. 2. It shows that finding an assignment for workers to minimize their total detour distance is not trivial. The challenge in the temporal domain occurs when we decide the rebalancing target for slice $t$. Actually, the target used in the example may not be good. If the demand of station 1 during slice $t+1$ is 'return 1', then an OoS event occurs again at station 1, since the station is full after time slice $t$. Under such a scenario, removing 2 bikes from station 1 before time slice $t$ may be a better choice. The number of time slices that the system should look ahead is uncertain.

In this paper, we propose a 3-approximation algorithm toward the WAP. Generally, our algorithm consists of two rounds of matching. It first finds the optimal match between rent and return stations, and then matches workers to paired stations. We investigate two heuristic approaches that consider different rebalance frequency granularities. One considers a fixed number of slices when deciding targets. The other one is inspired by [2] which greedily chooses the number of time slices to look ahead.

Our main contributions are summarized as follows:

- We propose a crowdsourcing-based scheme for hiring workers to rebalance BSSs. In the scheme, the uncertainty of the system in the temporal and spatial domains is decoupled by slicing the problem in the temporal domain.
- We formulate the general WAP to minimize workers' total detour cost in a time slice, show its NP-hardness, and propose a 3-approximation algorithm for the WAP when the detour cost is proportional to the detour distance.
- To determine the application frequency and target of the WAP in multiple slices, we investigate greedy heuristics rather than directly applying the method in [2].
- We simulate our algorithm on both real-world and synthetic datasets, and compare the performances of our algorithms and previous approaches in terms of the average total detour per slice.

## II. Related Work

With the rapid development of bike sharing, more and more researchers have devoted their effort to related issues including user demand prediction [3–8] , bike rebalancing strategies [2, 9, 10], station location optimization [11, 12], bike lane planning [13], and suggestion for users' journeys [14, 15]. We review researches on demand prediction and rebalancing strategy design in this section.

### A. Demand prediction

The success of our scheme strongly relies on the precise prediction of user demands. Researches on demand prediction can be categorized as station level prediction and cluster level prediction. The earlier demand prediction approach focuses on predicting the number of bikes at each station, such as [4, 5]. However, the station-level prediction does not take contextual information into account and may not always generate an accurate prediction [6]. Clustering similar stations is one way to address this problem. For example, Li et al. [6] proposed a hierarchical model that clusters stations based on their locations and transition patterns first, and then predicts the number of bikes rented from and returned to each station cluster. Chen et al. [7] further proposed a dynamic cluster-based scheme for over-demand prediction by considering opportunistic contextual factors such as social and traffic events. By utilizing demand prediction, our approach aims to minimize the number of bikes moved during rebalancing.

### B. *Rebalancing strategy design*

Two major approaches in the design of rebalancing strategies for BSSs are truck-based approaches [2, 16–18] and user-based approaches [9, 19–21].

In the truck-based approach, the BSS operator hires a fleet of trucks to transport bikes from overflow stations to underflow stations. In static models, the target inventory level for stations is constant. Liu et al. [2] proposed a method that first clusters bike stations according to geographic information and station status, and then assigns a truck to each cluster. The routing for each truck is modeled as an integer programming problem. For the dynamic model, Ghosh et al. [17] proposed an online rebalancing approach to minimize the occurrence of OoS events while considering the uncertainty in future demand. They turned the problem into an iterative game between the BSS operator and the environment to compute a strategy.

In the user-based approach, the BSS operator offers users monetary incentives and motivates them to rent/return bikes at certain stations [9, 19]. It is expected to achieve a self-balanced system to improve the overall service level by controlling the user's dynamics. Designing the pricing mechanism is challenging since the user cost is unknown. Waserhole [19] presented a dynamic pricing mechanism that incentivizes users to redistribute bikes by providing alternate rental prices. Singla et al. [9] proposed a crowdsourcing pricing mechanism to incentivize users based on the multi-armed bandit model. However, these approaches do not consider the spatial imbalance of BSS. Pan et al. [20] studied the rebalancing problem in dockless bike sharing systems and proposed a deep reinforcement learning algorithm for deciding the incentive price. The goal of the user-based approach is to achieve self-sustaining systems by controlling user dynamics instead of hiring workers to maintain the sustainability of the system.

## III. System Model

In our work, we propose a crowdsourcing-based bike station rebalancing scheme. The BSS operator conducts several rounds of rebalancing among bike stations at predefined intervals during the rebalancing period. In each round of rebalancing, the operator recruits a set of workers to move bikes from overflow stations to underflow stations. The rebalancing targets (number of bikes to move) for each round and the length of the interval between two rounds are decided based on states and demand predictions of stations in the BSS.

The BSS operator needs to gather sources and destinations of workers, before it can generate an assignment for each worker. A worker is denoted by $w$, and the worker set is denoted by $W$. Each worker has his/her source $s_w$ and destination $d_w$. The sets of $s_w$ and $d_w$ are denoted by $S=\{s_w|w\in W\}$ and $D=\{d_w|w\in W\}$, respectively.

TABLE I
Table of Notations

| Notations | Description |
|---|---|
| $W, S, D$ | the set of workers, their sources, and destinations |
| $w, s_w, d_w$ | a worker and his/her source, and destination |
| $B, N, P$ | the set of bike stations, rent, and return stations |
| $b, n, p$ | a bike station, a rent station, and a return station |
| $\eta_b$ | the capacity of station $b$ |
| $\boldsymbol{\eta}_B$ | the capacity vector for all stations in $B$ |
| $T, t$ | the set of time slices, and a time slice |
| $\phi_b(t)$, | the state of station $b$ at the beginning of slice $t$ |
| $\tau_b(t)$,$\rho_b(t)$ | the demand and rebalancing target of $b$ during $t$ |
| $\boldsymbol{\tau}_B(t)$,$\boldsymbol{\rho}_B(t)$ | the demand and rebalancing target vector of B |

The states and user demand predictions for bike stations are also necessary. A bike station is denoted by $b$, and the set of bike stations is denoted by $B$. The number of bike stations is $|B|$, where $|\cdot|$ represents the cardinality of a set. Each station $b$ has a capacity $\eta_b$ and can store at most $\eta_b$ bikes. Capacities of different stations can be different. For simplicity, we slice the rebalancing period in the temporal domain. The set of time slices is denoted by $T$, and each time slice is $t\in T$. The state of station $b$ for slice $t$ is defined as the number of on-dock bikes at the beginning of each time slice $t$, and it is an non-negative integer denoted by $\phi_b(t)$. The demand of $b$ during time slice $t$ is an integer denoted by $\tau_b(t)$, and it comes from usage predictors, such as [2, 7]. The demand $\tau_b(t)$ can be either positive or negative. A positive demand means that more bikes are returned to the station and the number of on-dock bikes increases during time slice $t$. A negative demand has the opposite meaning.

Based on the state and demand information, the operator can determine a rebalancing target $\rho_b(t)$ for each station $b$ during $t$. A positive $\rho_b(t)$ means that the station $b$ needs $\rho_b(t)$ bikes, and a negative $\rho_b(t)$ means $|\rho_b(t)|$ bikes shall be removed from the station. $\sum_{b\in B}\rho_b(t)$ should be 0 since the rebalancing operation does not affect the number of bikes in the system. In each time slice $t$, bike stations with negative targets can form a *rent station set* $N$, and stations with positive targets can form a *return station set* $P$. Formally, $N=\{b\in B|\rho_b(t)<0\}$ and $P=\{b\in B|\rho_b(t)>0\}$. Elements in sets $N$ and $P$ are denoted as $n$ and $p$, respectively.

In the paper, we assume the number of workers who can be recruited is large enough. If not, the BSS operator could use trucks for rebalancing. Truck-based rebalancing strategies are reviewed in section II. After recruiting a sufficient amount of workers, the operator needs to determine a *rebalancing assignment* $(w, n, p)\in W\times N\times P$ for each worker, meaning that the worker $w$ should rent a bike from station $n$ and return it to station $p$. The corresponding detour distance of worker $w$ is denoted by a real number $\delta_w$. Formally, $\delta_w = dis(s_w,n)+dis(n,p)+dis(p,d_w)-dis(s_w,d_w)$ , where $dis(\cdot,\cdot)$ is the distance function used to calculate the geographical distance between two locations.

The assignment for workers is modeled by the assignment graph $G=(V,E)$ shown in Fig. 3. The vertex set is constructed

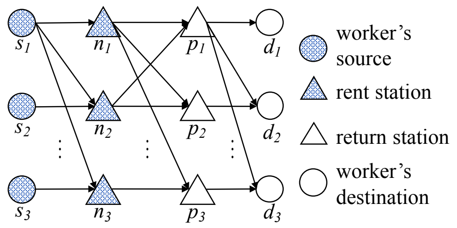


Fig. 3. An illustration of an assignment graph.

by workers' sources, destinations, and bike stations, i.e. $V = S \cup N \cup P \cup D$. The directed weighted edges have three types, including edges from $S$ to $N$, edges from $N$ to $P$, and edges from $P$ to $N$. Formally, $E = (S \times N) \cup (N \times P) \cup (P \times D)$. We use the function $e : V^2 \mapsto \mathbb{R}$ to denote the edge weights. The weight is quantified by the geographical distance. For example, the weight of an edge from $s_w$ to $n$ is the the corresponding distance, i.e., $e(s_w, n) = dis(s_w, n)$. A flow from $s_w$ to $d_w$ is equivalent to an assignment for $w$. The sum weight of edges on the flow is $w$'s moving distance, and is used to quantify $w$'s detour since the distance between $s_w$ and $d_w$ is constant.

The time evolution of the state of a station is modeled by discretized time series. As shown in Fig. 4, the state $\phi_b(t+1)$ is determined by the state $\phi_b(t)$, the demand $\tau_b(t)$, and the rebalance operation $\rho_b(t)$ during time slice $t$. Formally, $\phi_b(t+1) = \phi_b(t) + \tau_b(t) + \rho_b(t)$. Without the rebalancing operation $\rho_b(t)$, an OoS event may occur- either an overflow event with $\phi_b(t') > \eta_b$, or an underflow event with $\phi_b(t') < 0$.

## IV. Problem Formulation and Analysis

### A. Worker assignment problem

In a fixed time slice, we propose the Worker Assignment Problem (WAP). Given rebalancing targets of stations, sources and destinations of the workers, the WAP aims to find out an optimal assignment for workers that minimizes their overall detours when workers are moving bikes between stations.

The WAP can be seen as finding a minimum cost flow from $S$ to $D$. Based on the assignment graph, an assignment for worker $w$ is equivalent to a flow from $s_w$ to $d_w$. As shown in Fig. 3, a flow from $s_w$ to $d_w$ passing through a node $n$ and a node $p$ represents a worker $w$ rents a bike from $n$ and returns to $p$. Our WAP problem is to minimize the total cost of the $k$ flows. We use $f(\cdot, \cdot)$ to denote the flow rate, which is an integer, between two vertices.

This problem can be formulated by a Integer Programming model, which can be described as:

$$\min \sum_{W,N,P} f(s_w,n)e(s_w,n) + f(n,p)e(n,p) + f(p,d_w)e(p,d_w)$$

$$s.t. \sum_{N} f(s_w,n) = 1, \ \sum_{P} f(p,d_w) = 1, \forall w \in W \quad (1)$$

$$\sum_{W} f(s_w,n) = |\rho_n|, \sum_{W} f(p,d_w) = |\rho_p|, \forall n \in N, p \in P \quad (2)$$

$$f(n,p) = \sum_{W} (f(s_w,n) \cdot f(p,d_w)), \forall n \in N, p \in P \quad (3)$$

$$f(s_w,n), f(p,d_w) \in \{0,1\}, f(n,p) \in \mathbb{N} \quad (4)$$

Our objective is to minimize the total weight of the flow. For the WAP in a Euclidean plane, it is the overall detour distances.

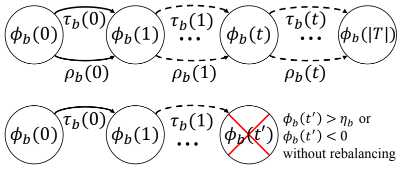


Fig. 4. Station state evolution in discretized time.

In a more general version, which is denoted as the *general WAP*, the edge weight could be any kind of detour cost, such as traveling time. Eq. (1) is the assignment constraint, which means that each worker should be assigned to a rent station in $N$ and a return station in $P$. Eq. (2) is the target constraint. It means that each station's positive or negative targets should be satisfied. Eq. (3) is the consistency constraint. If there is a flow from $s_w$ to $n$ and from $p$ to $d_w$ (i.e., the worker $w$ rents a bike from $n$ and returns to $p$), there should be a flow from $n$ to $p$. Eq. (4) is the flow-rate constraint. The rates of flows from $S$ to $N$ and flows from $P$ to $D$ can be either 0 or 1. The rates of flows from $N$ to $P$ should be a non-negative integer.

### B. Configuration design problem

In this subsection, we formulate the Configuration Design Problem (CDP) for multiple time slices. Given the demand prediction of each time slice, CDP aims to design a set of rebalancing targets which can minimize the number of bike movements over multiple time slices. We choose this objective as an effort to minimize workers' total detour over multiple slices. Formally, given the initial state vector $\boldsymbol{\phi}_B(0)$ and the demand vectors $\boldsymbol{\tau}_B(t)$ for all time slices $t \in T$. The objective is to find the rebalancing target vector $\boldsymbol{\rho}_B(t)$ for each time slice $t$, and minimize the total amount of moved bikes.

$$\min \sum_{t \in T} \sum_{b \in B} |\rho_b(t)|$$

$$s.t. \quad 0 \le \phi_b(t) \le \eta_b, \forall b \in B, \forall t \in T \quad (5)$$

$$\sum_{b \in B} \rho_b(t) = 0, \forall t \in T \quad (6)$$

$$\rho_b(t) \in \mathbb{N}, \forall b \in B, \forall t \in T \quad (7)$$

Note that $\sum_{t \in T} \sum_{b \in B} |\rho_b(t)|$ is actually twice the number of moved bikes, since moving a bike from a rent station to a return station can simultaneously fulfill a positive target and a negative target. Eq. (5) is the capacity constraint , which ensures that the number of bikes at each station cannot be lower than 0 or higher than the station capacity. Eq. (6) is the matching constraint which requires the sum of rebalancing targets among all stations at each time slice to be 0. This is because the total number of bikes should remain unchanged during rebalancing. Eq. (7) indicates that rebalancing targets are non-negative integers.

### C. Problem hardness

***Theorem 1:*** The general WAP for a time slice is NP-hard.
*Proof:* The proof shows that our optimization problem can be reduced from a weighted 3D matching problem. The decision problem of 3D matching is known to be one of Karp's 21

**Algorithm 1** Two-Round Matching Algorithm - Stage 1

```
Input: positive station set $P$ as well as the corresponding demand set $\{\rho_P\}$, and negative station set $N$ as well as the corresponding demands $\{\rho_N\}$
Output: rent/return station pairs which are used for worker allocation in the next stage
1: for each $n \in N$ and $\rho_n < -1$ do
2:     Copy station $n$ $(|\rho_n| - 1)$ times in $N$.
3: for each $p \in P$ and $\rho_p > 1$ do
4:     Copy station $p$ $(\rho_p - 1)$ times in $P$.
5: $V \leftarrow N \cup p$, $E \leftarrow \emptyset$
6: for $n \in N$, $p \in P$ do
7:     $E \leftarrow E \cup (n, p)$, $e(n, p) \leftarrow dis(n, p)$
8: $X \leftarrow$ min-cost perfect matching of $G(V, E)$.
9: return $X$ as the rent/return station pair
```

**Algorithm 2** Two Round Matching Algorithm - Stage 2

```
Input: worker set $W$, rent/return station pair set $X$
Output: rent and return allocation for each user, and workers' total travel distance $C$
1: $V' \leftarrow W \cup X$, $E' \leftarrow \emptyset$
2: for $w \in W$, $(n, p) \in X$ do
3:     $E' = E' \cup (w, (n, p))$,
       $e(w, (n, p)) \leftarrow dis(s_w, n) + dis(n, p) + dis(p, d_w)$
4: $M \leftarrow$ min-cost perfect matching of $G'(V', E')$.
5: return $M$ as the rent and return allocation for workers, and $||M||$ as workers' total travel distance.
```

NP-complete problems [22], and the weighted 3D matching problem belongs to the NP-hard problem class.

The general WAP can be reduced from a maximum 3D matching problem if we let tripartite sets denote the set $W$, $P$, and $N$. A 3D matching $M$ with $|M| = |W|$ is a legal assignment for bike rebalancing. That is, a triple $(w, n, p)$ in $M$ represents that worker $w$ should move a bike from station $n$ to station $p$. The weight of each triple is equal to a large positive constant minus the cost of the corresponding journey (i.e. the sum of the costs from $s_w$ to $n$, then to $p$, and finally to $d_w$). A maximum matching maximizes the sum of negations of journey costs. Therefore, it minimizes the sum of journey costs. The assignment is optimal for workers because it minimizes workers' total detour costs. An instance of the maximum 3D matching is therefore corresponding to an instance of the general WAP. ■

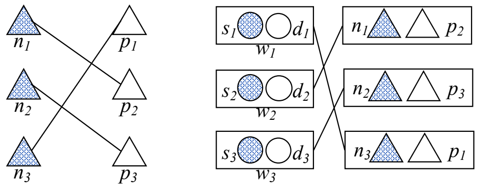
## V. Fixed Time Slice Optimization

### A. Overview

In this section, we introduce a 3-approximation algorithm to solve the WAP in a Euclidean plane. Our algorithm is based on two stages of matchings. In the first round of matching, we only consider the bike movement between stations and ignore workers' sources and destinations. This generates the rent/return station pairs, i.e., a bike rented from a station in $N$ should be returned to the corresponding station in $P$. In the second round of matching, the algorithm considers another bipartite graph containing set $W$ and $N \times P$ which is used to match workers with rent/return station pairs. An illustration of the algorithm procedure is shown in Fig. 5.

### B. Two-Round Matching algorithm

In the Two-Round Matching (TRM) algorithm, we we first apply the weighted bipartite matching algorithm on the bipartite graph constructed by $N$ and $P$. The weighted match found by the algorithm indicates the optimal combination of rent and return stations which can achieve the minimum moving distance. The procedure of the first round of matching is shown in Algorithm 1. Lines 1-4 in the algorithm are to construct the



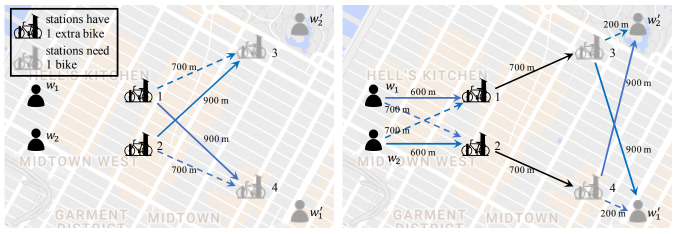
(a) Matching between stations

(b) Matching between workers and station pairs

Fig. 5. The procedures of Two-Round Matching algorithm.

bipartite matching graph. In the weighted matching algorithm, a vertex can only be matched once. Therefore, we duplicate the stations based on their rebalancing targets to guarantee that the target of each vertex in set $N$ (or $P$) is $-1$ (or 1). The vertex set is initialized in line 5 and the edge set is initialized in lines 6-7. We apply weighted matching on the bipartite graph in line 8. Notice that the weighted matching algorithms are designed for maximization. To find the non-zero minimum perfect matching, we modify the edge weight when using the maximum weighted matching algorithm. The modified weight is calculated by subtracting the original weight from a large number. It is $e' = LC - e$, where $LC$ is a large constant number and $e'$ is the modified edge weight.

The second round of matching generates the assignments for workers, which is also a bipartite matching problem. We construct another bipartite graph where one part of the graph represents workers $W$ and the other part represents rent/return station pairs $N \times P$. The weighted edge between a worker $w$ and a combination $(n, p)$ equals the total moving distance of $w$ if he/she rents a bike at station $n$ and returns it at $p$. Notice that we use the workers' total moving distance instead of their overall detour as the edge weight. Actually, the detour reaches the minimum when the total moving distance is minimized, because the total length of workers' original journeys is constant. The procedure of the second round of matching is shown in Algorithm 2. Line 1 initializes the vertex set $V' = W \cup X$ and edge set $E' = \emptyset$. The weight edges are added to $E$ in lines 2-3. Similarly, as in Algorithm 1, we modify the edge weights and apply the maximum weighted matching algorithm in line 5. Finally, an edge set $M$ is calculated, and it constitutes the assignments for workers. The workers' overall moving distance is the sum weight of edges in $M$, and is denoted by $\|M\|$.



(a) Matching between positive and negative stations (b) Matching between users and rent/return pairs

Fig. 6. Illustration of the two-round matching algorithm.

We apply the TRM algorithm to the example in Fig. 6. We first match rent and return stations. The dashed and solid line in Fig. 6(a) show 2 possible matches, and the total distance of the match indicated by dashed line is smaller. Therefore, the extra bikes in station 1 should be delivered to station 3 and the bikes in station 2 should be moved to 4, as shown by the dashed line. The second round of matching works to assign each worker to a proper pair of matched stations. Fig. 6(b) shows all possible assignments. The dashed blue line represents an assignment $\{(w_1,(2,4)),(w_2,(1,3))\}$. It means that worker $w_1$ is assigned to rent a bike from station 2 and then return it at station 4, and that worker $w_2$ is assigned to rent at station 1 and return at station 3. In this assignment, the total moving distance of workers is $3200\text{m}$. The solid blue line represents the assignment $\{(w_1,(1,3)),(w_2,(2,4))\}$. Here, the overall moving distance is $4400\text{m}$. Therefore, the algorithm picks $\{(w_1,(2,4)),(w_2,(1,3))\}$ as the final assignment. In comparison, the assignment generated by the simple greedy algorithm, which assigns each user to rent and return at the station nearest to his/her origin and destination respectively, is $\{(w_1,(1,4)),(w_2,(2,3))\}$. The total movement distance is $3400\text{m}$, which is larger than the distance found by the TRM.

Our algorithm can find a better assignment than the simple greedy algorithm because our algorithm checks more situations in the search space compared to the simple greedy algorithm. The search space represents all the possible combinations of workers, rent stations, and return stations. Clearly, the size of the search space is extremely large (i.e., $O((|B|!)^2)$), and cannot be traversed in polynomial time. Our algorithm can efficiently reduce the size of the search space by ignoring the combinations whose total moving distance between rent and return stations is not minimized. Although the optimal solution may be ignored in these reductions, we prove our algorithm has a 3 approximation ratio. Our algorithm balances the algorithm optimality and time complexity, and can be applied in large BSSs with hundreds of stations.

***Theorem** 2:* The time complexity of TRM is $O(|W|^3)$.

*Proof:* The time complexity of a maximum weighted bipartite matching algorithm can be $O(|V|^2\log|V|+|V|\cdot|E|)$ when using the Fibonacci heap [23]. We construct a complete bipartite graph with $O(|W|)$ nodes and $O(|W|^2)$ edges, and the time consumption of two rounds of matching is $O(|W|^3)$. The consumption of the remaining steps is $O(|W|)$. Therefore, the overall time consumption of our algorithm is $O(|W|^3)$. ■

***Theorem** 3:* The TRM is a 3-approximation algorithm for the WAP in the Euclidean space.

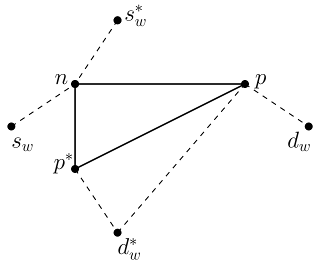


Fig. 7. The relation between the TRM output and the OPT.

*Proof:* The calculation of the 3-approximation ratio is based on the triangle inequality and the optimality of each matching stage of TRM. For each negative station $n\in N$, there is a corresponding positive station $p\in P$ which is assigned in the first round of TRM. In addition, the station pair $(n,p)$ is matched to a worker $w$ with source $s_w$ and destination $d_w$ in the second round of TRM. We assume that in the optimal solution, the station $n$ should be paired with station $p^*$, and the station pair $(n,p^*)$ should be balanced by the worker $w^*$ whose source and destination are $s_w^*$ and $d_w^*$ respectively. The relation among these nodes is shown in Fig. 7, which is a geometric graph in the Euclidean space. The total moving distance of workers generated by our algorithm is $\sum_{n\in N}(dis(s_w,n)+dis(n,p)+dis(p,d_w))$, and the optimal value is $\sum_{n\in N}(dis(s_w^*,n)+dis(n,p^*)+dis(p^*,d_w^*))$.

Based on the triangle inequality, we can conclude that $dis(p,p^*)\le dis(n,p)+dis(n,p^*)$ and $dis(p,d_w^*)\le dis(p,p^*)+dis(p^*,d_w^*)$ for each $n\in N$. According to the optimality of the first round of matching, we can conclude that $\sum_{n\in N}dis(n,p)\le\sum_{n\in N}dis(n,p^*)$. Besides, the optimality of the second round of matching guarantees that $\sum_{n\in N}(dis(s_w,n)+dis(p,d_w))\le\sum_{n\in N}(dis(s_w^*,n)+dis(p,d_w^*))$. Combining these inequity relationships, we have:

$$\begin{aligned}
&\sum_{n\in N}(dis(s_w,n)+dis(p,d_w))\le\sum_{n\in N}(dis(s_w^*,n)+dis(p,d_w^*))\\
&\le\sum_{n\in N}(dis(s_w^*,n)+dis(p,p^*)+dis(p^*,d_w^*))\\
&\le\sum_{n\in N}(dis(s_w^*,n)+dis(n,p)+dis(n,p^*)+dis(p^*,d_w^*))\\
&\le\sum_{n\in N}((dis(s_w^*,n)+2dis(n,p^*)+dis(p^*,d_w^*)).
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\sum_{n\in N}(dis(s_w,n)+dis(n,p)+dis(p,d_w))\\
&=\sum_{n\in N}(dis(s_w,n)+dis(p,d_w))+\sum_{n\in N}dis(n,p)\\
&\le\sum_{n\in N}((dis(s_w^*,n)+2dis(n,p^*)+dis(p^*,d_w^*))+\sum_{n\in N}dis(n,p^*)\\
&\le 3\sum_{n\in N}(dis(s_w^*,n)+dis(n,p^*)+dis(p^*,d_w^*))=3OPT.
\end{aligned}$$

The 3-approximation holds. ■

## VI. Multiple Slice Optimization

Although the future demand of the system can be predicted, it is hard to decide how many time slices the scheme should

**Algorithm 3** $k$-slice Greedy Algorithm.

```
Input: station capacity vector $\boldsymbol{\eta}_B$, current station state vector $\boldsymbol{\phi}_B(t_0)$, the demand prediction $\{\boldsymbol{\tau}_B(t)\}(t_0 \le t \le t_0 + k)$ of following time slices
Output: rebalancing target vector $\boldsymbol{\rho}_B(t_0)$
1: $\alpha_b \leftarrow \eta_b - (\phi_b(t_0) + \max[\sum_{l=t_0}^{t_0} \tau_b(l), \cdots, \sum_{l=t_0}^{t_0+k} \tau_b(l)])$, $\beta_b \leftarrow 0 - (\phi_b(t_0) + \min[\sum_{l=t_0}^{t_0} \tau_b(l), \cdots, \sum_{l=t_0}^{t_0+k} \tau_b(l)])$, $\forall b \in B$
2: positive target vector $\boldsymbol{\rho}_+ \leftarrow \mathrm{find}(\boldsymbol{\beta}_B > 0)$, negative target vector $\boldsymbol{\rho}_- \leftarrow \mathrm{find}(\boldsymbol{\alpha}_B < 0)$, $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho}_- + \boldsymbol{\rho}_+$
3: if $\mathrm{sum}(\boldsymbol{\rho}_+, \boldsymbol{\rho}_-) > 0$ then
4:     Iteratively decrease $\rho_b, \arg\min_b(\boldsymbol{\beta}_B)$ (use final state to break tie) by 1, until $\mathrm{sum}(\boldsymbol{\rho}_+, \boldsymbol{\rho}_-) = 0$
5: else if $\mathrm{sum}(\boldsymbol{\rho}_+, \boldsymbol{\rho}_-) < 0$ then
6:     Iteratively increase $\rho_b, \arg\max_b(\boldsymbol{\alpha}_B)$ by 1, until $\mathrm{sum}(\boldsymbol{\rho}_+, \boldsymbol{\rho}_-) = 0$
7: return $\boldsymbol{\rho}_B(t_0)$ as the rebalance target vector
```

look ahead due to the uncertainty of the system in the temporal domain. If the algorithm is short-sighted, some self-balancing situations may be ignored. On the other hand, infeasible situations may occur if the algorithm considers too many time slices. [2] proposes a greedy approach that tries to find rebalancing targets which can maximize the future living periods for each station. We find out that their approach may generate unnecessary bike movements and their algorithm does not consider that the sum of rebalance targets over overflow and underflow stations should be zero. Inspired by their idea, we investigate two algorithms that are outlined below.

## A. *$k$-slice greedy algorithm*

Firstly, we introduce a heuristic greedy algorithm: $k$-slice Greedy Algorithm ($k$GA). The idea is to look ahead $k$ time slices, and find a proper rebalancing target for the following $k$ time slices, where $k$ is constant and is set by the BSS operator. The operator conducts a round of rebalancing for every $k$ time slices. Recall that a proper target should minimize the number of bikes needed to move, and the sum of targets for all stations should be 0. In $k$GA, we firstly choose the target for each station greedily, and then check the sum of the chosen targets over all stations.

The procedures of $k$GA are illustrated in Algorithm 3. We first calculate the range of the feasible rebalancing target for each station. The largest value in the range for station $b$ is denoted by $\alpha_b$. A positive $\alpha_b$ means that at most, $\alpha_b$ bikes can be moved into $b$ within the following $k$ time slices. The smallest value in the range is denoted by $\beta_b$, and a negative $\beta_b$ means at most $|\beta_b|$ bikes can be removed from $b$. If $\alpha_b < 0$, the station $b$ will face overflow events in the following $k$ slices and has to remove bikes beforehand. In contrast, $\beta_b > 0$ has the opposite meaning. For example, in Table II, the initial state is $(0, 3, 3, 1)$, and the demand during the first slice is $(-1, +1, +2, -1)$. The capacity for each station is 5. In 1GA, $\alpha_1 = 5$ and $\beta_1 = 1$. It means that we can add a minimum of 1 bike and a maximum of 5 bikes to station 1 without causing OoS events. $\alpha_b$ and $\beta_b$ indicate the range of feasible rebalancing targets of station $b$. $\boldsymbol{\alpha}_B$ and $\boldsymbol{\beta}_B$ record the ranges for all stations in $B$. The $k$GA will greedily choose the smallest number in this range. However, the greedy strategy cannot guarantee that the sum of targets among all stations is 0. To solve this, $k$GA iteratively adjusts the target. If the sum of targets is larger than 0, $k$GA iteratively adds one negative target to the station that has the minimum $\beta_b$ and uses the highest number of bikes on-dock (after $k$ time slices) to break ties. If the sum is greater than 0, $k$GA iteratively adds one positive target to the station that has the maximum $\alpha_b$ and uses the highest number of docks to break ties. Notice that if the demand of a station exceeds the station capacity, $k$GA cannot find a feasible target. This happens when the adjustments in lines 4 or 6 cannot proceed. We assume that the operator can set the length of each time slice properly to avoid such cases.

**Algorithm 4** Greedily Look-ahead Algorithm

```
Input: station capacity vector $\boldsymbol{\eta}_B$, current station state vector $\boldsymbol{\phi}_B(t_0)$, the demand prediction $\{\boldsymbol{\tau}_B(t)\}(t_0 \le t \le |T|)$ of following time slices
Output: rebalancing target vector $\boldsymbol{\rho}_B(t_0)$
1: Calculate longest survival time $T_b$ under different targets $\rho_b(t_0)$ for each station $b$, i.e., $\arg\max_{\rho_b(t_0)}(T_b | 0 \le \sum_{t=t_0}^{t_0+T_b} \tau_b(t) + \phi_b(t_0) + \rho_b(t_0) \le \eta_b)$, $\forall b \in B$
2: $k \leftarrow \min(T_b, \forall b)$
3: Apply $k$GA to calculate $\boldsymbol{\rho}_B(t_0)$. Repeatedly decrease $k$ if $k$GA cannot find a feasible set of targets.
4: return $\boldsymbol{\rho}_B(t_0)$ as the rebalance target
```

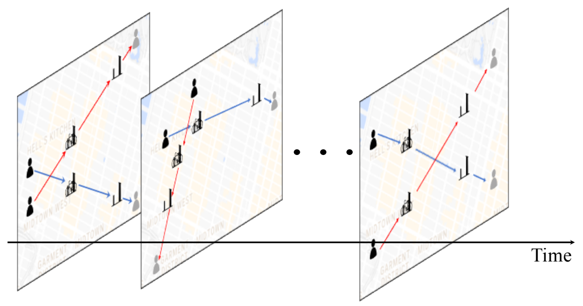


Fig. 8. An example of multiple slice rebalance scenario.

## B. *Greedily look-ahead algorithm*

Besides manually defining $k$ for $k$GA, we can also design a greedy algorithm to automatically select the largest $k$. Following this approach, we propose another target setting algorithm: Greedily Look-ahead Algorithm (GLA).

The procedures of the GLA are illustrated in Algorithm 4. The survival time (i.e., the time duration before OoS events occur) of a station is related to the rebalancing target. In line 1, GLA tests the survival time of each station $b$ under all possible rebalancing targets, and stores the longest one, which is denoted by $T_b$. In line 2, the GLA sets $k$ as the minimum $T_b, \forall b \in B$. Unfeasible situations may occur if $k$ is large. To deal with this case, the GLA repeatedly decreases $k$ by 1 and then calls $k$GA until a feasible target is found.

It seems looking up more time slices (i.e. larger $k$) can generate better results in terms of the total moved bikes

TABLE II

| Time slice | Demand $\tau$ | Greedily Look-ahead Algorithm | | 1-slice Greedy Algorithm | |
|---|---|---|---|---|---|
| | | State without rebalancing | Rebalancing target | State without rebalancing | Rebalancing target |
| 1 | $(-1,+1,+2,-1)$ | $(-1,4,5,0)$ | $(+1,-1,0,0)$ | $(-1,4,5,0)$ | $(+1,0,-1,0)$ |
| 2 | $(+3,-1,-3,+2)$ | $(3,2,2,2)$ | $(0,0,0,0)$ | $(3,3,1,2)$ | $(0,0,0,0)$ |
| 3 | $(+3,-4,+5,-3)$ | $(6,-2,7,-1)$ | $(-1,+2,-2,+1)$ | $(6,-1,6,-1)$ | $(-1,+1,-1,+1)$ |
| 4 | $(0,0,0,0)$ | $(5,0,5,0)$ | $(0,0,0,0)$ | $(5,0,5,0)$ | $(0,0,0,0)$ |

in a day. However, this is not the case, since the greedy algorithm cannot generate an optimal substructure in our problem. We use the following counterexample in Table II to illustrate this point. Assume the current state is (0, 3, 3, 1) and the demand vectors in the following time slices are $\tau(1) = (-1,+1,+2,-1)$, $\tau(2) = (+3,-1,-3,+2)$, $\tau(3) = (+3,-4,+5,-3)$ and $\tau(t) = (0,0,0,0)\ \forall t \geq 4$. Notice that the sum of demands in each time slice may not be 0, while the sum of targets should be 0. According to the table, three bikes need to be moved if the operator follows the targets generated by GLA. However, the operator only needs to move two bikes if the 1GA is used.

***Theorem 4:*** The time complexity of $k$GA and GLA is $O(|B|\log|B|)$.

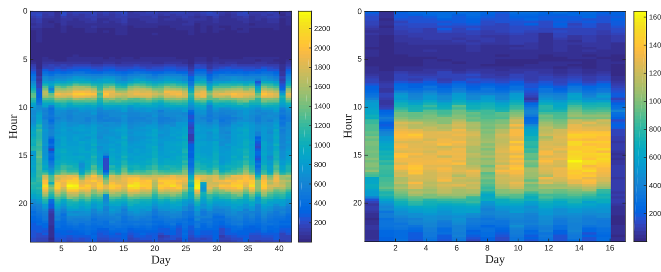
*Proof:* First we analyze the time complexity of $k$GA. The calculation of the largest/smallest feasible targets for all stations costs $O(|B|)$ time. It is the same for the calculation of the positive/negative demand vector, since they are founded by a traversal of each $\alpha_b/\beta_b$ (with constant size $k$). If the sum of positive and negative targets doesn't equal 0, the algorithm will bridge the gap by 1 in each iteration. Since the maximum difference between the sum of positive targets and the sum of negative targets does not exceed $|B|$, the maximum amount of iterations will not exceed $O(|B|)$. In each iteration, the algorithm chooses the station with largest lower/upper bound value. If a heap is maintained, $O(\log|B|)$ time is needed to adjust the head in each iteration. Therefore, the overall time complexity of $k$GA is $O(|B|\log|B|)$.

The difference between $k$GA and GLA is that the GLA will determine the value $k$ by itself and then apply $k$GA. The determination of $k$ requires a single traversal of the input time slices. Since the amount of time slices is a constant, the GLA has the same time complexity as $k$GA, which is $O(|B|\log|B|)$. ■

## VII. Experiment

### A. Real-world dataset and statistics

We use the public data of NYC Citi Bike to construct our real-world dataset: the NYC dataset. We use the history trip data from 8/1/2017 to 9/30/2017. The data contains the records of each trip including trip duration (in seconds), trip start/stop time and date, start/end station ID, and latitude/longitude, etc. Our NYC dataset contains more than 1.5 million trip records and 328 bike stations. The statistics of trip duration and temporal usage distribution of trips on 8/1/2017 (Tuesday) shows that more than 55% of trips are shorter than 10 minutes. It is reasonable to infer that most workers can ride a bike from one station to another within 10 minutes. It provides us with a clue for choosing the slice length.



(a) weekdays (b) weekends

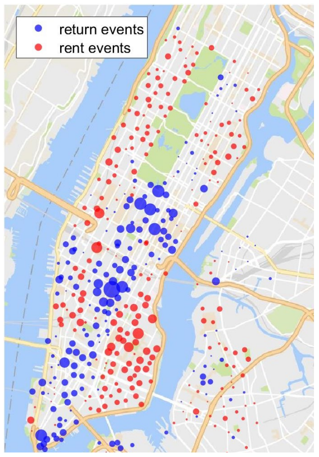
Fig. 9. Usage pattern in weekdays and weekends.

The temporal imbalance of the usage distribution is shown in Fig. 9 and the spatial imbalance is shown in Fig. 10. To illustrate temporal imbalance, we sum up the users' demands, including rent events and return events, among all stations for each 15-min time slice in a day. Fig. 9(a) and Fig. 9(b) illustrate the usage distribution on weekdays and weekends, respectively. It can be seen that there are two peak periods on weekdays and one on weekends. To show spatial imbalance, we plot the usage of each station in Fig. 10 for AM rush hours (8:00 - 10:00 AM) and PM rush hours (5:00PM - 7:00 PM). The blue nodes represent stations whose return events are more frequent (i.e. the number of bikes at these stations increases) in the given time window. The node's diameter is proportional to demands of the corresponding station. The red nodes have the opposite meaning. The demand is predictable because of the existence of the usage pattern. Usage prediction methods such as [2] can be added into our scheme to generate the near-future demand for $k$GA and GLA.

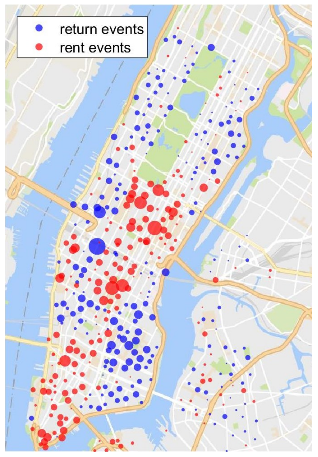
### B. Synthetic Dataset

The synthetic dataset is used as a supplement for the real-world dataset. The location of each bike station in the NYC dataset is static, and so is the density of bike stations. We want to test how the density of stations affects the performance of TRM, since the station density of BSSs in different cities may vary significantly. Therefore, we build a synthetic dataset which contains several station sets with different densities.

In our synthetic dataset, we randomly generate the locations of bike stations following a uniform distribution. The source and destination location distributions of workers are also uniform. The density of stations is measured by the expected number of stations in a $5 \times 5\ \mathrm{km}^2$ square. The capacity of each station is set as 20 and the initial inventory of each station is set as 3/4 of the capacity, which is 15. The number of rent and return events of stations in each time slice is generated by the



(a) AM rush hours (8:00 - 10:00 AM)



(b) PM rush hours (5:00 - 7:00 PM)

Fig. 10. The users' demands in morning and evening rush hours.

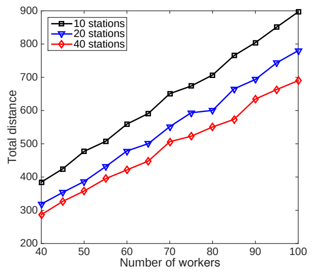
Poisson process with parameter $\lambda = 7$ which is the average number of daily rent events in the NYC dataset.

## C. *Comparison algorithms*

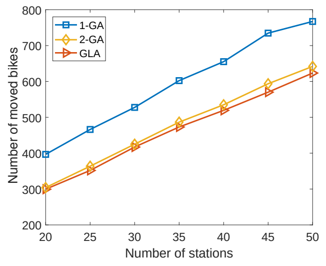
The *Branch-and-Bound* (BB) algorithm is a global optimization method which is used to find the optimal solution to the WAP problem. Although it cannot be used in realistic scenarios, it can provide an optimal result when the input size is small. The optimal result can be used to evaluate the performance of our two-rounds matching algorithm. The key issue in the branch-and-bound algorithm is designing a proper lower-bound heuristic for each state. It is defined as the sum weight of chosen $k$ stations plus the sum of the $n-k$ smallest weights. The complexity of calculating this lower-bound heuristic is factorial. Therefore, the comparison is conducted with a small input size.

The *Local Search* (LS) algorithm is a local optimization approach. Finding the weighted 3D matching can be seen as finding the minimum weighted subfamily of pairwise disjoint sets and the size of each set is equal to 3. To the best of our knowledge, the approximation algorithm with the tightest bound for the problem is proposed by Berman in [24]. The basic idea of the algorithm is local search. The approximation ratio of the local search algorithm is $(k+1+\epsilon)/2$ and the time complexity is $m^{O(k)}$, in which $m$ is the number of nodes in the intersection graph. In our problem $k=3$, and $m=|W|^3$. Therefore, the approximation ratio of the algorithm is equal to $2+\epsilon$ with time complexity $O(|W|^9)$. Assuming that a BSS only needs to move 50 bikes in each time slice, it costs more than 120 hours to calculate a result on a 4.5GHz CPU theoretically, which is not applicable. In contrast, our TRM costs 3.85s on a 4.5GHz CPU with an input size of 50, and 357s on a 1GHz CPU with an input size of 100.

In addition, a *Greedy* algorithm is used as a baseline approach. In the greedy algorithm, each worker chooses the nearest station in $N$ to rent a bike, and returns it to the station in $P$ that is nearest to his/her destination. If too many workers choose to rent/return at the same station, the situations can be arbitrarily broken, i.e., simply making a random selection.



(a) Under different station densities



(b) Comparing GLA with $k$GA

Fig. 11. Performance comparison.

## D. *Experimental Settings*

In the synthetic dataset, we focus on testing the performance of TRM on different station densities. Totally, we choose three densities: 10, 20, 40 to represent sparsely, regularly, densely distributed stations, respectively.

In the real-world dataset, we first compare the performance of TRM with others. The rebalancing target is calculated by 1GA. Notice that the branch-and-bound algorithm is extremely time-consuming, so we extract a subset of stations and choose a short slice length (20 min) to decrease the number of bikes need to be moved. The stations are randomly extracted from the NYC dataset in a $2 \times 2$ $\text{km}^2$ area. We use the average total detour in a time slice to measure the performance of the TRM. For generalization, we repeat the experiment 100 times over 10 different regions and record the average results.

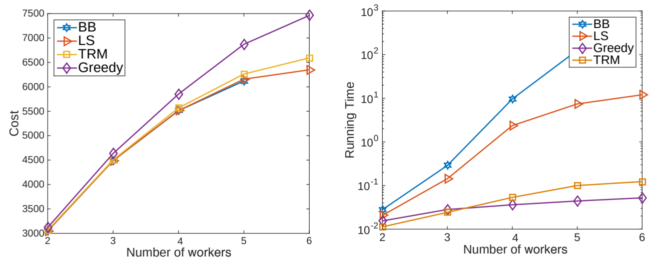
The other experiment based on real-world datasets focuses on the comparison between $k$GA and GLA. We assume the rebalancing period is from 0:00 AM to 11:59 PM. The initial state and capacity of each station are extracted from the NYC dataset. However, the OoS events are not recorded by the system. Therefore, the demand for each time slice is generated by the prediction algorithm in [2]. Similarly, to smooth out the result, the experiment is repeated 40 times.

## E. *Evaluation Results*

Firstly, we present the evaluation results of TRM with different densities in Fig.11(a). In total, there are 3 lines in the figure. Each line represents a unique station density. From the simulation result, we can find out that if the number of workers is fixed (meaning the number of bikes that need to be moved is fixed), a larger station density costs a smaller total move distance. This is easy to explain because higher station density may lead to smaller spatial intervals between stations, and thus leads to a smaller total moving distance.

Fig. 11(b) illustrates the comparison between GLA and $k$GA algorithms, we can find out that the GLA outperforms 1GA and 2GA. That is to say, although we find an example where 1GA outperforms GLA, GLA has better performance in general cases. It is not surprising since the GLA can adjust the time slice to look ahead automatically and it is more flexible than $k$GA. In the experiment, we also track the $k$ chosen by GLA in each iteration and find that the $k$ value barely exceeds 4, which gives a clue for deciding the rebalancing frequency.

The comparison between TRM and other algorithms is shown in Fig. 12. The number of workers represents the average number of workers recruited in each time slice. The



(a) Comparison on distance (b) Comparison on running time

Fig. 12. Comparison between TRM and existing algorithms.

cost represents the average total detour in each slice. From Fig. 12(a), we can conclude that the performances of TRM and LS are similar. They both outperform the greedy algorithm and are not far from the optimal solution. However, considering the running time in Fig. 12(b), we can conclude that the running time of LS is larger than that of TRM. If the problem size is larger, applying the LS to solve the WAP is no longer appropriate since it costs more than one day to provide the solution. Although TRM slightly underperforms compared to LS, TRM is more efficient in terms of its running time.

## VIII. Conclusion

In this paper, we propose a crowdsourcing BSS rebalancing scheme that recruits workers to rebalance the system. The user dynamics in spatial and temporal domains bring uncertainties when designing such a system. To decouple effects from these two domains, we slice the problem in the temporal domain and generate a sequence of slices with a fixed time duration. In each slice, we formulate the WAP in the spatial domain. The general WAP is proven to be NP-hard. A 3-approximation algorithm based on two rounds of matching is proposed to solve WAP in the Euclidean plane. Over multiple time slices, the rebalance frequency and targets affect the number of bikes that are moved. We investigate several approaches, including $k$GA (which only looks ahead for a single time slice) and GLA (which looks ahead for as many time slices as possible). The experiments are conducted based on both synthetic and real-world datasets. Results of the comparison between TRM and other algorithms in a fixed time slice show that although TRM slightly underperforms compared to a local-search approach algorithm, it runs much faster and can be applied to large-scale real-world BSSs. Over multiple time slices, the comparison between $k$GA and GLA shows that GLA can efficiently decrease the number of workers recruited over a day in the real-world dataset, although we show it may generate unnecessary bike movements in some cases.

## Acknowledgement

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128.

## References

[1] E. Fishman, "Bikeshare: A review of recent literature," *Transport Reviews*, vol. 36, no. 1, pp. 92–113, 2016.

[2] J. Liu, L. Sun, W. Chen, and H. Xiong, "Rebalancing bike sharing systems: A multi-source data smart optimization," in *Proc. of ACM KDD*, 2016, pp. 1005–1014.

[3] J. Liu, L. Sun, Q. Li, J. Ming, Y. Liu, and H. Xiong, "Functional zone based hierarchical demand prediction for bike system expansion," in *Proc. of ACM KDD*, 2017, pp. 957–966.

[4] J. Froehlich, J. Neumann, N. Oliver *et al.*, "Sensing and predicting the pulse of the city through shared bicycling." in *Proc. of IJCAI*, vol. 9, 2009, pp. 1420–1426.

[5] A. Kaltenbrunner, R. Meza, J. Grivolla, J. Codina, and R. Banchs, "Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system," *Pervasive and Mobile Computing*, vol. 6, no. 4, 2010.

[6] Y. Li, Y. Zheng, H. Zhang, and L. Chen, "Traffic prediction in a bike-sharing system," in *Proc. of ACM SIGSPATIAL*, 2015.

[7] L. Chen, D. Zhang, L. Wang, D. Yang, X. Ma, S. Li, Z. Wu, G. Pan, T.-M.-T. Nguyen, and J. Jakubowicz, "Dynamic cluster-based over-demand prediction in bike sharing systems," in *Proc. of ACM UbiComp*, 2016, pp. 841–852.

[8] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, 2014.

[9] A. Singla, M. Santoni, G. Bartók, P. Mukerji, M. Meenen, and A. Krause, "Incentivizing users for balancing bike sharing systems." in *Proc. of AAAI*, 2015, pp. 723–729.

[10] M. Charikar, S. Khuller, and B. Raghavachari, "Algorithms for capacitated vehicle routing," *SIAM Journal on Computing*, vol. 31, no. 3, pp. 665–682, 2001.

[11] J. Liu, Q. Li, M. Qu, W. Chen, J. Yang, H. Xiong, H. Zhong, and Y. Fu, "Station site optimization in bike sharing systems," in *Proc. of IEEE ICDM*, 2015, pp. 883–888.

[12] L. Chen, D. Zhang, G. Pan, X. Ma, D. Yang, K. Kushlev, W. Zhang, and S. Li, "Bike sharing station placement leveraging heterogeneous urban open data," in *ACM UbiComp*, 2015.

[13] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng, "Planning bike lanes based on sharing-bikes' trajectories," in *Proc. of ACM KDD*, 2017, pp. 1377–1386.

[14] J. W. Yoon, F. Pinelli, and F. Calabrese, "Cityride: a predictive bike sharing journey advisor," in *Proc. of IEEE MDM*, 2012.

[15] J. Zhang, P. Lu, Z. Li, and J. Gan, "Distributed trip selection game for public bike system with crowdsourcing," in *Proc. of IEEE INFOCOM*, 2018.

[16] D. Chemla, F. Meunier, and R. W. Calvo, "Bike sharing systems: Solving the static rebalancing problem," *Discrete Optimization*, vol. 10, no. 2, pp. 120–146, 2013.

[17] S. Ghosh, M. Trick, and P. Varakantham, "Robust repositioning to counter unpredictable demand in bike sharing systems," in *Proc. of IJCAI*, 2016, pp. 3096–3102.

[18] Y. Duan, J. Wu, and H. Zheng, "A greedy approach for vehicle routing when rebalancing bike sharing systems," in *Proc. of IEEE GLOBECOM*, 2018.

[19] A. Waserhole and V. Jost, "Pricing in vehicle sharing systems: Optimization in queuing networks with product forms," *EURO J. Transp. Logist.*, vol. 5, no. 3, pp. 293–320, 2016.

[20] L. Pan, Q. Cai, Z. Fang, P. Tang, and L. Huang, "Rebalancing dockless bike sharing systems," *arXiv preprint arXiv:1802.04592*, 2018.

[21] Y. Duan and J. Wu, "Optimizing rebalance scheme for dockless bike sharing systems with adaptive user incentive," in *Proc. of IEEE MDM*, 2019.

[22] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, p. 85.

[23] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987.

[24] P. Berman, "A d/2 approximation for maximum weight independent set in d-claw free graphs," in *Scandinavian Workshop on Algorithm Theory*. Springer, 2000, pp. 214–219.