Spatio-Temporal Matching for Urban Transportation Applications

DANIEL AYALA, Department of Computer and Mathematical Sciences, Lewis University
OURI WOLFSON and BHASKAR DASGUPTA, Department of Computer Science, University of Illinois at Chicago
JIE LIN, Department of Civil and Materials Engineering, University of Illinois at Chicago
BO XU, HERE Technologies

In this article, we present a search problem in which mobile *agents* are searching for static *resources*. Each agent wants to obtain exactly one resource. Both agents and resources are spatially located on a road network and the movement of the agents is constrained to the road network. This problem applies to various transportation applications including: vehicles (agents) searching for parking (resources) and taxicabs (agents) searching for clients to pick up (resources). In this work, we design search algorithms for such scenarios. We model the problem in different scenarios that vary based on the level of information that is available to the agents. These scenarios vary from scenarios in which agents have complete information about other agents and resources, to scenarios in which agents only have access to a fraction of the data about the availability of resources (uncertain data). We also propose pricing schemes that incentivize vehicles to search for resources in a way that benefits the system and the environment. Our proposed algorithms were tested in a simulation environment that uses real-world data. We were able to attain up to 40% improvements over other approaches that were tested against our algorithms.

CCS Concepts: • **Information systems** → **Location based services**; *Mobile information processing systems*; • **Theory of computation** → *Computational pricing and auctions*;

Additional Key Words and Phrases: Location based services, solution concepts in game theory, computational pricing and auctions, incomplete, inconsistent, uncertain databases

ACM Reference format:
Daniel Ayala, Ouri Wolfson, Bhaskar Dasgupta, Jie Lin, and Bo Xu. 2018. Spatio-Temporal Matching for Urban Transportation Applications. *ACM Trans. Spatial Algorithms Syst.* 3, 4, Article 11 (May 2018), 39 pages.
https://doi.org/10.1145/3183344

This work was supported in part by the NSF IGERT program under grant DGE-0549489 and NSF grants IIS-1213013, IIS-1160995, and CCF-1216096.

Authors' addresses: D. Ayala, Department of Computer and Mathematical Sciences, Box 298, Lewis University, One University Parkway, Romeoville, IL 60446, USA; email: ayalada@lewisu.edu; O. Wolfson and B. DasGupta, Department of Computer Science, 851 S. Morgan (M/C 152), Room 1120 SEO, Chicago, IL 60607-7053; emails: wolfson@cs.uic.edu, bdasgup@uic.edu; J. Lin, UIC Civil and Materials Engineering, 2095 Engineering Research Facility, 842 W. Taylor Street (M/C 246), Chicago, Illinois 60607-7023; email: janelin@uic.edu; B. Xu, 425 W. Randolph St., Chicago, IL 60606; email: xu@here.com.

https://doi.org/10.1145/3183344

1 INTRODUCTION

The problem addressed in this article concerns *mobile agents* searching for *stationary resources* in space. Each resource has a *fixed* spatial location, and each agent is moving through space to procure exactly one resource. A search problem of this nature arises in applications that are very commonplace in a typical urban transportation system such as the following:

—A set of taxicabs (mobile agents) are looking to pick up clients (stationary resources) in a given urban area.
—A set of travelers in vehicles (mobile agents) are looking for available parking slots (stationary resources).

Another more recent scenario in which this type of search problem is applicable is the *bike-sharing* initiatives in cities [11]. In this scenario, travelers on bicycles are the mobile agents, and each such traveler needs to find an available docking station (stationary resources) for her/his bicycle. A similar situation also arises when travelers in electric vehicles are searching for charging stations. In all these scenarios, the agents move based on a database of spatial information reflecting the locations and availabilities of the resources; the database changes *dynamically* as more and more information about the availability of resources is received. It bears mentioning that these resources are different from points-of-interest (POI's) that are studied in various search problems. These resources studied in this work can be used by only one agent at a time, whereas POI's are unlimited in this sense.

There are four possible scenarios that could be studied for the search problem studied in this article, depending on the information that can be obtained by the mobile agents. We illustrate these scenarios below with the help of Figure 1.

1.1 Zero Information (Oblivious) Scenario

The most common situation that happens in an urban transportation system is that vehicles (agents) are not aware of the positions of their desired resources. Then each agent wanders around until it comes upon an available desired resource. This is a case in which *no* information about the resources are available to the agents. The zero-information case is similar to the deterministic version of the so-called "honey-pot searching" problem originally motivated by U.S. Navy operations during the Second World War [9, 15, 26].

In the example shown in Figure 1, in the zero-information case, both the agents wander around trying to find a resource. Clearly, the agents would benefit from having some information available about the resources to make better decisions.

1.2 Incomplete Information Scenario

The proliferation of mobile devices, location-based services, and wireless sensors has given rise to applications that can help the agents to find their desired resources. For instance, in the taxicab example, potential clients (resources) could report to a server by using their cell phones, and the server could share the location information of the clients with the taxicab drivers (agents) that use their service. For the example of bike-sharing initiatives in cities, one could have sensors embedded in the docking stations informing a server of the locations of available docks (resources), and this location information could then be shared with the bike riders (agents) that wish to dock their bikes. Similar applications can also be found in the context of finding parking for vehicles [41, 45]. In a nutshell, these are examples of applications that share resource *availability data* with the agents, and the agents in turn use this data to make intelligent decisions to find resources. Thus, in

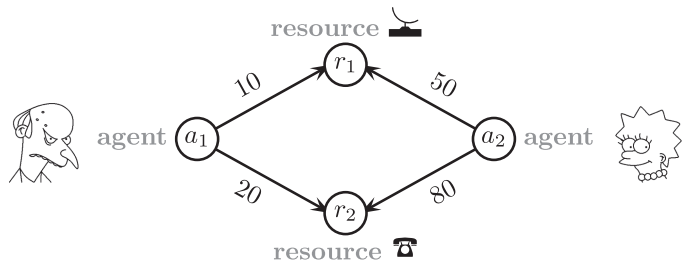

Fig. 1. An example of two mobile agents a_1 and a_2 and two stationary resources r_1 and r_2. The numbers represent (in seconds) the travel times for the agents to reach the corresponding resources.

the *incomplete information* scenario, the agents are aware of the location of the available resources, and act selfishly to optimize their own performances.

In the example shown in Figure 1, in the incomplete information case with each agent looking to minimize its immediate travel time to obtain a resource, both a_1 and a_2 will travel toward the resource r_1 to obtain it. However, only one of them, namely a_1, will be *satisfied* with the outcome by obtaining the resource, and the other agent a_2 will have to travel extra time to look for the next available resource r_2.

1.3 Complete Information Scenario

In a *complete information* scenario, each agent is able to obtain all the information that is necessary to make its decision, and agents may make choices that are a best response to what they know other agents may be doing. For example, the agents may receive information not only on the locations of the resources but also about the locations of the other agents as well.

For the example shown in Figure 1, in the incomplete information, scenario a_2 was left unsatisfied because of a wasted trip to grab resource a_1 only to find out that it was taken by a_1 ahead of its arrival. If a_2 had been aware of the proximity of a_1 to r_1, a_2 could have chosen to visit r_2 directly to save a wasted trip to r_1. Thus, with this additional information available to a_2, both the agents are left satisfied: a_1 because its travel time was minimized and a_2 because it did as best as it could given all the information.

The agent-resource assignment of a_1 to r_1 and a_2 to r_2 is actually a so-called *Nash equilibrium* [33], i.e., an assignment such that no agent can unilaterally further improve its performance by deviating from the assigned resource and selecting another resource instead (with other agents keeping their selection of assigned resources). For example, in Figure 1, a_1 cannot decrease its travel time by trying to obtain resource r_2, and a_2 cannot decrease its travel time by trying to obtain resource r_1 (since a_1 is closer to r_1 than a_2).

1.4 Complete Information with Currency Exchange Scenario

The Nash equilibrium discussed in the preceding section for the complete information scenario is an equilibrium situation in which agents act selfishly for their own purposes. This stands in sharp contrast to the so-called *system optimal* assignment of agents to resources for the greater good of the entire system; instead of agents optimizing their own performances, the objective now is to optimize the *overall* performance of the system even if this requires sacrifice of individual performances of some agents. For example, in Figure 1, the system optimal assignment matches a_1 to r_2 and a_2 to r_1 with a total travel time of 70 (as opposed to 90 as in the Nash equilibrium assignment). Thus, in this assignment, a_1 sacrificed its own performance for the good of the entire system.

However, a system optimal assignment may be very difficult to achieve in practice since mobile agents move around freely and make decisions on which resources to visit without thinking of helping anyone but themselves. Nevertheless, a system optimal assignment may have important environmental implications such as that of reducing the travel time of vehicles on the road. Thus, transportation authorities are often interested in obtaining these types of assignments when it comes to mobile agents looking for these resources. Suppose that now, besides having the complete information of the system, the agents could also exchange currency. Then we could foresee a scenario in which a_2 could pay a_1 some money in exchange for the guarantee that a_1 will not visit r_1. For example, suppose that each second is worth one cent to any agent, and a_2 offers a_1 20 cents for the right to take r_1. In this case, a_1 will be even happier than before if she/he accepts because a_1 will pay the cost of driving (20 cents) but will also earn 20 cents from a_2 and, thus, a_1 breaks even and pays no cost in this situation (as opposed to 10 cents before). On the other hand, a_2 will be even happier than before by paying a total of 70 cents (50 cents for driving to r_1 and 20 cents given to a_1) as opposed to 80 cents before. Finally, now the total system costs are also minimized due to the negotiation that occurred between a_1 and a_2.

1.5 Summary of Our Contribution

In this article, we study the four possible scenarios for our search problem as mentioned in Sections 1.1–1.4. We *improve* upon the state-of-the-art (zero information) scenario by studying more refined models that are feasible based on the type of information available to the agents and the ability to exchange currency, and presenting algorithms for these models that are shown to be efficient through simulations over *real-world data*. We study these models in several contexts, such as in *game-theoretic* contexts (where the goal is to compute and study properties of a Nash equilibrium) and in the context of *datasets with missing information* in which there are either errors in the resource availability data that is received by the agents or that not all of the data is available because not every available resource can be sensed as available or unavailable.

The rest of the article is organized as follows.

—In Section 2, we summarize prior research works related to the topics studied in this article.
—In Section 3, we provide notations and preliminary definitions related to the methods and algorithms studied subsequently.
—In Section 4, we define the system optimal model and describe an algorithm for computing a system optimal assignment.
—In Section 5, we provide basic concepts and terminologies for a game theoretic setup of our problems.
—In Section 6, we discuss our results on computing a deterministic Nash equilibrium strategy profile for the complete and incomplete scenarios for our search problem.
—In Section 7, we discuss our results on designing mechanisms for resource pricing schemes that apply to the complete information with a currency exchange scenario.
—In Section 8, we present distributed algorithms for a practical setting of the incomplete information scenario by using a gravitational paradigm that navigates the agents toward resources in an efficient manner.
—In Section 9, we present algorithms in the context of uncertain and missing data about resource availability.
—In Section 10, we present our simulation methodologies that are based on real-world data, and the results of the evaluations of our algorithms.
—Finally, in Section 11 we present some concluding remarks.

2 PRIOR RELATED WORK

2.1 Sensing of Spatial Resource Availability

Approaches for monitoring and sensing available spatial objects are commonplace nowadays due in part to the proliferation of geosensors and GPS-enabled mobile devices [12]. Information about the locations of these resources can be obtained via collection of the data using client-server architectures [27] or via decentralized data dissemination [54]. In Ref. [27], remote sensing of spatial resources is presented in which users report to a server to request services (such as a taxi service). On the other hand, peer-to-peer (P2P) mobile ad hoc networks are used in Ref. [54] for spatial resource discovery in a decentralized fashion.

Our work in this article focuses on navigation toward available resources. However, our work depends on having some type of resource availability data. Approaches for parking availability detection have been presented recently as well. In Ref. [38], ultrasonic sensor technology is used to determine the spatial dimensions of open parking slots, whereas wireless sensors are used in Ref. [36] to track open parking slots in a parking facility. These two works show how one can *detect* open slots. In Ref. [31], the authors *couple* detection with the sharing of the parking slot information in a mobile sensor network by presenting a methodology for vehicles driving past curbside parking slots to detect open ones and generating a map view of parking slot availability, as opposed to having to spend on equipping each parking slot individually with wireless sensors for monitoring. Crowdsourcing methods using end-user smartphones have appeared for the problem of detecting the availability of parking spaces, because it is easy to implement and inexpensive for larger scale (such as city-wide) services. For example, by classifying mobility patterns of smartphone users [30] or using Wi-Fi signature matching [34], parking and unparking activities for individual smartphone users and, hence, vehicles can be detected. In Ref. [57], Xu et al. introduce PhonePark, a software solution for detecting parking availability in blocks by using mobile phones and detecting mobility patterns of the mobile phone users. In this article, we assume that vehicles can receive information about open parking slots at any time using one of these parking availability detection approaches.

In [6], the question of how many probe vehicles are needed to collect on-street parking data while having a minimal amount of error is answered. They use trajectories of a fleet of taxis to answer the question and propose that this data could be collected by taking advantage of on-board sensors as side-scanning ultrasonic sensors or windshield-mounted cameras (as in Ref. [31]). They, thus, show the feasibility of having access to the accurate data that we need in this work.

The *value* of having location information of spatial resources like parking was tested in a P2P environment in Ref. [25]. Kokolaki et al. show, through simulations, how vehicles with access to data about available resources have an advantage over vehicles that do not. In Ref. [47], the relevance of parking reports in a vehicular ad hoc network was studied.

Some prior work was performed on dissemination of reports of open parking slots in Ref. [55]. In Ref. [55], a parking choice algorithm was presented that selected parking slots based on a relevance metric that included the age of the open parking report. Their work assumed that a driver knew the expected time a slot would remain available from the current time, and also knew how long it would take to travel there. In our context, this is similar to an agent a knowing the probability of another agent arriving at the resource before a. This is a strong assumption that we *do not make* in this article. Furthermore, the focus of Ref. [55] was on P2P dissemination of parking reports.

2.2 Search and Allocation of Spatial Resources

Wireless ad hoc networking was also used in Ref. [50] to search for open parking slots. The authors in Ref. [50] presented an algorithm based on the time-varying Traveling Salesperson problem to

compute a tour of the open slots in order for each vehicle to search for parking in the order of the computed tour. Similar to Ref. [55], their approach depended on knowing the probability that the parking slot would still be open after some time. These approaches assume that the locations of slots are known, do not consider the missing information and errors that the availability data might have, and, in principle, can be thought of as using a framework of the so-called prize-collecting Traveling Salesperson problem [22].

In Refs [7, 10, 17], reservation systems for parking slots were studied. A centralized reservation system was presented in Ref. [7] in which a server collects information from road-side units and other vehicles and reserves slots for vehicles. In Ref. [10], the reservation system was distributed amongst peers in a vehicular ad hoc network and slots were reserved by requesting slots to a specified peer called the coordinator for each slot. In Ref. [17], a reservation system was proposed in which vehicles were matched to parking slots by solving a mixed integer linear program that optimized based on the current state of the parking information. Also, in Ref. [48], a service-oriented architecture is presented in which a server matches requests for a service or resources from users. These systems attempt to circumvent the competition for parking slots by using reservations. In our work, we design algorithms for parking that assume the parking system *is* competitive. Indeed, existing parking systems are inherently competitive rather than reservation-based.

A good survey of smart parking solutions, in various of the contexts mentioned here, can be found in Ref. [28].

In Ref. [21], the problem of matching spatial datasets is considered. In this problem, a centralized server assigns ”customers” to ”service providers” such that the total distance of the assignment is minimized. This problem is similar to the centralized problem presented in Section 4. The authors of Ref. [21] reduce their problem to the minimum cost network flow (MCNF) problem as we do. They present an efficient algorithm to solve the MCNF problem by various optimizations such as pruning the distance-based bipartite graph. Their algorithm could be used to solve the MCNF problem formulation from Section 4.

Some prior work was also performed in the setting where mobile agents are searching for static resources, but when there is uncertainty with the data about the resources. In Refs [20] and [19], a framework is presented for searching for resources when probabilities of finding the resources are available for each block of a road network. The authors present algorithms that look for an optimal path in this probabilistic setting, optimizing the probability of finding a resource. In this work, we present algorithms that work in situations where there is inherent uncertainty in the data as well. However, our uncertainty may be due to missing reports or errors in the availability of data and not due to the probabilistic nature of the available data.

2.3 Pricing of Spatial Resources

Pricing of resources to obtain some system-wide objectives, studied in this article, was also studied in the past in other contexts of transportation applications. In the transportation literature, this is commonly known as “congestion pricing” [51]. The most common type of congestion pricing is that of toll-like prices assessed on major urban areas or major roads to decrease the demand of entering to these areas and roads, and pricing strategies of a similar type have been famously implemented in the central business district of Singapore [39] and in other major cities across the world. This article investigates the pricing problems in the context of algorithmic game theory, which has a rich history (e.g., see textbooks such as Ref. [35]). In Ref. [32], the parking problem is modeled as an online bipartite matching problem in which users have private preferences. Their solutions present pricing mechanisms in the form of posted prices that seek to maximize the social welfare.

2.4 Gravitational Models

Some of our algorithms in this article are based on using gravitational force to model the attractiveness of regions with resources. Gravitational models have been employed in other computing applications that use Euclidean data. For example, in Refs [18, 56], such models were used for clustering data in the Euclidean space.

2.5 Comparison with our Prior Work on Matching

Finally, this work generalizes, subsumes, and extends our prior work on the topic of spatio-temporal resource search in Refs [1–4], on various fronts. In terms of the problem setup, our work here generalizes the work and proofs presented in those articles by allowing agents to enter the system at arbitrary times and not having all agents enter the system at the same time. This makes for a more realistic setup for modeling real-life situations. Here, then, we present updates for all of the proofs and methodologies (in Sections 3–8), which include this time component as well. Furthermore, here, we present a proof for Theorem 6.2, which was previously not presented for space considerations in Ref. [2]. This proof shows how a Nash equilibrium can be computed in our game-theoretic setup by computing a stable matching between the agents and resources. In terms of the simulation setup, the results presented in our previous work were from simulations that made use of synthetic data in two fronts: the resources and agents were synthetically generated, and the space that the agents traversed was as well. For this work, we made use of real-world data of resource availability, obtained from the SFPark database [41, 45]. We also simulated the movement of agents in a real-world road network by modeling the geographic features of the Fisherman's Wharf area in the city of San Francisco. Finally, in this work, we extend our methodologies to study how they could work in the setting where there is missing information due to errors and due to vehicles that do not report about availability of resources to our system (Section 9).

3 BASIC NOTATIONS AND PROBLEM SETUP

The general setup of our spatio-temporal matching problem is as follows:

- —Throughout the article, we will use $\mho$ to indicate a sufficiently large positive number; the precise nature of the "largeness" of $\mho$ will be specified when needed.
- —There are the following two types of *objects*:
  - —We have a set of n *agents* $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$.
  - —We have a set of m *stationary resources* $\mathcal{R} = \{r_1, r_2, \ldots, r_m\}$, and a *dummy* resource $\Xi \stackrel{\text{def}}{=} r_{m+1}$. It is commonplace to use such dummy items in algorithmic game theory applications to simplify description and case analysis, e.g., see Ref. [35, Section 11.4.3].[1]
- —The locations of the agents in $\mathcal{A}$ and the resources in $\mathcal{R}$ over all given times are points in some connected space $\mathcal{M}$. The dummy resource Ξ is *not* located in $\mathcal{M}$.
- —The following parameters are associated with each agent $a_i \in \mathcal{A}$:
  - —a *start time* $t_i^{\mathcal{A}}$, namely the time when a_i starts to look for a resource,
  - —a *location* $\ell_i^{\mathcal{A}}(t)$, namely the location of a_i at any time $t \geq t_i^{\mathcal{A}}$, and
  - —a *starting location* $\ell_i^a \stackrel{\text{def}}{=} \ell_i^{\mathcal{A}}(t_i^{\mathcal{A}})$, namely the location of a_i at the start time $t_i^{\mathcal{A}}$.
- —The following parameters are associated with each resource $r_j \in \mathcal{R} \cup \{\Xi\}$:
  - —a start time $t_j^{\mathcal{R}}$, namely the time when r_j becomes available, and
  - —a location $\ell_j^{\mathcal{R}}$ of r_j that does not change over time.

[1] The reader may think of Ξ as an artificial resource introduced for simplifying descriptions and case analysis in proofs, much in the same manner as dummy records are used as *sentinels* in data structures to simplify descriptions of algorithms that manipulate them.

For the dummy resource $\Xi \stackrel{\text{def}}{=} r_{m+1}$, $t^{\mathcal{R}}_{\Xi} \stackrel{\text{def}}{=} t^{\mathcal{R}}_{m+1} = 0$, and $\ell^{\mathcal{R}}_{\Xi} \stackrel{\text{def}}{=} \ell^{\mathcal{R}}_{m+1}$ is used only as a symbol without any value.

—A resource in $\mathcal{R}$ can be used by *at most one* agent at any given time, and the agent has to be located at the same place as the resource in order to use it. This means that an agent has to travel, for some *travel time*, to a resource to use it; and the resource has to be available for use. If agent a_i uses resource r_j, then the earliest time at which a_i starts using r_j, i.e. $\max\{t^{\mathcal{A}}_i + \text{time}(\ell^a_i, \ell^{\mathcal{R}}_j), t^{\mathcal{R}}_j\}$, is called the time when the agent *obtains* the resource. The time function is a travel time between two locations and will be defined below.

The dummy resource Ξ can be used by *any* number of agents at a specific time.

—The following three functions are used in our setup:

Travel time function time: $\mathcal{M} \times \mathcal{M} \mapsto \mathbb{R}^+$ $\text{time}(x, y)$ is the time taken by any agent to travel from location $x \in \mathcal{M}$ to location $y \in \mathcal{M}$.

For the dummy resource Ξ, we assume that $\text{time}(\ell^{\mathcal{A}}_i(t), \ell^{\mathcal{R}}_{\Xi}) = \mho$ for any agent $a_i \in \mathcal{A}$ and for any time t, where $\mho \gg \sum_{i=1}^{n} \sum_{j=1}^{m} \text{time}(a_i, r_j)$ is a *sufficiently large* positive number.

Matching function g: $\mathcal{A} \mapsto \mathcal{R}$ g is a *partial* function that is *bijective* (one-one correspondence). Essentially, $g(a_i) = r_j \in \mathcal{R}$ indicates that agent a_i is *supposed* to obtain resource r_j, and, in this case, we will often loosely use the term "*agent a_i was matched with resource r_j*" to indicate that a_i obtained or will obtain r_j.

If the partial function did *not* assign a value to $g(a_i)$ for some $a_i \in \mathcal{A}$, then we will often describe it by saying that "$g(a_i) = \Xi \stackrel{\text{def}}{=} r_{m+1}$" or by saying that "*$a_i$ was assigned to* Ξ " or also by saying that "*a_i was not assigned*".

Cost function cost: $\mathcal{A} \times \mathcal{R} \times \mathbb{R}^+ \mapsto \mathbb{R}$ $\text{cost}(a_i, r_j, t)$ denotes the *cost* incurred by an agent $a_i \in A$ to find and obtain some available resource $r_j \in R$ at time t. This cost may depend on the locations $\ell^{\mathcal{A}}_i(t)$ and $\ell^{\mathcal{R}}_j$ of a_i and r_j. The exact value of the cost is dependent on the application being considered and can be determined in a different way for different agents.[2] Some factors that may influence the cost function are:

—travel time to the resource,
—price of a resource, and
—the safety of the location of the resource.

—Broadly speaking, each agent looks to minimize her/his cost of obtaining a resource. Specific details of the objective will be spelled out in the relevant context.

We characterize this setup as a Spatio-temporal matching problem. The *spatial* component of the matching arises from the locations that the agents and resources have over the space $\mathcal{M}$. The *temporal* component of the matching comes from the instances when the agents request or start to look for the resources, from the instances when the resources become available, and from the time duration it takes for an agent to obtain a resource.

As was discussed in Section 1.4, we will study two separate methods of computing a *good* matching between agents and resources. They are differentiated by the way that the mobile agents will make decisions. In a *system optimal* setting, some centralized authority makes routing decisions for the agents. In a *Nash equilibrium* setting, agents are selfish and make their own choices. In

[2]We assume that $\text{cost}(a_i, \Xi, t) = \mho$, for a sufficiently large positive integer $\mho \gg \sum_{i=1}^{n} \sum_{j=1}^{m} \max_t \{\text{cost}(a_i, r_j, t)\} + \sum_{i=1}^{n} t^{\mathcal{A}}_i$, corresponding to the dummy resource Ξ.

the following sections, we will provide algorithms for computing these matchings in the problem setup we described above.

4 SYSTEM OPTIMAL SPATIO-TEMPORAL MATCHING (CENTRALIZED MODEL)

In this section, we give algorithms for computing a *system optimal* matching for our problem. In the context of algorithmic game theory, this approach is usually referred to as optimizing the *social welfare*. A system optimal matching is realized by a model in which a centralized authority (e.g., a transportation authority) is interested in optimizing the *total* performance of all the agents in the system. In our case, we need to compute a matching that minimizes the total costs accrued by all the agents, i.e., we seek to find a matching function g between the agents and resources that minimizes the total cost $\sum_{i=1}^{n} \mathit{cost}(a_i, g(a_i), t_i^{\mathcal{A}})$. We can easily compute such a mapping g in polynomial time by reducing our problem to an instance of the minimum-cost network flow problem on a directed bipartite graph.

Theorem 4.1. *A mapping g that minimizes $\sum_{i=1}^{n} \mathit{cost}(a_i, g(a_i), t_i^{\mathcal{A}})$ can be computed in $O((n+m)^3)$ time complexity.*

Proof. We reduce our problem to an instance of the *minimum-cost network flow* problem on a directed graph $G = (V, E)$ for which a polynomial time exact solution is well-known (e.g., see Ref. [8]). G has the following $n + m + 2$ nodes:

- a node a_i for every agent $a_i \in \mathcal{A}$,
- a node r_j for every resource $r_j \in \mathcal{R}$,
- a source node u, and
- a sink node v.

Furthermore, G has the following $nm + n + m$ directed edges:

- a directed edge (a_i, r_j) of capacity 1 and cost $\mathit{cost}(a_i, r_j)$ for every resource-agent pair $a_i \in \mathcal{A}, r_j \in \mathcal{R}$, and
- a set of $n + m$ directed edges $\{ (u, a_i) \mid 1 \le i \le n \} \cup \{ (r_j, v) \mid 1 \le j \le m \}$, each of zero cost and capacity 1.

Letting flow : $E \mapsto \mathbb{R}^+$ denote the flow function; our social welfare optimization problem is equivalent to the following minimum-cost network flow problem:

> *find a minimum-cost flow of value* $\sum_{i=1}^{n} \text{flow}(\,(u, a_i)\,) = \min\{m, n\}$ *from* u *to* v *in* G.

Since all the edge capacities are integral (0 or 1), the flow function is integral-valued (e.g., see Ref. [37]), and therefore, due to capacity constraints, flow(e) is 0 or 1 for any edge $e \in E$. The desired mapping g can now be easily computed as

$$g(a_i) = \begin{cases} r_j, \text{ if flow } ((a_i, r_j)) = 1, \text{for any } j \\ \Xi, \text{ otherwise} \end{cases}$$

□

Remark 1. Since our cost function is of a very general nature, it can model social welfare optimization in many alternate optimization objectives that are of interest in urban transportation systems. We provide three such examples below.

Minimizing total driving time. A system optimal matching that minimizes the total *driving time* of all the agents is one which a centralized authority would be interested in

because minimizing the total driving time of all agents is good for an urban transportation system (less congestion) and also for the environment (less pollution). This can be computed by setting $\mathit{cost}(a_i, g(a_i), t_i^{\mathcal{A}}) = \mathit{time}(\ell_i^a, \ell_{g(a_i)}^{\mathcal{R}})$.

Minimizing the total travel and wait time to obtain resources. We can also compute a system optimal matching that *is* affected by the arrival times of agents and availability times of resources. In this case, a central authority wants to minimize the *total* (driving plus waiting) time it takes for all the agents to *obtain* their resources. Minimizing this total time is *not* the same as just minimizing the total driving time since in the former case an agent could potentially wait for a resource to become available if waiting for that resource is more cost-effective than driving to obtain another resource that is currently available. This case can be easily captured in our formulation by setting:

$$\mathit{cost}\left(a_i, g\left(a_i\right), t_i^{\mathcal{A}}\right) = \begin{cases} t_j^{\mathcal{R}} - t_i^{\mathcal{A}}, & \text{if } t_i^{\mathcal{A}} + \mathit{time}\left(\ell_i^a, \ell_{g(a_i)}^{\mathcal{R}}\right) \leq t_j^{\mathcal{R}} \\ \mathit{time}\left(\ell_i^a, \ell_{g(a_i)}^{\mathcal{R}}\right), & \text{otherwise} \end{cases}$$

The first case in the above equation is one in which the resource will not be available to the agent upon arrival to it and she/he will have to wait some time for the resource to become available, whereas the second case is one in which the resource is already available by the time that the agent could reach it.

Minimizing the sum of the times when resources are obtained. If the central authority wants to minimize the sum of the times when the agents obtain their resources, this can be done by setting

$$\mathit{cost}\left(a_i, g\left(a_i\right), t_i^{\mathcal{A}}\right) = \begin{cases} t_j^{\mathcal{R}}, & \text{if } t_i^{\mathcal{A}} + \mathit{time}\left(\ell_i^a, \ell_{g(a_i)}^{\mathcal{R}}\right) \leq t_j^{\mathcal{R}} \\ t_i^{\mathcal{A}} + \mathit{time}(\ell_i^a, \ell_{g(a_i)}^{\mathcal{R}}), & \text{otherwise} \end{cases}$$

5 GAME THEORETIC SETUP – BASIC CONCEPTS AND TERMINOLOGIES

System optimal agent-resource assignments, as discussed in Section 4, show some desirable properties such as being efficiently computable, and may also serve a good purpose for the environment and for the welfare of all agents, but they are difficult to justify *in practice* in distributed settings. For example, a system optimal assignment in Figure 1 (a_1 assigned to r_1 and a_2 assigned to r_1) forces agent a_1 to sacrifice for the good of others since a_1 could pay lesser costs (10 instead of 20) by making its own decision rather than following the choice that is suggested by a system optimal assignment. In a distributed setting, agents make their own (selfish) choices instead of following a system optimal choice to minimize only their own costs. This type of setting can be modeled and analyzed using game theoretic concepts such as a Nash equilibrium. *Informally*, a *game* has three essential components: a set of *players*, a set of possible *strategies* (choices) for the players, and a *payoff* (cost) function [42]. The payoff function determines the cost incurred by each player for a given *strategy profile*, where a *strategy profile* refers to a vector in which the i^{th} component represents the strategy selected by the i^{th} player. *More formally*, we define the components of the game for our spatio-temporal matching problem as follows:

—The set of *players* is $\mathcal{A}$ (the set of n agents).
—In this article, we will consider only *deterministic* strategies for agents. A (deterministic) strategy for an agent a_i will be denoted by the variable $\pi_i \in \mathcal{R} \cup \{\Xi\}$ with the following convention:

$$\pi_i = \begin{cases} r_j, & \text{if agent } a_i \text{ opts for the resource } r_j \in \mathcal{R} \\ \Xi, & \text{if agent } a_i \text{ does not opt for any resource} \\ & \text{or equivalently } a_i \text{ opts for the dummy resource } \Xi \end{cases}$$

The vector of variables $\overrightarrow{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ is called a *strategy profile vector*; a specific *strategy profile* is obtained from this vector by assigning a specific value to each π_j for $1 \leq j \leq n$.

—The *payoffs* (costs) for each player (agent) in this game can be defined by the $cost$ function introduced in Section 3. Let $\mathcal{S} = (\pi_1, \pi_2, \ldots, \pi_n) \in (\mathcal{R} \cup \{\Xi\})^n$ be a strategy profile chosen by the players, i.e., $\pi_i \in \mathcal{R} \cup \{\Xi\}$ is the chosen resource by agent a_i for $1 \leq i \leq n$. Then, the cost (payoff) to the player a_i corresponding to this strategy profile $\mathcal{S}$, which we will denote by $cost_{\mathcal{S}}(a_i)$, is as follows:

—$cost_{\mathcal{S}}(a_i) = cost(a_i, \pi_i, t_i^{\mathcal{A}})$ if and only if agent a_i would obtain the resource π_i *before*[3] any other agent a_j that also opted for this resource, i.e., for any a_j such that $\pi_j = \pi_i$.

—Otherwise, $cost_{\mathcal{S}}(a_i) = \mho$ for a sufficiently large positive number $\mho \gg \sum_{i=1}^{n} \sum_{j=1}^{m} cost(a_i, r_j, t_i^{\mathcal{A}})$.

For notational convenience, let $cost_{\mathcal{S}} = \sum_{i=1}^{n} cost_{\mathcal{S}}(a_i)$.

The Nash equilibrium [33] is a standard desired strategy profile that is used to model the stability of individual choices of players in a game. In such a strategy profile, no player can further improve its performance by changing its own strategy unilaterally. For a strategy profile $\mathcal{S} = (\pi_1, \pi_2, \ldots, \pi_n)$, let $\mathcal{S}_{-i}$ denote the set of strategy profiles $\{ (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_\alpha, \pi_{i+1}, \ldots, \pi_n) \mid \pi_\alpha \neq \pi_i \}$. The standard definition of a *deterministic* Nash equilibrium translates to the following definition for our spatio-temporal matching problem.

Definition 5.1 (Deterministic Nash equilibrium for our spatio-temporal matching problem). A strategy profile $\mathcal{S}$ is a Nash equilibrium strategy profile if and only if the following holds:

$$\forall\, 1 \leq i \leq n \; \forall \mathcal{T} \in \mathcal{S}_{-i} \; : \; cost_{\mathcal{S}}(a_i) \leq cost_{\mathcal{T}}(a_i)$$

For the remainder of the article, *equilibrium* and *Nash equilibrium* will be used interchangeably. A Nash equilibrium strategy profile stands in sharp contrast to a system optimal matching because of the distributed selfish nature of the mobile agents in real-world transportation applications. Indeed, agents in transportation applications act according to their own self-interests.

6 COMPUTATION OF A DETERMINISTIC NASH EQUILIBRIUM STRATEGY PROFILE

In this section, we provide our results related to computing a deterministic Nash equilibrium for the game theoretic setup of our spatio-temporal matching problems as described in the previous section.

6.1 Complete Information Scenario

The complete information scenario was described in Section 1.3. In terms of the general setup of our spatio-temporal matching problem described in Section 3, this scenario ensures that each agent has knowledge of all the parameters of the setup (including those belonging to other agents) as described in the setup. Note that this framework is more general than the framework in our previous research works in [1, 2] because of introduction of parameters such as $t_i^{\mathcal{A}}$, $\ell_i^{\mathcal{A}}(t)$ and $t_j^{\mathcal{R}}$. These

[3]If a resource is available and two or more agents are at distance zero from the resource, then we assume that the agent with the earliest arrival time obtains the resource before the others; and if two or more agents have the same arrival time at the resource, then the agent with the smallest index obtains the resource before the others.

new parameters could be especially useful for spatio-temporal matching applications that have reserved pickups or reservation times for resources; e.g., in the application involving taxicabs and clients mentioned before, a client (resource) $r_j \in \mathcal{R}$ could ask a cab for pickup at a *specific* time $t_j^{\mathcal{R}}$.

We show below how to compute a deterministic Nash equilibrium strategy profile for this complete information scenario in *polynomial time*. For this purpose, we will use the well-known concept of a stable matching.

Definition 6.1 (Stable matching between agents and resources). Assume that each agent $a_i \in \mathcal{A}$ has an associated strict total order relation $\prec_{a_i} \stackrel{\text{def}}{=} r_{i_1} \prec_{a_i} r_{i_2} \prec_{a_i} \cdots \prec_{a_i} r_{i_m} \prec_{a_i} \Xi$ over $\mathcal{R} \cup \{\Xi\}$ (the "preference list" for agents), and, similarly, each resource $r \in \mathcal{R} \cup \{\Xi\}$ has an associated strict total order $\prec_r \stackrel{\text{def}}{=} a_{j_1} \prec_r a_{j_2} \prec_r \cdots \prec_r a_{j_n}$ over $\mathcal{A}$ (the "preference list" for resources). The matching function g defines a stable matching[4] between agents and resources if and only if there are no two matched pairs $(a_i, g(a_i))$ and $(a_j, g(a_j))$ such that *both* of the following conditions hold:

—a_j prefers $g(a_i)$ over $g(a_j)$, i.e., $g(a_i) \prec_{a_j} g(a_j)$, and
—$g(a_i)$ prefers a_j over a_i, i.e., $a_j \prec_{g(a_i)} a_i$.

Any matching function g defines a natural strategy profile $\mathcal{S}^g = (\pi_1^g, \pi_2^g, \ldots, \pi_n^g)$ where $\pi_i^g = g(a_i)$. The Gale-Shapley deferred acceptance algorithm in Ref. [16] can be used to compute a stable matching between agents and resources in polynomial time. To use the above concept of a stable matching, we then need to define preference lists for all the agents and resources. This is done as follows.

—(*agent preference lists*) For later notational convenience, let $x_{a_i,r} = \mathit{cost}(a_i, r, t_i^{\mathcal{A}})$ for $r \in \mathcal{R} \cup \{\Xi\}$. The preference list $\prec_{a_i} \stackrel{\text{def}}{=} r_{i_1} \prec_{a_i} r_{i_2} \prec_{a_i} \cdots \prec_{a_i} r_{i_m} \prec_{a_i} \Xi$ for agent a_i is defined by

$$\forall\, i_s \in \{i_1, i_2, \ldots, i_{m-1}\} \,:\, r_{i_s} \prec_{a_i} r_{i_{s+1}} \;\equiv\; x_{a_i, r_{i_s}} \leq x_{a_i, r_{i_{s+1}}}$$

Intuitively, this means that each agent orders the resources increasingly by their cost.

—(*resource preference lists*) Let $y_{a_i,r_j} = t_i^{\mathcal{A}} + \mathit{time}(\ell_i^a, \ell_j^{\mathcal{R}})$ be the earliest possible time when an agent $a_i \in \mathcal{A}$ may arrive to obtain resource $r_j \in \mathcal{R}$ if she/he chooses it as a strategy. Then, the preference list $\prec_{r_j} \stackrel{\text{def}}{=} a_{j_1} \prec_{r_j} a_{j_2} \prec_{r_j} \cdots \prec_{r_j} a_{j_n}$ for a resource $r_j \in \mathcal{R}$ is defined by

$$\forall\, j_s \in \{j_1, j_2, \ldots, j_{n-1}\} \,:\, a_{j_s} \prec_{r_j} a_{j_{s+1}} \;\equiv\; y_{a_{j_s}, r_j} \leq y_{a_{j_{s+1}}, r_j}$$

Intuitively, this means that each resource orders the agents increasingly by the earliest time they could obtain the resource.

The preference list $\prec_\Xi$ for the dummy resource Ξ is simply the n agents in $\mathcal{A}$ listed in an arbitrary order.

We can now prove the following result.

Theorem 6.2 (Stable matching and Nash equilibrium). *If g is a stable matching between agents and resources with the agent preferences determined by the ordered agent preference lists $\prec_{a_i}$'s and the resource preferences determined by the ordered resource preference lists $\prec_{r_j}$'s and $\prec_\Xi$, then the strategy profile $\mathcal{S}^g$ is a Nash equilibrium strategy profile.*

Proof. Let g be a stable matching between agents and resources with the agent preferences determined by the ordered agent preference lists $\prec_{a_i}$'s and the resource preferences determined

[4]Recall that $g(a_i) = \Xi$ if a_i was not matched to any resource.

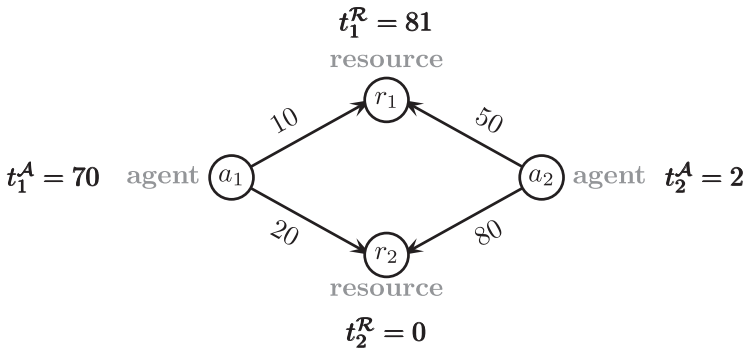

Fig. 2. An example of two mobile agents a_1 and a_2 and two stationary resources r_1 and r_2. The numbers on edges represent (in seconds) the travel times for the agents to reach the corresponding resources. $t_1^{\mathcal{A}}, t_2^{\mathcal{A}}, t_1^{\mathcal{R}}$, and $t_2^{\mathcal{R}}$ are the start times for agent a_1, agent a_2, resource r_1, and resource r_2, respectively.

by the ordered resource preference lists $\prec_{r_j}$'s and $\prec_{\Xi}$. Note that $g^{-1}(r_j)$ is defined for every resource $r_j \in \mathcal{R}$ in any stable matching.

We prove the result by contradiction. Suppose, for contradiction, that $\mathcal{S}^g = (\pi_1^g, \pi_2^g, \ldots, \pi_n^g)$ is *not* a Nash equilibrium strategy profile. Then, there exists an index i and a strategy $\mathcal{T} = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_\alpha, \pi_{i+1}, \ldots, \pi_n) \in \mathcal{S}_{-i}^g$ such that $\mathit{cost}_{\mathcal{S}}(a_i) > \mathit{cost}_{\mathcal{T}}(a_i)$. Note that $\mathit{cost}_{\mathcal{S}}(a_i) > \mathit{cost}_{\mathcal{T}}(a_i)$ implies $\pi_\alpha \neq \Xi$ and also implies that agent a_i must be able to obtain the resource π_α before any other agent that also opted for this resource. We now show below that the existence of $\mathcal{T}$ violates the conditions of a stable matching.

—Since agent a_i obtains π_α before any other agent that also opted for this resource, $y_{a_i,\pi_\alpha} < y_{g^{-1}(\pi_\alpha),\pi_\alpha}$, which in turn implies

$$a_i \prec_{g(g^{-1}(\pi_\alpha))} g^{-1}(\pi_\alpha) \tag{1}$$

—Since $\mathit{cost}_{\mathcal{T}}(a_i) < \mathit{cost}_{\mathcal{S}}(a_i)$, $x_{a_i,\pi_\alpha} < x_{a_i,\pi_i}$, which in turn implies

$$g\left(g^{-1}(\pi_\alpha)\right) = \pi_\alpha \prec_{a_i} \pi_i = g(a_i) \tag{2}$$

If we consider the matched pairs $(a_i, g(a_i))$ and $(g^{-1}(\pi_\alpha), \pi_\alpha)$, Relations (1) and (2) provide a violation of the stability of the matching (c.f. Definition 6.1). □

Examples and Discussion. Here we demonstrate the use of the last theorem and the fact that the introduction of different starting times change the Nash equilibrium matching. Assume that the cost function of an agent is the earliest time when it obtains a resource. Consider again the example shown in Figure 1, which has no start times for the agents and resources (or, equivalently, all the start times are zero). A system optimal matching and a Nash equilibrium strategy profile for this example can be computed by using the algorithms in Ref. [1]. They are as follows:

—A system optimal assignment matches a_1 to r_2 and a_2 to r_1 with a total travel time of 70.
—As observed before in Section 1.3, a Nash equilibrium strategy profile can be obtained by a_1 opting for r_1 and a_2 opting for r_2 with a total travel time of 90. The corresponding matching function g_{ne} is given by $g_{ne}(a_1) = r_1$ and $g_{ne}(a_2) = r_2$.

Now, let us consider the same example but with start times for all the objects. Figure 2 shows the new example. Assume that the objective of each agent is to minimize the time when it obtains a

resource, i.e.,

$$\mathit{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right) = \begin{cases} t_j^{\mathcal{R}}, & \text{if } (t_j^{\mathcal{R}} - t_i^{\mathcal{A}}) \geq \mathit{time}(\ell_i^a, \ell_j^{\mathcal{R}}) \\ t_i^{\mathcal{A}} + \mathit{time}(\ell_i^a, \ell_j^{\mathcal{R}}), & \text{otherwise} \end{cases}$$

To compute a Nash equilibrium strategy profile, we first need to compute the x_{a_i,r_j} and y_{a_i,r_j} values used in fixing the agent and resource preference lists (since the number of agents is equal to the number of resources, calculations involving the dummy resource Ξ is not needed). These values are as follows:

$$x_{a_1,r_1} = 81 \quad x_{a_1,r_2} = 90 \quad x_{a_2,r_1} = 81 \quad x_{a_2,r_2} = 82$$
$$y_{a_1,r_1} = 80 \quad y_{a_1,r_2} = 90 \quad y_{a_2,r_1} = 52 \quad y_{a_2,r_2} = 82$$

It can be verified that the only matching that is stable according to these values is the one that matches a_1 with r_2 and a_2 with r_1. It can be easily verified that none of the agents has an incentive to deviate from their choices:

—If a_1 deviates to opt for r_1, she/he will not obtain it because a_2 will obtain it first.
—If a_2 deviates to opt for r_2, she/he will obtain it a one time unit later than before so she/he has no incentive to deviate.

The corresponding new matching function g'_{ne} is given by $g'_{ne}(a_1) = r_2$ and $g'_{ne}(a_2) = r_1$. We can, thus, see how adding start times for the objects changes the Nash equilibrium matching.

A greedy algorithm to compute a Nash equilibrium strategy profile (i.e., the function g_{ne}) in the static context (i.e., with identical start times) was presented in Ref. [1]. The algorithm sorts the weights (travel times) of the edges in the bi-partite graph of agents and resources, and matches agents and resources in that order. A straightforward adaptation of such an algorithm would not work in the dynamic context, i.e., with start times, regardless of the cost function used.

6.2 Incomplete Information Scenario

In the game theoretic formulation of this scenario, the agents do not have parametric information to compute the payoff function. This means that an agent does not know what its payoff will be in a given strategy profile since, for example, it does not know the locations and start times of the other agents.

In Ref. [1], we introduced a formulation in which each agent makes probabilistic assumptions about the locations and start times of the other agents in the game and the analysis done based on the expected performances. In other words, this scenario leads to computing the Nash equilibrium strategy profiles in a Bayesian setting, where the selfish goal of an agent is to minimize its *expected* cost conditional on the above-mentioned prior distributions. In Ref. [1], we showed how to calculate the Nash equilibrium for a simple special case in which agents move on a straight line and all starting times are ignored.

7 DESIGNING PRICING MECHANISMS FOR COMPLETE INFORMATION WITH CURRENCY EXCHANGE SCENARIO

In the previous section, we sought to compute a Nash equilibrium strategy profile assuming that the agents in transportation applications are *inherently* selfish and seek to optimize their own costs only. However, as we also have observed before in Section 6.1 and elsewhere, this is sub-optimal for the transportation system as a whole. And in some applications, sub-optimality can have major societal and environmental implications.

Take our parking application, for example. Cruising for parking by driving around an urban area looking for available parking slots has been shown to be a *major* cause of congestion in urban

areas. For example, in Ref. [43], studies conducted in 11 major cities revealed that the average time to search for curbside parking was 8.1 minutes, and cruising for these parking slots accounted for 30% of traffic congestions in those cities. This means that each parking slot would generate 4,927 vehicle miles traveled (VMT) per year [44]. For example, in a big urban city like Chicago with over 35,000 curbside parking slots [49], the total number of VMT becomes 172 million VMT per year due to cruising while searching for parking. Furthermore, this would account for a waste of 8.37 million gallons of gasoline, and over 129,000 tons of CO_2 emissions.

To reduce these environmental costs, it would be great to have agents *guided* toward system optimal matchings. However, as has been discussed before, this is difficult to justify in practice because of the individual costs that some agents could end up sacrificing. Furthermore, computing a system optimal matching requires complete information, but agents do not necessarily have incentives to share information about their locations (for privacy concerns or other reasons). Therefore, our aim in this section is to combat the problem in a different way. The central question that we wish to tackle can be described informally as:

> *Can we propose a pricing scheme on the resources that will incentivize the agents to move in a system optimal manner?*

7.1 Mechanism Design for Agent-Independent Resource Pricing Scheme

In this section, we address the following question: How can we impose an agent-independent monetary toll on using the resources such that the Nash equilibrium matching, when considering the toll cost (i.e., the original cost plus the toll), is a system optimum (or as close as possible to it) when considering the cost alone. Doing so would imply that selfish agents in an anarchical system would naturally settle into a system optimum matching, regardless of how the non-toll cost is defined. By agent-independence, we mean that the toll imposed on a resource is the same, regardless of the agent using the resource. This is similar to a metered parking slot, which has the same price for any driver that uses it and different from the tolls discussed in the next section.

Let g^{opt} be a system optimal matching function that gives a system optimal total cost of value OPT, i.e.,

$$\text{OPT} = \sum_{i=1}^{n} \mathit{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right) = \min_{g:\mathcal{A}\mapsto\mathcal{R}} \left\{ \sum_{i=1}^{n} \mathit{cost}\left(a_i, g\left(a_i\right), t_i^{\mathcal{A}}\right) \right\}$$

An *agent-independent resource pricing scheme* is a vector $\mathcal{P} = (p_1, p_2, \ldots, p_m)$ where p_i is the extra price for resource $r_j \in \mathcal{R}$ that any agent must pay to use it, i.e., with the pricing scheme introduced, the cost of agent a_i to obtain resource r_j is now modified to

$$\mathit{cost}_{\mathcal{P}}\left(a_i, r_j, t_i^{\mathcal{A}}\right) = \mathit{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right) + p_j$$

Ideally, we would like to compute a matching function $g_{\mathcal{P}} : \mathcal{A} \mapsto \mathcal{R}$ that matches each agent a_i to a resource $g_{\mathcal{P}}\left(a_i\right)$ such that:

$$\mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) = \min_{1\le k\le m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} \tag{3}$$

If this is possible, then agent a_i would have no incentive to deviate to another strategy, and, thus, this strategy would be a Nash equilibrium strategy for a_i. If, furthermore, this condition holds for *all* agents with their matched and obtained resources, then such a matching is indeed a Nash equilibrium strategy profile. In addition, we, of course, would like to *bound the difference* between the sum of the costs of all the agents in this new matching $g_{\mathcal{P}}$ and the original system optimal cost

OPT; i.e., we would like to find a Δ (the smaller the better) such that

$$\sum_{i=1}^{n} \mathit{cost}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \mathrm{OPT} + \Delta \tag{4}$$

If, for example, the cost function cost reflects a measure of environmental pollution created by driving, then the above bound indicates that the pricing scheme makes sure that the new level of pollution increases by at most Δ above the minimum possible.

Then, as is the norm in the algorithms community, we are led to investigate a *bi-criteria* approximation of these two goals. For this purpose, we consider a relaxed version of Equation (3) by introducing the notion of a ε-approximate equilibrium in a manner similar to that in the algorithmic game theory community (e.g., see Refs. [13, 24, 29]). Let $\varepsilon > 0$ be a positive number. Then, define an agent a_i as being in *ε-almost at equilibrium* or being in an *ε-approximate equilibrium* provided

$$\mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \min_{1 \leq k \leq m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon \tag{3'}$$

For a given pricing scheme, we say that an agent-resource matching is ε-almost at equilibrium (or is at ε-approximate equilibrium) when the above Equation (3′) holds for *all* agents with their matched and obtained resources. For notational convenience, let $\mu = \max_{i,j}\{\mathit{cost}(a_i, r_j, t_i^{\mathcal{A}})\}$. We prove the following result.

Theorem 7.1. *For every $\varepsilon > 0$, we can compute an agent-independent pricing scheme $\mathcal{P} = (p_1, p_2, \ldots, p_m)$ and a matching function $g_{\mathcal{P}} : \mathcal{A} \mapsto \mathcal{R}$ in $O\left(n^2\mu/\varepsilon\right)$ time that satisfy both the following:*

(♠) *$g_{\mathcal{P}}$ induces a ε-approximate equilibrium,* i.e.,

$$\mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \min_{1 \leq k \leq m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon$$

(♠♠) $\sum_{i=1}^{n} \mathit{cost}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \mathrm{OPT} + n\,\varepsilon.$

Proof. Our algorithm is based on the auction algorithm in Ref. [5]. The algorithm executes in "rounds" or iterations starting with an arbitrary initial matching. We assume that we start with all prices set to 0. In this discussion, we assume that $m = n$; however, as Ref. [5] indicates, this requirement can be relaxed. There is a matching and a set of prices at the end of each round. If all the agents are at ε-almost equilibrium with their matched resources at the end of any round, then the algorithm terminates. Otherwise, an agent that is *not* ε-almost at equilibrium, say agent a_i, is selected. Let r_j be the resource that has minimal cost for a_i, i.e., let

$$\alpha = \mathit{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right) + p_j = \min_{1 \leq k \leq m} \left\{ \mathit{cost}\left(a_i, r_k, t_i^{\mathcal{A}}\right) + p_k \right\}$$

and let r_q (with $q \neq j$) be the resource that has *second* minimal cost for a_i (the resource second most preferred by a_i), i.e., let

$$\beta = \mathit{cost}\left(a_i, r_q, t_i^{\mathcal{A}}\right) + p_q = \min_{\substack{1 \leq k \leq m \\ k \neq j}} \left\{ \mathit{cost}\left(a_i, r_k, t_i^{\mathcal{A}}\right) + p_k \right\} \text{ for some } q \neq j$$

Then, the following steps are executed:

—a_i exchanges slots with the agent assigned to r_j at the beginning of the next round.
—a_i increases the price of his/her best resource r_j from the current value of p_j to the new value $p_j + (\beta - \alpha) + \varepsilon$. Basically, $\beta - \alpha + \varepsilon$ is the highest value to which r_j's price can be

(∗ initialization ∗)

set $p_j \leftarrow 0$ for all $1 \le j \le m$

let $g_{\mathcal{P}} : \mathcal{A} \mapsto \mathcal{R}$ be an arbitrary agent-resource matching

(∗ rounds of auctions ∗)

while *not all agents are at* ε*-almost equilibrium* **do**

let i be an index such that $\mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}(a_i), t_i^{\mathcal{A}}\right) > \min\limits_{1 \le k \le m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon$

(∗ compute bid increment ∗)

let $\alpha \leftarrow \mathit{cost}_{\mathcal{P}}\left(a_i, r_j, t_i^{\mathcal{A}}\right) = \min_{1 \le k \le m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\}$

let $\beta \leftarrow \mathit{cost}_{\mathcal{P}}\left(a_i, r_q, t_i^{\mathcal{A}}\right) = \min_{\substack{1 \le k \le m \\ k \ne j}} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\}$ for some $q \ne j$

set $p_j \leftarrow p_j + (\beta - \alpha) + \varepsilon$

(∗ reassign resources ∗)

set $g_{\mathcal{P}}\left(g_{\mathcal{P}}^{-1}(r_j)\right) \leftarrow g_{\mathcal{P}}(a_i)$

set $g_{\mathcal{P}}(a_i) \leftarrow r_j$

endwhile

(∗ output ∗)

return $\mathcal{P} = (p_1, p_2, \ldots, p_m)$ and $g_{\mathcal{P}}$ as the solution

Fig. 3. The algorithm for pricing resources in Theorem 7.1 based on the auction algorithm in Ref. [5].

increased while still being a_i's preferred resource in an ε-almost equilibrium. Note that $\beta - \alpha + \varepsilon \ge \varepsilon$.

This algorithm continues in a sequence of rounds until all agents are at ε-almost equilibrium. The complete algorithm is shown in Figure 3. The iterative approach can be viewed as an *auction* where a_i raises the price of his/her bid on resource r_j by the bidding increment $\beta - \alpha + \varepsilon$.

An analysis similar to that in Ref. [5] shows that the auction algorithm in Figure 3 is guaranteed to terminate in $O\left(n\,\mu/\varepsilon\right)$ rounds, and a naive implementation of each round gives a total running time of $O\left(n^2\mu/\varepsilon\right)$. Since the algorithm terminates when *all* agents are ε-almost at equilibrium, it also computes an ε-approximate equilibrium for the agent-resource matching induced by $g_{\mathcal{P}}$, which proves condition (♠). Thus, it only remains to prove condition (♠♠). Using condition (♠) of the auction algorithm and simple algebraic manipulation, we get

$$\begin{aligned}
&\forall\, i : \mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}(a_i), t_i^{\mathcal{A}}\right) \le \min\nolimits_{1 \le k \le m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon \\
\Longrightarrow\ &\forall\, i : \mathit{cost}\left(a_i, g_{\mathcal{P}}(a_i), t_i^{\mathcal{A}}\right) + p_{g_{\mathcal{P}}(a_i)} \le \mathit{cost}\left(a_i, g^{\mathrm{opt}}(a_i), t_i^{\mathcal{A}}\right) + p_{g^{\mathrm{opt}}(a_i)} + \varepsilon \\
\Longrightarrow\ &\sum_{i=1}^{n} \left[\mathit{cost}\left(a_i, g_{\mathcal{P}}(a_i), t_i^{\mathcal{A}}\right) + p_{g_{\mathcal{P}}(a_i)} \right] \le \sum_{i=1}^{n} \left[\mathit{cost}\left(a_i, g^{\mathrm{opt}}(a_i), t_i^{\mathcal{A}}\right) + p_{g^{\mathrm{opt}}(a_i)} + \varepsilon \right] \\
\Longrightarrow\ &\sum_{i=1}^{n} \mathit{cost}\left(a_i, g_{\mathcal{P}}(a_i), t_i^{\mathcal{A}}\right) \le \sum_{i=1}^{n} \left[\mathit{cost}\left(a_i, g^{\mathrm{opt}}(a_i), t_i^{\mathcal{A}}\right) \right] + n\,\varepsilon = \mathrm{OPT} + n\,\varepsilon \\
&\qquad\qquad \text{since} \quad \sum_{i=1}^{n} p_{g_{\mathcal{P}}(a_i)} = \sum_{i=1}^{n} p_{g^{\mathrm{opt}}(a_i)} = \sum_{j=1}^{n} p_j
\end{aligned}$$

This proves (♠♠) and concludes the proof. □

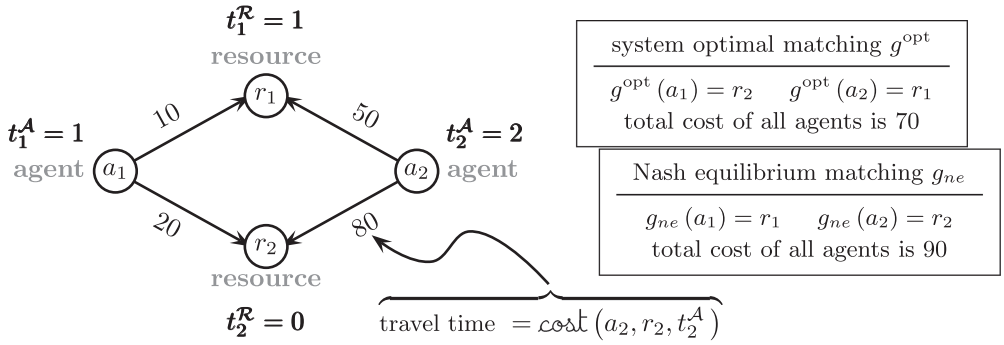

Fig. 4. An example of two mobile agents a_1 and a_2 and two stationary resources r_1 and r_2. The numbers on edges represent the travel times for the agents to reach the corresponding resources. *Assume that these travel times also represent the costs for the agents to obtain the corresponding resources.*

Remark 2 (Primal-dual interpretation of the auction algorithm). Readers familiar with the primal-dual approach for solving linear programs by iteratively satisfying complementary slackness conditions [8] will realize that the auction algorithm in Figure 3 can be interpreted as a primal-dual schema in the following manner: start with a feasible (not necessarily optimal) solution of the dual linear program for the spatio-temporal matching problem [35, Section 11.3.1] by, for example, setting all the dual variable (prices) to zeros and iteratively increasing dual variables until all complementary slackness conditions are satisfied.

Examples and Discussion. Now we demonstrate the pricing algorithm. Consider the example shown in Figure 4. Successive rounds of the algorithm in Figure 3, with all starting prices at 0 and with the initial matching as the system optimal matching shown in Figure 4, are as follows:

Initialization (before round 1)	$p_1 = 0$	$p_2 = 0$	$g_{\mathcal{P}}(a_1) = r_2$	$g_{\mathcal{P}}(a_2) = r_1$
after round 1	$p_1 = 10 + \varepsilon$	$p_2 = 0$	$g_{\mathcal{P}}(a_1) = r_1$	$g_{\mathcal{P}}(a_2) = r_2$
after round 2 (final round)	$p_1 = 30 + \varepsilon$	$p_2 = 0$	$g_{\mathcal{P}}(a_1) = r_2$	$g_{\mathcal{P}}(a_2) = r_1$

Thus, at the conclusion of the algorithm, r_1 has a price of $30 + \varepsilon$ and r_2 is free. Indeed, it is easy to see that for these prices, the equilibrium matching when considering the prices and costs (a_1 obtains r_2, and a_2 obtains r_1) is a system optimum when considering the costs alone.

The reader may wonder if it is possible to set $\varepsilon = 0$ in the algorithm in Figure 3. Unfortunately, it is easy to adopt an example from Ref. [5] to show that the algorithm will run forever for this example input if $\varepsilon = 0$.

7.2 Mechanism Design for Agent-Dependent Resource Pricing Scheme

The agent independent pricing scheme has a disadvantage in the sense that the agents are not incentivized to participate in the scheme, since the resource prices constitute an additional tax. In this section, we remedy the situation by showing how to impose the tax, and refund it, in a way that guarantees that each agent is better off than in an anarchical system.

To achieve this, the resource prices imposed are *agent-dependent*. This means that a resource price (or toll) may depend on the agent that obtains the resource. In contrast, in the agent-independent previous scheme discussed in the previous section, the price of a resource does *not* depend on the agent that was matched to it.

Thus, an agent-dependent pricing scheme can be denoted by $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n)$, where each $\mathcal{P}_i$ for $1 \leq i \leq n$ is a vector $(p_{i,1}, p_{i,2}, \ldots, p_{i,m})$, and each $p_{i,j}$ represents the price that agent a_i

would have to pay to obtain r_j. The prices in this scheme will again be designed to incentivize agents into making resource choices in a system optimal manner. Similar to the previous section, with the agent-dependent pricing scheme introduced, the cost of agent a_i to obtain resource r_j is now modified to

$$\mathit{cost}_{\mathcal{P}}\left(a_i, r_j, t_i^{\mathcal{A}}\right) = \mathit{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right) + p_{i,j}$$

We use the following notations:

—$g^{\text{opt}} : \mathcal{A} \mapsto \mathcal{R}$ denotes a system optimal matching function that gives a system optimal total cost of value OPT *before* pricing, i.e.,

$$\text{OPT} = \sum_{i=1}^{n} \mathit{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right) = \min_{g:\mathcal{A}\mapsto\mathcal{R}} \left\{ \sum_{i=1}^{n} \mathit{cost}\left(a_i, g\left(a_i\right), t_i^{\mathcal{A}}\right) \right\}$$

—$g_{ne} : \mathcal{A} \mapsto \mathcal{R}$ denotes a Nash equilibrium matching function *before* pricing.

The Pricing Scheme

Let $\mho \gg \sum_{i=1}^{n} \sum_{j=1}^{m} \mathit{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right)$ be a sufficiently large number. Given a Nash equilibrium matching g_{ne}, consider the following pricing scheme $\mathcal{P}$:

$$p_{i,j} = \begin{cases} \max\left\{0,\ \mathit{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right)\right\}, & \text{if } g^{\text{opt}}\left(a_i\right) = r_j \\ \mho, & \text{otherwise} \end{cases} \tag{5}$$

We can immediately observe that the pricing given by Equation (5) has the following desirable property.

Lemma 7.2. *g^{opt} is a Nash equilibrium matching function when pricing is used, i.e., when $\mathit{cost}_{\mathcal{P}}$ is used as the cost function.*

Proof. Straightforward; since $\mho$ is sufficiently large, each a_i has no incentive to choose another resource over $g^{\text{opt}}\left(a_i\right)$. □

Compensation for Unhappy Agents by the Pricing Authority

Two important consequences for the pricing scheme in Equation (5) are the following:

Happy agents: If $\mathit{cost}(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}) \geq \mathit{cost}(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}})$, then agent a_i has to pay an extra cost of $\mathit{cost}(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}) - \mathit{cost}(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}})$ in addition to the original cost $\mathit{cost}(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}})$. Thus, the total cost, namely the sum of these two costs, incurred by a_i is exactly $\mathit{cost}(a_i, g_{ne}(a_i), t_i^{\mathcal{A}})$, which is the cost (before pricing) that it would have paid originally with the assignment g_{ne}. We call such an agent a *happy* agent since she/he does not pay any extra amount.

Unhappy agents: However, if $\mathit{cost}(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}) < \mathit{cost}(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}})$, then agent a_i gets its resource at no cost, but it ends up paying $\mathit{cost}(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}})$, which is strictly more than the cost (before pricing) $\mathit{cost}(a_i, g_{ne}(a_i), t_i^{\mathcal{A}})$ that it would have paid originally with the assignment g_{ne}. We call such an agent an *unhappy* agent.

We now explain how such an unhappy agent can be compensated. Consider the following compensation method:

(★) *The (central) pricing authority pays back to each unhappy agent a_i the amount of $\mathit{cost}(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}) - \mathit{cost}(a_i, g_{ne}(a_i), t_i^{\mathcal{A}})$.*

Then, agent a_i will end up paying a total of

$$\mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) - \left(\mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)\right) = \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right),$$

which is precisely the cost a_i would have paid originally with the Nash equilibrium matching g_{ne}. This, therefore, would make a_i a happy agent.

Thus, it follows that with this reimbursement scheme, all the agents will be happy because they will simply pay the same cost that they would have paid originally with the Nash equilibrium matching. The only question that is left to be answered is *if the pricing authority will also make a profit as well.* The following lemma answers this question in the affirmative.

Lemma 7.3. *The agent-dependent pricing scheme in Equation* (5) *along with the reimbursement of costs by the pricing authority as described in* (★) *yields a* non-negative *total profit for the pricing authority.*

Proof. Using simple algebraic manipulations, the total profit (payment received minus payment made) for the pricing authority can be ultimately written as:

$$\begin{aligned}
&\sum_{i:\, \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) > \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)\right] \\
&\qquad - \sum_{i:\, \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) \geq \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)\right] \\
&= \sum_{i:\, \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) > \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)\right] \\
&\qquad + \sum_{i:\, \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) \geq \mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)\right] \\
&= \sum_{i=1}^{n} \left[\mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) - \mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)\right] \\
&= \sum_{i=1}^{n} \left[\mathit{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)\right] - \sum_{i=1}^{n} \left[\mathit{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)\right] \geq 0,
\end{aligned}$$

where the last inequality follows from the fact that a system optimal matching has a total minimum cost. □

Remark 3. Thus, based on the results in Lemma 7.3 and 7.2, the pricing authority could act as a broker in the following sense:

- —it collects payments from the agents that incur a lower cost (compared to equilibrium) in a system optimal matching,
- —it pays agents that incur a higher cost in a system optimal matching, and
- —it makes a profit.

Since the agents pay exactly what they would have paid originally with the Nash equilibrium and the pricing authority is also making a profit, everybody is "happy." Furthermore, the pricing authority could potentially make the agents even "happier" by splitting a fraction of the profits with them and still keeping the other fraction of the profits for itself. In this case, each agent will pay a total cost that is *lower* than the one that they would have paid by being selfish.

Example. Consider, again, the example in Figure 4 but with our new agent-dependent pricing scheme. The following prices are obtained by the scheme:

$$p_{1,1} = \mho \quad p_{1,2} = 0 \quad p_{2,1} = 30 \quad p_{2,2} = \mho$$

This means that the pricing authority will charge $80 - 50 = 30$ to a_2 for r_1, pays back $20 - 10 = 10$ to a_1, and makes a net profit of $30 - 10 = 20$ as a broker. This net profit of 20 could potentially be used to further compensate the agents for driving optimally by giving them a fraction of it.

7.3 Some Remarks on Trust Issues and Strategy-proof Mechanism Designs

The pricing schemes presented in the previous two sections fulfill the purpose for which they are designed by incentivizing agents to move in a system optimal manner. However, they do have the following shortcoming:

> *When using these schemes, agents may have an incentive to lie to the pricing authorities about their costs and locations.*

Consider, again, the example in Figure 4. If a_2 lies to the pricing authority about its start time $t_2^{\mathcal{A}}$ or its initial location ℓ_i^a such that $\mathit{cost}(a_2, r_1, t_2^{\mathcal{A}})$ decreases, then the pricing authority will still think that the system optimal matching will be to send a_2 to r_1 as before, but it will actually charge the agent a *lesser* price than before. Then the agents can lie to possibly pay less or to possibly obtain resources that they would otherwise not obtain.

To circumvent these types of problems, these schemes, therefore, need to be accompanied with some assurances about trust in the location/costs reported by the agents. Finding such pricing schemes that are strategy-proof (i.e., agents have *no* incentive to lie) is discussed in Refs [52, 58].

8 DISTRIBUTED ALGORITHMS FOR THE INCOMPLETE INFORMATION SCENARIO

The model studied in the previous section assumes that knowledge about the locations, the arrival times of all agents, and resources are available to all agents and the central authority in the system. This assumption may be difficult to justify in practice for various reasons. For example, it raises privacy and security concerns because some agents may not wish to share their location information with other agents.

In this as well as the next section, we study our problem in a setting where the location and start time of an agent is *not* known to other agents. This is a more realistic setting for our spatio-temporal matching problem. Each agent, therefore, would act as if she/he is computing her/his destination dynamically in an online fashion based on her/his receipt of *dynamic updates* of resource availability *without* any knowledge about the arrivals or locations of other agents. Thus, each agent only has partial or incomplete information to make her/his decision regarding which resources to visit and try to obtain.

8.1 Basic Gravitational Algorithmic Paradigm

The heuristic algorithmic approach we describe here, then, is one that pushes the agents toward areas where they are *most likely* to find a resource or areas with a *higher density* of available resources, taking into account the agent's location and its proximity to the surrounding resources. Assuming the agents to be distributed uniformly across space, this is expected to increase the agent's probability of finding a resource by arriving in an area with a larger number of resources. The *Gravitational Parking Algorithm* (GPA), a heuristic using the gravitational algorithmic paradigm, was originally introduced in Ref. [1] and encompasses these properties. GPA was used in Ref. [1] to guide agents toward areas of the map when they do not have information about other agents that are competing with them for the resources. Though GPA was originally designed

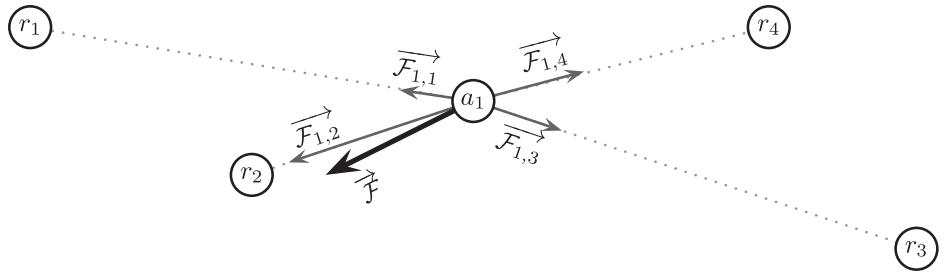

Fig. 5. A schematic diagram of the basic gravitational algorithmic paradigm. $\overrightarrow{\mathcal{F}_{1,1}}, \overrightarrow{\mathcal{F}_{1,2}}, \overrightarrow{\mathcal{F}_{1,3}}, \overrightarrow{\mathcal{F}_{1,4}}$ are the force vectors induced on agent a_1 by resources r_1, r_2, r_3, r_4, respectively, and $\overrightarrow{\mathcal{F}} = \overrightarrow{\mathcal{F}_{1,1}} + \overrightarrow{\mathcal{F}_{1,2}} + \overrightarrow{\mathcal{F}_{1,3}} + \overrightarrow{\mathcal{F}_{1,4}}$ is the resultant force vector.

for parking applications in Ref. [1], it can also be applied to the general framework presented in this article involving agents and resources.

In GPA, resources are said to have a *gravitational pull* on the agents. At any point in time, each resource has a gravitational force vector acting on the agent whose magnitude and direction depend on the distance of the resource from the agent and the spatial location of the resource. Then, all of these vectors are added and the agent moves in the direction of the *resultant vector* (total gravitational force) for a specified time interval and the same process is repeated at the beginning of the next time interval. The intuition behind this approach is that agents are expected to be pulled toward areas with a *higher* density of available resources, thus increasing the probability of finding one. A schematic diagram for this approach is shown in Figure 5.

The following simplified formula,[5] which did not include the masses of the objects or a gravitational constant like in the *classical* Newton's law of gravitation, was used to generate the magnitudes of the gravitational force vectors in Ref. [1]:

$$\left|\overrightarrow{\mathcal{F}_{i,j}}\right| \stackrel{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, r_j\right)}\right| = \frac{1}{\left[\,\mathit{time}\left(\ell_i^{\mathcal{A}}(t),\ \ell_j^{\mathcal{R}}\right)\right]^2}, \tag{6}$$

where $\overrightarrow{\mathcal{F}\left(a_i, r_j\right)}$ is the force vector generated by resource r_j on agent a_i. Note that in Equation (6) $\ell_i^{\mathcal{A}}(t)$, location of the agent at time t, is a function of time and, therefore, changes continuously as the agent moves. Thus, the GPA approach as described here is really a *navigation algorithm* rather than just an algorithm that computes an agent-resource matching. For general costs given by our cost function, Equation (6) can be generalized to the following:

$$\left|\overrightarrow{\mathcal{F}_{i,j}}\right| \stackrel{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, r_j\right)}\right| = \frac{1}{\left[\,\mathit{cost}\left(a_i,\ r_j,\ t\right)\right]^2} \tag{7}$$

With Equation (7), one can compute the force vector by considering the current *general cost* as opposed to simply using the current travel time between an agent and the resources. This general cost could potentially include many *additional* factors such as the *walking time* from the resource to the actual destination of the agent or the *price* of the resource.

8.2 Revising the Gravitational Algorithmic Paradigm for Road Networks

The GPA approach presented in the previous section was shown to work well in the two-dimensional Euclidean plane. However, in a real transportation network, movements of agents

[5] $|\overrightarrow{a}|$ denotes the magnitude of a vector $\overrightarrow{a}$.

are restricted to the available roads. Thus, the direction of the resultant vector $\mathcal{F}$, while posing no problem for the Euclidean plane, may actually point to a direction that does *not* coincide with one of the available roads. Then, the gravitational algorithmic paradigm as discussed above needs to be revised.

Let $G = (V, E)$ be the directed graph representing the given road network with edges in E representing roads and nodes in V representing intersections of these roads. On such a graph G we will still use a gravitational algorithmic approach based on Equation (7). However, now an agent can change her/his routing direction only upon arrival to an intersection (i.e., a node in G) as opposed to any time in the case of the Euclidean plane. Therefore, Equation (7) will be used by an agent to update its routing *only* at each intersection.

ALGORITHM 1: Algorithm for computing the magnitudes of the κ direction vectors for agent $a_i \in \mathcal{A}$ at time t. Instead of adding up all the vectors for all available resources (as in the Euclidean plane), a_i aggregates the force vectors for all resources into spatial direction vectors corresponding to each possible direction out of the intersection.

```
(* location(v) denote the location of node v ∈ V *)
(* For v ∈ V, cost_v(a_i, r_j, t) is equal to cost(a_i, r_j, t) assuming ℓ_i^A(t) = location(v) *)
(* For e = (u, v) ∈ E, cost_e(a_i, t) is equal to cost(a_i, r_j, t) assuming ℓ_j^R = location(v) *)
(* Agent a_i is in node v at time t; v has κ outgoing edges e_1 = (v, v_1), ..., e_κ = (v, v_κ) ∈ E *)

|D_1| ← |D_2| ← ··· ← |D_κ| ← 0
for all r_j ∈ R do
    for all q = 1, 2, ..., κ do
        cost(a_i, r_j, t) ← cost_{e_q}(a_i, r_j, t)
        |D_q| ← |D_q| + 1 / [cost(a_i, r_j, t)]^2
    end for
end for
```

For any node $v \in V$, let location(v) denote the actual physical location of v. Also, suppose that at time t, an agent a_i is at this intersection (node) v (i.e., $\ell_i^{\mathcal{A}}(t) = \text{location}(v)$) and v has κ outgoing edges $e_1 = (v, v_1), e_2 = (v, v_2), \ldots, e_\kappa = (v, v_\kappa) \in E$. Let $\mathit{cost}_{e_q}(a_i, r_j, t)$ denote the value of $\mathit{cost}(a_i, r_j, t)$ *assuming that the agent a_i first travels through edge e_q*. Furthermore, there will be κ direction vectors $\overrightarrow{\mathcal{D}_1}, \overrightarrow{\mathcal{D}_2}, \ldots, \overrightarrow{\mathcal{D}_\kappa}$ where $\overrightarrow{\mathcal{D}_q}$ will point to the direction of edge e_q. The initial magnitudes of all these κ vectors are set to zeros. Then, for each *available* resource $r_j \in \mathcal{R}$ and each possible direction corresponding to each $\overrightarrow{\mathcal{D}_q}$, the agent performs the following computations:

- —first computes the cost of r_j assuming that a_i's travel will begin[6] with the edge e_q.
- —then computes the magnitude of force vector $\overrightarrow{\mathcal{F}_{i,j}}$ using this value of $\mathit{cost}(a_i, r_j, t)$ in Equation (7), and
- —then updates $|\overrightarrow{\mathcal{D}_q}|$ to $|\overrightarrow{\mathcal{D}_q}| \leftarrow |\overrightarrow{\mathcal{D}_q}| + |\overrightarrow{\mathcal{F}_{i,j}}|$.

This computation is summarized in Algorithm 1 for the convenience of the reader.

[6]In the case when the general cost reflects the travel time, i.e., when $\mathit{cost}(a_i, r_j, t) = \mathit{time}(\ell_i^{\mathcal{A}}(t), \ell_j^{\mathcal{R}})$, this step corresponds to computing a *shortest path* from $\ell_i^{\mathcal{A}}(t)$ to $\ell_j^{\mathcal{R}}$ that begins with the edge e_q.

After repeating this procedure for each available resource and edge that exits v, the agent uses the computed direction vectors $\overrightarrow{\mathcal{D}_1}, \overrightarrow{\mathcal{D}_2}, \ldots, \overrightarrow{\mathcal{D}_\kappa}$ to make its route choice. There are various options for choosing the direction of travel according to the magnitudes of these vectors. In Ref. [4], several options were experimentally tested, and the *Deterministic Magnitude Gravitational* (Dm-Gra) algorithm was found to show the best results. In Dm-Gra, the agent travels from v along the edge having the vector $\overrightarrow{\mathcal{D}_q}$ of largest magnitude. Algorithm 2 shows the specification of Dm-Gra.

ALGORITHM 2: Deterministic Magnitude Gravitational (Dm-Gra) algorithm for choosing travel direction.

compute direction vectors $\overrightarrow{\mathcal{D}_1}, \overrightarrow{\mathcal{D}_2}, \ldots, \overrightarrow{\mathcal{D}_\kappa}$ using Algorithm 1
let q be the index such that $|\overrightarrow{\mathcal{D}_q}| = \max_{1 \le j \le \kappa} \{|\overrightarrow{\mathcal{D}_j}|\}$
agent a_i moves along edge e_q

9 SPATIO-TEMPORAL MATCHING WITH MISSING OR ERRONEOUS INFORMATION

The methods that have been described so far depend on the existence of real-time availability data, i.e., *every* available resource and its location is accounted for in the database at any point in time. The collection of this *resource availability data* depends on wireless sensors. For example, in the context of finding parking for vehicles, applications such as Refs [41, 45] make use of static sensors that are embedded in the road pavement and detect whether the resource (parking slot) is available or not in real time. These sensors, then, can be used by applications for resource discovery, giving the agents a *complete* and *real-time* view of the resource availability data. Nevertheless, the implementation and maintenance of these sensors present many difficult challenges, e.g., they may have a very high monetary cost and so it is infeasible to cover a whole city with these *static* sensors. Thus, other methods are often needed for collection of resource availability data.

To overcome the difficulties of employing static sensors, various other methods exist for collection of this availability data that make use of *mobile devices*. For example, in Ref. [46], mobile *phones* are used to detect automatically when a traveler parks or de-parks from a parking slot in an urban setting. Even for a spatial resource search application such as the application involving taxicabs and clients mentioned before, one could have travelers requesting taxicabs by using their mobile phones as described above. In both these applications, the mobile phone acts as a sensor that determines the availability of a resource. Applications like these have been shown to be quite useful [57]. However, these applications do suffer from having a low *penetration rate* (or ratio). The penetration rate for these mobile applications is the percentage of users of the transportation system that use their mobile phones as sensors for collecting the availability data. Then, an important question to ponder about is: *how useful can these mobile sensors be in the face of low penetration rate and the limited availability data*?

In the previous sections, we studied the problem of how to *navigate* users in the road network to find their desired resources given complete and correct resource availability data. The methods presented there work well for settings that have *full* access to the *ground truth data* about resource availability. However, in more practical settings, it is usually *infeasible* to have access to this type of data. Real-life systems will have access to uncertain or inaccurate availability data. The uncertainty stems from the fact that, when a resource is reported to be available by some sensor, one does not know if the resource is still available when a user will search for it because perhaps some other user that does not report to the system (due to a low penetration rate) has taken the resource. Uncertainty may also stem from the fact that a resource that is reported to be unavailable may actually be *available* because reports are missing from users that do not report

to the system. In a nutshell, the resource availability data may be erroneous and incomplete (with missing information).

The uncertainty and errors in the resource availability data may come in various forms such as:

Missing reports: A resource was last reported to be available according to the database, but it is no longer available; or a resource was last reported to be unavailable according to the database, but it is available.

Erroneous reports: These reports are due to errors in sensing, i.e., the mobile device detects availability or unavailability in an incorrect manner.

Unknown exact locations: We may not know the exact location of an individual resource inside a *block* of resources (edge of the network). Thus, we assume that a reportedly available individual resource lies *somewhere* inside the block.

Given the existence of *limited* and *uncertain* resource availability data, we wish to investigate how the agents should move across the road network to obtain their desired resources. In this section, we develop methods for this setting with missing and erroneous reports. In the next section, we will test these methods via simulation in a setting that is based on real-world ground truth data.

9.1 Setup and Preliminary Definitions for Spatio-Temporal Matching with Missing and Erroneous Reports

Our setup uses the following definitions and terminologies, which are in addition to those presented in Section 3 and pertain only to this section and the experiments that were run:

—Let $G = (V, E)$ be the directed graph representing a road network with edges in E representing roads and nodes in V representing intersections of these roads. We will also use the term block to refer to these edges, as in, a block of resources that is contained in an edge.

—*Blocks* of resources are available on some edges of the network. Each block contains a set of resources. For each block $(v_i, v_j) \in E$, there exists a ground-truth resource availability number $K_{i,j} \geq 0$.

—The set of agents are split into two separate groups: the *sensing* agents and the *non-sensing* agents.

—The *penetration rate* of the system is the percentage of agents in the system that are sensing agents.

—A database receives reports about availability or unavailability of resources from the sensing agents. The non-sensing agents do not provide any reports. The reports can be classified as:

***Resource-level Availability*:** A new resource has been sensed as available or made available by one of the sensing agents.

***Resource-level Unavailability*:** A previously available resource has been sensed as unavailable by one of the sensing agents.

***Block-level Unavailability*:** No available resources were found when a sensing agent passed through a block.

—The reports that the agents send to the database contain the following information:

***Classification*:** resource-level availability or unavailability, or block-level unavailability.

Block: the location of the block where the report originates.
Timestamp: the time when the report originated.

—After T time units have passed from the creation of a report in the database, the report is discarded and is no longer considered to be relevant in the database. T is a parameter of the system that is used to determine when the received reports become *stale*.
—The sensing agents will have the resource availability data to help them make their routing decisions. This data could contain erroneous reports and will not contain reports that are missing due to non-sensing agents not sharing any reports. For each block (edge) $(v_i, v_j) \in E$, they will have access to the following quantities (which in some cases will have to be estimated):

$k_{i,j}$: the number of resources that are *reportedly available* in the block (v_i, v_j) according to previous resource-level reports from other sensing agents, which are still *not stale*.
$u_{i,j}$: the number of resources that are *reportedly unavailable* in the block (v_i, v_j) according to previous resource-level reports from other sensing agents, which are still *not stale*.

—If the database receives a block-level unavailability report, at some time t, for block $(v_i, v_j) \in E$, then $k_{i,j}$ is set to 0 and $u_{i,j}$ is set to its maximum value (the number of resources that exist in the block). Also, availability reports, which were received before t, are marked as stale.
—If an agent passes by a block (v_i, v_j) and $K_{i,j} > 0$, then the agent will stop its search and obtain an available resource. If $K_{i,j} = 0$, then the agent will continue its search in other blocks of the road network and if the agent is also a sensing agent, she/he will send a block-level unavailability report for this block, stating that no parking is currently available in that block.

9.2 Navigation Using Only Availability and Block-Unavailability Reports

As stated before, there are two types of resource-level reports that the database receives: the availability reports and the unavailability reports. Availability reports are received when a sensing agent makes a resource available (e.g., leaves a parking slot), whereas unavailability reports are received when a sensing agent occupies a previously available resource. The database can also receive block-level unavailability reports.

In this section, we consider a setting that has access *only* to the resource-level availability reports and to the block-level unavailability reports. For example, in the context of finding parking for vehicles, this is a system that can only detect when vehicles leave a parking spot, but it does not collect data about when a spot is taken. This system can also detect block-level unavailability reports whenever a sensing agent that is searching for a resource, passes by a block and does not obtain a resource.

This type of system is somewhat akin to the system presented in Ref. [31]. They present a system in which vehicles act as mobile sensors of availability by driving past curbside parking slots to detect open ones. These mobile sensors generate a map view of parking slot availability. Thus, this type of system generates availability reports but not unavailability reports. In the application involving taxicabs and clients, this is a system in which a client (the resource being searched for) requests a cab service, but does not notify the service if she/he found another taxi.

Here, we present two algorithms for navigation in this setting based on the gravitational algorithmic paradigm discussed in the previous section. One uses merely the number of reportedly available slots; the other uses in addition the *age* of each of the availability reports.

9.2.1 Gravitational Navigation Without Aging. We will use this same gravitational algorithm that was presented in Section 8.2 but now we also incorporate the uncertainty of availability values for each block in the algorithm. Before, the gravitational forces were computed individually for each resource, but now the gravitational forces will be *aggregated* at the block level. For this purpose, we modify Equation (7) in the following manner. The magnitude of the gravitational force of a block $(v_p, v_q) \in E$ to an agent $a_i \in \mathcal{A}$ is now defined as

$$\left|\overrightarrow{F_{i,p,q}}\right| \stackrel{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, v_p, v_q\right)}\right| = \frac{k_{p,q}}{\left[\,\mathit{cost}\left(a_i,\, r_j,\, t\right)\,\right]^2} \text{ with } \ell_j^{\mathcal{R}} = \begin{cases} \text{midpoint of the line connecting} \\ \text{location}(v_p) \text{ and location}(v_q), \end{cases} \tag{7'}$$

where we assume that the location of the block coincides with the location of the midpoint of the line connecting nodes v_p and v_q. Also, the value of $k_{p,q}$ is the number of availability reports that have been received and are not *stale*. The agent, then, again proceeds in the direction of the block with the highest accumulated gravity force as in Section 8.2.

9.2.2 Gravitational Navigation With Aging. We can refine our approach discussed in the preceding section by incorporating the *age* of the resource-level availability reports in Equation (7′). We can do so by giving a *relevance score* to each availability report according to its age. The older the availability report, the less relevant it should be since the probability that somebody has obtained the resource increases over time, i.e., the newer the report is, the higher the relevance score should be. We assume a *linear decay* for the relevance score of a report. That is, if $t' \leq T$ is the age of an availability report q, then the relevance score $R(q)$ for q is defined as:

$$R(q) = 1 - \frac{t'}{T}$$

Note that $R(q)$ linearly decreases from a value of 1 when $t' = 0$ to a value of zero when $t' = T$. Let $Q_{i,j} = \{q_1, q_2, \ldots, q_k\}$ be the set of k resource availability reports that are still relevant (i.e., of age no more than T) for a block $(v_i, v_j) \in E$, and let t_ℓ be the age of report $q_\ell \in Q_{i,j}$. Then we can compute the *estimated availability* at block (v_i, v_j) as:

$$\mathcal{E}_{i,j} = \sum_{\ell=1}^{k} \left(1 - \frac{t_\ell}{T}\right)$$

We then redefine the gravitational pull of a block, when considering the age of reports, for an agent $a_i \in \mathcal{A}$ by modifying Equation (7′) as:

$$\left|\overrightarrow{F_{i,p,q}}\right| \stackrel{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, v_p, v_q\right)}\right| = \frac{\mathcal{E}_{i,j}}{\left[\,\mathit{cost}(a_i,\, r_j,\, t)\,\right]^2} \tag{7''}$$

This new gravitational force may then be used in the same way as in the previous section.

9.3 Navigation Using Both Availability and Unavailability Reports

In this section, we discuss navigation algorithms that use *both* resource-level availability and unavailability reports. This would be a system like the one described in Ref. [46] where both types of activities (availability as well as unavailability) are detected. As in the previous section, the database will also receive block-level unavailability reports. In this setting, we need to compute an estimated availability value, based on all of the received reports, to be used by a gravitational algorithm. In Section 9.2, we had only availability reports to deal with, but now we also need to

process the resource-level unavailability reports and combine them with the availability reports in a suitable manner.

9.3.1 Computing the Estimated Availability—A Queue-based Approach. Recall that the database keeps only those reports that were received within the last T time units and aggregates these reports at the *block level*. Thus, for each block $e = (v_i, v_j) \in E$, we have a sequence of reports, say d in number, of the form: $\langle c_1, e, t_1\rangle, \langle c_2, e, t_2\rangle, \ldots, \langle c_d, e, t_d\rangle$, where c_i is the *classification* of the report (available or unavailable), e is the identification of the block where the report originated, and t_i is the time when the report was created. From this sequence of reports, it is impossible to exactly compute the true availability $K_{i,j}$ in the block because we only keep reports from the last T time units and because of the inherent uncertainty of the information (due to the penetration rate). Since we do not know which resources are still available or unavailable, given a sequence or reports for a block, we could compute different true availabilities for any given sequence of reports. Therefore, we need to compute an *estimated availability* that is based only on the information that we have access to (aggregated resource-level reports).

In this section, we propose a *queue-based* approach. We have a queue of resource-level reports for each block. Initially, all the queues are empty. We continue to process the reports as they come. Whenever a resource-level availability report for some block is received, the report is added (enqueued) to that block's queue (increasing its size by one). If a resource-level unavailability report is received for a block and the queue for that block is not empty, then the oldest report is deleted (dequeued) from that block's queue. Whenever a report that is in any queue reaches an age greater than T, the report is removed from the queue as well. Then, at any point in time, the *availability estimate* for the block is simply the *current size* of the queue.

The block-level unavailability reports are not necessarily saved and are just processed as denoted in Section 9.1, i.e., all resource-level availability reports received before the time of the block-level unavailability report are marked as stale. In this queue-based approach, this would be equivalent to emptying the queue whenever this type of report is received (denoting that the estimate of availability should be 0).

9.3.2 The Navigation Algorithm. The algorithm that will be used for navigation will be the same gravitational approach discussed in the preceding section, except that the numerator $k_{i,j}$ in Equation (7′) will be replaced by the current size of the queue for the block $(v_p, v_q) \in E$.

10 SIMULATIONS AND EXPERIMENTAL RESULTS

In this section, we experimentally evaluate the previously presented algorithms in a simulation framework that uses real-world data from the SFPark project [41].

10.1 Simulation Data

In this section, we describe the real-world data that is used in this work to test our spatio-temporal matching algorithms. The source of the data is a project developed by the San Francisco Municipal Transportation agency called SFPark [41, 45]. They have embedded wireless sensors on the pavement of parking slots to determine their availability on a real-time basis, and publish real-time data of availability changes, at a per-block level, for each block that their sensors cover. As described before, the problem of finding parking for vehicles is a prime example of our spatio-temporal resource matching problem, and, thus, this SFPark dataset is *very* relevant for our purposes.

A tuple in this SFPark database has the schema $\langle blockId, availability, timestamp\rangle$, and is a report that at time *timestamp*, the availability on the block *blockId* has changed to the given *availability*. We then transform this database into a database of availability reports by scanning these tuples

in order and saving the previous availabilities for each block. Then, when checking the current availability for a given block, if the previous availability is smaller than the current one, then we publish a *deparking* event (resource-level availability) report to the reports database, and if the current value is smaller than the previous availability, then we publish a *parking* event (resource-level unavailability) report to the reports database. Thus, after scanning the whole database, we have created a *reports database* where each tuple is of the form $\langle event, blockId, timestamp\rangle$ with $event \in \{parking, deparking\}$ indicating a classification of an unavailability or availability report.

To generate data to test the algorithms presented in Section 9, suppose that a given penetration rate is $0 \leq p \leq 1$. Then, to *simulate this given penetration rate p* for the system, all we need is to choose to keep each tuple with probability p. This would give us a reports database with a penetration rate of p.

We can also use the original database to compute all values of $K_{i,j}$ (the ground truth) at any point in time. To do this, we search for the previous tuple that occurs in the database, for that block, before the current timestamp. We use this method of querying the database to test our algorithms with certain information. To test our algorithms that use missing information, we may also use this ground truth to determine if an agent can obtain the resource (parking slot) that the agent is passing by on a block.

Since no exact locations of parking slots are available, we make a simplifying assumption that the available slots on a block are located in the middle of the block, which is consistent with Equation (7′).

10.2 Simulation Setup

Our simulation tests the spatio-temporal search algorithms presented previously. We test these algorithms against a *greedy* algorithm that moves the agents to the *closest reportedly available* resource in the scenario where the algorithms have access to ground truth data, and against an algorithm that tries to maximize the probability of finding a resource, in the scenario where the algorithms have access to uncertain (missing and erroneous) information. The simulation uses real-world data from the SFPark [41] database (as described before). The availability reports from the SFPark data were taken for the tourist area of San Francisco called the Fisherman's Wharf. We built a graph of the Fisherman's Wharf region based on the block data given by the SFPark database.

We *created n* vehicles (agents) at the beginning of each simulation run, and placed them at uniformly distributed random locations in the graph. These *n created agents* are all classified as sensing agents. The simulation starts at a given time t_0 of the day. After placing all the vehicles, each resource search navigation algorithm was tested.

The agents (vehicles) move through the network over time *at a constant speed* of 20 mph, following the navigation algorithm being tested. If an agent passes by a block and there is parking (resource) available while it is in the middle of the block, then the vehicle is parked (i.e., the agent is matched with this resource). If no parking is available on the block, then the agent continues to move through the network and she/he sends a block-level unavailability report to the database. When an agent finds an available resource, the time it took for it to find that resource is saved. The simulation stops when *all* agents are able to obtain a resource (i.e., when all vehicles are parked). The *average time* to park all the vehicles is also computed and saved.

It bears mentioning that there exists a distinction between these *n created* agents and the *real agents*, which created the original dataset. These real agents are an implicit part of the simulation since they are the ones that created the original park/depark reports. They also influence the simulation because some reports occur at the time when the simulation is run. For example, say that

the simulation uses simulation data from a specific day at 4:00 p.m. of that day, then every report that is received in that hour, and that exists in our database, was sent by one of these real agents.

Then, these real agents differ from our created agents in various ways. The real agents can send park and depark reports, whereas the created agents only send a report when they finally park (and then they are no longer part of the simulation). Also, the created agents are allowed to send a *ZERO-PARKING* report when they pass by a block that contains zero available resources (parking spots). Also, the real agents are not controlled by our simulation either; they are just implicitly part of the simulation (based on the SFPark data that was used).

For each experiment, 10,000 different simulation runs were generated and evaluated for each of the tested algorithms. The Java software for the simulation environment is available at Ref. [40].

10.3 Tested Algorithms

The algorithms that were tested are labeled as follows for subsequent referral:

***ZeroInfo*:** In this algorithm, the vehicles have *no* information about resource availability. The algorithm searches *blindly* across the network, using a random walk approach, until it finds a resource. In this random walk approach, the agent chooses randomly (using a uniform distribution) among all the possible edges that it can take at each intersection, excluding the edge that it just came from (i.e., no U-turns). This algorithm models a vehicle *without* a navigation system.

***Gravity*:** This is the basic gravitational algorithm presented in Section 8.2 with access to the ground truth data about resource availability.

***Greedy*:** In this algorithm, the agents, with access to the ground truth data about resource availability, move toward the *closest* available slot. This is a naive approach for searching resources in that it has no guarantee of obtaining a resource, but it is a very common strategy for searching resources.

***SysOpt*:** This is the system optimal algorithm for matching between agents and resources (presented in Section 4). It does not require the navigation aspect that is described above. For these tests, agents are simply assigned to their system optimal resources. This approach is tested here to motivate the efficiency of the system when using the pricing schemes presented in Section 7. This algorithm uses complete information about the locations of the agents and resources, which means that the penetration rate for these tests is 100%.

***UGravityQ*:** This is the queue-based algorithm presented in Section 9.3.1 that uses the gravitational algorithmic paradigm. This algorithm has access only to a dataset with missing information (reports), i.e., the availability data is based on a given penetration rate and the value of T.

***UGravity*:** This is the approach presented in Section 9.2.1. It uses the gravitational algorithm but with only the received availability reports. This algorithm has access only to a dataset with missing reports.

***PM*:** This is a probability maximization approach, which was introduced in Ref. [23]. This algorithm aims to choose a path with the maximum overall probability of finding a resource.

We should note that the algorithm that was presented in Section 9.2.2, which takes into consideration the age of the received reports, was tested in the simulations but is not included here because it did not significantly alter the results.

The *PM* algorithm, which is tested in the setting with missing and erroneous information, seeks to compute paths that maximize the probability of finding available resources. The probabilities

that they compute are based on both historical data and real-time availability data. For the historical component, they seek to compute two metrics: $1/\lambda$ is the expected time an available resource remains available, and $1/\mu$ is the expected time a consumed resource remains in this consumed state.

We estimate these two metrics from the historical SFPark data for each block in the tested urban area. Since a report in the SFPark database is not labeled with the parking space it pertains to, it is uncertain which parking event ends an available time interval started by a deparking event, or which deparking event ends an unavailable time interval started by a parking event. We, therefore, adopt a First-In-First-Out strategy to match parking events and deparking events; in fact, any matching strategy (e.g., Last-In-First-Out) will work equally well. Specifically, for each block in the SFPark database, we generate a list of parking and deparking events in ascending order of their timestamp. Then, we visit these events sequentially. For each visited parking event, we match it to the earliest deparking event before it is in the list. We compute the time difference of the two events as the time of availability of a parking slot in that block. The matched deparking event is then removed from the list. After all events are visited, we average all of the computed times of availability for that block and that average is the value for $1/\lambda$, the expected time a resource remains available. Similarly, we match a visited deparking event to the earliest parking event before it, and compute $1/\mu$, the expected time a consumed resource remains in this consumed state. It should be noted that we estimate the two metrics from the original SFPark database without introducing any missing and erroneous information. This treatment favors the G2 algorithm because, in reality, the estimation of the two metrics could well be contaminated by missing and erroneous information.

Then, we incorporate the values of λ and μ into the probability computations, which have a time component of when a resource became available or consumed, to be able to then compute the availability probabilities for each block in our road network (given the current availability reports for each block). The algorithm from Ref. [23] that we use for comparison is the G2 algorithm. Like our gravitational navigation algorithm, it uses a greedy approach, and, thus, is a suitable comparison for our method. It is a heuristic that computes a path by extending the path with edges that maximize the probability of finding the resource but also taking into account the distance traveled. It keeps extending a path until a probability of obtaining a resource on that path is greater than some threshold ρ. Josse et. al. also presented a G1 heuristic that was not used in our comparisons because they showed in their work that the G2 algorithm was superior to G1.

The G2 algorithm is implemented as follows in our experiments. For each block, we maintain a list of available timestamps. When a deparking event report is received, an available timestamp is added to the list. When a parking event report is received, the earliest available timestamp (if there is any) is matched and removed from the list. When an agent arrives at a node, it chooses which of the attached blocks to move on depending on the probability that there is at least one resource available at an attached block. This probability is computed based on the number of resources reported available and the probabilities that they are currently available. According to Equation (1) introduced in Ref. [23], with exponentially distributed available time and possible re-appearance, the probability that a resource that is available at time 0 is available at time t is $\frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t}$. This availability probability is then divided by the distance to travel on that block (G2 heuristic) to give a distance-conditioned probability of availability. The agent will then choose to move toward the block that maximizes this probability of availability.

10.4 Tested Parameters

The following parameters are varied in the simulations as mentioned below:

Table 1. Summary of Several Key Conclusions of Our Simulation Results. See Relevant Section for Further Details and Explanations

Algorithms with access to ground truth data	▷	Time to find parking increases as n increases.
	▷	*Gravity* always outperforms *Greedy*.
	▷	Growth rate of average time to find parking is smallest (excluding *SysOpt*) for *Gravity* as n increases.
Algorithms with access to a dataset that could have missing or erroneous data	▷	Best performances were obtained by *UGravityQ* and *UGravity*. Best results for these algorithms were obtained with higher T.
	▷	Even when $n = 1$, the gravitational algorithms outperform *PM*.
UGravity vs. PM	▷	For most n and p, *UGravity* was better than *PM*.
Environmental improvements	▷	In a big urban city like Chicago, with gravitational algorithms and improvement of over 20%, we would save over 34.4 million VMT, 1.67 million gallons of gasoline, and over 25,800 tons of CO_2 emissions per year.

- ***n*** The number of created agents that are generated. The values that were tested for this parameter were $n \in \{1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60\}$.
- ***p*** The penetration rate of the system, i.e., the percentage of agents in the transportation system that are assumed to be sensing agents in the scenarios with missing and erroneous information. The values that were tested for this parameter were $p \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.0\}$.
- ***T*** The time threshold for discarding reports from the database in scenarios with missing data. This parameter is tunable and a representative one was chosen for the presentation of results. The values that were tested for this parameter (in minutes) were $T \in \{5, 15, 30, 45, 60, 75, 90, 105, 120\}$.

The *PM* algorithm, which was tested for comparison purposes, keeps extending a path until a probability of obtaining a resource on that path is greater than some threshold ρ. For the simulation, we decided to choose a value of $\rho = 0.9$. This was the largest value of ρ that was tested in Ref. [23].

10.5 Simulation Results for Various Environments

In this section, we discuss the results of our simulation in various environments and parameter settings. For the convenience of the reader, some of the key conclusions reached due to these results are summarized in Table 1 below.

10.5.1 Results for Algorithms with Access to Ground Truth Data. These simulations were run in an environment in which the algorithms had access to the ground truth data of resource availability. For this case, the algorithms that were tested were *ZeroInfo*, *Gravity*, *Greedy*, and *SysOpt*. These algorithms did not depend on the values of T and p. The *ZeroInfo* algorithm did not have access to any data, but we wanted to compare it to these approaches to be able to see the contrast between having and not having access to the ground truth data.

Figure 6 shows results for the average time to park for vehicles using different algorithms. As expected, the *ZeroInfo* algorithm serves as an *upper bound* for the time to park. We also observe that

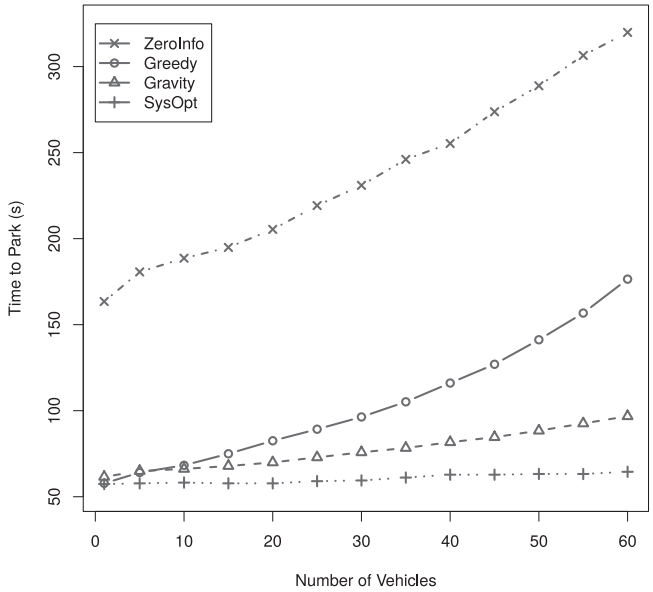

Fig. 6. Average time to park for algorithms with access to the ground truth data of resource availability.

for all algorithms, except the *SysOpt* algorithm, the time to find parking increases as the number of vehicles that are looking for parking increases; for the *SysOpt* algorithm, this increase is marginal and imperceptible in the graph due to scaling. Also observe that the *Gravity* algorithm has better time to find parking over the *Greedy* algorithm, and the growth rate of average time to find parking is smallest (excluding the *SysOpt* algorithm) for the *Gravity* algorithm as the number of vehicles increases.

Figure 7 shows the percentage improvement of the *SysOpt* and the *Gravity* algorithm over the *Greedy* algorithm. The improvement keeps increasing for both algorithms as the number of vehicles increases. Observe that the *Gravity* algorithm achieves a significant improvement on the average time to find parking (up to 40%) over *Greedy* even though, unlike the *SysOpt* algorithm, it does *not* have access to the information of the locations of all other vehicles in the system. Nevertheless, the clear winner in improving environmental and driving times, as expected, is the *SysOpt* algorithm. This further motivates the use of the pricing schemes presented in Section 7.

10.5.2 Simulation Results for Algorithms with Missing or Erroneous Information. Figure 8 shows some of our obtained results for algorithms that use a dataset with missing information due to different values of penetration rate. These algorithms can be classified into two categories: those that make use of *both* the resource availability and the resource unavailability reports (using the queue-based approach for handling these reports), and those that have access *only* to availability reports. The algorithm that makes use of the queue-based approach is *UGravityQ*.

In Figure 8, we see the results for varying values of the penetration rate, with $n = 30$ and $T = 120$. It also includes the results for the *ZeroInfo* algorithm and the Gravity algorithm (gravitational approach with access to all the information). We show this so one can see the gaps that are present in the performance of the algorithms.

So, for this case, we can see that the Gravitational approaches are around two and a half to three times better than the *ZeroInfo* algorithm (depending on the value of n being tested). This

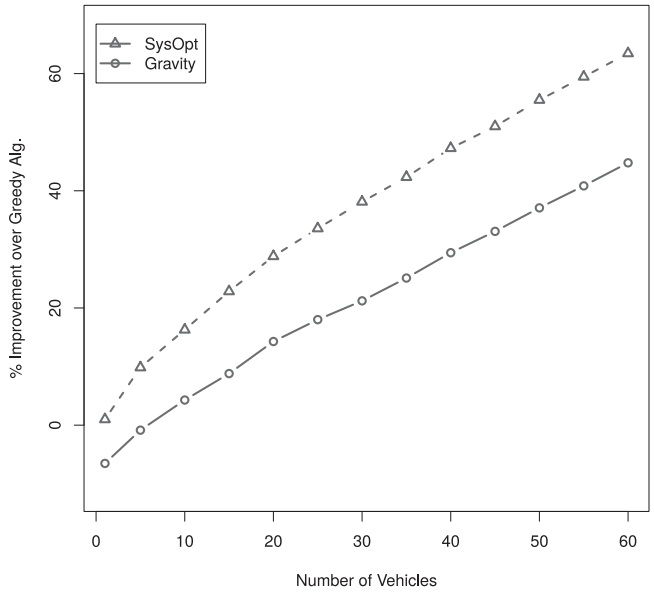

Fig. 7. Percentage improvement of the average time to park for *SysOpt* and *Gravity* over *Greedy*.

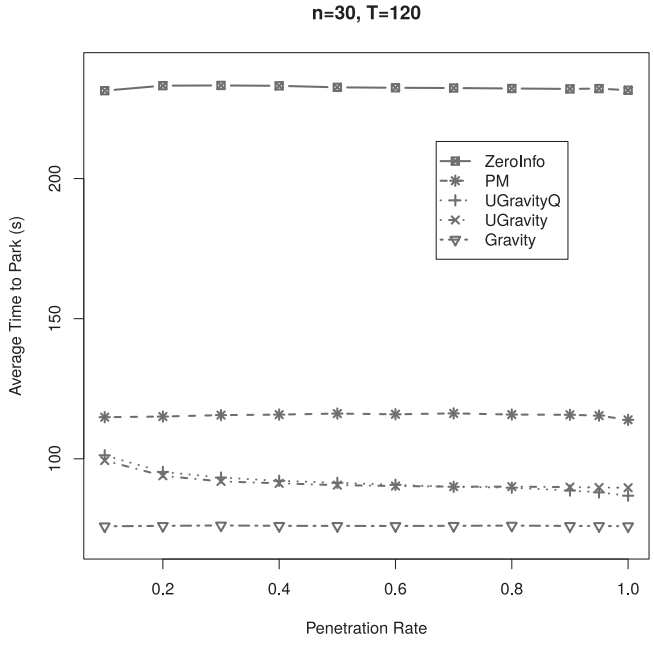

Fig. 8. Average time to park for the tested algorithms, when $n = 30$ and $T = 120$.

gives us an idea of what is the value of having access to data, since the *ZeroInfo* algorithm does not have access to any data. For our gravitational approaches, this ratio of the average time an agent can find a resource with the *ZeroInfo* algorithm, over the time it takes with the gravitational approaches, ranged between two and three for most test cases.

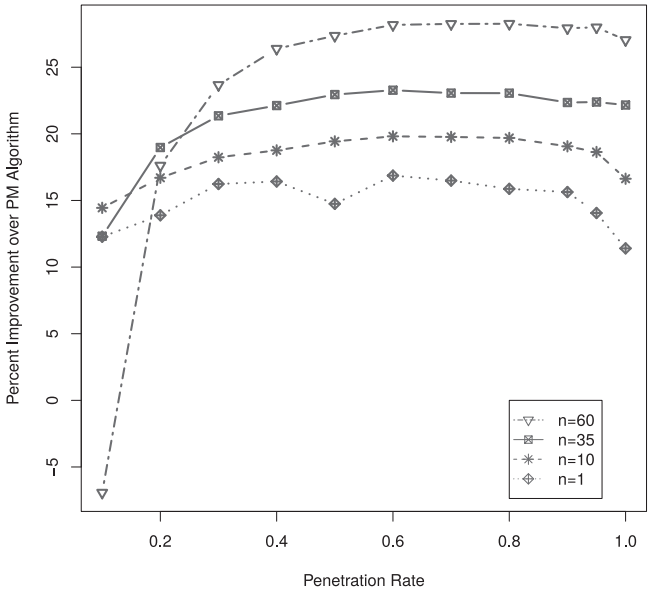

Fig. 9. Percentage improvement of the average time to park of *UGravity* over *PM* with $T = 120$.

In Figure 8, we can also see the gap between the gravitational approaches that have access to the dataset with missing or erroneous information against the *Gravity* algorithm, which had access to all the perfect information. This gives us an idea of what is the price or the penalty for not having access to all of the data about parking and deparking events. If we were to define a ratio of the average time an agent can find a resource with one of the *UGravity* algorithms over the time it took for the *Gravity* algorithm, this ratio ranged between 1.2 and 1.6 for most of the cases that were tested.

Combining the results from the previous two paragraphs, we can see that the value of using our gravitational algorithms, even in cases with a low penetration rate, over the *ZeroInfo* algorithm, is much larger than the penalty or price we get for not having access to all of the data (like with the *Gravity* algorithm).

Figure 8 is a good representation of all the simulations that were executed. In almost all cases, the gravity algorithms outperformed their probability maximization counterpart (*PM*). Except in the very extreme cases of the penetration rate being at its lowest and the competitive ratio (n) being at its highest. For this graph, we chose the highest value of T because this was a system parameter and all the algorithms performed at their best with the higher value of T.

In Figure 8, the best performing algorithms were the gravitational algorithms and both versions of the gravitational algorithms showed very similar type of performances. We then, further, wanted to quantify the level of improvement that we obtained with these approaches over the *PM* algorithm in their setting. We selected *UGravity* and compared it to its counterpart *PM*. This *PM* algorithm is a suitable comparison for *UGravity* because it works with uncertain data by having access to probabilities and choosing paths that maximize the probability of finding a resource. This *PM* approach depended on computing probabilities of finding parking in each block from the accessible reports.

Figure 9 shows the percentage improvement of the average time to park of *UGravity* over *PM* with differing values of penetration rate p and number of vehicles n. We observe that, as p increases,

the percentage improvement over the baseline algorithm *also* increases implying that, as more and more data are available, *UGravity* can make better choices. However, even for very small values of p, in most cases, *UGravity* was better than *PM*.

Figure 9 also shows that as the level of competition for resources increases (i.e., higher values of n), the performance of *UGravityAge* also improves since gravitational navigation algorithms are very likely to direct vehicles toward spatial areas with higher probabilities of finding resources. This is especially important when dealing with datasets with missing information. The datasets with missing data can be interpreted in a probabilistic manner, so leading agents toward areas with higher probabilities of resource availabilities is very important for a good performance of any navigation algorithm. This observation holds except when the level of competition is the highest ($n = 60$) and penetration rates are low ($p = 0.1$ or $p = 0.2$). Observe that, for this extreme case of competition, having access to data becomes *even* more important.

The simulation results discussed above show improvements of the average time to park of over 20% in some cases. We now provide estimates of the *environmental impact* of such improvements. In Ref. [43], studies conducted in 11 major cities revealed that the average time to search for curbside parking was 8.1 minutes, and cruising to find these parking slots accounted for 30% of the traffic congestion in those cities. This means that each parking slot would generate 4,927 vehicle miles traveled (VMT) per year, and, thus, the total VMT generated would be this number multiplied by the number of parking slots in the city. For example, in a big urban city like Chicago with over 35,000 curbside parking slots [49], the total number of VMT generated would be 172 *million* VMT per year due to cruising to search for parking, and would, therefore, account for a waste of 8.37 *million* gallons of gasoline and over 129,000 *tons* of CO_2 emissions. Then, with the gravitational algorithms and an improvement of over 20%, *we would be saving over* 34.4 *million VMT per year,* 1.67 *million gallons of gasoline, and over* 25,800 *tons of* CO_2 *emissions.*

11 CONCLUSION AND FUTURE WORK

In this article, we have studied a spatio-temporal matching problem in which agents are looking to obtain a resource in a transportation network. We have studied the problem by modeling the problem as a competition between the agents for the resources in various settings. We formulated the matching problem as a game and were able to compute a Nash equilibrium in a complete information context. For the incomplete information case, we presented a heuristic based on a gravitational algorithmic paradigm. We also presented two pricing schemes for this matching problem in which agents are incentivized to act in a system optimal way that is beneficial for the system and the environment. We then presented another version of the problem that has access only to limited data that could have missing information and erroneous information, where only a fraction of the agents in the transportation system report on available resources. We were able to adapt our gravitational algorithmic paradigm to this setting as well.

Finally, through simulations, we showed the effectiveness of our proposed heuristics. The simulations were based on real-world data that was obtained from the SFPark project. The simulations showed how our gravitational approaches can attain over 20% improvements over the probability maximization approaches in the uncertain case, and up to 40% when the agents have access to the ground truth data. This meant that, according to previous studies, with our navigation heuristics, we would potentially be saving up to 68.8 million vehicle miles traveled per year, 3.35 million gallons of gasoline, and over 51,600 tons of CO_2 emission (in the certain case with 40% improvement). If the pricing schemes were used to incentivize vehicles to search for the resources optimally, the improvements would be even higher.

The present work addresses matching between elements of two sets, agents and resources. The work can be extended, and has applications, in matching between elements of a single set. Practi-

cally, such situation arises in matching or grouping of passengers for the purpose of ridesharing. And, in fact, we started work along this line in Ref. [14] and [53].

Another subject of future work is to evaluate the effects of price incentives in large-scale experiments. This type of study may enable governments to estimate the potential revenue generated from the incentive scheme. These experiments can also assess the scalability of our approach. That is, in the case th at hundreds of vehicles needed to find parking space around the same time, will the computational cost be too high in large-scale optimizations?

REFERENCES

[1] D. Ayala, O. Wolfson, B. Xu, B. Dasgupta, and J. Lin. 2011. Parking slot assignment games. In *Proc. of the 19th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2011)*. ACM, 299–308.

[2] D. Ayala, O. Wolfson, B. Xu, B. DasGupta, and J. Lin. 2012. Parking in competitive settings: A gravitational approach. In *Proc. of 13th Intl. Conf. on Mobile Data Management (MDM)*. IEEE, 27–32.

[3] D. Ayala, O. Wolfson, B. Xu, B. DasGupta, and J. Lin. 2012. Pricing of parking for congestion reduction. In *Proc. 20th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS)*. ACM, 43–51.

[4] D. Ayala, O. Wolfson, B. Xu, B. DasGupta, and J. Lin. 2012. Spatio-temporal matching algorithms for road networks. In *Proc. 20th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS)*. ACM, 518–521.

[5] D. P. Bertsekas. 1990. The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces* 20, 4, The Practice of Mathematical Programming (July 1990), 133–149.

[6] F. Bock and S. Di Martino. 2017. How many probe vehicles do we need to collect on-street parking information? In *Proc. 5th IEEE Intl. Conf. on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. Napoli, Italy, 538–543.

[7] J. L. Boehlé, L. J. M. Rothkrantz, and M. van Wezel. 2008. CBPRS: A city based parking and routing system. *ERIM Report Series Reference No. ERS-2008-029-LIS* (May 2008).

[8] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. 1998. *Combinatorial Optimization*. John Wiley & Sons, New York, NY.

[9] B. DasGupta, J. P. Hespanha, J. Riehl, and E. Sontag. 2006. Honey-pot constrained searching with local sensory information. *Journal of Nonlinear Analysis: Hybrid Systems and Applications* 65, 9 (2006), 1773–1793.

[10] T. Delot, S. Ilarri, S. Lecomte, and N. Cenerario. 2013. Sharing with caution: Managing parking spaces in vehicular networks. *Mobile Information Systems* 9 (2013), 69–98.

[11] P. DeMaio. 2009. Bike-sharing: History, impacts, models of provision, and future. *Journal of Public Transportation* 12 (2009), 41–56.

[12] M. Duckham. 2013. *Decentralized Spatial Computing: Foundations of Geosensor Networks*. Springer Verlag, Berlin.

[13] E. Even-Dar, A. Kesselman, and Y. Mansour. 2003. Convergence time to Nash equilibria. In *Proc. of the 30th Intl. Conf on Automata, Languages and Programming*. Springer Verlag, 502–513.

[14] L. Foti, J. Lin, O. Wolfson, and N. Rishe. 2017. The Nash equilibrium among taxi ridesharing partners. In *Proc. 25th Intl. Conf. on Advances in Geographic Information Systems (ACM GIS)*. ACM.

[15] J. R. Frost and L. D Stone. 2001. *Review of Search Theory: Advances and Applications to Search and Rescue Decision Support*. Technical Report CG-D-15-01. U.S. Coast Guard Research and Development Center, Gronton, CT. Retrieved from http://www.rdc.uscg.gov/reports/2001/cgd1501dpexsum.pdf.

[16] D. Gale and L. S. Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15.

[17] Y. Geng and C. G. Cassandras. 2011. A new "smart parking" system based on optimal resource allocation and reservations. In *Proc. of 14th Intl. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE, 1129–1139.

[18] J. Gomez, D. Dasgupta, and O. Nasraoui. 2003. A new gravitational clustering algorithm. In *Proc. of SIAM Conf. on Data Mining (SDM)*. SIAM, 83–94.

[19] Q. Guo and O. Wolfson. 2016. Finding geospatial resources using uncertain data. In *Proc. of the 17th IEEE Intl. Conf. on Mobile Data Management (MDM 2016)*. IEEE.

[20] Q. Guo, O. Wolfson, and D. Ayala. 2015. A framework on spatio-temporal resource search. In *Proc. of the 11th IEEE Intl. Wireless Communications and Mobile Computing Conference (IWCMC 2015)*. IEEE.

[21] L. Hou, K. Mouratidis, M. L. Yiu, and N. Mamoulis. 2010. Optimal matching between spatial datasets under capacity constraints. *ACM TODS* 35, 2 (2010), 1–44.

[22] D. S. Johnson, M. Minkoff, and S. Phillip. 2000. The prize collecting Steiner tree problem: Theory and practice. In *Proc. of 11th ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 760–769.

[23] G. Josse, K. A. Schmid, and M. Schubert. 2015. Probabilistic resource route queries with reappearance. In *Proc. of 18th Intl. Conf. on Extending Database Technology (EDBT)*. 445–456.

[24] M. Kearns and Y. Mansour. 2002. Efficient Nash computation in large population games with bounded influence. In *18th Conference in Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 259–266.
[25] E. Kokolaki, M. Karaliopoulos, and I. Stavrakakis. 2011. Value of information exposed: Wireless networking solutions to the parking search problem. In *Proc. of 8th Int. Conf. on Wireless On-Demand Network Systems and Services (WONS)*. IEEE, 187–194.
[26] B. O. Koopman. 2000. *Search and Screening: General Principles and Historical Applications*. Pergamon Press, New York, NY.
[27] R. Kühne. 2003. Potential of remote sensing for traffic applications. In *Proc. of IEEE Intelligent Transportation Systems*. IEEE, 745–749.
[28] T. Lin, H. Rivano, and F. Le Mouël. 2017. A survey of smart parking solutions. *IEEE Transactions on Intelligent Transportation Systems* 18, 12 (2017), 3229–3253.
[29] R. J. Lipton, E. Markakis, and A. Mehta. 2003. Playing large games using simple strategies. In *Proc. 4th ACM Conf. on Electronic Commerce*. ACM, 6–41.
[30] S. Ma, O. Wolfson, and B. Xu. 2014. Updetector: Sensing parking/unparking activities using smartphones. In *Proc. of 7th Int. Workshop on Computational Transportation Science (IWCTS)*.
[31] S. Mathur, T. Jin, N. Kasturirangan, J. Chandrashekharan, W. Xue, M. Gruteser, and W. Trappe. 2010. ParkNet: Drive-by sensing of road-side parking statistics. In *MobiSys*. ACM, 123–136.
[32] R. Meir, Y. Chen, and M. Feldman. 2013. Efficient parking allocation as online bipartite matching with posted prices. In *Proc. of the 12th Intl. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'13)*. St. Paul, MN.
[33] J. Nash. 1950. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences* 36, 1 (1950), 48–49.
[34] S. Nawaz, C. Efstratiou, and C. Mascolo. 2013. Parksense: A smartphone based sensing system for on-street parking. In *Proc. of the 19th Annual Int. Conf. on Mobile Computing and Networking (MobiCom)*. ACM, 75–86.
[35] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. 2007. *Algorithmic Game Theory*. Cambridge University Press, New York, NY.
[36] B. Panja, B. Schneider, and P. Meharia. 2011. Wirelessly sensing open parking spaces: Accounting and management of parking facility. In *Proc. of the 17th Americas Conf. on Information Systems*. Association for Information Systems, Detroit, Michigan.
[37] C. H. Papadimitriou and K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, New Jersey.
[38] W. Park, B. Kim, D. Seo, D. Kim, and K. Lee. 2008. Parking space detection using ultrasonic sensor in parking assistance system. In *IEEE Intelligent Vehicles Symposium*. IEEE, 1039–1044.
[39] S.-Y. Phang and R. S. Toh. 2004. Road congestion pricing in Singapore: 1975 to 2003. *Transportation Journal* 43, 2 (2004), 16–25.
[40] http://bit.ly/2nsUhEN. 2018. (January 2018).
[41] http://sfpark.org/. 2018. (January 2018).
[42] E. Rasmusen. 2006. *Games and Information* (4th ed.). Blackwell Publishing, Hoboken, New Jersey.
[43] D. Shoup. 2005. *The High Cost of Free Parking*. American Planning Association, Chicago, IL.
[44] D. Shoup. 2006. Cruising for parking. *Transport Policy* 13 (2006), 479–486.
[45] D. Simons. Winter 2012. SFPark: San Francisco knows how to park it. *Sustainable Transport* 23 (Winter 2012), 26–27.
[46] L. Stenneth, O. Wolfson, B. Xu, and P. S. Yu. 2012. PhonePark: Street parking using mobile phones. In *Proc. of 13th Intl. Conf. on Mobile Data Management (MDM)*. IEEE, 278–279.
[47] P. Szczurek, B. Xu, J. Lin, and O. Wolfson. 2010. Spatio-temporal information ranking in VANET applications. *Intl. J. of Next-Generation Computing* 1, 1 (July 2010), 52.
[48] M. Tilly and S. Reiff-Marganiec. 2011. Matching customer requests to service offerings in real-time. In *Proc. of the 2011 ACM Symposium on Applied Computing*. 456–461.
[49] Transportation Alternatives (www.transalt.org) 2008. *Pricing the Curb: How San Francisco, Chicago and Washington D.C. are Reducing Traffic with Innovative Curbside Parking Policy*. Transportation Alternatives (www.transalt.org), New York, NY.
[50] V. Verroios, V. Efstathiou, and A. Delis. 2011. Reaching available public parking spaces in urban environments using ad-hoc networking. In *Proc. of 12th Intl. Conf. on Mobile Data Management (MDM)*. IEEE, 141–151.
[51] B. W. Wie and R. L. Tobin. 1998. Dynamic congestion pricing models for general traffic networks. *Transportation Research Part B - Methodological* 32, 5 (1998), 313–327.
[52] O. Wolfson and J. Lin. 2014. A marketplace for spatio-temporal resources and truthfulness of its users. In *Proc. of the 7th ACM SIGSPATIAL Intl. Workshop on Computational Transportation Science*. ACM, 1–6.
[53] O. Wolfson and J. Lin. 2017. Fairness versus optimality in ridesharing. In *Proc. of the 18th IEEE Intl. Conf. on Mobile Data Management (MDM 2017)*. IEEE, 118–123.

[54] O. Wolfson and B. Xu. 2004. Opportunistic dissemination of spatio-temporal reseource information in mobile peer-to-peer networks. In *Proc. of the 1st Intl. Workshop on P2P Data Management, Security and Trust (PDMST'04), DEXA Workshops 2004*. IEEE, 954–958.

[55] O. Wolfson, B. Xu, and H. Yin. 2004. Dissemination of spatio-temporal information in mobile networks with hotspots. In *Proc. of the 2nd Intl. Workshop on Databases, Information Systems, and Peer-to-Peer Computing*. Springer-Verlag, 185–199.

[56] W. E. Wright. 1977. Gravitational clustering. *Pattern Recognition* 9 (1977), 151–166.

[57] B. Xu, O. Wolfson, J. Yang, L. Stenneth, P. S. Yu, and P. C. Nelson. 2013. Real time street parking availability estimation. In *Proc. of 14th Intl. Conf. on Mobile Data Management (MDM)*. IEEE, 16–25.

[58] Bo Zou, Nabin Kafle, Ouri Wolfson, and Jie Lin. 2015. A mechanism design based approach to solving parking slot assignment in the information era. *Transportation Research Part B*, 81, Part 2 (2015), 631–653.

Received April 2017; revised November 2017; accepted January 2018