Reaching Available Public Parking Spaces in Urban Environments using Ad-hoc Networking

Vasilis Verroios [1], **Vasilis Efstathiou** [2] **and Alex Delis** [3]
University of Athens, GR15784, Athens, Greece
{verroios[1], v.efstathiou[2], ad[3]}@di.uoa.gr

***Abstract*—A fundamental application in vehicular ad-hoc networks (VANETs) is the discovery of available parking spaces as vehicles navigate through urban road networks. Vehicles are now capable of finding such parking spots using their on-board sensing and computational infrastructure and then they can disseminate this information for use by other members of the travelling community in the geographic vicinity. In this context, we examine the problem of locating an available parking space for a vehicle entering an urban network of roads. Upon its entry, a vehicle has to determine the best way to visit parking spots reported to be free. In deciding this, the vehicle has to consider the time required to reach each candidate position, its distance from the final destination should the driver walk, and of course, the probability that the spot(s) will be still-free once the vehicle shows up at location. We formulate the question at hand as a Time-Varying Travelling Salesman problem and we propose an approach for computing the route that a vehicle must traverse in order to visit all parking spaces known to be available. Our method takes into account the limited computational resources of vehicles and attempts to find the best feasible trip. This is done in conjunction with a cost function that estimates the probability to find a space filled. In order to ascertain the effectiveness of our proposal, we compare it with a best-first approach and examine computational overheads. We also investigate how close to optimal results our approach comes.**

***Keywords*-parking space problem; VANETs; Time-Varying Travelling Salesman Problem; TSP; incremental computing; ad-hoc networking;**

I. Introduction

Recent work in vehicular ad hoc networks (VANETs) has yielded equipment that can help vehicles identify available parking spaces while navigating in city environments [1]. Clearly, such information is very useful as if disseminated properly it could help the movement of cars, help individuals get into their destination faster and at the same time save monetary resources expended by both drivers and local governments. Modern vehicles are equipped with sensory equipment that is able to detect spaces between parked vehicles [1]. This equipment recognizes a spot as an available parking space through the combined use of its sensory GPS-enabled devices and a local geographic database. Such databases contain detailed maps for urban areas along with city-designated spaces where public parking is permitted; they also feature parking lots that inform the public about their occupancy and their available parking spaces through adjacent roadside-units [2].

A number of protocols has been proposed [3], [4], [5], [6] to help disseminate information about the status of the road network and in particular information about parking. The above information could be used in conjunction with the probability to find the available space still free when a vehicle finally reaches the spot [7]. In this paper, we exploit the information about identified available parking spaces that dynamically becomes available so that a driver can be efficiently directed to the available spots that are closest to her final destination. The *criteria* for this decision are: *a)* the time needed to arrive to each space available, *b)* the probability to find the spot in question available at the time of arrival, *c)*the walking distance to the final destination (and possibly the parking fee for the space). We formulate the above question as a travelling salesman problem in which the points-to-be-visited are available public parking spots, located close to the final destination of the driver.

We adopt a solution that attempts to minimize the aforementioned criteria and yield a minimal cost. This cost is time-dependent as the likelihood to find a space available depends upon the time of arrival. For this reason, the classic version of *Travelling Salesman Problem (TSP)* cannot sufficiently address our requirement and so, we resort to the generalized *Time-Varying TSP* [3]. The latter uses as input, both a transit time $b(x,y,t)$ as well as a transit cost $c(x,y,t)$ for each edge (x,y) at time t. Here, the problem is to find a dynamic path that starts at a pre-specified vertex s and visits each vertex only once in a way that the total transit cost of traversed path is minimal. As the *Time-Varying TSP* is inherently *NP*-hard, an exact solution would undoubtly require substantial computational resources. We follow a dual approach: firstly, we deploy clustering algorithms to reduce the problem size by mostly working with representative points of our input in order to offer a near-optimal solution. Secondly, we dynamically use disseminated-by-others information as a vehicle proceeds towards its destination in order to correct and re-adjust the path currently pursued.

Our approach takes into consideration the limited resources vehicular devices demonstrate when it comes to designating a vehicle trajectory in order to select an available spot. It is worth mentioning that the continuous update of information related to available parking as necessitated by the roaming of the vehicles, may take a toll on the computational load on the car devices. Our aim is to reduce the computational load vehicle devices experience while at the same time to find a near-optimal trip. The rest of this paper is organized as follows: Section II states the problem we address and Section III outlines related work. Section IV presents our proposed method for compiling a trip around known parking spaces. The way we evaluate our approach and the outcomes of experiments are discussed in Section V. Section VI provides concluding remarks.

II. Problem and Assumptions

Vehicles equipped with an automatic parking system feature sensors that can detect the length and the width of a space between stable objects [1]. Once a vehicle detects a space, it must locate its position on its map possibly with the help of GPS. The map should also have information about the municipality-sanctioned areas where parking is allowed so that the vehicle can independently determine whether it has encountered an available spot. In an orthogonal manner, parking lots may inform potential customers about their utilization and available capacities with the help of roadside units. Through PDAs or smart-phones, vehicles may collect information transmitted by such units [2]. Vehicles store the information for detected spaces and/or availability in parking lots and through their communication resources, they disseminate this information to others near by. The dissemination could be performed through an appropriate protocol for parking information [3], [4], [5], [6]. Apart from the above parking space availability, vehicles maintain statistics about average time periods needed to traverse road segments, the average number of spaces one may need to visit before an available spot is reached, and the average period of time that a parking space remains free. The space and time granularity of such statistics depend on the storage capabilities of a vehicle's device.

The dispensed-by-others parking information has to be exploited by a vehicle so that it will locate a spot within a short period of time and close to its final destination. The decision for the trip, the vehicle will follow, around the known parking spaces, may be critical for the time needed to park and the distance of the parking position to destination. Consider Fig. 1 where a vehicle in position s has its final destination position d and is aware that spaces 1, 2, 3 and 4 close to her final destination are currently available. If the vehicle goes directly to space 3, geographically the closest to destination d, it may arrive there too late and find it taken. An alternative route would be to make a detour, to check spaces 1 and 2, which are the closest to the current position, before visiting space 3, if 1 and 2 are taken. Another detour could be made to visit space 4, after 1 and 2, and before 3. But parking at 4 means that the driver will have to walk some distance to her destination. Also due to detours taken, the vehicle may finally arrive to 3 with some delay, should all previous choices prove to be infeasible.

We can compute the expected time to destination for a specific trip if we have at our disposal an estimation for the probability to find each of the visited spaces free during the trip in question. This expected time is a function of the time required to reach every single available position and the time needed to walk from there to destination. Let us consider

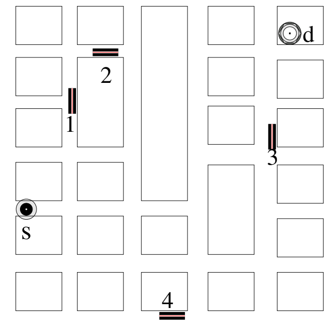

Fig. 1. Example for parking space problem

the trip with the minimum expected time to destination be the best choice. For n parking places, the number of distinct trips traversing them all, is factorial. Clearly, this would pose significant strain on the computing resources of the vehicle device rendering an exhaustive search infeasible. Even if the value of n is limited, the fact that the disseminated parking information stored by a vehicle is frequently updated, it will cause intense use of pertinent resources. The method discussed in Section IV ultimately proposes a trip that minimizes a cost function related to the criteria stated in the introduction. At the same time, our approach takes into consideration the fact that information about parking spots is continously received by moving vehicles whose computational resources by and large remain limited.

III. Related Work

We anticipate that moving vehicles use a dissemination protocol in order to exchange information about parking availability. Actually, a number of such proposals have already been proposed in [3], [4], [5], [6]. In [3], a spatio-temporal relevance function is proposed to compute the relevance of information, so that the dissemination of information can be carried out solely based on its relevance. In [4], mobile nodes build content summaries of their local information and disseminate them to peers; through a routing mechanism queries are forwarded to nodes that have a high probability of providing results. The protocol discussed in [5] disseminates aggregated parking information about areas that cover a lot of parking terminals, while taking the local relevance and age of information into account. In [6] the proposed protocol does not disseminate the same information to all vehicles, in order to avoid competition among them for the same parking spaces, and tries to coordinate them, so that each vehicle "reserves" a different space. Vehicle self-coordination is also discussed in [8] where a protocol for traffic flow management in intersections is suggested. In [9], we discuss a decentralized and collaborative approach to bypass congested areas in large road-networks based on information disseminated among moving vehicles.

In this paper, we express the quality of a trip that visits reported parking places in a sequence using a cost function. This function depends on the probability to actually locate a spot available for parking. In a similar context, [7] predicts the occupancy of a parking lot, by modeling it as a queue and using a Markov chain. Another work related to problem of finding a free parking space, is presented in [2], where *vehicle-to-vehicle (V2V)* and *vehicle-to-infrastructure (V2I)* communications are used to help vehicles navigate inside large parking lots. Two basic approaches for the parking space search problem are investigated in [10]: the first is a decentralized approach that uses *V2V* communication in order to disseminate parking space information among vehicles and the second is a centralized approach where a server acts as a reservation system for vehicles' parking requests. The system presented in [11], consists of a central server that collects reports for parking spaces detected by vehicles' sensors, in order to produce a real-time map of parking space information. Parking space allocation and routing to a reserved space is provided by the system proposed in [12], using aggregate parking and traffic information from vehicles and roadside units. In this work, we build on existing dissemination protocols [3], [4], [5], [6] and through collected statistics (i.e., average rates "seen by others thus far" on the road network) we propose methods for helping a vehicle rapidly approach available space near the stated destination.

IV. The Salesman Method

With the help of the dissemination protocol, a vehicle seeking a spot uses data accumulated thus far and stored on board to determine a trajectory to be visited. The data entails all identified parking positions located close to the destination. The degree of proximity can be driver designated and indicates how far she is willing to walk. Candidate positions have to be large enough to accommodate the vehicle in question. In this section we outline three different approaches:

- the *exact* approach in which we determine the vehicle trajectory with the data available on-board at request time.
- the *clustering*-based approaches in which instead of dealing with all candidate parking spots, we initially group all currently on-board available spots in geographic extends. We then work with these groups to provide a solution, and finally,
- the *live* approach in which we continuously receive updates regarding parking spots from oncoming (in communication range) cars and dynamically re-calibrate the trajectory taken to destination.

In all of the above approaches clearly additional factors may affect the final decision of where to actually park. For instance, paying a certain fee or seeking a free-of-charge space would be constraints to take into consideration. Such constraints could be easily incorporated in our methods but this discussion is omitted for brevity here.

A. The Exact Algorithm

Consider the setting of Fig. 2 in which a car departs from location s and has as its final destination location d. As soon as the driver requests service, the on-board equipment works with reported spots in locations 1, 2, 3 and 4. The vehicle could travel from s to any of these four locations. These moves

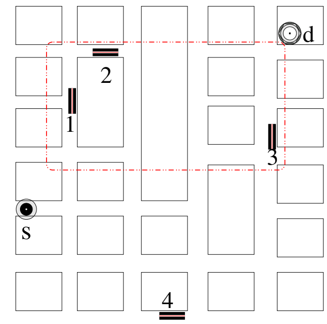

Fig. 2. Rationale of the dynamic-programming algorithm

constitute four feasible virtual edges to be taken and we need to estimate the cost of each; it turn, we need to carry out the same process going from the first selected spot to second one and so on until a full trajectory has been formed. The cost for a single "virtual directed edge" that connects two parking spaces a and b is defined as follows:

$$C(a, b, t_{tot}) = t_{ab} + p(t_{tot}) * w_b + [1 - p(t_{tot})] * D \qquad (1)$$

where t_{ab} is the time required to drive from space a to space b, w_b is the time needed to walk from space b to the final destination, D is a time penalty taken if space b is not available when the vehicle arrives there, and t_{tot} represents the accrued time until spot b has been reached. $p(t_{tot})$ is the probability to find space b still available after elapsed time t_{tot}.

The cost $C(a, b, t_{tot})$ consists of the time t_{ab} to drive from a to b as well as the walking time w_b required to reach destination d if b is still-available or the penalty D that we have to pay otherwise. The factor w_b is weighted by probability $p(t_{tot})$ whereas factor D by its complement. Clearly, if b is close to d (i.e., a short walk apart), the designated probability $p(t_{tot})$ needs to be high so that b is visited as early as possible; in this case w_b is expected to be much smaller than D. If the vehicle delays its visit to b, the weight $[1 - p(t_{tot})]$ of penalty D will be high. In order to avoid paying a large percentage of the time penalty, the vehicle will be forced to choose a space b, that is close to the previous space a, in the trajectory that connects all the reported free spaces. This way, the possibility that the vehicle will always arrive at a taken space, becomes highly unlikely. This cost function also leads vehicles towards producing trajectories that are minimal in terms of time.

In order to compute t_{ab}, a vehicle uses the statistics, maintained in its storage, for the traverse times of road segments, to find the shortest path from a to b. In [9], we discuss an approach to estimate t_{ab}. A number of average values, based on reported statistics, are used in the proposed cost function of Eqn. 1. The "space average life-time" (*salt*) is the average period that a parking space remains free once vacated. The average number of parking spaces visited before a vehicle finds a spot available is designated as "average spot missed" (*asm*). Statistics maintained in the vehicles storage, provide the values for both *salt* and *asm*. Based on the data we currently have at hand, we can compute the following two factors: first, the average time to drive from all spaces to all others or "spot to spot" (*sts*) and second, the "average walk time" (*wat*) from all spaces to d. Provided the above parameters, we can designate the factors $p(t_{tot})$ and D of Eqn. 1 as follows:

$$p(t_{tot}) = \frac{salt}{t_{tot} + salt} \tag{2}$$

$$D = asm * sts + wat \tag{3}$$

Our rationale is to select a decreasing function $p(t_{tot})$ that suggests $p(0) = 1$ and $p(salt) = 0.5$; $p(t_{tot})$ asymptotically reaches 0 when time goes to ∞. Time penalty D is an estimation of the average time we will need to park plus to walk using *asm*, *sts* and *wat*.

In order to find the trip that visits all spots and minimizes the proposed cost function, we apply the exact dynamic-programming Alg. 1 which is based on the *TVTSP-ZW* algorithm [13]. The input for this algorithm is the set of the known available spaces when service was solicited and the time T needed for the "longest" trip that visits all these spaces. This "longest" trip serves as an upper-bound for the maximum time a trip may incur. Appendix 1 discusses how Alg. 4 finds an approximation for T.

The rationale of the dynamic-programming Alg. 1 is depicted in Fig. 2. We search for the best trip that starts at the vehicle's current position s, visits spaces 1, 2, 3, ends in space 4; once in 4, we impose the requirement that time t has elapsed. Here, $1 \rightarrow 4$, $2 \rightarrow 4$ and $3 \rightarrow 4$ are the three previous alternative virtual edges. We find the best route by recursively examining the cost of reaching nodes 1, 2 and 3 and adding onto these costs the respective costs for traversing the above three virtual edges.

In Alg. 1, *B(s,x,S,t)* expresses the cost of the "best" trip that starts from s, visits all the spaces of set S, ends in space x and needs time t to complete. If no such trip can be furnished, then *B(s,x,S,t)* $= \infty$. In general, *B(s,x,S)* is the cost of the "best" trip for all t.

B. Grouping Spaces into Clusters

The time complexity of Alg. 1 is $O(n^3 T 2^n)$[13], where T is the time of the longest trip and n is the number of parking spaces. The main factor in this complexity is the factor 2^n. In order to lower the computational load implied by this time complexity, we decrease the problem's size by grouping parking spaces located geographically close into clusters. A vehicle groups the spaces known to be available, that are stored in its device, into clusters, and applies Alg. 1 using as input the clusters of parking spaces instead of distinct spots. We propose to locate the best trip around the clusters by using the *Exact Algorithm* and visit the spaces within each cluster using a best-first approach with respect to the cost function $C(a, b, t_{tot})$. Our conjecture is that through the use of clusters we can expedite the process of determining the trajectory to visit all known spaces by shedding computations. In Section V, we compare the optimal trip around distinct spaces and the trip around clusters, and evaluate what is the impact of applying the clustering, in the solution's quality.

Algorithm 1 `Optimal Trip Based on` $C(a,b,t)$

Input: $parkingSpaces$: The set of parking spaces inside the maximum walking range

T: The time of the longest trip that visits all the parking spaces

Output: The trip that starts from the current position, visits all the spaces, and has the minimum cost

Begin

s := vehicle's current position
for each space $y \in parkingSpaces$ **do**
  $time$:= time to drive from s to y
  $B(s, y, \emptyset, time) := C(s, y, time)$
  store s as the predecessor of space y in $B(s, y, \emptyset, time)$
end for
$spaces := |parkingSpaces|$
for $size = 2, 3, ..., spaces$ **do**
  for each subset of spaces S such that $|S| = size$ **do**
    for $t = 1, 2, ..., T$ **do**
      for each space $y \in S$ **do**
        for each space $x \in S - \{y\}$ **do**
          $time$:= time to drive from x to y
          $u := t - time$
          if $B(s, x, S - \{x, y\}, u)$ is equal to ∞ **then**
            continue with the next space x
          end if
          $addedCost := C(x, y, t)$
          if $B(s, x, S - \{x, y\}, u) + addedCost < B(s, y, S - \{y\}, t)$ **then**
            $B(s, y, S - \{y\}, t) := B(s, x, S - \{x, y\}, u) + addedCost$
            store x as the predecessor of y in $B(s, y, S - \{y\}, t)$
          end if
        end for
      end for
    end for
  end for
end for
$bestCost := \infty$
for each space $y \in parkingSpaces$ **do**
  $cost := B(s, y, parkingSpaces - \{y\})$
  if $cost < bestCost$ **then**
    $bestCost := cost$
  end if
end for
return The trip that refers to $bestCost$

End

The clustering impact can be limited, if the parking spaces that are grouped into a cluster, are spaces that are "very close" to each other. Ideally, spots that ultimately appear in clusters are partial spot-sequences appearing in the optimal trajectory produced by Alg. 1. We elect to use *Quality Threshold (QT) Clustering*[14] as we do not want to designate the number of resulting clusters and we use the Euclidean distance metric. Fig. 3 shows 16 parking spaces that are grouped into 5 clusters. The vehicle at point s, will decide the order to visit 5 clusters, using Alg. 1. The spaces within each cluster are to be visited in a best-first order fashion.

C. Cutting-off Clusters

QT Clustering uses a threshold for the maximum radius of the clusters it generates. If we use a value for this threshold, that is less than the minimum distance of all the pairs of parking spaces, we will end up with one cluster per parking space. The opposite case is when we use a large threshold and we end up with just one cluster with all the parking spaces inside. An appropriate value for the threshold of *QT* clustering is strongly related to the city where the vehicle moves. The resulting clusters that will be the input to the *Exact* Alg. 1 may

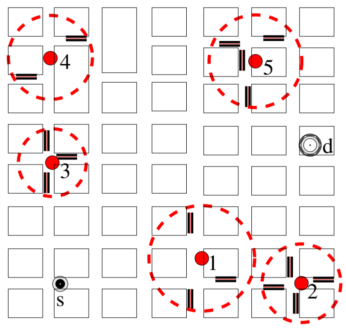

Fig. 3. Clustering Example

still inflict substantial load on the computational capabilities of a vehicle's device. In order to adjust the problem's size in its operational environment, a vehicle can use a subset of all the clusters as input in Alg. 1. The size of this subset critically affects the computational load and must be selected according to the computational resources of the on-board equipment.

The choice for the subset of clusters is not trivial. One option would be to keep the *top-k* clusters, based on the cost function $C(a, b, t_{tot})$. Since our aim is to be close to the solution for the optimal trip, we choose to identify *k* areas for the parking spaces and keep one cluster as a representative of each area. Fig. 4 shows that, for the 5 clusters of Fig. 3, we identify three areas. One area for clusters 1 and 2, one for clusters 3 and 4, and one for cluster 5. The representative clusters are 1, 4 and 5, respectively. The method we apply, in order to identify the areas and choose the subset of the representative clusters is the *k-medoid* clustering and the *Partitioning Around Medoids (PAM)* approach[15]. The data points that are given as *PAM* input are the cluster centers.

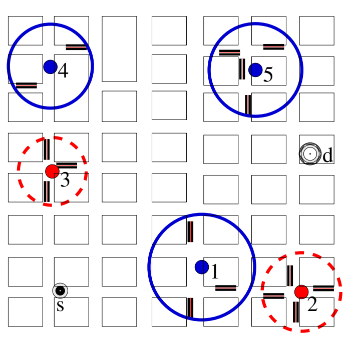

Fig. 4. Cutting-off Example

The rest of the clusters that are not selected in the subset of representatives, will be inserted, one by one, in the trip generated by Alg. 1. At every step, we select the cluster center that is the closest to one of those already selected. The chosen cluster will be inserted into the position, in the sequence of already inserted clusters, that gives the best total cost for a trip. Alg. 2 termed *Synchronization Mode*, depicts a combined approach that carries out clustering, selection of best representatives, use of the *exact* algorithm and finally insertion of cut-off clusters. We use the Euclidean distance as the distance metric and the distance between two clusters is the one between their centers. Note that the time to traverse a cluster, is the time to traverse the spaces contained in it, using a best-first approach.

Algorithm 2 `Synchronization Mode`

Input: $parkingSpaces$: The set of parking spaces inside the maximum walking range

k: The size of the set that will be given as input to Alg. 1

Output: The cluster sequence which corresponds to the proposed trip around the spaces

Begin

$clusters$:= **apply** QT clustering with $parkingSpaces$ as input
$topK$:= **apply** PAM algorithm with $clusters$ and k as input
$remainder$:= $clusters - topK$
$approximation$:= **apply** Alg. 4 with $topK$ as input
$clusterSeq$:= **apply** Alg. 1 with $topK$ and $3 * approximation$ as input
while $remainder \neq \emptyset$ **do**
  $toBeInserted$:= The cluster $c \in remainder$ with the minimum distance to a cluster $l \in \{clusters - remainder\}$
  insert $toBeInserted$ into the position of $clusterSeq$ that gives the best cost for the corresponding trip
  update $clusterSeq$ with the new insertion
  $remainder$:= $remainder - \{toBeInserted\}$
end while
return $clusterSeq$

End

D. Live-Mode Approach

While a vehicle is in motion, it communicates with others encountered within its radio range and continually receives reports (updates) about parking spaces. An update informs that either a spot previously reported free is taken or a new opening exists in the vicinity. Evidently, a set of such updates may be received at any time in the form of a batch. These change may necessitate that the vehicle re-adjusts the trip initially computed and currently followed. A likely option would be to have Alg. 2 invoked every time an update is received. This would clearly pose significant burden on the on-board computing resources. Here, we present an *incremental* approach for possibly adjusting the trip initially generated by Alg. 2 once a vehicle receives an update. Moreover, we define the condition that specifies when a vehicle has to invoke Alg. 2 anew in order to regenerate a near-optimal trip for the heavily-updated parking space set stored on-board.

A vehicle examines updates from a batch one-by-one and in doing so, re-adjusts its trajectory connecting all known spaces. For every update, the vehicle detects which of the already-formed clusters that make up its current route is affected. If the update reports a spot already-taken, the vehicle has to reflect this fact on its storage so that others may be notified properly; in addition if this space is already part of a cluster, the vehicle marks this cluster in the trajectory as affected. Should the update report a new position, the vehicle has to detect which of its trajectory clusters is the best candidate to incorporate this "newly-discovered" space. The best such cluster candidate is the one that maximizes the scoring function $SF = 1/(r * r_C^2)$, where r is the radius of a cluster, and r_C is the Euclidean distance of the new space to the center C of a cluster. If a cluster consists of just a parking space, then $SF = 1/r_C^3$. This scoring function SF favors clusters with smaller radius, that have a center closer to the new space. If the new space is not within the maximum radius permitted for any of the current clusters, we generate a new cluster that consists of only this new parking spot.

After detecting the cluster affected by a single update, the vehicle must examine if the clusters must be visited in a different order. More specifically, if the vehicle follows a trip that sequentially traverses n clusters, it must decide whether to re-order the position of the affected cluster in the sequence. In this regard, each of the $n+1$ possible positions corresponds to a new trip. We consider the position that corresponds to the trip with the *minimum total cost* as the best position for the affected cluster. The worst-case time-complexity for handling a single update is *O(n*m)* where m is the number of pre-existing spaces and n is the number of clusters. Each of the n+1 trips that have to be computed has an individual cost of *O(m)*. We experimentally investigate how computationally intensive is the above update procedure in section V-A.

Updates may either trigger repositioning of existing clusters or form new clusters; these changes essentially reflect alterations that affect the initial cluster sequence compiled by Alg. 2. Should the update-batches heavily modify the status of the parking space set a vehicle works with, we might have to re-invoke Alg. 2. The question here is whether we can continue using incremental adaptations of the cluster-sequence or we have to resort to something more drastic which is the recompilation of the sequence from scratch (with the new data). We define a distance metric that quantifies the degree of change between two routes: the route initially designated [1] and the travelling sequence maintained incrementally. Below, we first provide an example for this distance metric and we then offer its respective closed-formula.

Consider the situation in Fig. 5. Here, a vehicle was aware of the available spaces 1, 2 and 3 the last time Alg. 2 was invoked. When the vehicle reaches point s, it receives a batch-update that collectively reports spaces 1, 2 and 3 as occupied and spaces 4 and 5 as available. The initial sequence of say $1 \rightarrow 2 \rightarrow 3$ changes and through incremental computations becomes $4 \rightarrow 5$. The vehicle at s calculates the distance between the above two trips before it decides to use Alg. 2. Each spot of the adjusted trip, which visits less spaces, corresponds to $3/2$ spaces of the initial one. We match the first space of the adjusted trip, 4, with the first $2(=\lceil 3/2 \rceil)$ spaces, 1 and 2, of the initial trip, and space 5 with space 2 and space 3. The distance between the two trips, is a weighted sum of the distances of the matched spaces. In this example, the weight for each matching pair is $\frac{3/2}{2} = 0.75$. For the four matching pairs 4 with 1, 4 with 2, 5 with 2, 5 with 3 the Euclidean distances are respectively 4, 4, 6, and 5. The distance between the two trips is set at $0.75(4+4+6+5) = 14.25$. More formally, we define the distance between two proposed trips S and L as follows:

$$w\{\sum_{j=1}^{\lceil\frac{|L|}{|S|}\rceil} d(s_1, l_j) + \sum_{i=2}^{|S|}\sum_{j=f(i)}^{to(i)} d(s_i, l_j) + \sum_{j=to(|S|)}^{|L|} d(s_{|S|}, l_j)\}$$

where

$$f(i) = \lceil\frac{|L|}{|S|}\rceil + r + (i-2)[\lceil\frac{|L|}{|S|}\rceil - 1 + r],$$

$$to(i) = \lceil\frac{|L|}{|S|}\rceil + (i-1)[\lceil\frac{|L|}{|S|}\rceil - 1 + r],$$

$$w = \frac{|L|}{|S|} / \lceil\frac{|L|}{|S|}\rceil$$

Without loss of generality, we assume that trip S visits $|S|$ spaces (designated as s_i for i=1...$|S|$) which are less than the $|L|$ spaces, (designated as l_j for j=1...$|L|$), visited by trip L. A specific parking spot s_z that is part of the short trip S matches with the spaces of the long trip L, that start from $f(z)$, and end to $to(z)$. The variable r is equal to 1 when $|L| mod |S| = 0$ and 0 otherwise. The distance $d(s,l)$ between two spaces s and l is the Euclidean distance.

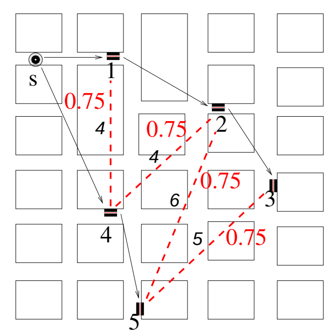

Fig. 5. Distance between two routes

The interaction between our proposed incremental computing approach and Alg. 2 is presented in Alg. 3. Every vehicle runs this algorithm as a deamon that creates and incrementally updates the trajectory to follow in the presence of continuously disseminated information on the status of spots. The *SynchThres* threshold designates the maximum distance between two routes and we use it to determine if the *Synchronization Mode* has to be immediately applied, instead of carrying on the incremental updating. The value of this threshold depends on how frequently the computing device of a vehicle can reserve computational resources to apply Alg. 2.

V. Evaluation

In order to evaluate the proposed method, we built a simulator and we performed experiments on a part of the Paris road network. A sreenshot of the simulator's GUI is given on Fig. 6. This simulator uses a simple dissemination protocol for the parking information, that can be described as follows: each vehicle stores for each known available parking space, spatial data and a timestamp, that denotes when this space

[1] the last time we run the *Synchronization Mode* Alg. 2

Algorithm 3 `Live-Mode Deamon`

Initialization:

$synchSeq$:= **apply** Alg. 2 with the known parking spaces as input

$curSeq$:= $synchSeq$

Begin

```
while true do
  if there is an addition or deletion in the set of known parking spaces then
    for each change do
      if a space s was taken then
        affectedCluster := The cluster that contains s
        curSeq := curSeq − {affectedCluster}
      end if
      if a new space s is available then
        affectedCluster := The best cluster to contain s, based on SF
        if no cluster could contain s then
          affectedCluster := A new cluster that contains only s
        else
          curSeq := curSeq − {affectedCluster}
        end if
      end if
      insert affectedCluster into the position of curSeq that gives the best
        cost for the corresponding trip
      update curSeq with the new insertion
    end for
    distance := The distance between the trips for curSeq and synchSeq
    if distance > SynchThres then
      synchSeq := apply Alg. 2 with the known parking spaces as input
      curSeq := synchSeq
    end if
  end if
end while
```

End

was reported available. Vehicles exchange parking information with other in-range vehicles, and keep the most up-to-date information. When a vehicle traverses a road segment, it is able to detect free parking spaces. A final destination is assigned to each vehicle, and a vehicle will park only to a parking space that is within a maximum distance range from the final destination and is larger than the vehicle's length.

The experiments performed are divided in two parts: in the first, we ascertain the effectiveness of the incremental computing and clustering techniques. The metrics used in the first part of evaluation, are the execution time for the computation of a trip around a set of parking spaces and the cost difference of a trip from the optimal trip, using the cost function $C(a, b, t_{tot})$ described in section IV-A. In the second part of our experiments, we compare our method to a Best-First approach, where each vehicle goes directly to the most promising (or "best" at the time) parking space. We use the following metrics: *a)* time needed to reach a free parking space, *b)* time required to walk from the selected space to the final destination and *c)* the percentage of vehicles that managed to park.

The simulator has been developed on *Java 6* and all the experiments were performed on a machine equipped with an Intel Core2 Duo 3.00GHz CPU and 4 GB RAM. Throughout the experiments, we assumed a wireless communication range of 50m, a bandwidth of 2 Mbps and a vehicle density of 20 vehicles per km.

A. Clustering and Cutting Off Impact

In the experiments presented here, we monitored the computational time needed by one vehicle to decide the trip around parking spaces, using the methods we propose in this paper. We also estimate how "close" a trip remains to the optimal trip, when we apply the *Clustering* and *Cutting Off* methods. The cost difference of a trip from the optimal, states this fact; the cost in question is expressed by the function $C(a, b, t_{tot})$ of section IV-A. The parameters affecting the geographic distribution of parking spaces are the average, ave, and standard deviation, std, for the distance between every two spots. In the results we present in this paper, ave was set to 522m and std to 395m.

Fig. 7 depicts the effect of *QT Clustering* with 3 different values for quality threshold $QThres$. This threshold expresses the maximum radius a cluster may display. We choose ave and std as the parameters which affect the maximum radius. The three levels of clustering are designated as follows:

- Cluster1: $QThres = ave/3$
- Cluster2: $QThres = ave/3 + std/2$
- Cluster3: $QThres = ave/3 + 2 * std$

The x-axis expresses the number of parking places known by the vehicle that computes the trip around them. The y-axis expresses the execution time for the computation of the trip in milliseconds and logarithmic scale. The execution time needed by Alg. 1 which finds the optimal trip, increases exponentially to the number of known parking places. The execution time for Cluster1 also rises when the number of parking spaces increases, as the number of clusters that are formed becomes larger. Cluster2 and Cluster3 form less clusters than Cluster1 and have reduced execution times.

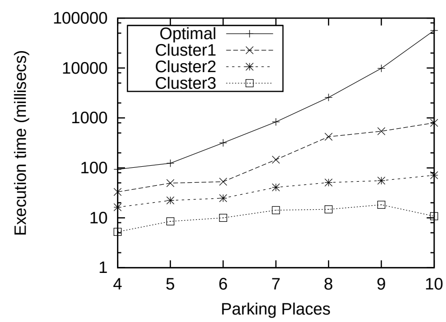

Fig. 7. Time to compute the trip around parking places for the optimal and three clustering levels

Fig. 8 presents the cost difference from optimal for the same experiments with those depicted in Fig. 7. Cluster1 has the best cost difference in most cases, since it forms clusters with parking spaces that are close to each other, and it is likely that these spaces are visited sequentially, in the optimal trip. The fact that the curves of Cluster2 and Cluster3 are rough, is an indication that the best first approach for visiting large subsets of parking spaces, gives near-optimal results in some cases, and high-cost results in other cases.

In Figs. 9 and 10, we compare the *top-k* method and the *k-medoids* method for deciding the *k*, out of 10 spaces [2], that will

[2]The computational resources were not sufficient for computing the optimal solution for a problem size over 10 using the *Exact* approach.

Fig. 6. Screenshot of the simulator's GUI

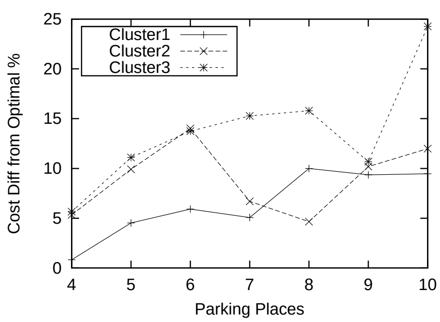

Fig. 8. Cost difference from optimal for three clustering levels

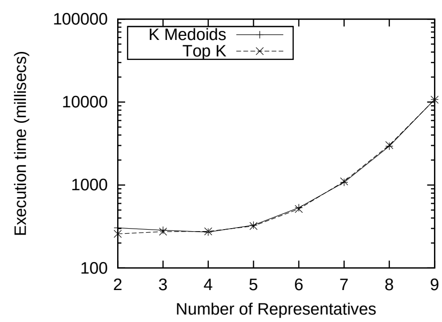

Fig. 9. Time to compute the trip around parking places, using *top-k* and *k-medoids* methods, to produce a subset of parking spaces

be used as input in Alg. 1, as described in subsection IV-C. The *top-k* method selects the *k* spaces with the lower cost, using cost function $C(a, b, t_{tot})$ described in section IV-A. The x-axis in both figures is the number, *k*, of spaces that will be selected by the two methods. We observe that the differences in the execution time of the two methods are minimal, since they will always run Alg. 1 with the same input size, *k*. We also observe that when 30% to 70% of the parking spaces are used as representatives, the *k-medoids* method shows significantly better cost difference. This can be explained by the fact that *top-k* method selects the subset of the "best" parking spaces, that may be positioned close together, while *k-medoids* selects a subset that represents in a better way, the topology of the entire parking space set.

In Figs. 11 and 12 we apply clustering and then we keep *k* of the clusters, as representatives, using the *k-medoids* method. The x-axis denotes *k* in both figures, and the three clustering levels are the same as in Figs. 7 and 8. We observe that the cost

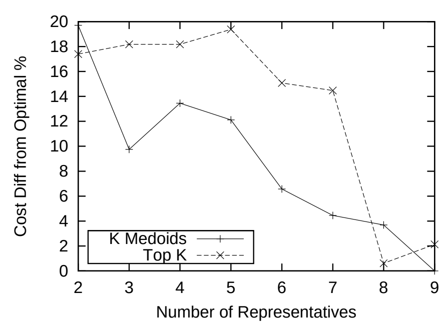

Fig. 10. Cost difference from optimal, using *top-k* and *k-medoids* methods, to produce a subset of parking spaces

difference remains the same as *k* moves over 4, for Cluster2, and remains the same for all the values of *k*, for Cluster3. This is because, in most cases, Cluster2 does not form, more than 4 clusters, and Cluster3 does not form more than 2. Cluster1 forms more clusters, and the cost difference from optimal is better than Cluster2 only when a sufficiently large number of these clusters are used as representatives. Note that when more than 5 clusters are used as representatives, the cost difference from optimal is less than 10%, for Cluster1.

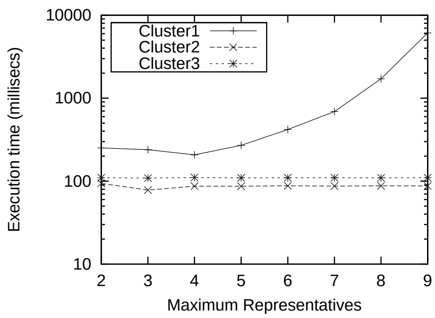

Fig. 11. Time to compute the trip around parking places, using clustering and cutting off

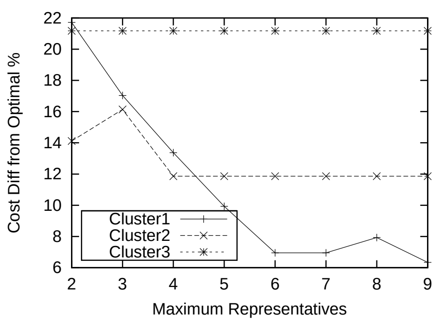

Fig. 12. Cost difference from optimal, using clustering and cutting off

A comparison between the clustering and cutting off methods, is given in Figs. 13 and 14. For each experiment performed using clustering, that produces L clusters, we run an experiment using *k-medoids*, keeping L representatives for all the spaces. In both cases the L representatives make up the input to Alg. 1. We used 5 clustering levels which are more fine-grained than the 3 levels used before. As Figs. 13, 14 show, cutting off gives a significantly better outcome than the clustering method, for all the clustering levels, while the execution time is slightly more in most cases. This outcome indicates that it is better to find the optimal trip for a number L of representatives for the whole parking space set, and then insert the rest of the spaces in this trip one by one, than finding the optimal trip for L clusters, and then visit the spaces inside a cluster using a best-first approach.

Figs. 15 and 16 respectively depict the computational overhead and the deviation from the optimal trip when *Live-Mode* is in use. In Fig. 15, a vehicle re-adjusts its trajectory, for a number of updates. The execution time for handling these updates, starts from 50ms for one or two updates, then increases linearly, and finally reaches 500ms for 8 updates. Fig. 16 presents the cost difference from the optimal trip, if instead of applying *Live-Mode*, a vehicle "synchronizes"

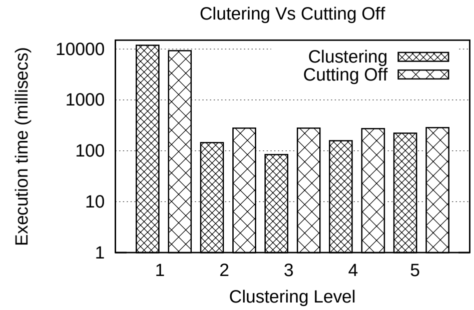

Fig. 13. Time to compute the trip around parking places for clustering and cutting off methods

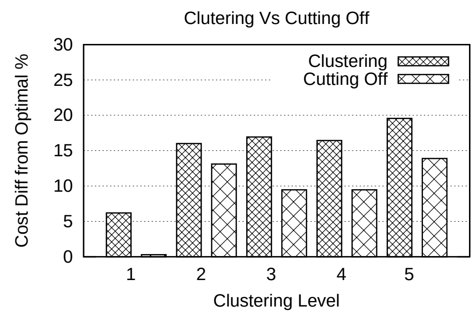

Fig. 14. Cost difference from optimal for clustering and cutting off methods

(i.e., executes Alg. 2) its trip after every update is received. When the number of updates, since the last time the vehicle synchronized its trip, is less than 4, the cost difference remains below 2%. Cost difference rises to 4% for 5 to 7 updates, and reaches 9% for 8 updates.

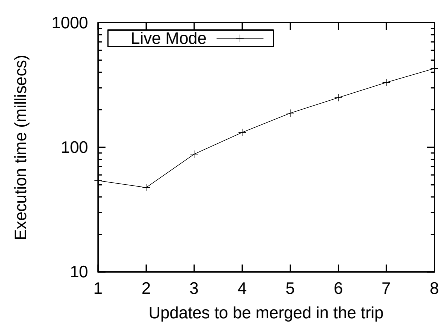

Fig. 15. Execution Time for merging a number of updates, using the *Live-Mode*

The experiments we carried out in this section overall indicate that *Live-Mode* features low computational overheads and the re-adjusted route it yields is close to the optimal trip. When clustering is performed with a large radius for clusters or when cutting off is performed with a small number of representatives, the formed route may be considerably worse than the optimal. The most efficient way to keep the size of the problem small and attain a near-optimal solution is to initially deploy clustering with a small radius and subsequently, to use *k-medoids* to select *k* representative clusters.

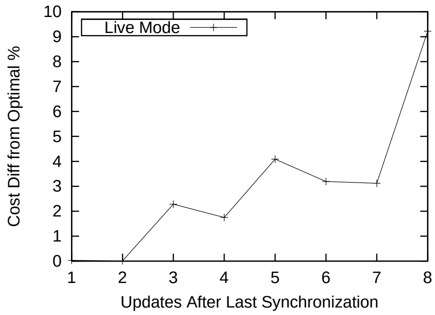

Fig. 16. Cost difference from optimal, after a number of updates, using the *Live-Mode*

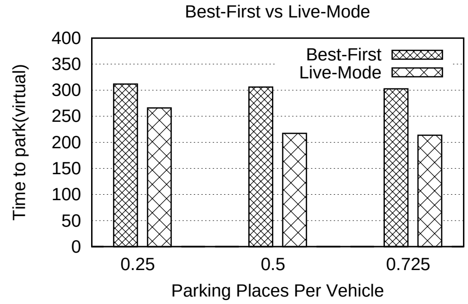

Fig. 17. Time to find a parking space free for the *Live-Mode* and Best-First approaches

B. *Live-Mode versus Best-First*

In this set of experiments, we examine how much time a vehicle gains following a route that optimizes the cost function $C(a, b, t)$ compared to a trip realized using a best-first approach at every iteration. Regardless of the approach taken the cost function remains the same and takes into account the time needed to arrive to parking space, the time required to walk from this spot to the final destination and finally, the probability to find the spot still free.

Three different ratios, 0.25, 0.5, 0.725, are selected for the number of available parking spaces per vehicle. The starting point and final destination of a vehicle are chosen using uniform distribution, inside predefined regions of the road network. We generate pairs of vehicles, having the same starting point and final destination. One vehicle is using our *Live-Mode* approach while the second uses the best-first technique. The specific settings for the *Live-Mode* are *ave - std* for the maximum radius of clusters, and 7 for the maximum number of representatives. In all our experimentation, we maintain the space per vehicle ratio constant by generating new pairs of vehicles and new parking spaces as needed.

Figs. 17, 18 and 19 respectively show the average time needed to find a free parking space, the time needed to walk from that parking space to the final destination, and the percentage of vehicles that found a free space using the two approaches. *Live-Mode* demonstrates better results in all three cases. Vehicles that operate *Live-Mode* choose to, first, visit areas where there is a high density of parking spaces. In contrast, Best-First vehicles elect to visit spots that are their best option regarding their final destination and current position, even if these spots lie on areas with low-density of available spaces. The fierce competition among vehicles using the Best-First approach does not allow any significant improvement when the ratio of spaces per vehicle increases.

Fig. 17 shows that for all space ratios examined, *Live-Mode* attains better average time if compared with its Best-First counterpart. The biggest gains appear for rations 0.5 and 0.725 where *Live-Mode* can help vehicle effectively travel around empty spots. For space ratio 0.25, the difference of time required to park between *Live-Mode* and Best-First is noticeably smaller due to higher competition for spaces.

As Fig. 18 depicts the walking time to the destination is

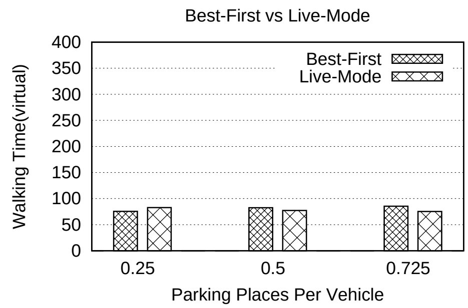

Fig. 18. Walking time to the final destination for *Live-Mode* and Best-First approaches

close for both approaches, in all cases. In general, we would expect that a vehicle using the Best-First approach would perform better than those enabled with *Live-Mode* as it tries to aggressively select a spot very close to the destination. However this does not frequently occur in our experiments since Best-First vehicle find their top picks occupied and so they have to settle for inferior choices.

As Fig. 19 shows, the percentage of vehicles finding a free space consistently remains higher for the *Live-Mode* approach. The results of Fig. 19 are in line with those presented in Figs. 17 and 18; simply the differences become more pronounced for ratios 0.5 and 0.725.

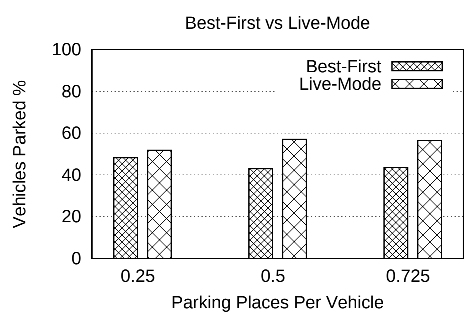

Fig. 19. Percentage of vehicles that found a parking space free for *Live-Mode* and Best-First approaches

VI. Conclusions

In this paper, we address the problem a driver faces while trying to locate a parking position in an urban environment. We assume that vehicles are equipped with *vehicle-to-vehicle* communication equipment which also features computational and storage capabilities. As the vehicle navigates, it continuously receives reports about available spots close to the area the driver intends to park from oncoming traffic. We formulate the problem using the *Time-Varying TSP* approach and propose methods that reduce complexity for this *NP*-hard problem. Our methods attempt to present a good trade-off between optimality of the solution adopted and computational requirements. We developed a detailed simulation engine that helped us experimentally evaluate our approach. Using a Best-First baseline approach we demonstrate that our *Time-Varying TSP* approach can have substantial performance benefit. We showed that our methods, if combined properly, can produce a near-optimal trip with respect to the deployed cost-function. Finally, our experimentation indicates that the incremental updating approach (*live-mode*) maintains close-to-optimal trajectories in the presence of continuously disseminated parking information.

Acknowledgment

We would like to thank Herald Kllapi for his help with the simulation environment and his comments. This work has been partially supported by the *D4Science II* FP7-project funded by the European Commission.

References

[1] W. Park, B. Kim, D. Seo, D. Kim, and K. Lee, "Parking Space Detection using Ultrasonic Sensor in Parking Assistance System," in *IEEE Intelligent Vehicles Symposium*, Eidhoven, The Netherlands, June 2008, pp. 1039–1044.

[2] R. Lu, X. Lin, H. Zhu, and X. Shen, "SPARK: a New VANET-based Smart Parking Scheme for Large Parking Lots," in *Proc. of the IEEE INFOCOM Conference*, Rio De Janeiro, Brazil, April 2009.

[3] B. Xu, A. Ouksel, and O. Wolfson, "Opportunistic Resource Exchange in Inter-Vehicle Ad-Hoc Networks," in *Proc. of Int. Conf. on Mobile Data Management*, Berkeley, CA, January 2004, pp. 4–12.

[4] T. Repantis and V. Kalogeraki, "Data dissemination in mobile peer-to-peer networks," in *Proc. of the Int. Conf. on Mobile Data Management*, Ayia Napa, Cyprus, May 2005, pp. 211–219.

[5] M. Caliskan, D. Graupner, and M. Mauve, "Decentralized Discovery of Free Parking Places," in *Proc. of the 3rd Int. Workshop on Vehicular Ad-hoc Networks*, ser. VANET '06. Los Angeles, CA: ACM, September 2006, pp. 30–39.

[6] T. Delot, N. Cenerario, S. Ilarri, and S. Lecomte, "A Cooperative Reservation Protocol for Parking Spaces in Vehicular Ad-hoc Networks," in *Proc. of the 6th Int. Conf. on Mobile Technology, Application and Systems (Mobility'09)*. Nice, France: ACM, 2009, pp. 1–8.

[7] M. Caliskan, A. Barthels, B. Scheuermann, and M. Mauve, "Predicting Parking Lot Occupancy in Vehicular Ad-hoc Networks," in *Proc. of the IEEE Int. Conf. on Vehicular Technology Conference (VTC'07)*. Dublin, Ireland: IEEE, April 2007, pp. 277–281.

[8] A. Khekdar and V. Kumar, "Self-Synchronizing Moving Objects Using Contextual Information," in *Proc. of the 2010 Int. Conf. on Information and Knowledge Engineering (IKE'10)*. Las Vegas Nevada, USA: CSREA Press, 2010, pp. 330–335.

[9] V. Verroios, K. Kollias, P. K. Chrysanthis, and A.Delis, "Adaptive Navigation of Vehicles in Congested Road Networks," in *Int. Conf. on Pervasive Services (ICPS'08)*. Sorrento, Italy: ACM, July 2008.

[10] E. Kokolaki, M. Karaliopoulos, and I. Stavrakakis, "Value of information exposed: wireless networking solutions to the parking search problem," in *Proc. of the 8th Int. Conf. on Wireless On-demand Network Systems and Services(IFIP/IEEE WONS)*, Bardonecchia, Italy, 2011.

[11] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrasekaran, W. Xue, M. Gruteser, and W. Trappe, "ParkNet: drive-by sensing of road-side parking statistics," in *Proc. of the 8th Int. Conf. on Mobile Systems, Applications, and Services(MobiSys '10)*. San Francisco, USA: ACM, 2010, pp. 123–136.

[12] J. Boehle, L. Rothkrantz, and M. V. Wezel, "CBPRS: A City Based Parking and Routing System," *ERIM Report Series Reference No. ERS-2008-029-LIS*, May 2008.

[13] X. Cai, D. Sha, and C. Wong, *Time-Varying Network Optimization*, 1st ed. Springer Publishing, 2007.

[14] L. Heyer, S. Kruglyak, and S. Yooseph, "Exploring Expression Data: Identication and analysis of Coexpressed Genes," in *Genome Research 9*, 1999, pp. 1106–1115.

[15] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.

[16] M. Fisher, L. Nemhauser, and L. Wolsey, "An Analysis of Approximations for Finding a Maximum Weight Hamiltonian Circuit," *Operations Research*, vol. 27, no. 4, pp. 799–809, July 1979.

Appendix 1

The problem of finding the longest trip can be formulated to the *MAX-TSP*[16], which is also *NP*–hard. We select a greedy algorithm that offers a $1/3$-approximation of the time needed for the longest trip[16]; this appears in Alg. 4. The algorithm initially chooses a directed edge of maximum weight and then continues to choose edges of maximum weight subject to the requirement that the collection be contained in a directed cycle with all the nodes. In our work, nodes are the parking spaces and the weight of a directed edge is the drive time from the first corresponding space to the second. The value obtained by Alg. 4 is multiplied by 3 before provided as input to Alg. 1; in this manner, we approximate the value of the longest trip throughout all nodes.

Algorithm 4 `"Longest" Trip Approximation`

Input: $parkingSpaces$: The set of parking spaces inside the maximum walking range

Output: A value that is at least the $1/3$ of the time of the longest trip that visits all the parking spaces

Begin

for each space $x \in parkingSpaces$ **do**
  for each space $y \in parkingSpaces - \{x\}$ **do**
    store the time to drive from x to y as the cost for the virtual directed edge from x to y
  end for
end for
$selectedSet := \emptyset$
while $selectedSet$ does NOT form a cycle containing all the spaces **do**
  $edgeAdded$:= The edge with the maximum cost, which satisfies the constraint that each space will have at most one incoming and one outgoing edge, if it is added in the $selectedSet$
  $selectedSet := selectedSet \cup \{edgeAdded\}$
end while
s := vehicle's current position
$maxFromSource := 0$
for each space $x \in parkingSpaces$ **do**
  $time$:= The time to drive from s to x
  if $time > maxFromSource$ **then**
    $maxFromSource := time$
  end if
end for
$cycleTime$:= The sum of the edges' cost in $selectedSet$
return $maxFromSource + cycleTime$

End