Capturing Unattended Demand In Bike-Sharing Systems

Manuel Henrique de Almeida Rodrigues

manuel.a.rodrigues@tecnico.ulisboa.pt - 78457

1 Introduction

The BSS consists in an platform that allows people to use bicycles as a way of transportation without having to bring their own personal bicycles. The main concept is to pick up a bicycle on one of the many stations spread across the city, and then drop it off on another station for somebody else to use it after [1,2]. Bicycles can be manual or have electric assistance [3]. Paris, London and New York are just a few examples of cities that already deployed an extensive bike-sharing station network [4,5].

1.1 Analyzing infrastructural BSS requirements

Deploying a fully functional and efficient BSS is something that comes as a great challenge. If a BSS is operating under bad planning and without a constant search for improvement, things won't work, and people won't look at shared bicycles as a valid alternative to their fossil fuel powered vehicles. Above all other, **convenience** is the main factor when deciding between using shared bikes or not [5]. The low pollution and physical activity factors are obviously very relevant, but at the end of the day people will choose the fastest, more functional and less stressful way of transportation. To achieve appropriate levels of convenience and operability, cities should launch BSSs that are as big and dense as possible, also covering low-income neighborhoods as well as other high density areas [5]. Station balance and demand analysis is directly related to this. If the stations are not placed taking the consumer's needs and the city's human flow into account, we may end up having considerable issues. Along others, we may have stations that never have available bikes to pick up, increasing waiting times; stations that are always full, making them unavailable for dropping off bicycles; and people that are forced to take a path so much longer than they needed, due to unavailable or unexisting stations, that the trip was simply not worth it [6,7]. All these events affect convenience for the customer of the service. This is why deciding **station capacity, density and placement** is directly related to the quality of the BSS, since users do not want to have large waiting times, do not want to walk too far to pick-up a bicycle, and do not want to drop off a bike too far from their final destination either [5,8]. In fact, a study carried on the US suggests that, for the service to be usable, the stations should not be placed more than 300 meters apart from each other, since people are not willing to walk more than 5 minutes to pick-up a bicycle or to reach their destination [5].

Summing up, the most relevant questions that need to be answered concerning infrastructural BSS requirements are: Where do we place our bike stations? What should be their capacity? How to efficiently expand the station network?

Currently, those decisions are mostly made based on direct observations of the network. For example, by measuring the pick-up and drop-off demands of a station it is possible to make assumptions about its balancing and performance [2,7]. In addition, conducting satisfaction surveys to the users of the BSS and other crowd sourced methods is also a common way to get feedback [9,1]. However, we believe that this methodology is too superficial and rather incomplete. Stations can have balanced demand values and still present issues, and surveys rarely catch and effectively measure all the problems. Our vision is that by monitoring the trajectories made by the users of the BSS we will be able to obtain a detailed and more realistic perspective of daily events.

Furthermore, with this proposal we hope to identify a new scenario that is currently not being monitored: **silent unattended demand**. As mentioned before, a very common scenario when using a BSS is being unable to pick-up/drop-off a shared bicycle on the more convenient station (unattended demand) [7,6,10]. In most cases the user will simply decide to use another station without giving any feedback to the system, which makes it difficult to register these occurrences (silent fault). To target this, simply

monitoring the trajectory between two stations is not enough. Inferring what is the user's starting point before renting a shared bicycle, and also his/her final destination after dropping off the bicycle, becomes relevant. If we're able to know these two physical coordinates of each trajectory, we can automatically determine if the user utilized the most convenient station considering his starting point/destination. And if he didn't, we can raise a new set of questions: Was the station empty/full? Is there an undesirable external factor? Is it for some reason hard to reach? Should it be repositioned?

1.2 Approaching Mobile Crowdsensing

The main goal of this proposal is to be able to automatically detect when a **silent unattended demand** situation occurs using sophisticated and precise techniques based on the sensorial capabilities of the smartphone. Then, the trip meta-data will be stored in an external database, from which a back-end server will fetch the data in order to apply clustering techniques and output structured files that will allow to generate visual representations of the data. By doing so, we are helping the BSS operator improve their service, since it gives them a tool to monitor and visualize the occurrence of an event that they could not monitor before.

Nevertheless, in order to know precisely when to start trying to detect the user's final destination, it is relevant to know the moment the user drops-off his/her bicycle and starts walking, since that moment works as a trigger. We are assuming, through some experience, that usually the BSS apps prompt the user to click a button when he is about to pick-up a bicycle. In contrast, there is no user input for drop-offs, since the user simply locks the bike on the station and the server registers the drop-off automatically. For that matter, the drop-off moment of the trajectories will have to be detected automatically in runtime.

We intend to achieve the mentioned objectives, and help optimizing existing BSSs, by monitoring and processing user trajectories. The method that we propose matches with the definition of mobile crowdsensing, which is the collection and analysis of data, extracted from a large group of people using a mobile application that is capable of taking advantage of the sensorial capabilities that mobile devices have nowadays [11,12].

With that said, our proposal consists in a functional **library** that contains an **Application Programming Interface (API)** that can be integrated on the architecture of BSS mobile applications. The purpose of the library is to provide an API with a set of methods that will enable the main application to collect and automatically detect relevant points in a trajectory, such as the moment the user drops-off his bicycle and starts walking towards the final destination, and also that same final destination. Furthermore, the library's outputs will be connected to a back-end server that will process, cluster and generate visual representations of the collected data. The visualizations will consist on web application with an intuitive dashboard.

1.3 Recognizing the technical challenges

While developing the proposed platforms many challenges came our way, concerning both the server side component and the mobile client component.

First of all, we had to **automatically detect** the moment of drop-off and also the final destination of the user, in order to identify silent unattended demand situations. It is not a safe practice to assume that the users will always be willing to interact with the app on these specific moments, and so we will had to apply some rather challenging techniques to effectively infer these segments in real-time. These techniques mostly rely on the sensorial capabilities of the mobile device, such as location, speed, acceleration and bearing.

On the server side, we'll need to store all user interactions with the mobile app for further analysis, including trip meta-data and station statistics. This process immediately raises **anonymity and privacy** concerns. To mitigate them, we will explore some approaches to achieve acceptable degrees of privacy of the data, especially personal data like user final destinations, which contain information about where the user has been and at what time.

We also provide an **energy efficient** application, since the location sensors are one of the most battery-consuming on a smartphone [13]. Given that this application would require high levels of location accuracy, overcoming this challenge may be more complex compared to apps where accuracy is not that important.

Besides energy efficiency, the location also brings out another challenge: even using high-

accuracy settings, it's fairly difficult for the sensor to be precise on every coordinate throughout the trajectory [14]. After evaluating our library through diverse kinds of metrics, we believe to have achieved an accurate, effective, useful, and energetically efficient solution, which was our purpose.

2 Related Work

2.1 Bike station demand, balance and expansion analysis

Measuring and analyzing the performance of a bike-sharing service is a crucial factor concerning the quality and future improvements of the network. As mentioned before, the quality of a BSS can be measured by studying its bike-stations, namely their respective demand flow and density [2,5]. This section introduces the necessary concepts that need to be taken into account in order to further explore these indicators.

2.2 Mobile Crowdsensing

A few years back, mobile phones were limited to making calls, sending text messages and maybe including a few games. As time went by, phones started to become technologically advanced objects, until transforming into what we see today. Smartphone is the new 'technical' term. Smartphones have a lot of capabilities, being really distant from the days where mobile phones only made calls and sent messages. They now include useful sensors of many kinds, which is the main topic of this section.

2.3 Mining sensed trajectories

Data mining is subject of great importance on the Computer Science area, since it allows us to extract relevant pieces of information from large sets of raw data. ACM SIGKDD [15] defines it as the computational process for discovering patterns in sets of data, involving methods that cross machine learning techniques, statistics, and database systems.

Monitored sensed data, and trajectories in particular, are a great source for mining BSS-related information. Through data mining techniques, those indicators can be used to infer relevant events that happened throughout the trajectory, such as significant stops and final destinations.

On this section we will approach techniques for mining trajectories individually in order to extract relevant stops, which as mentioned before is a very relevant topic for our work.

2.4 Data Privacy and Anonymity

Many companies and institutions that collect personal information of its clients often need to provide their data to other entities [16]. For example a hospital that provides their client's medical records to researchers that will study certain diseases. This data contains sensitive pieces information that directly or indirectly can be used to identify the client (e.g. names or addresses). Furthermore, even if these sensitive attributes are hidden, it is possible to recover that information using combinations with external databases. This section contains methods and solutions to contain this problem.

3 Proposed Solution

Our solution consists in a reusable library of methods, exposed through an API, ready to be easily invoked by any BSS mobile application. The goal is to have a service fully capable of working alongside any BSS app implemented in any technology. Although this library can have some interaction with the app on which it is integrated, the main requirement is for it to be able to fully run in background, without depending on user interaction to achieve our goals.

3.1 System Features

This work includes a library, intended to serve BSS mobile applications, that allows the users to take a participatory role on the improvement of the BSS by simply using it. The main purpose is to detect **silent unattended demand** situations, which occur when a user was for some reason forced to drop off his bicycle on a station that theoretically isn't the most convenient according to his final destination. For this matter, the system includes two main features on which we will dip in next.

- *Monitor and store trajectories.*
- *Data Processing.*
- *Visual representations of data.*

3.2 System Architecture Overview

To achieve our objectives we propose an architecture that includes a client library containing a trip monitoring API that will be used by BSS mobile application, as well as a back-end server to process the data and generate outputs.

3.3 Mobile Client Library

Working as the main source of data, the mobile component is a very important module of the architecture and will run on the client side using an API as access endpoint. The BSS mobile app will use the API's methods to make requests and trigger the real-time monitoring and processing of trajectories made by the users off the BSS. At the end of each trip, it's processed information is sent to an external database.

Location Monitoring and Silent Unattended Demand. The API is invoked for the first time when the user requests the BSS mobile app to unlock a bicycle from a station (usually by clicking a button). From that moment on, the trajectory starts to be monitored, and our goal is set: detect a possible **silent unattended demand** situation, which happens when the user silently chooses to drop-off the bicycle in another station because the closest one was for some reason not available. Furthermore, and for this purpose, there are two main pieces of information to be extracted from the user's trajectory in real-time, knowing that extracting the first is the key to extract the other. The first significant event that is detected is when the user drops off the bicycle at a certain station. This event is important because it is a trigger for our system to know that the user is no longer riding the bicycle and will now start moving towards his final destination. It is necessary to automatically infer the drop-off moment because currently most BSS apps do not have direct user input of this event, since this information is given automatically to the operator's server when the bicycle is returned to the station. After detecting the drop-off moment of the bicycle, we will start trying to infer the user's final destination, since it enables us to determine if the bicycle was returned to the closest station considering the destination. If it was not, we may have achieved our goal of identifying a silent unattended demand occurrence. To determine the closest station to the user's final destination we opted to use an euclidean distance calculation, since walking segments are unpredictable, diverse, and will not always match the street road network.

Detecting silent unattended demand. To face this challenge we first need to detect the user's drop-off by analyzing the standard motion behavior of someone who is dropping-off a bicycle. Typically there would be a deceleration of the user's motion as he approaches the station, followed by moments where the speed is close to zero as he places the bicycle on a dock. The next segments of the trajectory will be significantly slower in comparison to the average of the trajectory until then, since the user is now walking.

After detecting a drop-off, we are ready to start detecting the user's final destination, and hopefully **silent unattended demand** situations. The methodology is a bit more complex since the standard behaviors are not as obvious as for the drop-off moment. It is required to dip into further detail. There are a few indicators on which it is possible to rely in order to detect the user's arrival at his destination, namely motion and time.

When the user arrives at a destination his/her behavior may be to stand completely still (office, restaurant etc.) or to slowly move in random directions inside the same spatial area (monument, shops etc.). The solution we used is simpler, but still very effective. It consists in continuously monitoring if the user stays in the same geographical area (geofence) for a predetermined amount of time, which in our tests is 8 minutes. The geofence is updated every time the user leaves his geofence limits. Furthermore, when the final destination is detected we notify the user by asking for a confirmation, or for a correction in case it was not correct. In case no response is received within 10 seconds, it is assumed that the destination was detected correctly.

Energetic Efficiency. The techniques we propose to implement are demanding for the battery life of the user's mobile device since they mostly rely on sensors, such as location, motion and user activity sensors. Specially considering that the smartphone will be using the app for a fair amount of time.

Although we intend to develop energy efficient solutions, it is important to keep in mind that accuracy is also a very important factor for our methodology, and the most relevant sensors of this solution tend to get more expensive with the increase of accuracy. For instance, applications with constant location monitoring tend to be somewhat battery consuming, since they generally need to know the user's location with high accuracy and a time a time interval of seconds between location updates. Additionally, it is also relevant to mention that the BSS in Lisbon features smartphone chargers incorporated on the bicycles. However, we found it important to develop a solution that is also usable by BSSs that do not include that feature.

3.4 Database and Server Side Component

Like most of the systems that deal with large amounts of data, our proposal relies on a database and a back-end server. The main role of the server is to process and cluster data stored in the database, which is collected by the client library component. The final output is a set of Excel files which will later feed visual representations and heatmaps, which will allow the BSS operator to easily identify problems and limitations of the network.

Data Processing. In order to transform the raw TLO's into aggregated and understandable pieces of information, some processing has to be done. This processing outputs files with clustered data that will feed a final dashboard with visualizations and charts.

In our architecture, this important step is made through a Java Application that can run on a server. The final goal is to have organized and detailed information about each station concerning its performance, mainly in unattended demand occurrences.

Final Dashboard. The creation of a User Interface Dashboard showcasing the collected data is the last crucial step of this solution, since it provides the BSS operator with a faster and more intuitive way of analyzing the current state of the network and hopefully make make more sustained decisions.

Privacy and Anonymity. Through this API we are collecting very specific and individualized data, namely user trajectories that contain time and place information about the user's location. Additionally, the data we collect will need to be in some form provided to the operators of the BSS so they can take make their use of it.

Since our solution proposes to work as a library of tools to support an already existing mobile application, we are assuming that it is most likely that we do not have access to personal information about the users (e.g. name, address etc.), and so there is some sensitive information that we may not need to worry about hiding. However, this is not always true. For instance, if there's a certain regularly made trajectory that geographically stands out from the rest, it may be possible to infer an individual's identity by looking at the starting and end location points (house address, office etc.). For that reason, we intend to implement **Generalization** and **K-Anonymity** techniques that will help us normalize large sets of data and possibly eliminate very specific trajectories.

4 Implementation

4.1 Mobile Client Library

Implementation Platforms One of the first decisions to be made was the operating system on which to target this client library. Since most people have Android based devices, and there was few time for development, we decided to develop our proposal in Android, using *Java* as the main programming language. In order to function properly, the BSS app must be running at least in *Android 4.1 Jelly Bean*, and at most in *Android 7.1 Nougat*. .

LocationMonitor Service The LocationMonitor Service is mainly responsible for tasks related to the geographical location of the user. As we have explained before, monitoring the user's location in real time is essential for practically everything we do (trajectory, drop-offs, final destination etc.). For this, we resorted to the Google Fused Location API[1]. As the name implies, this API uses more than one source for obtaining the device's geolocation: GPS, cellular towers and

[1] https://developers.google.com/location-context/fused-location-provider/

WiFi are the most reliable. This way, we can almost always guarantee a precision around 10 meters or less.

The location request interval can be set from less than a second to a few hours. In our scope, that value varies accordingly to the state of the trip or the motion of the user in order to preserve the device's battery life.

Although this service is launched by the BSS mobile app (indirectly, through the API), it is an unidirectional communication. That is, the LocationMonitor Service only communicates with the TripManager Service to keep it updated about the current location of the user, and never broadcasts back to the BSS mobile app.

TripManager Service The TripManager Service is the core of all monitoring, and probably the most important component of this implementation. The purpose of this service is to manage and act upon everything that happens from the moment the user picks up a bicycle until the his/her final destination.

This service has many tasks, such as:

- **Listening, in real time, for location updates coming from the LocationMonitor Service.** These updates are used to keep record of the user's trajectory.
- **Broadcasting location updates to the BSS client mobile app.** Every time the TripManager Service receives a location update, it checks if the main activity of the BSS app is alive. If it is, channels that update to the app if the respective receiver is placed in the client-side.
- **Broadcasting key events to the BSS client mobile app.** The TripManager is also able to inform the client mobile app about important events that have occurred in the trip, such as bicycle drop-offs and final destination detection. This may be relevant if BSS operator intends to inform the user about the detection of these events, or even prompt them with confirmation requests.
- **Managing the trip's state machine.** The service always keeps track of the current state of the user's trip (off bike, bike picked-up, on bike, bike dropped-off, walking to destination, on destination).
- **Detecting the user's current activity.** It uses the Google Activity Recognition API[2] to detect if the user is walking, cycling etc. in a pre-defined interval of time between activity requests.
- **Monitoring Geofences.** Uses the Google Geofencing API[3] to detect final destinations (same geographic area for a certain set amount of time) and if the user is near a station (useful for bicycle pick-ups).

API Documentation This subsection aims to list all the available methods provided by the API to the BSS mobile application.

- ***connect()***
- ***disconnect()***
- ***sendPickUpIntent()***
- ***updateActivityState(String state)***

Battery Management Concerning battery usage management, we concluded that there are three main variables on which our implementation can have a preponderant role in guaranteeing an energetically efficient solution:

- *Time interval between location updates.* The frequency on which a device requests a location update has a direct correlation with battery consumption, since more calls to the server mean more data exchange and CPU work. Since location has a critical role in detecting relevant situations of the trips in real-time (drop-off station and final destination), our high accuracy location updates cannot be too sparse, and for this reason we will be testing updates between 1 and 5 seconds of interval. The purpose is to find that sweet spot where battery consumption is acceptable and our library's effectiveness is not negatively affected.
- *Time interval between activity updates.* Just like the location updates, user activity updates have a significant battery impact and also need to be frequent in order to accurately detect bicycle drop-offs (by detecting walking and stationary moments in real time). However, this sensor has a variant: more sparse activity updates can present more reliable information than closer ones, since the library has more time to cement its complex calculations. For this reason we will be testing updates between 1 and 5 seconds of interval.

[2] https://developers.google.com/location-context/activity-recognition/

[3] https://developers.google.com/location-context/geofencing/

– *Active sensors management.* Logically, not all sensors are needed simultaneously at all times. For instance, the activity recognition API is only needed after a bicycle pick-up, since it will be determinant in detecting the drop-off. Receiving activity updates from this API before a pick-up or after a drop-off is plain energy waste, since those updates will be discarded. A similar scenario happens for the location update API. Although we need it to be working throughout the whole trip, it is useless to receive location updates while the device's screen is off and no trip has started yet. For this purpose, our TripManager Service class will also be responsible for activating and deactivating the respective sensors accordingly with the state of the trip.

4.2 Server Side and Visual Dashboard

Java Server Application. We have mentioned that the clustering and aggregation of TLO meta-data, created by our library, has a big role in creating organized statistical information about stations and unattended demand occurrences. Its outputs are mainly important to feed visual representations of data, which will be presented in the next subsection.

As the title of this subsection suggests, this component was developed using the *Java* programming language. It takes a set of TLOs stored in the database as **input**, and it returns a pair of Excel files as **output** after all the necessary processing is done.

This application is meant to be run in a server that is capable of effectively deal with the expected large sets of data, that a big station network of a big city can bring.

Final Dashboard. Our dashboard is a web application, which was implemented using the OutSystems platform, that guarantees useful, simple, easy to understand, and responsive visualizations of data. This component shows tables, statistics, filters and heatmaps in a way that gives immediate and user friendly feedback to the BSS operator.

5 Evaluation

Now that we have explained the implemented solution of our proposal, it's time to prove it's reliability against the goals that were set. There are two main indicators that need to show extremely positive results in order to consider that the objectives were met: **unattended demand detection (UDD) and energetic efficiency**. If one of these indicators present unsatisfactory results, than this whole system is not yet reliable enough to be used.

However, there are some pre-indicators, which compose the UDD, that need to be tested first. Before thinking of testing if unattended demand situations are being inferred correctly, we need to test and validate the detection of drop-offs and final destinations, since these are crucial steps for UDD. Therefore, we will be presenting the drop-off detection results in Section 5.1 and the final destination detection results in Section 5.2, before dipping into UDD and energetic efficiency in the following sections.

All tests in this section were made using a Samsung Galaxy A6 smartphone.

5.1 Drop-off Detection

Automatically detecting when a user drops off a bicycle in a bike station is probably the most complex indicator to achieve highly effective results. The reason for this is that the user movements while performing tasks can be incredibly unpredictable, even though we can try to estimate an approximation of what is a common behavior for that task. After intense empirical tuning, we concluded that to detect a drop-off we should listen for stationary moments in the trajectory, followed by walking segments (see Section 3.3 for more insight).

As we mentioned in Section 4.1, we used Google Activity Recognition API for detecting user activities (walking, stationary, etc.). This brought up a relevant challenge: API accuracy varies accordingly to the real-life context of the user. For instance, it may be easier or harder to detect walking segments consonant the device is in a pocket, in the user's hand, or in a backpack. The current state of the BSS application may also be relevant (open, paused or closed). This means that every possible scenario should be tested. We wish to achieve high success ratios while the device in on the user's and or pocket, since these are probably the most common cases.

Additionally, every time an activity is detected, it comes with an associated confidence value (from 0 to 100). After some research, including the API's documentation, we found that

activities with less than 70 of confidence should be discarded, since they are not reliable enough.

5.2 Final Destination Detection

The detection of a bicycle drop-off is followed by the detection of the user's final destination. As we've explained in Section 3.3, this is done using a Geofencing technique, on which it is detected whether the user stays inside a radius of 40 meters for a pre-defined amount of time. The time which we set was 8 minutes. As we can see, this test is pretty straightforward and the actual destination only needs to stay within the 40 meters for it so succeed.

However, since the location sensor may be affected by external factors like tunnels, trees or high buildings, the detection of the correct final destination may be compromised. For this reason, two test cases were defined: normal external conditions (open spaces) and adverse conditions (anything from being under trees, near high buildings and inside buildings).

5.3 Unattended Demand Detection (UDD)

After evaluating and guaranteeing that the detection of drop-offs and final destinations is effective, UDD validation becomes simpler. As we know, the final step of UDD is to verify if the station used for drop-off was the most convenient considering the user's final destination, using distance measures and thresholds as metric. Since this final step only consists in evaluating a simple *If* condition, and it does not have multiple test variations like the previous ones, we will present the UDD evaluation as a global test case. For this we made complete trips (including pick-up, drop-off, and final destination) with an average of 18 minutes of duration. The trips forcibly included unattended demand occurrences. These tests were performed near Intitute Superior Tecnico, and also around Belem, including diverse types of conditions (near and inside buildings, open spaces, parks etc.).

Besides evaluating the success ratio of UDD, we also found it very relevant to test whether it should be made on the user's mobile device at the end of each trip or in a bulk by the Java web application running in the server. The idea is to measure the impact that this processing can have on a mobile device, namely by looking the at size, in bytes, of the TLO sent to the database through the network, and the time, in milliseconds, it takes to detect a UD occurrence.

For this, we implemented the UDD in both mobile library and Java server application.

5.4 Energetic Efficiency

As we have mentioned on the beginning of the chapter, evaluating the energetic impact of our solution in the user's mobile device is a crucial step in validating the viability of the proposal. Furthermore, it is important to also guarantee effectiveness in the monitoring of each stage of the trip.

In order to test the energetic efficiency, we tested the battery consumption of a trip when using different values for two main variables: time interval between location updates and time interval between activity updates.

6 Conclusions

As we have seen, the BSSs are emerging in many cities across the globe. Some cities already have an extensive station network that needs to be monitored 24/7 in order to avoid undesirable events that may displease the users. Most of the BSSs use direct observations of station demands and balance to conclude if a certain station, and the overall network, is performing well. It's through these direct observations that important decisions are made, namely expansion, re-position and capacity. However, this thesis argues that these methods are incomplete and do not address very specific and silent faults. Furthermore, we believe that by crowdsensing user trajectories we will obtain much more specific and reliable information. For instance, our main focus is to detect when a user intended to use a determined bike-station but ended up using one that was less convenient. This usually happens because the intended station was for some reason unavailable, and the system will rarely get a feedback that this situation happened (silent unattended demand). Our goal was to detect, process and visually represent this currently undetected and unmonitored event, as well as a few other useful indicators.

Our solution includes an architecture composed by three main modules: a mobile application library for Android exposed by an API, to collect data; a Java application to be ran on

a server, to process and aggregate the data; and a web application as a dashboard for the BSS operator, to visualize the data. We believe, and our proven features and tests support it, that we achieved the crucial goals of this proposal.

We hope that this work can help current BSSs evolve quickly and provide a better experience for their customers. We believe that we have developed and implemented this idea well enough to achieve that purpose.

References

1. J. Owen, C. Neita, and J. Froehlich, "Crowdsourcing Bikeshare Transit Planning: An Empirical Investigation of Washington D.C. and New York City," no. April 2012, pp. 1–16, 2015.
2. D. Tomaras, I. Boutsis, and V. Kalogeraki, "Lessons Learnt from the analysis of a bike sharing system," *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '17*, pp. 261–264, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3056540.3076206
3. "Bicicletas partilhadas de Lisboa comecam a "Girar"," *Shifter.pt*, 2017. [Online]. Available: https://shifter.pt/2017/09/gira-bicicletas-em-lisboa/
4. L. E. Ogden, "Cycling in cities: the search for the world's most bike-friendly metropolis," 2014. [Online]. Available: https://www.theguardian.com/cities/bike-blog/2014/mar/04/cycling-cities-search-bike-friendly-metropolis
5. NACTO, "Walkable Station Spacking is Key to Successful, Equitable Bike Share," *Report*, no. April, pp. 1–5, 2015. [Online]. Available: https://www.bicyclenetwork.com.au/media/vanilla{_}content/files/NACTO{_}Walkable-Station-Spacing-Is-Key-For-Bike-Share.pdf
6. G. Lubin, "Here's Which Citi Bike Stations Are Most Often Empty," 2013. [Online]. Available: http://www.businessinsider.com/where-citi-bike-stations-are-often-empty-2013-8
7. J. Liu, L. Sun, Q. Li, J. Ming, Y. Liu, and H. Xiong, "Functional Zone Based Hierarchical Demand Prediction For Bike System Expansion," *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*, pp. 957–966, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3097983.3098180
8. E. Jaffe, "The Methodology of Bike-Share Station Placement in New York City," 2011. [Online]. Available: https://www.citylab.com/transportation/2011/10/how-new-york-city-will-choose-its-bike-share-stations/248/
9. J. Zhang, X. Pan, M. Li, and P. S. Yu, "Bicycle-Sharing Systems Expansion : Station Re-Deployment through Crowd Planning," 2016.
10. N. Jian, D. Freund, H. Wiberg, and S. Henderson, "Simulation optimisation for a large-scale bikesharing system," *Proceedings of the 2016 Winter Simulation Conference*, no. 2013, pp. 602–613, 2016.
11. Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh, "A framework of energy efficient mobile sensing for automatic user state recognition," *Proceedings of the 7th international conference on Mobile systems, applications, and services - Mobisys '09*, p. 179, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1555816.1555835
12. N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A Survey of Mobile Phone Sensing," *IEEE Computer*, vol. 48, no. 9, pp. 140–150, 2010. [Online]. Available: http://www.cs.dartmouth.edu/{~}niclane/pubs/ieee{_}coms10.pdf
13. F. B. Abdesslem, A. Phillips, and T. Henderson, "Less is more: energy-efficient mobile sensing with senseless," *MobiHeld*, pp. 61–62, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1592621
14. S. Fang and R. Zimmermann, "EnAcq: Energy-efficient GPS trajectory data acquisition based on improved map matching," *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pp. 221–230, 2011.
15. KDD, "Data Mining Curriculum," *ACM*, 2014.
16. X. Xiao and Y. Tao, "Personalized privacy preservation," *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, p. 229, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1142473.1142500