ParkForU: A Dynamic Parking-Matching and Price-Regulator Crowdsourcing Algorithm for Mobile Applications

Ellen Mitsopoulou  
Department of Informatics  
Athens University of Economics and Business  
Athens, Greece  
Email: mitsopoulou@aueb.gr

Vana Kalogeraki  
Department of Informatics  
Athens University of Economics and Business  
Athens, Greece  
Email: vana@aueb.gr

Abstract—Large metropolitan cities are getting busier and busier everyday. Overpopulation has caused parking related problems which in turn have severe external effects such as traffic congestion, air-pollution, social anxiety and inefficient resource distribution. To alleviate those effects infrastructure-based parking information systems have been proposed. However, they incur extreme costs due to extensive hardware installations. A promising alternative, that has shown great interest in recent years, is the use of crowdsourcing using mobile phones. In this work we propose a crowdsourcing system that aims to find the available and most suitable parking options for users in a smart city. We have developed ParkForU, a parking-matching and price-regulator algorithm. ParkForU, unlike existing approaches where a large unfiltered number of parking possibilities is given to the users, provides the best matched parking results while at the same time provides an effective way for dynamically re-adjusting the parking providers’ price. Through extensive simulations, we show how ParkForU performs and benefits both users and parking providers.

I. INTRODUCTION

With the high percentage of vehicle ownership, parking has become an immense problem in all major urban cities. There are several issues involved, including inadequate information for drivers on parking availability and pricing. Therefore, the drivers are often frustrated when they are promised plenty or free parking but find limited or expensive parking or spend an excessive amount of time searching for a parking spot in a reasonable waking distance, especially during times of day with high demand. Recent studies [1] have shown that drivers spend on average 8.1 minutes each time they circle in a U.S. city to find a parking spot. This extra driving around the city has serious impact on traffic as it results in excessive car usage, increased traffic congestion and results in frustrated drivers trying to find parking on a busy street.

To alleviate the problem, significant effort has focused on designing reliable and real-time parking information systems. A few infrastructure-based systems have been proposed, including the SFPark system [2] that uses wireless sensors to collect and disseminate up-to-date information about parking availability in San Francisco and the Parknet system [3] where ultra-sonic sensors have been installed on vehicles to monitor parking availability when vehicles are moving. However, such sensor-based systems require extremely high cost for the complete system and hinder large scale deployments.

As the infrastructure-based parking information systems are expensive to setup and manage, a great amount of research has focused on developing mobile applications with the help of the human crowd [4] [5] [6]. Crowdsourcing-based approaches offer higher agility, lower cost, and larger coverage as they utilize the availability of vast number of mobile phone users. Therefore, it seems reasonable to assume that in the near future the larger part of urban parking will be managed and paid through smartphones. For example, PocketParker [7] leverages the crowd’s smartphone low-power sensors (i.e., accelerometer and GPS) in order to detect users’ movement and derive the status of parking or unparking. The ParkUs system [8] exploits smartphones’ accelerometer and magnetometer sensors, in order to detect parking activity within a city environment. Moreover, applications such as SpotHero\(^1\) and ParkAround\(^2\),

\(^1\)SpotHero: www.spothero.com  
\(^2\)ParkAround: www.parkaround.gr
which allow users to view parking spaces location and additionaly pay for them, have seen wide adoption. However, such approaches typically display a large and unfiltered number of parking options on a map. Thus, users can easily get “lost” and confused, making it quite difficult to decide which parking best suits them. An example illustrating the problem is shown in Figure 1. The query is for the center of New York city through the SpotHero application. As can be seen, a visually overcrowded map is generated, full of pin markers representing the available parking lots.

In this paper we present ParkForU, a parking-matching and price-regulator algorithm to be deployed in our crowdsourcing system that aims to help drivers in urban areas find and reserve the best-matched parking options. Our approach allows users to specify their destination along with a set of preferences regarding price and distance (from the parking lot to their destination), as well as a maximum number of results they wish to receive. The system filters the results based on the driver’s input and makes the appropriate suggestions. Additionally, the system notifies parking providers regarding drivers’ preferences and selections. Thereby, parking providers are able to dynamically regulate their prices based on supply and demand. Our approach aims at saving time, money, fuel spent during extra circling, minimize walking time to the destination and reduce the traffic congestion. We perform extensive simulations to evaluate our approach and show that ParkForU effectively meets the requested user demands.

II. System Model

In this section we present the entities and multi-attribute utility function of our system model.

Drivers. These are the smartphone users that use their devices to access the service through the corresponding application on the smartphone. To search for a parking lot, they fill in the following information: destination, desired parking duration and preferences regarding price and distance. The preferences are entered by selecting 1 among 5 class intervals. Each price class denotes a minimum/maximum acceptable price, whereas each distance class denotes the minimum/maximum distance the drivers are willing to walk from the parking lot to their destination. Lastly, they set a maximum number of results they wish to receive. The application forwards the query to the application server which in turn sends back information about suitable and available parking lots. Then, the application creates a map-visible list where the parking results are ranked in order of priority. The chosen parking can be reserved and booked online.

Parking Providers. The parking providers are in sync with the application servers and provide information regarding their coordinates, supplied services along with the corresponding price, current number of available parking spaces and their characteristics (e.g., garages, open-area parking, on- and off-street parking etc). In order to be able to dynamically adjust their prices, the parking providers receive updated information about the drivers’ preferences upon their parking lot selection (i.e., whether these parking providers were preferred or not among others in the list).

Grid. In our system we represent a city’s urban area with a grid. Let \( G(X, Y) \) denote a two-dimensional Euclidean plane. We will use the notation \( G(i, j) \), where \( i \in \{1, \ldots, X \} \) and \( j \in \{1, \ldots, Y \} \), to indicate a specific location on the grid.

We consider that there are \( D \) drivers searching for a paid parking lot. Each driver \( d \in D \) is associated with the following set of attributes:

- \( destination_d \): a specific grid cell \( G(id, jd) \) which represents the driver’s destination on the grid.
- \( parking_duration_d \): a desired parking duration of \( t \in \{1, \ldots, T \} \) time intervals (e.g., hours).
- \( \langle weight_price_d, weight_distance_d \rangle \): these weights are set based on each driver’s denoted preferences on price and distance respectively. The weights reflect how sensitive or indifferent a driver \( d \) is on these values. Their sum must be equal to the value of one.
- \( satisfaction_threshold_d \): a numerical value to be used as an upper bound and determines whether the driver selects to park or not.
- \( max_results_d \): the maximum number of parking options to be returned as results.

We consider that there are \( P \) parking providers. Each parking provider \( p \in P \) is associated with the following set of attributes:

- \( location_p \): a specific grid cell \( G(ip, jp) \) which represents the parking provider’s location on the grid.
- \( capacity_p \): the maximum number of vehicles that can fit into the parking.
- \( price_p \): the current price per time interval that will be charged when a service to a driver \( d \) is offered.
- \( occupied_spots_p \): the current number of occupied spots.
- \( selections_p \): a counter that increases every time a driver selects to park at the parking provider \( p \).
- \( declines_p \): a counter that increases every time a parking provider appears on a driver’s results but was not selected.
- \( selections_threshold_p \): an upper bound on the \( selections_p \) that determines whether the \( price_p \) needs to be raised.
- \( declines_threshold_p \): an upper bound on the number of \( declines_p \). This essentially indicates to the parking provider whether the \( price_p \) needs to decrease.

Multi-Attributive Utility Function. Utility is a well known concept that is used widely in economics. Utility reflects the satisfaction a consumer gets from using, owning or doing something and can be used as a measure of preference over some set of goods or services. Although there are many types of utility functions that we can plug in to our system, in this work we use a linear function, which is very popular and has been used in several works [9] [10] [11] to represent user preferences. Particularly, we use a Multi-Attributive Utility function \( S(x) \) to reflect the overall value of a parking provider \( p \) for a driver \( d \) based on its attributes:
\[ S(x) = \sum_{i=1}^{n} w_i S(x_i) \quad \text{subject to} \quad \sum_{i=1}^{n} w_i = 1 \quad (1) \]

where \( w_i \) is the weight factor, \( n \) is the number of attributes and \( S(x_i) \) denotes an individual attribute. In our model we assume that there are two fundamental attributes a driver cares about: total price and distance. We define as \( \text{total price}_{d,p} \) the amount a driver \( d \) will pay for the time intervals that her vehicle will be parked at the parking provider \( p \). This is calculated as follows:

\[ \text{total price}_{d,p} = \text{parking duration}_{d,p} \times \text{price}_p \quad (2) \]

The \( \text{distance}_{d,p} \) between a driver’s \( \text{destination}_d \) and a parking provider’s \( \text{location}_p \) is calculated as the Euclidean distance between the corresponding coordinates on the grid. Therefore, for \( G = (i_d, j_d) \) and \( G = (i_p, j_p) \) the distance is given by:

\[ \text{distance}_{d,p} = \sqrt{(i_d - i_p)^2 + (j_d - j_p)^2} \quad (3) \]

As we discussed earlier, each driver \( d \) has weights on price and distance that reflect her preferences on these attributes. For example, if a driver is more concerned about the distance than the price, then the weight for distance will be higher than the weight of the price and vice versa. The sum of the corresponding price and distance weights is equal to one. Hence, in our case, for a driver \( d \) and a parking provider \( p \) the multi-attributive utility function is defined as:

\[ S(x)_{d,p} = \text{weight price}_{d} \times \text{total price}_{d,p} + \text{weight distance}_{d} \times \text{distance}_{d,p} \quad (4) \]

Now that we have defined the attributes of the utility function, we need to clarify that the effect they have on the parking providers’ value is negative. It is assumed that lower prices and distances are preferred to higher ones. Hence, the lower the utility score the better it is.

### III. The ParkForU Algorithm

ParkForU is a dynamic parking-matching price-regulator algorithm designed to identify the most suitable parking options for a driver \( d \) based on her personal preferences with respect to price and distance. In order to find the most appropriate parking options, we employ a reverse auction scheme (i.e., winner is the lowest bid). In particular, in our approach of reversed auction, a driver \( d \) is the auctioneer and each parking provider \( p \), coupled with available spots, is a bidder. The bid, of each parking provider \( p \), is represented by the Multi-Attributive Utility Function score that is associated with driver \( d \). As we discussed in section II, the attributes we use in the Multi-Attributive Utility Function have a negative effect. Therefore, the winner bid (or the winner parking provider) is the one with the lowest utility score.

ParkForU executes each time a driver \( d \) searches for a parking lot through the system’s application. ParkForU returns a list with the results, where the list of results can be pruned based on the maximum number of results denoted by the

\[ d \text{, a list with the results, where the list of results can be pruned based on the maximum number of results denoted by the} \]

driver. The list includes the parking providers with the lower utility scores, which are suggested in descending order (i.e., the first parking appearing in the list is suggested as the best option, while the remaining ones are suggested in descending order). The driver is able to select among all options appearing in the list. Once the driver makes a selection, the algorithm makes the appropriate changes in the parking provider’s state. In addition, the algorithm notifies all parking providers appearing in the driver’s list whether they were selected or not. This step is important because the parking providers can use this knowledge to dynamically adjust their prices in order to attract more customers. Finally, to simulate the case where there is no parking satisfying a driver’s needs, we compare the \( \text{satisfaction threshold}_d \) with the lower utility scores, and if all scores are above that threshold then \( d \) is not satisfied and the driver does not park. The pseudocode of ParkForU is shown in Algorithm 1.

**Algorithm 1: ParkForU**

### IV. Simulations

**A. Simulation Setup**

We perform an extensive set of simulations to evaluate the performance of ParkForU. We assume a grid \( G(1000,1000) \)
which represents the downtown area in an urban city, 1000 drivers and 50 parking providers. The drivers’ destination and parking providers’ location on the grid are randomly generated based on a Uniform distribution. Additionally, we use the same distribution to generate randomly drivers’ parking duration with values from 1 to 8.

Regarding the drivers’ \( (weight_{price,d}, weight_{distance,d}) \), we evaluate three weight-pairs categories. A price-focused \((0.9, 0.1)\) where drivers are more interested into economic parking suggestions, a balanced \((0.5, 0.5)\) where they care equivalently for price and distance, and a distance-focused \((0.1, 0.9)\) where drivers prefer the closer to their destination parking solutions. Again, the Uniform distribution is used for generating each driver’s weight-pair. The max_results of to be returned at each driver \( d \) is set to 5. We assume that the driver selects the best option suggested by the system, otherwise he does not park and drops out of the system.

As we discussed in section II each parking provider \( p \) charges its clients the same price \( p \) per time interval. In order to draw conclusions, we consider that each parking provider belongs to one of the three pricing-policy categories. There is a low-priced category, a medium-priced and a high-priced. For the sake of simplicity, we set the price at 1 monetary unit for the low-priced category, at 2 units for the medium-priced and at 3 units for the high-priced. Nevertheless, while parking providers with ParkMatch keep the same price throughout the process, with ParkForU they regulate their prices. Therefore, parking providers with ParkForU start with an initial price, and we assume that they fluctuate their prices every 10 continuous selections of \( p \) and decrease when \( p \) has 20 continuous declines (i.e., selections_threshold = 10 and declines_threshold = 20).

Each parking provider’s capacity is set to 100, therefore the initial value of occupied spots, for each \( p \), is generated randomly with values up to 100. The utility functions’ attributes of total_price and distance are normalized. Hence, the maximum utility score that can occur is 1. Finally, for all drivers, the satisfaction threshold is set to 0.7.

In our simulations, we compare ParkForU with the ParkMatch algorithm [11] developed in our previous work. ParkForU uses a static approach for setting the parking prices and is not concerned about informing and regulating the price of parking providers. In order to identify pros and cons of each approach, we compare both algorithms under the same data input. We run simulations for each of the driver categories to evaluate the working of our approach. We run 50 simulations with different random seeds and the results are averaged.

B. Simulation Results

1) Case 1: In ParkForU implementation each parking provider increases \( p \) by 10% and decreases by 5% when required by the system. Figure 2 presents the percentage that the profit of parking providers increased with ParkForU compared to ParkMatch, for each of the parking categories. As can be seen, the profit was increased for all parking categories and for all driver categories simulations. Remarkable is the benefit achieved by parking providers that follow the high-priced model, which though ParkForU lowered their price, it managed to increase profit by 96% and 100%.

Figures 3 and 4 illustrate how drivers were distributed, based on the parking providers they selected to park and their pricing models. In Figure 3 where the weight on price is 0.9 (price-focused), most drivers preferred low-priced parking providers. Additionally, in Figure 4 with weight 0.9 on distance (distance-focused), where drivers care mostly for parking next to their destination and are price-irrelevant, the distribution of drivers among pricing model categories is almost equal.

Figure 5 presents how ParkForU affected parking providers’ prices compared to ParkMatch.
percentages how the price increased or decreased. We can observe that parking providers following the low-priced model have raised their prices in all driver categories. This happens since even with small increases on price, low-priced providers are still selected. In contrast, parking providers following the high-priced model decreased their prices in order to be competitive (except from the simulation with distance-focused drivers, which is expected).

2) Case 2: In this simulation case we consider that all parking providers belong to the same price category, which is the medium-priced. Additionally, with ParkForU implementation, we examine 3 sets of percentage variations for increasing/decreasing price respectively. These are: 10%-10%, 10%-5% and 10%-20%.

Fig. 6: Percentage of drivers that found a more economic parking through ParkForU.

Fig. 7: Percentage of drivers that dropped out of the system.

Figure 6 shows the percentage of drivers who found a better price through ParkForU compared to ParkMatch. We observe that the ParkForU performance was better in all percentage variations. Remarkable is the 39% that was achieved in the 10%-20% category. Figure 7 illustrates the percentage of drivers who finally dropped out of the system and did not park. We can observe that for all set of percentage variations ParkForU achieved lower percentage than ParkMatch. This indicates that through ParkForU drivers found lower prices and parked, while with ParkMatch they did not.

Figure 8 shows the percentage change in profit achieved with ParkForU compared to ParkMatch. As can be seen, ParkForU achieved higher revenue comparing to ParkMatch for all percentage variations. Figure 9 presents ParkForU effect on parking providers’ prices compared to ParkMatch. The comparison shows in percentage change how the price increased or decreased. We can see that for the sets of 10%-10% and 10%-5% the price increased while for the set of 10%-20% decreased. This indicates that depending on the percentage variation a parking provider decides to follow, results may differ.

V. RELATED WORK

In recent years a great number of research schemes have focused on parking related issues. From the infrastructure-based perspective, the pilot SFpark program [2] uses wireless sensors and wireless beacons to collect and disseminate information about parking availability in order to raise or lower meter and garage prices. Additionally, Parknet [3] installs ultra-sonic sensors on vehicles, and detects parking availability when vehicles are moving. However, the main issue with both approaches is the extremely high cost of the complete system that they require, and which hinders a large scale deployment.

In [12] an interesting survey regarding driver needs from a smart parking infrastructure is conducted. The results in [12] confirm that in contrast to infrastructure-based parking solutions, mobile crowdsensing via smartphones is the most
economical way to obtain parking availability information. Furthermore, they confirm that dynamic pricing is currently the most efficient way to regulate parking occupancy status and traffic congestion. As a consequence, there have been developed a great number of mobile crowdsourcing applications aim at simplifying peoples’ every day life and economic potential. Park Here! [13] utilised an accelerometer as a well as a gyroscope sensor in order to detect parking activity. Park Here!’s classification was binary; driving or not driving. By recording changes between the two states, it was able to detect parking activities. ParkUs [8] leverages smartphones’ accelerometer and magnetometer sensors to perform parking activity detection in the background without user involvement and with low energy consumption. ParkSense [14] is a smartphone based sensing system that detects if a driver has vacated a parking spot. The application uses a more robust WiFi beacon reception ratio as opposed to signal strength, to detect drivers’ unparking events. Regarding crowdsensing simulation tools, in [15] [16] the authors simulated crowdsensing activities for urban parking using the MASON multi-agent simulation toolkit and displayed the results on OpenStreetMap. All above mentioned crowdsourcing applications focus on finding vacant street parking spaces that are free of charge. However, there is no technique that we know of, that aims to give personalized suggestions for paid parking lots as well as deal with dynamic pricing. Finally, regarding the crowdsourcing challenges, there has been a growing body of work that addresses various challenges including auction-based and micropayment models [17] [10] [18] [19].

VI. CONCLUSIONS

In this work we have studied the parking problem in urban cities. We propose the ParkForU algorithm, to be deployed in a crowdsourcing system, which aims at finding the best-matched paid parking for drivers, based on their requirements. ParkForU suggests the most appropriate parking options based on the driver needs. At the same time, ParkForU updates the state of the parking providers and informs them about the drivers’ choices. This way, they are able to properly adjust their prices. We conduct detailed simulations where we show how our algorithm performs based on different driver demands and parking providers’ pricing-policies.

VII. ACKNOWLEDGMENTS

This research has been financed by the European Union through the FP7 ERC IDEAS 308019 NGHCS project and the Horizon2020 688380 VaVeL project.

VIII. ACKNOWLEDGMENTS

REFERENCES