Spatio-temporal Matching for Urban Transportation Applications

by

Daniel Ayala
B.S. (Universidad de Puerto Rico, Recinto de Río Piedras) 2003
M.C.S. (University of Illinois at Urbana-Champaign) 2008

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2017

Chicago, Illinois

To my parents,

Harry and Myriam,

who have blessed me in every way.

ACKNOWLEDGMENTS

Reflecting on the years of work towards this degree, I am reminded of the warning that is given in the book of Ecclesiastes, Chapter 12:

"*Of making many books there is no end, and much study wearies the body.*"

I have experienced firsthand this weariness that the Preacher warns us about and it just makes me feel even more appreciative of everyone who has supported me along the way.

I would first off like to thank my advisor Dr. Ouri Wolfson, for all his time and effort in supporting and advising me. The same gratitude goes to my co-advisors Dr. Bhaskar DasGupta and Dr. Jie Lin. I am also indebted to Dr. Bo Xu for all of his help during my research. They all are great scholars who are an example to me. I am thankful to Dr. Tanya Berger-Wolf and Dr. A. Prasad Sistla for being part of my committee and offering their guidance.

I would also like to thank all my friends that were of great support to me throughout these years. Listing all of them would be a futile endeavor and would be unfair to the ones that I would mistakenly leave out. So here I would say that I am deeply grateful to anyone who has given me a helping hand when I felt down and to anyone who has brought me joy and made me smile when I needed it most. I am also thankful for my church families in Puerto Rico and in Chicago, who I know have supported me in prayer and have given me much needed strength and faith.

Finally, I am most thankful for my family. My parents and sister are a foundation of love, faith, strength, and support without which I would not have finished this work.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

POI	Point-of-Interest
GPS	Global Positioning System
P2P	Peer-to-Peer
MCNF	Minimum Cost Network Flow
POA	Price of Anarchy
GPA	Gravitational Parking Algorithm
DM-GRA	Deterministic Magnitude Gravitational Algorithm

SUMMARY

Vehicles nowadays are increasingly being equipped with computers, sensors, and communication capabilities. These devices on cars lead to the rise of novel applications that aid travelers in their daily travels. The vehicle itself becomes a mobile device and tool that can be used in many different ways. The vehicle has the capability of receiving updates on the driving conditions around it or data about locations across the city. With this updated traffic information travelers can be assisted to make optimal decisions while driving. For example, the driver can be assisted in choosing an optimal route to his/her destination or could search for places of interest around the vehicle's current location.

The urban transportation system as a whole can also be improved by using the computing capabilities that vehicles have. Vehicles can act as sensors that collect data and share with the system. City and transportation planning agencies can use this data to aggregate and share with other users or to make assessments on how the system is being used. Systems can also be implemented that guide or incentivize vehicles to make decisions that benefit the transportation system. For example, a city can implement congestion pricing on roads to reduce congestion on specific areas of a city. Or in emergency situations, vehicles can be guided through an evacuation planning system to be able to drive outside of an area in an organized and timely manner.

In this work we will study such an application. We will study the very common transportation problem of searching for resources in an urban space. The problem has been formulated as a spatio-temporal matching problem in which agents are looking to obtain a resource in a transportation network. We have studied the problem by modelling 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.

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 up to 25% improvements over other approaches that use probability maximization 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).

CHAPTER 1

INTRODUCTION

The problem addressed in this work 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 (1). 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 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 dissertation, depending on on the information that can be obtained by the mobile agents. We illustrate these scenarios below with the help of Figure 1.

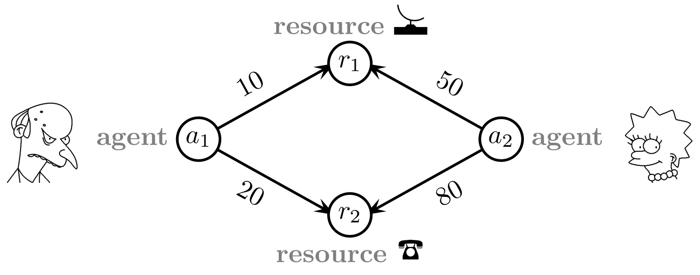


Figure 1. An example of two mobile agents a_1 and a_2 and two stationary resources r_1 and r_2. The numbers represent (in seconds) the travel times for the agents to reach the corresponding resources.

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 (2; 3; 4).

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 (5; 6). 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 find resources. Thus, in 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 towards 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*, *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 minimize 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 and Outline

In this work, 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 dissertation is organized as follows.

- In Chapter 2, we summarize prior research works related to the topics studied in this work.
- In Chapter 3, we provide notations and preliminary definitions related to the methods and algorithms studied subsequently.

- In Chapter 4, we define the system optimal model and describe an algorithm for computing a system optimal assignment.
- In Chapter 5, we provide basic concepts and terminologies for a game theoretic setup of our problems.
- In Chapter 6, we discuss our results on computing a deterministic Nash Equilibrium strategy profile for the complete and incomplete scenarios for our search problem.
- In Chapter 7, we discuss the gap that exists between the System Optimal and Nash Equilibrium formulations of our problem.
- In Chapter 8, we discuss our results on designing mechanisms for resource pricing schemes that apply to the complete information with currency exchange scenario.
- In Chapter 9, we present distributed algorithms for a practical setting of the incomplete information scenario by using a gravitational paradigm that navigates the agents towards resources in an efficient manner.
- In Chapter 10, we present algorithms in the context of uncertain and missing data about resource availability.
- In Chapter 11, we present our simulation methodologies that are based on real-world data, and the results of the evaluations of our algorithms.
- Finally, in Chapter 12 we present some concluding remarks.

CHAPTER 2

PRIOR RELATED WORK

Approaches for monitoring and sensing available spatial objects are commonplace nowadays due in part to the proliferation of GPS-enabled mobile devices. Information about the locations of these resources can be obtained via collection of the data using client-server architectures (7) or via decentralized data dissemination (8). In (7), 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 (8) for spatial resource discovery in a decentralized fashion.

Our work in this dissertation focuses on navigation towards 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 (9), ultrasonic sensor technology is used to determine the spatial dimensions of open parking slots, whereas wireless sensors are used in (10) to track open parking slots in a parking facility. These two works show how one can *detect* open slots. In (11), 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 citywide) services. For example, by classifying mobility patterns of smartphone users (12) or using Wi-Fi signature matching (13), parking and unparking activities for individual smartphone users and hence vehicles can be detected. In (14), 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 work, we assume that vehicles can receive information about open parking slots at any time using one of these parking availability detection approaches.

The *value* of having location information of spatial resources like parking was tested in a P2P environment in (15). Kokolaki *et al.* show through simulations how vehicles with access to data about available resources have an advantage over vehicles that do not. In (16), 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 (17). In (17), 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 work. Furthermore, the focus of (17) was on peer-to-peer (P2P) dissemination of parking reports.

Wireless ad-hoc networking was also used in (18) to search for open parking slots. The authors in (18) 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 (17), their approach depended on knowing the probability that the parking slot will 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 a using a framework of the so-called prize-collecting Traveling Salesperson problem (19).

In (20; 21; 22), reservation systems for parking slots were studied. A centralized reservation system was presented in (20) in which a server collects information from road-side units and other vehicles and reserves slots for vehicles. In (21), 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 (22), 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 (23), 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.

In (24), 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 Chapter 4. The authors of (24) 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 Chapter 4.

Pricing of resources to obtain some system-wide objectives, studied in this work, was also studied in the past in other contexts of transportation applications. In the transportation literature this is commonly known as “congestion pricing” (25). 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 similar type has been famously implemented in the central business district of Singapore (26) and in other major cities across the world. This work investigates the pricing problems in the context of algorithmic game theory which has a rich history (*e.g.*, see textbooks such as (27)).

Some of our algorithms in this work 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 (28; 29) such models were used for clustering data in the Euclidean space.

CHAPTER 3

BASIC NOTATIONS AND PROBLEM SETUP

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

▶ Throughout this work, 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 $\Xi \overset{\text{def}}{=} r_{m+1}$. It is commonplace to use such dummy items in algorithmic game theory applications to simplify description and case analysis, *e.g.*, see (27, 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,

[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 description of algorithms that manipulate them.

- ▷ 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* $\ell_i^a \overset{\text{def}}{=} \ell_i^{\mathcal{A}}\left(t_i^{\mathcal{A}}\right)$, 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.

For the dummy resource $\Xi \overset{\text{def}}{=} r_{m+1}$, $t_{\Xi}^{\mathcal{R}} \overset{\text{def}}{=} t_{m+1}^{\mathcal{R}} = 0$, and $\ell_{\Xi}^{\mathcal{R}} \overset{\text{def}}{=} \ell_{m+1}^{\mathcal{R}}$ 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.* $\max\{t_i^{\mathcal{A}} + \mathit{time}(\ell_i^a, \ell_j^{\mathcal{R}}), t_j^{\mathcal{R}}\}$, 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: $\mathcal{M} \times \mathcal{M} \mapsto \mathbb{R}^+$ $\mathit{time}(x, y)$ 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 $\mathit{time}\left(\ell_i^{\mathcal{A}}(t), \ell_{\Xi}^{\mathcal{R}}\right) = \mho$ for any agent $a_i \in \mathcal{A}$ and for any time t, where $\mho \gg \sum_{i=1}^{n} \sum_{j=1}^{m} \mathit{time}\,(a_i, r_j)$ 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 "$g(a_i) = \Xi \overset{\text{def}}{=} r_{m+1}$ " or by saying that "*a_i was assigned to* Ξ " or also by saying that "*a_i was not assigned*".

***Cost function* cost**: $\mathcal{A} \times \mathcal{R} \times \mathbb{R}^+ \mapsto \mathbb{R}$ $\mathit{cost}\,(a_i, r_j, t)$ 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 $\ell_i^{\mathcal{A}}(t)$ and $\ell_j^{\mathcal{R}}$ of a_i and r_j. The exact value of the cost is obviously dependent on the application being considered[1]. 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.

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

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

CHAPTER 4

SYSTEM OPTIMAL SPATIO-TEMPORAL MATCHING (CENTRALIZED MODEL)

In this chapter 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 $\sum_{i=1}^{n} \mathscr{cost}\left(a_i, g\left(a_i\right), t_i^{\mathcal{A}}\right)$. 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 1. *A mapping g that minimizes $\sum\limits_{i=1}^{n} \mathscr{cost}\left(a_i, g\left(a_i\right), t_i^{\mathcal{A}}\right)$ can be computed in $O\big((n+m)^3\big)$ 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 (30)). 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 $\{\,(u, a_i)\,|\,1 \le i \le n\,\} \cup \{\,(r_j, v)\,|\,1 \le j \le m\,\}$, each of zero cost and capacity 1.

Letting $\mathsf{flow}\colon E \mapsto \mathbb{R}^+$ 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 $\sum_{i=1}^{n} \mathsf{flow}\,(\,(u, a_i)\,) = \sum_{j=1}^{m} \mathsf{flow}\,(\,(r_j, v)\,) = \min\{m, n\}$ *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 (31)), and therefore, due to capacity constraints, $\mathsf{flow}(e)$ is 0 or 1 for any edge $e \in E$. The desired mapping g can now be easily computed as

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

□

Remark 2. *Since our $\mathcal{cost}$ 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* $\mathcal{cost}(a_i, g(a_i), t_i^{\mathcal{A}}) = \mathcal{time}\left(\ell_i^a, \ell_{g(a_i)}^{\mathcal{R}}\right)$.

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 is easily captured in our formulation by setting:*

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

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*

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

CHAPTER 5

GAME THEORETIC SETUP – BASIC CONCEPTS AND TERMINOLOGIES

System optimal agent-resource assignments, as discussed in Chapter 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 (32). 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 work, we will consider only *deterministic* strategies for agents. A (deterministic) strategy for an agent a_i will be denoted by the variable $\mathscr{r}_i \in \mathcal{R} \cup \{\Xi\}$ with the following convention:

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

The vector of variables $\overrightarrow{\mathscr{r}} = (\mathscr{r}_1, \mathscr{r}_2, \ldots, \mathscr{r}_n)$ is called a *strategy profile vector*; a specific *strategy profile* is obtained from this vector by assigning a specific value to each $\mathscr{r}_j$ for $1 \leq j \leq n$.

▷ The *payoffs* (costs) for each player (agent) in this game can be defined by the $\mathscr{cost}$ function introduced in Section 3. Let $\mathcal{S} = (\mathscr{r}_1, \mathscr{r}_2, \ldots, \mathscr{r}_n) \in (\mathcal{R} \cup \{\Xi\})^n$ be a strategy profile chosen by the players, *i.e.*, $\mathscr{r}_i \in \mathcal{R} \cup \{\Xi\}$ 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 $\mathscr{cost}_{\mathcal{S}}(a_i)$, is as follows:

- $\mathscr{cost}_{\mathcal{S}}(a_i) = \mathscr{cost}\left(a_i, \mathscr{r}_i, t_i^{\mathcal{A}}\right)$ if and only if agent a_i would obtain the resource $\mathscr{r}_i$ *before*[1] any other agent a_j that also opted for this resource, *i.e.*, for any a_j such that $\mathscr{r}_j = \mathscr{r}_i$.
- Otherwise, $\mathscr{cost}_{\mathcal{S}}(a_i) = \mho$ for a sufficiently large positive number $\mho \gg \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{m} \mathscr{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right)$.

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

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 $\mathcal{S} = (\mathscr{r}_1, \mathscr{r}_2, \ldots, \mathscr{r}_n)$, let $\mathcal{S}_{-i}$ denote the set of strategy profiles $\Big\{ (\mathscr{r}_1, \mathscr{r}_2, \ldots, \mathscr{r}_{i-1}, \mathscr{r}_\alpha, \mathscr{r}_{i+1}, \ldots, \mathscr{r}_n) \mid \mathscr{r}_\alpha \neq \mathscr{r}_i \Big\}$. The standard definition of a *deterministic* Nash equilibrium translates to the following definition for our spatio-temporal matching problem.

Definition 3 (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:*

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

[1]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.

For the remainder of the dissertation, *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.

CHAPTER 6

COMPUTATION OF A DETERMINISTIC NASH EQUILIBRIUM STRATEGY PROFILE

In this chapter, 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 chapter.

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 Chapter 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 (34; 35) because of introduction of parameters such as $t_i^{\mathcal{A}}$, $\ell_i^{\mathcal{A}}(t)$ and $t_j^{\mathcal{R}}$. These 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 4 (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*[1] *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 $\mathcal{S}^g = (\mathfrak{s}_1^g, \mathfrak{s}_2^g, \ldots, \mathfrak{s}_n^g)$ where $\mathfrak{s}_i^g = g(a_i)$. The Gale-Shapley deferred acceptance algorithm in (36) 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 $x_{a_i,r} = \mathrm{cost}\left(a_i, r, t_i^{\mathcal{A}}\right)$ 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

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

---

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

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

- ▷ (*resource preference lists*) Let $y_{a_i,r_j} = t_i^{\mathcal{A}} + \mathsf{time}(\ell_i^a, \ell_j^{\mathcal{R}})$ 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 $\prec_{r_j} \overset{\text{def}}{=} a_{j_1} \prec_{r_j} a_{j_2} \prec_{r_j} \cdots \prec_{r_j} a_{j_n}$ for a resource $r_j \in \mathcal{R}$ is defined by

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

  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 5 (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 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 $\mathcal{S}^g = (\mathfrak{r}_1^g, \mathfrak{r}_2^g, \ldots, \mathfrak{r}_n^g)$ is *not* a Nash equilibrium strategy profile. Then there exists an index i and a strategy $\mathcal{T} = (\mathfrak{r}_1, \mathfrak{r}_2, \ldots, \mathfrak{r}_{i-1}, \mathfrak{r}_\alpha, \mathfrak{r}_{i+1}, \ldots, \mathfrak{r}_n) \in \mathcal{S}_{-i}^g$ such that $\mathfrak{cost}_{\mathcal{S}}(a_i) > \mathfrak{cost}_{\mathcal{T}}(a_i)$. Note that $\mathfrak{cost}_{\mathcal{S}}(a_i) > \mathfrak{cost}_{\mathcal{T}}(a_i)$ implies $\mathfrak{r}_\alpha \neq \Xi$ and also implies that agent a_i must be able to obtain the resource $\mathfrak{r}_\alpha$ 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 $\mathfrak{r}_\alpha$ before any other agent that also opted for this resource, $y_{a_i,\mathfrak{r}_\alpha} < y_{g^{-1}(\mathfrak{r}_\alpha),\mathfrak{r}_\alpha}$, which in turn implies

$$a_i \prec_{g(g^{-1}(\mathfrak{r}_\alpha))} g^{-1}(\mathfrak{r}_\alpha) \tag{6.1}$$

- Since $\mathfrak{cost}_{\mathcal{T}}(a_i) < \mathfrak{cost}_{\mathcal{S}}(a_i)$, $x_{a_i,\mathfrak{r}_\alpha} < x_{a_i,\mathfrak{r}_i}$, which in turn implies

$$g\left(g^{-1}(\mathfrak{r}_\alpha)\right) = \mathfrak{r}_\alpha \prec_{a_i} \mathfrak{r}_i = g(a_i) \tag{6.2}$$

If we consider the matched pairs $(a_i, g(a_i))$ and $\left(g^{-1}(\mathfrak{r}_\alpha), \mathfrak{r}_\alpha\right)$, relations (Equation 6.1) and (Equation 6.2) provide a violation of the stability of the matching (*c.f.* Definition 4). □

6.1.1 Examples and Discussion

Here we demonstrate the use of the last theorem and the fact that the introduction of different starting times changes 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 (34). 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.*,

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

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:

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

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

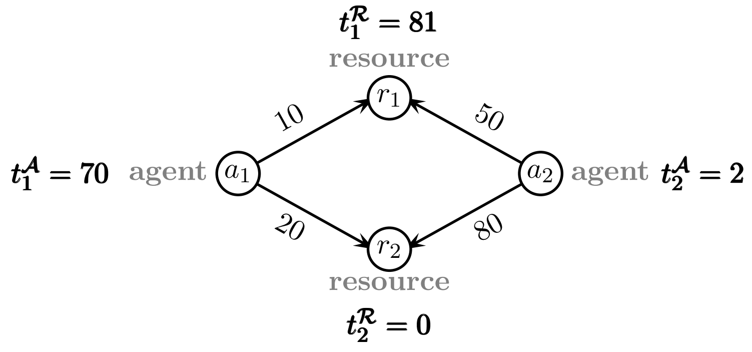


Figure 2. An example of two mobile agents a_1 and a_2 and two stationary resources r_1 and r_2. The numbers on edges represent (in seconds) the travel times for the agents to reach the corresponding resources. $t_1^{\mathcal{A}}, t_2^{\mathcal{A}}, t_1^{\mathcal{R}}$ and $t_2^{\mathcal{R}}$ are the start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

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 1 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 (34). 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 straight-forward adaptation of such an algorithm would not work in the dynamic context, i.e., with start times, regardless of the $\mathfrak{cost}$ 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 (34) 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 to minimize its *expected* cost conditional on the above-mentioned prior distributions. This will be further discussed in chapter 9.

CHAPTER 7

STUDYING THE GAP BETWEEN THE SYSTEM OPTIMAL AND NASH EQUILIBRIUM FORMULATIONS

In this chapter we will explore the gap that exists between the system optimal and Nash Equilibrium formulations of our problems. We will present a theoretical study of this gap, by means of the computation of the *price of anarchy*, and a practical study that was performed through simulations.

7.1 Theoretical Gap

The *price of anarchy* (POA) of a game is the ratio of the total cost paid in the equilibrium assignment over the total cost paid by the players in the assignment that minimizes the social welfare (system optimal assignment) (27). The POA of the class of games defined in chapter 5 will be the largest such ratio that could be found for an instance that type of game. In this chapter we show that the POA of these games with complete information is unbounded. This is a fundamental result which indicates that for the resource search problem studied here, when travelers are selfish and make their own choices, costs can get arbitrarily worse than the optimal assignment. More specifically, the ratio between the costs of the equilibrium and system optimal assignments can grow unboundedly as the size of the problem increases.

Theorem 6. *The Price of Anarchy in the Complete Information Resource-search game is unbounded.*

Proof. Consider a game with n agents and n available resources. Let $D = \{d_{ij}\}$ form a matrix, where the entry $d_{ij} = \mathcal{cost}(a_i, r_j, t_i^{\mathcal{A}})$.

Let the cost function be defined by the following matrix:

$$\{\mathcal{cost}(a_i, r_j, t_i^{\mathcal{A}})\} = \begin{bmatrix} n & 2n & \cdots & n \cdot n \\ n^2 & 2n^2 & \cdots & n \cdot n^2 \\ \vdots & \vdots & \ddots & \vdots \\ n^n & 2n^n & \cdots & n \cdot n^n \end{bmatrix}$$

Then $\mathcal{cost}(a_i, r_j, t_i^{\mathcal{A}}) = jn^i$. Any matching in this game will lead to a total cost (adding the costs of all players) of some polynomial of degree at least n. In this case, also assume that $\mathcal{time}(\ell_i^{\mathcal{A}}, \ell_j^{\mathcal{R}}) = \mathcal{cost}(a_i, r_j, t_i^{\mathcal{A}})$.

The equilibrium assignment will be one that is obtained by performing the Gale-Shapley Algorithm. In this special case, the smallest entry, d_{ij}, in the matrix will be computed and a_i will be assigned to resource r_j. Then that column and row will be removed from the matrix and the smallest entry will be computed in this updated matrix. It's clear that the polynomial that is obtained from this algorithm will be the one that adds up the diagonal entries of matrix A. Then the equilibrium assignment will have a total cost of:

$$EQ = \sum_{k=0}^{n-1} (n-k)n^{n-k} = n^{n+1} + \sum_{k=1}^{n-1} (n-k)n^{n-k} \tag{7.1}$$

Then, the equilibrium solution leads to a total cost that is given by a polynomial of degree $n+1$.

Now consider an assignment that tries to choose the smallest distance for the larger cost agents. So then for a_n, resource r_1 will be assigned so as to minimize the large cost factor that it has of n^n. This assignment chooses the diagonal that starts on position d_{n1} and ends in position d_{1n}. For this special assignment, a_i is assigned slot r_{n-i+1}. Let the cost of this assignment be defined as X. Then:

$$X = \sum_{k=0}^{n-1} (k+1)n^{n-k} = n^n + \sum_{k=1}^{n-1} (k+1)n^{n-k} \tag{7.2}$$

If we take the ratio of EQ/X then we get a polynomial of degree 1 which is a function of n. Let OPT be the cost of the assignment that optimizes the social welfare. By definition $OPT \leq X$. Therefore $EQ/X \leq EQ/OPT$ and then EQ/OPT will be greater than some function of n. Therefore, the POA given by the EQ/OPT ratio will be unbounded because an instance of our game can be constructed by the previous construction that will be larger than any proposed bound. $\square$

The price of anarchy is also used for computing bounded approximations to problems in a distributed manner when the POA is bounded (*e.g.* (37; 27)). When the POA is bounded, individual agents using Nash equilibrium strategies compute an approximation for the optimal problem with the POA being the approximation ratio. The fact that the POA is unbounded makes this technique impossible for this problem.

7.2 Practical Gap

In this section we use simulations to determine empirically, on average, the benefits of system optimal matching over nash equilibrium matchings.

7.2.1 Simulation Setup

The goal of our simulation is to empirically ascertain how much better off is the transportation system and the environment, when agents make choices in a system optimal manner.

Our simulations were run on a road network (grid) that was embedded in an Euclidean space. The positions of the agents, the motion of the agents and locations of the resources were restricted to be on the network. The number of agents (n) was a parameter of the simulation. The values of n that were tested were $n = 25k$ for $k = \{1, 2, \ldots, 12\}$. A system optimal assignment and a Nash equilibrium were computed for each configuration of agents and resources, and the total distance traveled by all the agents based on these two assignment was saved. This means that the cost for these simulations was simply the driving distance ($\mathit{time}(\ell_i^{\mathcal{A}}, \ell_j^{\mathcal{R}})$) between the agents and the resources. This choice was made to determine what are the environmental benefits that are obtained from system optimal agent movement.

We also tested a varying number of *competitive ratios*. Say now that there are m available resources and n agents. Then define the competitive ratio as n/m. The higher the competitive ratio, the bigger the competition for the available resources.

We also tested a varying number of regional skews of the locations of resources. In reality, available resources are not uniformly distributed across a road network. Thus, we generate skewness as follows. The map is partitioned into 16 equal-sized square regions. A random permutation of the regions is generated (uniform distribution) and is used as the ranking of the popularity of each region for available resources. To choose the location of each of the available resources, first a random number is generated to determine in which region to place the slot. The Zipf distribution with its skew parameter and the regional popularity previously generated are used to generate this random number. Then a random position in the grid (uniform) is chosen from the region denoted by the Zipf number. The n agents' initial positions are generated using the uniform distribution on the grid.

For each value of simulation configuration under consideration, 1000 different simulation runs were tested and averaged. Each test was done to compute what is basically an average *price of anarchy* (PoA). PoA is the ratio of the total cost in the Nash equilibrium assignment to the total cost of the system optimal assignment (27). In these simulations we computed the average PoA to determine the average benefit of system optimal agent choices.

The parameters for the simulation are:

- n - the number of agents.
- n/m - the competitive ratio between agents and resources.
- *skew* - the regional skew of the Zipf distribution.

The values that were tested for each parameter are detailed in table Table I. For each configuration of the parameters, 100 different simulation runs were generated and tested.

7.2.2 Simulation Results

Figure 3 shows the results for the average POA computation for various values of n. It also shows the results for varying values of competitive ratio (n/m).

Parameter	**Symbol**	**Range**
Agents	n	{25,50,...,275,300}
Competitive Ratio	n/m	$\{2,\frac{4}{3},1\}$
Regional Skew	-	{0, 1, 2, 3}

TABLE I

PARAMETERS TESTED ON PRICE OF ANARCHY SIMULATION

We can see that the highest values of average POA were attained when the competitive ratio was lowest ($n/m = 1$). This means that as the availability of resources increases, so does the congestion cost incurred by the system based on the Nash equilibrium assignment compared to the System optimal assignment.

We can see that at $n = 300$ and $n/m = 1$, the average POA that was obtained was around 1.3. This was the highest POA obtained in all simulations. This means that for every mile traveled by each agent with the system optimal assignment, the agents will travel 1.3 miles when using the Nash equilibrium assignment on an average. That in turn means that the percent improvement of the system optimal assignment can be up to $1 - 1/1.3 \approx 23\%$.

Figure 4 shows some results with varying skew. We can also see that the highest values of average POA were obtained when the skew was 0, *i.e.* slots were distributed uniformly across the road network.

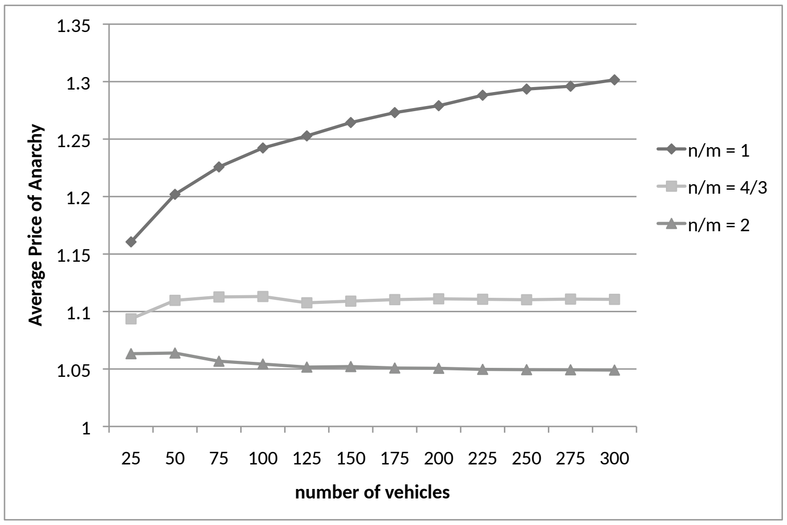


Figure 3. Average POA with varying values of n and varying competitive ratio (skew = 0).

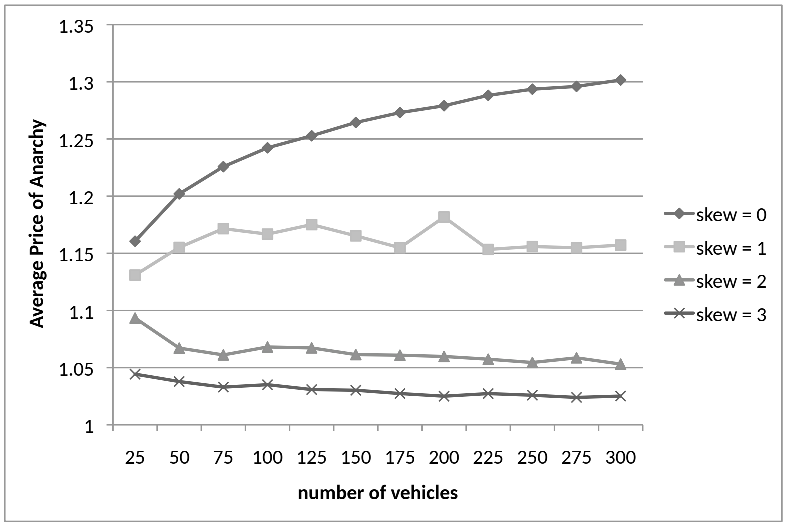


Figure 4. Average POA with varying values of n and varying regional skew (n/m = 1).

CHAPTER 8

DESIGNING PRICING MECHANISMS FOR COMPLETE INFORMATION WITH CURRENCY EXCHANGE SCENARIO

In chapter 6, 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 (38), 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 (39). For example, in a big urban city like Chicago with over $35,000$ curbside parking slots (40), 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* towards system optimal matchings. The results from chapter 7 show that there is a clear gain in having agents make choices in a system optimal way, rather than a selfish way. However, as has been discussed before, this is difficult to justify in practice because of the individual costs that the 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?*

8.1 Mechanism Design for Agent-Independent Resource Pricing Scheme

In this subsection we address the following question: How to 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 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.*,

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

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 $g_{\mathcal{P}}\colon \mathcal{A} \mapsto \mathcal{R}$ that matches each agent a_i to a resource $g_{\mathcal{P}}\left(a_i\right)$ such that:

$$\mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) = \min_{1 \le k \le m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} \tag{8.1}$$

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

$$\sum_{i=1}^{n} \mathcal{cost}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \mathsf{OPT} + \Delta \tag{8.2}$$

If, for example, the cost function $\mathcal{cost}$ 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 8.1) by introducing the notion of a ε-approximate equilibrium in a manner similar to that in the algorithmic game theory community (*e.g.*, see (41; 42; 43)). 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

$$\mathcal{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \min_{1 \leq k \leq m} \left\{ \mathcal{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon \tag{Equation 8.1'}$$

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 condition (Equation Equation 8.1') holds for *all* agents with their matched and obtained resources. For notational convenience, let $\mu = \max_{i,j} \left\{ \mathcal{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right) \right\}$. We prove the following result.

Theorem 7. *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 $g_{\mathcal{P}} \colon \mathcal{A} \mapsto \mathcal{R}$ in $O\left(n^2\mu/\varepsilon\right)$ time that satisfy both the following:*

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

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

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

Proof. Our algorithm is based on the auction algorithm in (44). 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 (44) 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

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

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

This algorithm continues in a sequence of rounds until all agents are at ε-almost equilibrium. The complete algorithm is shown in Figure 5. 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 (44) shows that the auction algorithm in Figure 5 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 $g_{\mathcal{P}}$

```
(* initialization *)
    set $p_j \leftarrow 0$ for all $1 \leq j \leq m$
    let $g_{\mathcal{P}} \colon \mathcal{A} \mapsto \mathcal{R}$ be an arbitrary agent-resource matching
(* rounds of auctions *)
    while not all agents are at $\varepsilon$-almost equilibrium do
        let $i$ be an index such that $\mathit{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}(a_i), t_i^{\mathcal{A}}\right) > \min_{1 \leq k \leq m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon$
    (* compute bid increment *)
        let $\alpha \leftarrow \mathit{cost}_{\mathcal{P}}\left(a_i, r_j, t_i^{\mathcal{A}}\right) = \min_{1 \leq k \leq m} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\}$
        let $\beta \leftarrow \mathit{cost}_{\mathcal{P}}\left(a_i, r_q, t_i^{\mathcal{A}}\right) = \min_{\substack{1 \leq k \leq m \\ k \neq j}} \left\{ \mathit{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\}$ for some $q \neq j$
        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
```

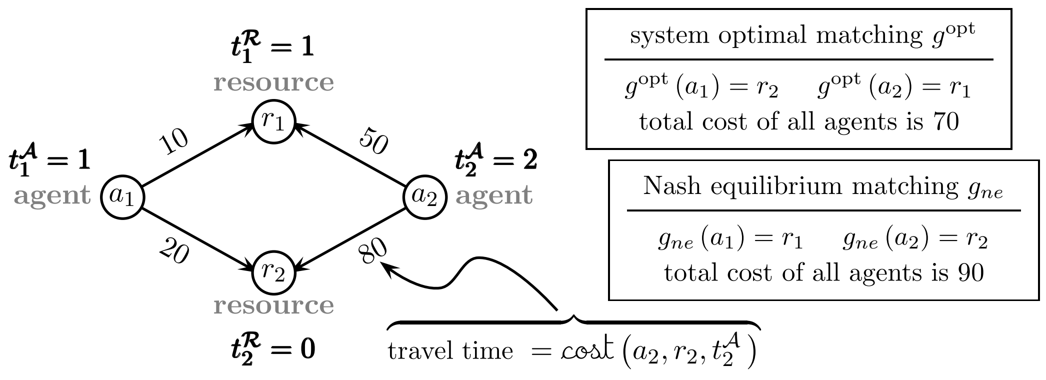
Figure 5. The algorithm for pricing resources in Theorem 7, based on the auction algorithm in (44).

which proves condition (♠). Thus, it only remains to prove condition (♠♠). Using condition (♠) of the auction algorithm and simple algebraic manipulation, we get

$$
\begin{aligned}
& \forall\, i\colon \mathcal{cost}_{\mathcal{P}}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \min_{1\leq k\leq m}\left\{ \mathcal{cost}_{\mathcal{P}}\left(a_i, r_k, t_i^{\mathcal{A}}\right) \right\} + \varepsilon \\
\Longrightarrow\; & \forall\, i\colon \mathcal{cost}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) + p_{g_{\mathcal{P}}(a_i)} \leq \mathcal{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right) + p_{g^{\text{opt}}(a_i)} + \varepsilon \\
\Longrightarrow\; & \sum_{i=1}^{n}\left[\mathcal{cost}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) + p_{g_{\mathcal{P}}(a_i)}\right] \leq \sum_{i=1}^{n}\left[\mathcal{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right) + p_{g^{\text{opt}}(a_i)} + \varepsilon\right] \\
\Longrightarrow\; & \sum_{i=1}^{n}\mathcal{cost}\left(a_i, g_{\mathcal{P}}\left(a_i\right), t_i^{\mathcal{A}}\right) \leq \sum_{i=1}^{n}\left[\mathcal{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right)\right] + n\,\varepsilon = \mathsf{OPT} + n\,\varepsilon \\
& \qquad\qquad\qquad \text{since } \sum_{i=1}^{n} p_{g_{\mathcal{P}}(a_i)} = \sum_{i=1}^{n} p_{g^{\text{opt}}(a_i)} = \sum_{j=1}^{n} p_j
\end{aligned}
$$

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

Remark 8 (Primal-dual interpretation of the auction algorithm)**.** *Readers familiar with the primal-dual approach for solving linear programs by iteratively satisfying complementary slackness conditions (30) will realize that the auction algorithm in Figure 5 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 (27, Section 11.3.1) by, for example, setting all the dual variable (prices) to zeroes and iteratively increase dual variables until all complementary slackness conditions are satisfied.*



$t_1^{\mathcal{R}} = 1$
resource
r_1
$t_1^{\mathcal{A}} = 1$
agent
a_1
10
20
50
80
$t_2^{\mathcal{A}} = 2$
a_2
agent
r_2
resource
$t_2^{\mathcal{R}} = 0$
travel time $= \mathtt{cost}\left(a_2, r_2, t_2^{\mathcal{A}}\right)$
system optimal matching g^{opt}
$g^{\text{opt}}(a_1) = r_2 \quad g^{\text{opt}}(a_2) = r_1$
total cost of all agents is 70
Nash equilibrium matching g_{ne}
$g_{ne}(a_1) = r_1 \quad g_{ne}(a_2) = r_2$
total cost of all agents is 90

Figure 6. 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.*

8.1.1 Examples and Discussion

Now we demonstrate the pricing algorithm. Consider the example shown in Figure 6. Successive rounds of the algorithm in Figure 5, with all starting prices at 0 and with the initial matching as the system optimal matching shown in Figure 6, 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 5. Unfortunately, it is easy to adopt an example from (44) to show that the algorithm will run forever for this example input if $\varepsilon = 0$.

8.2 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 subsection 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

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

We use the following notations:

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

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

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

8.2.1 The Pricing Scheme

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

$$p_{i,j} = \begin{cases} \max\left\{0, \mathcal{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathcal{cost}\left(a_i, r_j, t_i^{\mathcal{A}}\right)\right\}, & \text{if } g^{\text{opt}}\left(a_i\right) = r_j \\ \mho, & \text{otherwise} \end{cases} \tag{8.3}$$

We can immediately observe that the pricing given by (Equation 8.3) has the following desirable property.

Lemma 9. *g^{opt} is a Nash equilibrium matching function when pricing is used,* i.e.*, when $\mathfrak{cost}_{\mathcal{P}}$ is used as the cost function.*

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

□

8.2.2 Compensation for Unhappy Agents by the Pricing Authority

Two important consequences for the pricing scheme in (Equation 8.3) are the following:

Happy agents: If $\mathfrak{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) \geq \mathfrak{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)$ then agent a_i has to pay an extra cost of $\mathfrak{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) - \mathfrak{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)$ in addition to the original cost $\mathfrak{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)$. Thus, the total cost, namely the sum of these two costs, incurred by a_i is exactly $\mathfrak{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)$, 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 $\mathfrak{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) < \mathfrak{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)$ then agent a_i gets its resource at no cost, but it ends up paying $\mathfrak{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)$, which is strictly more than the cost (before pricing) $\mathfrak{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)$ 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:

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

Then agent a_i will end up paying a total of

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

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 10. *The agent-dependent pricing scheme in* (Equation 8.3) *along with the reimbursement of costs by the pricing authority as described in* ($\star$) *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:

$$\sum_{i:\, \mathsf{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) > \mathsf{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathsf{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathsf{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right)\right]$$

$$- \sum_{i:\, \mathsf{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) \geq \mathsf{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathsf{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathsf{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right)\right]$$

$$= \sum_{i:\, \mathsf{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right) > \mathsf{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathsf{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathsf{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right)\right]$$

$$+ \sum_{i:\, \mathsf{cost}\left(a_i, g^{\text{opt}}(a_i), t_i^{\mathcal{A}}\right) \geq \mathsf{cost}\left(a_i, g_{ne}(a_i), t_i^{\mathcal{A}}\right)} \left[\mathsf{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathsf{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right)\right]$$

$$= \sum_{i=1}^{n} \left[\mathsf{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right) - \mathsf{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right)\right]$$

$$= \sum_{i=1}^{n} \left[\mathsf{cost}\left(a_i, g_{ne}\left(a_i\right), t_i^{\mathcal{A}}\right)\right] - \sum_{i=1}^{n} \left[\mathsf{cost}\left(a_i, g^{\text{opt}}\left(a_i\right), t_i^{\mathcal{A}}\right)\right] \geq 0$$

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

Remark 11. *Thus, based on the results in Lemma 10 and 9, 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.*

8.2.2.1 Example

Consider again the example in Figure 6 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.

8.3 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 6. 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 $\mathfrak{cost}\left(a_2, r_1, t_2^{\mathcal{A}}\right)$ 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 (45; 46).

CHAPTER 9

DISTRIBUTED ALGORITHMS FOR THE INCOMPLETE INFORMATION SCENARIO

The model studied in previous chapters assumes that knowledge about the locations and 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 chapter, 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.

9.1 Incomplete Information Game in Simple Scenarios

In the incomplete information version of our game, agents will now make some prior probabilistic assumptions about the locations of the other agents in the game and the analysis is performed based on the expectations given by the prior distributions. One can compute the expected costs based on the distribution that is used to denote the location of an agent. Then an agent will be looking to minimize its expected cost. In this context, the analysis will compute the Nash equilibrium strategies for the players but considering expected costs. This equilibrium is analogous to the Nash equilibrium for but instead of using cost given by the cost functions, it uses *expected cost*.

In this section we develop equilibrium strategies for two simple scenarios in terms of game setup and compare them with a naïve strategy. These games will be played on the number line and will only have two players.

9.1.1 2 vehicles, 3 slots on the Line

a) Game Setup and Expected Cost Formula

Consider the incomplete information resource search game played in the [0,1] line. Let $n = 2$ and let one player be named a_x and the other named a_y. Let $x \in [0,1]$ and $y \in [0,1]$ be the locations of the players respectively. Let $m = 3$, where the location of r_1 is 0 (left end of line) and the location of r_2 and r_3 is 1 (right end). In this model, each player does not know the location of the other and assumes that its location is distributed uniformly in $[0,1]$.

We'll assume that the strategies can be randomized (mixed) and that they will depend on the players' locations. In randomized strategies players choose each strategy with a probability. Let $p : [0,1] \rightarrow [0,1]$ be a function that maps the location of a player to a probability. In other words, $p(x)$ defines the probability that a_x chooses to move to the left to find a resource and $1 - p(x)$ will be the probability of him moving to the right. Since the game is symmetric, in terms of the players, we assume that the function will be the same for each player.

Denote by $C^*(x)$ the expected cost of a_x finding a resource under the condition that a_x is at location x. For this simple scenario this expected cost will be based solely on the distance traveled by a_x. $C^*(x)$ can be computed as follows. When $x \leq 1/2$,

$$\begin{aligned} C^*(x) &= \int_0^x \{p(y)[p(x)(1-x+2y)+(1-p(x))(1-x)] \\ &\quad +(1-p(y))[p(x)(x)+(1-p(x))(1-x)]\}dy \\ &+ \int_x^{\frac{1}{2}+x} \{p(y)[p(x)(x)+(1-p(x))(1-x)] \\ &\quad +(1-p(y))[p(x)(x)+(1-p(x))(1-x)]\}dy \\ &+ \int_{\frac{1}{2}+x}^{1} \{p(y)[p(x)(x)+(1-p(x))(1-x)] \\ &\quad +(1-p(y))[p(x)(x)+(1-p(x))(2+x-2y)]\}dy \end{aligned} \tag{9.1}$$

When $1/2 < x \leq 3/4$,

$$\begin{aligned} C^*(x) &= \int_0^x \{p(y)[p(x)(1-x+2y)+(1-p(x))(1-x)] \\ &+ (1-p(y))[p(x)(x)+(1-p(x))(1-x)]\}dy \\ &+ \int_x^{\frac{3}{2}-x} \{p(y)[p(x)(x)+(1-p(x))(1-x)] \\ &+ (1-p(y))[p(x)(x)+(1-p(x))(1-x)]\}dy \\ &+ \int_{\frac{3}{2}-x}^{1} \{p(y)[p(x)(x)+(1-p(x))(1-x)] \\ &+ (1-p(y))[p(x)(3-x-2y)+(1-p(x))(1-x)]\}dy \end{aligned} \tag{9.2}$$

When $3/4 < x \leq 1$,

$$\begin{aligned}
C^*(x) &= \int_0^{\frac{3}{2}-x} \{p(y)[p(x)(1-x+2y) + (1-p(x))(1-x)] \\
&+ (1-p(y))[p(x)(x) + (1-p(x))(1-x)]\}dy \\
&+ \int_{\frac{3}{2}-x}^{x} \{p(y)[p(x)(1-x+2y) + (1-p(x))(1-x)] \\
&+ (1-p(y))[p(x)(3-x-2y) + (1-p(x))(1-x)]\}dy \\
&+ \int_x^1 \{p(y)[p(x)(x) + (1-p(x))(1-x)] \qquad (9.3) \\
&+ (1-p(y))[p(x)(3-x-2y) + (1-p(x))(1-x)]\}dy
\end{aligned}$$

b) Nash Equilibrium Strategy

Now let $p^*(x)$ be defined as follows:

$$p^*(x) = \begin{cases} 1 & \text{if } x \leq 3/8 \\ 0 & \text{if } x > 3/8 \end{cases} \qquad (9.4)$$

This strategy is: if the agent is at position 3/8 or smaller, with probability 1 move to the resource at 0; otherwise move to the resources at 1 with probability 1.

Theorem 12. *In an incomplete information resource-search game played on the [0,1] line with $n = 2$, $m = 3$, with the location of the slots at 0,1, and 1; the strategy $p^*(x)$ (as defined by Equation 9.4) is a Nash equilibrium strategy.*

Proof. Let $n = 2$ with agents named a_x and a_y with $x \in [0,1]$ and $y \in [0,1]$ being their positions respectively. Let $m = 3$ with the positions of each slot being 0,1, and 1.

Suppose that $x < 3/8$ and suppose that a_y uses strategy p^*. We prove the theorem by showing that the smallest expected cost for a_x is attained by using the same strategy. Using Equation 9.1 and based on a_y's strategy choice, the expected cost becomes:

$$\begin{aligned} C^*(x) &= \int_0^x [p(x)(1-x+2y) + (1-p(x))(1-x)]dy \\ &+ \int_x^{3/8} [p(x)(x) + (1-p(x))(1-x)]dy \\ &+ \int_{3/8}^{\frac{1}{2}+x} [p(x)(x) + (1-p(x))(1-x)]dy \\ &+ \int_{\frac{1}{2}+x}^{1} [p(x)(x) + (1-p(x))(2+x-2y)]dy \end{aligned} \tag{9.5}$$

Now after simplifying the updated equation we obtain a linear function in terms of $p(x)$. Then the expected cost equation becomes of the form $C^*(x) = ap(x) + b$. In this case $a = 2x - \frac{3}{4}$. When $a < 0$, $p(x) = 1$ is optimal for a_x and when $a > 0$ then $p(x) = 0$ is optimal for a_x.

Then solving for x when $a < 0$ gives us that $p(x) = 1$ when $x < 3/8$ and $p(x) = 0$ when $x > 3/8$, in which case $p(x) = p^*(x)$. Thus it turns out that randomized strategies are not needed. Based on the player's location, he'll know which strategy to choose.

The same result holds when $x > 3/8$. □

c) Comparison with Naïve Strategy

We define a *naïve strategy* in this incomplete information game to be one where each player simply moves to the closest available slot. Formally, for the setup on the line, the naïve strategy $p(x)$ is defined as follows:

$$p(x) = \begin{cases} 1 & \text{if } x \leq 1/2 \\ 0 & \text{if } x > 1/2 \end{cases} \tag{9.6}$$

By plugging $p(x)$ from Equation 9.6 into Equation 9.1, Equation 9.2, and Equation 9.3 and integrating from 0 to 1 for all possible values of x, we can compute the expected cost for a_x when using the naïve strategy:

$$\begin{aligned} & \int_0^{1/2} \left[\int_0^x (1 - x + 2y) dy + \int_x^1 x dy \right] dx \\ + \quad & \int_{1/2}^1 \int_0^1 (1 - x) dy dx = \frac{1}{3} \approx 0.333 \end{aligned} \tag{9.7}$$

Similarly by plugging $p^*(x)$ from Equation 9.4 into Equation 9.1, Equation 9.2, and Equation 9.3 and integrating from 0 to 1 for all possible values of x, we can compute the expected cost of a_x when using the equilibrium strategy:

$$\begin{aligned} & \int_0^{\frac{3}{8}} \left[\int_0^x (1 - x + 2y) dy + \int_x^1 x dy \right] dx \\ + \quad & \int_{\frac{3}{8}}^{\frac{1}{2}} \left[\int_0^{\frac{1}{2}+x} (1 - x) dy + \int_{\frac{1}{2}+x}^1 (2 + x - 2y) dy \right] dx \\ + \quad & \int_{\frac{1}{2}}^1 \left[\int_0^1 (1 - x) dy \right] dx = \frac{163}{512} \approx 0.318 \end{aligned} \tag{9.8}$$

This gives the Nash equilibrium strategy an expected percent improvement of approximately 4.5% over the naïve (greedy) strategy for this example.

9.1.2 2 vehicles, 2 slots on the Line

If we take the same example as in the previous section but remove one of the slots on the right end of the line then the following theorem applies:

Theorem 13. *In an incomplete information resource-search game played on the [0,1] line with $n = 2$, $m = 2$, with the location of the slots at 0 and 1; the naïve strategy as defined in Equation 9.6 is a Nash equilibrium strategy.*

9.2 Basic Gravitational Algorithmic Paradigm

Solving the incomplete information game for arbitrary values of n and m is difficult in general because of the different combinations of strategies to consider from all players when constructing the expected cost formula. The general number of resources also increases the number of strategies for each agent and further complicates the expected cost formula. We then wish to propose a heuristic based on the results of section 9.1 with which vehicles can compute their strategies in an incomplete information context. The heuristic we will introduce is based on a gravitational force model.

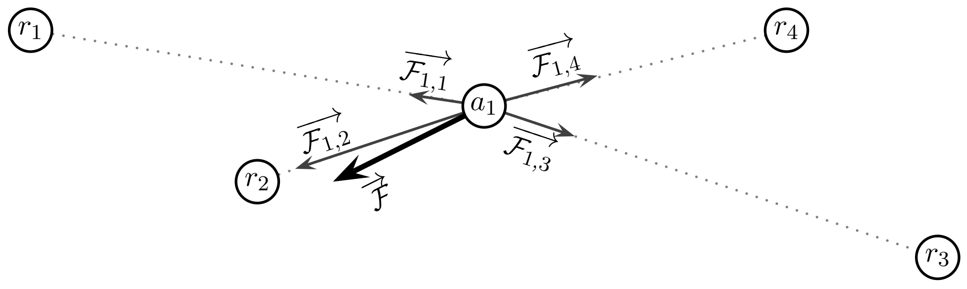
Before presenting our heuristic approach, let us discuss the suitability of several alternative approaches. One technique that could be suitable for this problem is greedy algorithms. We will test our algorithm against a greedy approach that chooses to move towards the closest resource at all times. An approach that models the problem as a Traveling Salesman problem (18) would not be suitable because the availability of resources changes as the agent is traveling and therefore there is no point of planning a tour that visits all the resources. Dynamic programming methods would also be unsuitable because the problem cannot be broken up into smaller subproblems due to the lack of information. Flow-based methods would not work either because there is no notion of flow when an agent does not know the locations of other agents.

From the examples on the line, we know that the equilibrium strategies are ones in which a player should always choose a resource (no randomization) depending on his location on the map. For the example in section 9.1.2 the player should always choose r_1 (move left) if he's located in $[0, 1/2]$ and move right otherwise. For the problem in section 9.1.1 he will move left when located in $[0, 3/8]$ and move right otherwise. So then the fact that there are two available parking slots on the right changes the bounds at which it will be more profitable for the player to choose to move left or right.

This observed phenomenon can be modeled as resources having some type of gravitational pull on the agents. In physics, *gravitational force* is determined by the masses of the objects (resources and agents) and the distance between them. If we set the masses of all resource and agents to be constant then it is expected that in the $n = 2$, $m = 2$ problem in section 9.1.2 the point where the forces are equal would be right in the middle of the line segment (distances are the same). This point changes when adding the third available resource because the two resources to the right will have more gravitational pull than the lone slot in the left. That is why it is an equilibrium for a player to move to the right whenever it is located anywhere in $(3/8, 1]$.

The heuristic algorithmic approach we describe here then is one that pushes the agents towards 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 (34) and encompasses these properties. GPA was used in (34) to guide agents towards 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 for parking applications in (34), it can also be applied to the general framework presented in this work 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 towards 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 7.



r_1
r_4
$\overrightarrow{\mathcal{F}_{1,1}}$
$\overrightarrow{\mathcal{F}_{1,4}}$
a_1
$\overrightarrow{\mathcal{F}_{1,2}}$
$\overrightarrow{\mathcal{F}_{1,3}}$
r_2
$\overrightarrow{\mathcal{F}}$
r_3

Figure 7. 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.

The following simplified formula[1], 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 (34):

$$\left|\overrightarrow{\mathcal{F}_{i,j}}\right| \overset{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, r_j\right)}\right| = \frac{1}{\left[\mathsf{time}\left(\ell_i^{\mathcal{A}}(t),\, \ell_j^{\mathcal{R}}\right)\right]^2} \tag{9.9}$$

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 9.9, $\ell_i^{\mathcal{A}}(t)$, the 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 *naviga-*

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

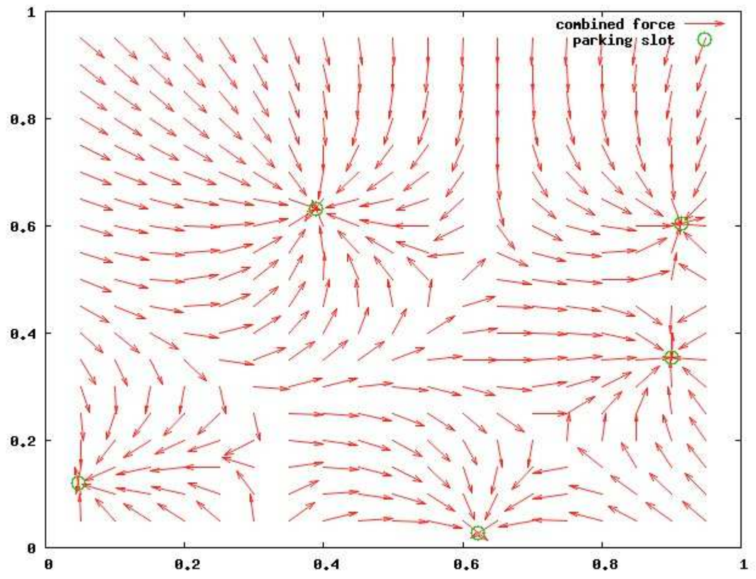


Figure 8. Force field generated by 5 slots

tion algorithm rather than just an algorithm that computes an agent-resource matching. For general costs given by our $\mathscr{cost}$ function, (Equation 9.9) can be generalized to the following:

$$\left|\overrightarrow{\mathcal{F}_{i,j}}\right| \overset{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, r_j\right)}\right| = \frac{1}{\left[\,\mathscr{cost}\left(a_i,\, r_j,\, t\right)\right]^2} \tag{9.10}$$

With (Equation 9.10), 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.

Figure 8 shows what a gravitational force field generated by five sample resources would look like. The arrows represent the direction at which an agent will move when it is located at the start point of the arrow and the small dots represent the resources. This diagram gives us an idea of how agents move across the map when using GPA and it shows that they will eventually converge to a resource. Also, it shows how if we changed the locations of the resources, the field would also change as well, and thus the strategies will be affected.

In (34) and (35), the GPA approach was evaluated in simulations that were performed in a general free-space and with synthetic data. The evaluation of the approach here in this dissertation will be left until chapter 11, where the evaluation will use real-world data and will be done for the version of the algorithm which works with movement that is restricted to a road network (presented in the next section).

9.3 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 *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 9.10). 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 9.10) will be used by an agent to update its routing *only* at each intersection.

For any node $v \in V$, let $\mathsf{location}(v)$ 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.* $\ell_i^{\mathcal{A}}(t) = \mathsf{location}(v)$) 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 $\mathrm{cost}_{e_q}(a_i, r_j, t)$ denote the value of $\mathrm{cost}(a_i, r_j, t)$ *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 zeroes. 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[1] with the edge e_q.
- then computes the magnitude of force vector $\overrightarrow{\mathcal{F}_{i,j}}$ using this value of $\mathrm{cost}(a_i, r_j, t)$ in (Equation 9.10), and
- then updates $\left|\overrightarrow{\mathcal{D}_q}\right|$ to $\left|\overrightarrow{\mathcal{D}_q}\right| \leftarrow \left|\overrightarrow{\mathcal{D}_q}\right| + \left|\overrightarrow{\mathcal{F}_{i,j}}\right|$.

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

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

[1]In the case when the general cost reflects the travel time, *i.e.*, when $\mathrm{cost}(a_i, r_j, t) = \mathrm{time}\left(\ell_i^{\mathcal{A}}(t), \ell_j^{\mathcal{R}}\right)$, 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.

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.

```
(* $\mathsf{location}(v)$ denote the location of node $v \in V$ *)
(* For $v \in V$, $\mathit{cost}_v(a_i, r_j, t)$ is equal to $\mathit{cost}(a_i, r_j, t)$ assuming $\ell_i^{\mathcal{A}}(t) = \mathsf{location}(v)$ *)
(* For $e = (u, v) \in E$, $\mathit{cost}_e(a_i, t)$ is equal to $\mathit{cost}(a_i, r_j, t)$ assuming $\ell_j^{\mathcal{R}} = \mathsf{location}(v)$ *)
(* Agent $a_i$ is in node $v$ at time $t$; $v$ has $\kappa$ outgoing edges $e_1 = (v, v_1), \ldots, e_\kappa = (v, v_\kappa) \in E$ *)

$\left|\overrightarrow{\mathcal{D}_1}\right| \leftarrow \left|\overrightarrow{\mathcal{D}_2}\right| \leftarrow \cdots \leftarrow \left|\overrightarrow{\mathcal{D}_\kappa}\right| \leftarrow 0$
for all $r_j \in \mathcal{R}$ do
  for all $q = 1, 2, \ldots, \kappa$ do
    $\mathit{cost}(a_i, r_j, t) \leftarrow \mathit{cost}_{e_q}(a_i, r_j, t)$
    $\left|\overrightarrow{\mathcal{D}_q}\right| \leftarrow \left|\overrightarrow{\mathcal{D}_q}\right| + \frac{1}{\left[\mathit{cost}(a_i, r_j, t)\right]^2}$
  end for
end for
```

options for choosing the direction of travel according to the magnitudes of these vectors. In (47), 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 $\left|\overrightarrow{\mathcal{D}_q}\right| = \max_{1 \le j \le \kappa} \left\{ \left|\overrightarrow{\mathcal{D}_j}\right| \right\}$
agent $a_i$ moves along edge $e_q$
```

CHAPTER 10

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 (5; 6) 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 (48) 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 (14). 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.

10.1 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 chapter 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, that 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.

10.2 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 (11). 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.

10.2.1 Gravitational Navigation Without Aging

We will use this same gravitational algorithm that was presented in Section 9.3 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 9.10) 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

$$\left|\overrightarrow{F_{i,p,q}}\right| \overset{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, v_p, v_q\right)}\right| = \frac{k_{p,q}}{\left[\mathit{cost}\left(a_i,\, r_j,\, t\right)\right]^2} \text{ with } \ell_j^{\mathcal{R}} = \begin{cases} \text{midpoint of the line connecting} \\ \mathsf{location}\left(v_p\right) \text{ and } \mathsf{location}\left(v_q\right) \end{cases} \tag{10.1}$$

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 9.3.

10.2.2 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 10.1). 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 10.1) as:

$$\left|\overrightarrow{F_{i,p,q}}\right| \overset{\text{def}}{=} \left|\overrightarrow{\mathcal{F}\left(a_i, v_p, v_q\right)}\right| = \frac{\mathcal{E}_{i,j}}{\left[\mathit{cost}\left(a_i,\, r_j,\, t\right)\right]^2} \tag{10.2}$$

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

10.3 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 (48) 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 10.2 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.

10.3.1 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 for each block. Initially all the queues are empty. We continue to process the reports as they come. Whenever an availability report for some block is received, the report is added (enqueued) to that block's queue (increasing its size by one). If an 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 10.1, *i.e.* all 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).

10.3.2 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 10.1) will be replaced by the current size of the queue for the block $(v_p, v_q) \in E$.

CHAPTER 11

SIMULATIONS AND EXPERIMENTAL RESULTS WITH REAL-WORLD DATA

In this section we experimentally evaluate the previously presented algorithms in a simulation framework that uses real-world data from the SFPark project (5).

11.1 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 (5; 6). 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 10, 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 time stamp. 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 10.1).

11.2 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. The simulation uses real-world data from the SFPark (5) 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 having 40 nodes and 63 edges 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 ran. For example, say that the simulation uses simulation data from a specific day at 4pm 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 created 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 which 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.

11.3 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 9.3 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 towards 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 chapter 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 chapter 8. 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 10.3.1 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 ratio and the value of T.

***UGravity*:** This is the approach presented in Section 10.2.1. 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 (49). 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 10.2.2, 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 for the setting with missing or 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 were able to compute these two metrics from the historical SFPark data for each block in the tested urban area. 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 (49) 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.

11.4 Tested Parameters

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

n The number of 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 ratio 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 that 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 $\rho = 0.9$, which was the largest value that was tested in (49).

11.5 Simulation Results for Various Environments

In the remaining subsections of this section, we discuss the results of our simulation in various environments and parameter settings.

11.5.1 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 9 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 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 10 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

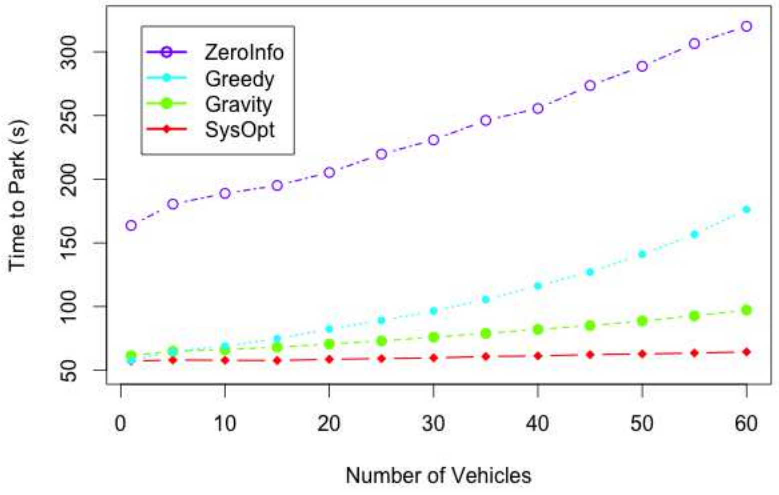


Figure 9. Average time to park for algorithms with access to the ground truth data of resource availability.

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 8.

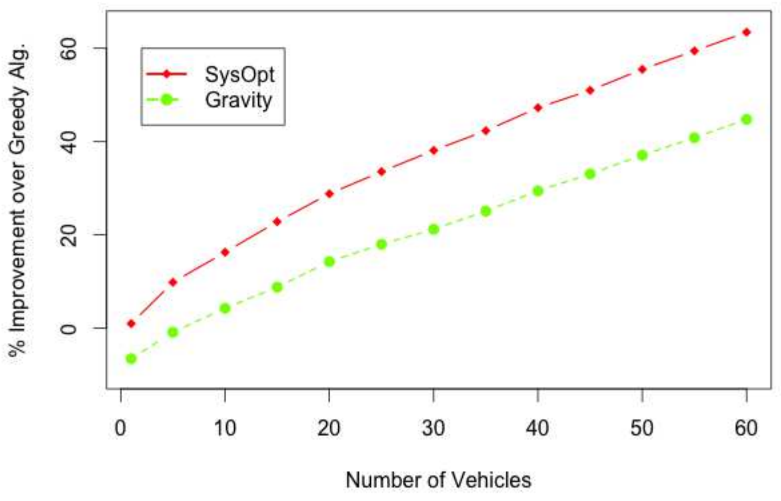


Figure 10. Percentage improvement of the average time to park for *SysOpt* and *Gravity* over *Greedy*.

11.5.2 Simulation Results for Algorithms with Missing or Erroneous Information

Figure 11 shows some of our obtained results for algorithms that use a dataset with missing information due to different values of penetration ratio. 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

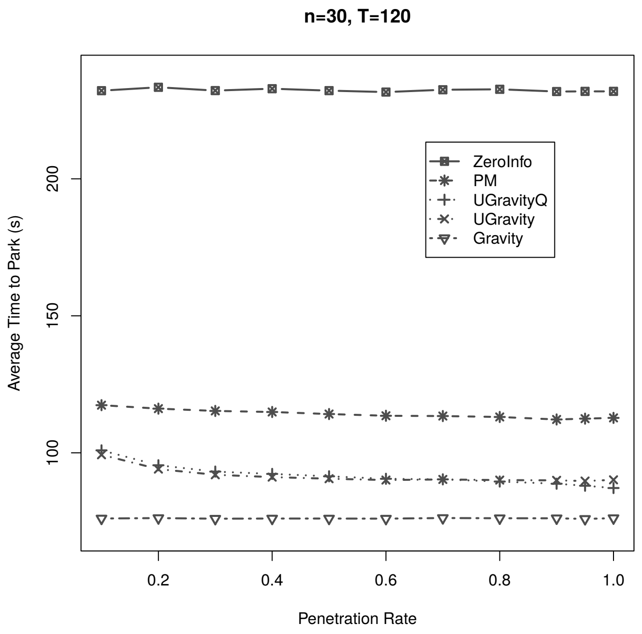


Figure 11. Average time to park for the tested algorithms, when $n = 30$ and $T = 120$.

that have access *only* to availability reports. The algorithm that makes use of the queue-based approach is *UGravityQ*.

In Figure 11 we see the results for varying values of the penetration ratio, 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. This gives of 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 2 and 3 for most test cases.

In Figure 11, 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 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).

This figure 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 ratio 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 the all the algorithms performed at their best with the higher value of T.

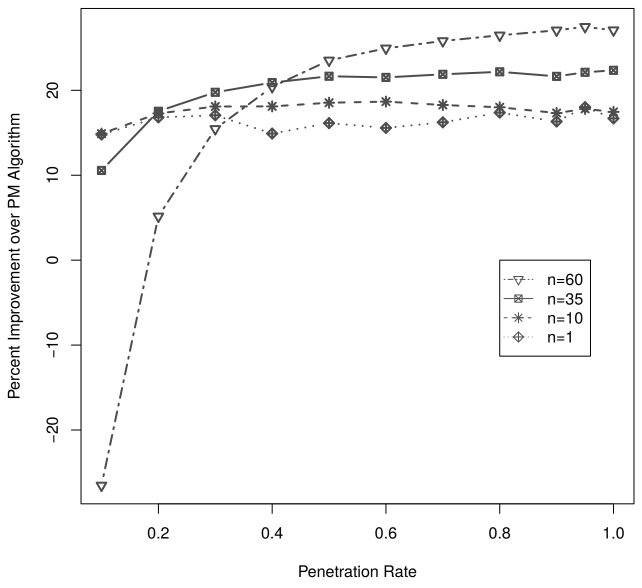


Figure 12. Percentage improvement of the average time to park of *UGravity* over *PM* with $T = 120$.

In Figure 11 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 obtain 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 accesible reports.

Figure 12 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 12 also shows that as the level of competition for resources increases (*i.e.*, higher values of n) the performance of *UGravity* also improves since gravitational navigation algorithms are very likely to direct vehicles towards 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 towards 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 ratios 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 of the average time to park of over 20%. We now provide estimates of the *environmental impact* of such improvements. In (38) 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 (40), 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 emission. 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 *emission.*

CHAPTER 12

CONCLUSION

In this dissertation 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 modelling 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 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.

CITED LITERATURE

1. DeMaio, P.: Bike-sharing: History, impacts, models of provision, and future. Journal of Public Transportation, 12:41–56, 2009.

2. Frost, J. R. and Stone, L. D.: 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, September 2001.

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

4. Koopman, B. O.: Search and Screening: General Principles and Historical Applications. New York, NY, USA, Pergamon Press, 2000.

5. `http://sfpark.org/`: ,

6. Simons, D.: Sfpark: San francisco knows how to park it. Sustainable Transport, 23:26–27, Winter 2012.

7. Kühne, R.: Potential of remote sensing for traffic applications. In Proceedings of IEEE Intelligent Transportation Systems, pages 745–749, Shanghai, China, Oct. 12-15, 2003. IEEE.

8. Wolfson, O. and Xu, B.: Opportunistic dissemination of spatio-temporal reseource information in mobile peer-to-peer networks. In Proc. of 1st International Workshop on P2P Data Management, Security and Trust (PDMST'04), DEXA Workshops 2004, pages 954–958, Zaragoza, Spain, Sept. 2004. IEEE.

9. Park, W., Kim, B., Seo, D., Kim, D., and Lee, K.: Parking space detection using ultrasonic sensor in parking assistance system. In IEEE Intelligent Vehicles Symposium, pages 1039–1044, Eindhoven, The Netherlands, June 2008. IEEE.

10. Panja, B., Schneider, B., and Meharia, P.: Wirelessly sensing open parking spaces: Accounting and management of parking facility. In Proceedings of the 17th Americas

Conference on Information Systems, page paper 270, Detroit, Michigan, August 2011. Association for Information Systems.

11. Mathur, S., Jin, T., Kasturirangan, N., Chandrashekharan, J., Xue, W., Gruteser, M., and Trappe, W.: Parknet: Drive-by sensing of road-side parking statistics. In MobiSys, pages 123–136, San Francisco, CA, June 2010. ACM.

12. Ma, S., Wolfson, O., and Xu, B.: Updetector: Sensing parking/unparking activities using smartphones. In Proc. of 7th Int. Workshop on Computational Transportation Science (IWCTS), 2014.

13. Nawaz, S., Efstratiou, C., and Mascolo, C.: Parksense: a smartphone based sensing system for on-street parking. In Proc. of the 19th Annual Int. Conf. on Mobile Computing and Networking (MobiCom), pages 75–86. ACM, 2013.

14. Xu, B., Wolfson, O., Yang, J., Stenneth, L., Yu, P. S., and Nelson, P. C.: Real time street parking availability estimation. In Proc. of 14th Intl. Conf. on Mobile Data Management (MDM), pages 16–25, Milan, Italy, June 3-6, 2013. IEEE.

15. Kokolaki, E., Karaliopoulos, M., and Stavrakakis, I.: 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), pages 187–194, Bardonecchia, Italy, January 2011. IEEE.

16. Szczurek, P., Xu, B., Lin, J., and Wolfson, O.: Spatio-temporal information ranking in vanet applications. Intl. J. of Next-Generation Computing, 1(1):52, July 2010.

17. Wolfson, O., Xu, B., and Yin, H.: 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, pages 185–199, Toronto, Canada, 2004. Springer-Verlag.

18. Verroios, V., Efstathiou, V., and Delis, A.: Reaching available public parking spaces in urban environments using ad-hoc networking. In Proc. of 12th Intl. Conf. on Mobile Data Management (MDM), pages 141–151, Lulea, Sweden, 2011. IEEE.

19. Johnson, D. S., Minkoff, M., and Phillip, S.: The prize collecting steiner tree problem: theory and practice. In Proc. of 11th ACM-SIAM Symposium on Discrete Algorithms, pages 760–769, San Francisco, CA, 2000. SIAM.

20. Boehlé, J., Rothkrantz, L. J. M., and van Wezel, M.: Cbprs: A city based parking and routing system. ERIM Report Series Reference No. ERS-2008-029-LIS, May 2008.

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

22. Geng, Y. and Cassandras, C. G.: A new "smart parking" system based on optimal resource allocation and reservations. In Proc. of 14th Intl. Conf. on Intelligent Transportation Systems (ITSC), pages 1129–1139, Washington, DC, USA, October 5-7 2011. IEEE.

23. Tilly, M. and Reiff-Marganiec, S.: Matching customer requests to service offerings in real-time. In Proc. of the 2011 ACM Symposium on Applied Computing, pages 456–461, 2011.

24. Hou, L., Mouratidis, K., Yiu, M. L., and Mamoulis, N.: Optimal matching between spatial datasets under capacity constraints. ACM TODS, 35(2), 2010.

25. Wie, B. W. and Tobin, R. L.: Dynamic congestion pricing models for general traffic networks. Transportation Research Part B - Methodological, 32(5):313–327, 1998.

26. Phang, S.-Y. and Toh, R. S.: Road congestion pricing in singapore: 1975 to 2003. Transportation Journal, 43(2):16–25, 2004.

27. Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V.: Algorithmic Game Theory. New York, NY, USA, Cambridge University Press, 2007.

28. Gomez, J., Dasgupta, D., and Nasraoui, O.: A new gravitational clustering algorithm. In Proc. of SIAM Conf. on Data Mining (SDM), pages 83–94, San Francisco, CA, 2003. SIAM.

29. Wright, W. E.: Gravitational clustering. Pattern Recognition, 9:151–166, 1977.

30. Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A.: Combinatorial Optimization. New York, NY, John Wiley & Sons, 1998.

31. Papadimitriou, C. H. and Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Upper Saddle River, New Jersey, Prentice-Hall, Inc., 1982.

32. Rasmusen, E.: Games and Information. Hoboken, New Jersey, Blackwell Publishing, 4th edition, 2006.

33. Nash, J.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences, 36(1):48–49, 1950.

34. Ayala, D., Wolfson, O., Xu, B., Dasgupta, B., and Lin, J.: Parking slot assignment games. In Proc. of the 19th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2011), pages 299–308, Chicago, IL, November 2011. ACM.

35. Ayala, D., Wolfson, O., Xu, B., DasGupta, B., and Lin, J.: Parking in competitive settings: A gravitational approach. In Proc. of 13th Intl. Conf. on Mobile Data Management (MDM), pages 27–31, Bengaluru, India, July 23-26 2012. IEEE.

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

37. Marden, J. R. and Roughgarden, T.: Generalized efficiency bounds in distributed resource allocation. In IEEE CDC, pages 2223–2238, Atlanta, GA, Dec. 2010.

38. Shoup, D.: The High Cost of Free Parking. Chicago, IL, American Planning Association, 2005.

39. Shoup, D.: Cruising for parking. Transport Policy, 13:479–486, 2006.

40. Transportation Alternatives (`www.transalt.org`): , New York, NY Pricing the Curb: How San Francisco, Chicago and Washington D.C. are reducing traffic with innovative curbside parking policy, July 2008.

41. Even-Dar, E., Kesselman, A., and Mansour, Y.: Convergence time to nash equilibria. In Proc. of the 30th Int. Conference on Automata, Languages and Programming, pages 502–513, Eindhoven, The Netherlands, 2003. Springer Verlag.

42. Kearns, M. and Mansour, Y.: Efficient nash computation in large population games with bounded influence. In 18th Conference in Uncertainty in Artificial Intelligence, pages 259–266, Edmonton, Alberta, Canada, 2002. Morgan Kaufmann.

43. Lipton, R. J., Markakis, E., and Mehta, A.: Playing large games using simple strategies. In 4th ACM Conference on Electronic Commerce, pages 36–41, San Diego, CA, 2003. ACM.

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

45. Wolfson, O. and Lin, J.: A marketplace for spatio-temporal resources and truthfulness of its users. In Proc. of the 7th ACM SIGSPATIAL International Workshop on Computational Transportation Science, pages 1–6, Dallas, Texas, November 2014. ACM.

46. Zou, B., Kafle, N., Wolfson, O., and Lin, J.: A mechanism design based approach to solving parking slot assignment in the information era. Transportation Research Part B, conditionally accepted, 2015.

47. Ayala, D., Wolfson, O., Xu, B., DasGupta, B., and Lin, J.: Spatio-temporal matching algorithms for road networks. In 20th International Conference on Advances in Geografic Information Systems (ACM SIGSPATIAL GIS), pages 518–521, Redondo Beach, CA, November 6-9 2012. ACM.

48. Stenneth, L., Wolfson, O., Xu, B., and Yu, P. S.: Phonepark: Street parking using mobile phones. In Proc. of 13th Intl. Conf. on Mobile Data Management (MDM), pages 278–279, Bengaluru, India, 2012. IEEE.

49. Josse, G., Schmid, K. A., and Schubert, M.: Probabilistic resource route queries with reappearance. In Proc. of 18th Intl. Conf. on Extending Database Technology (EDBT), pages 445–456, Brussels, Belgium, 2015.

50. McVitie, D. G. and Wilson, L. B.: Stable marriage assignment for unequal sets. BIT Numerical Mathematics, 10(3):295–309, 1970.

51. Delot, T., Cenerario, N., Ilarri, S., and Lecomte, S.: A cooperative reservation protocol for parking spaces in vehicular ad hoc networks. In Proc. of the 6th Int. Conference on Mobile Techonology, Application and Systems, Nice, France, September 2009.

52. http://faspark.com/: ,

53. Ma, S., Zheng, Y., and Wolfson, O.: T-share: A large-scale dynamic taxi ridesharing service. In Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE), 2013.

54. Roth, A. E. and Sotomayor, M. A. O.: Two-sided Matching: A Study in Game-theoretic Modeling and Analysis. Econometric Society monographs. Cambridge, UK, Cambridge University Press, 1990.

55. Yoon, J. W., Pinelli, F., and Calabrese, F.: Cityride: a predictive bike sharing journey advisor. In Proc. of 13th Intl. Conf. on Mobile Data Management (MDM), Bengaluru, India, July 23-26 2012.

56. Yuan, N. J., Zheng, Y., Zhang, L., and Xie, X.: T-finder: A recommender system for finding passengers and vacant taxis. IEEE Transactions on Knowledge and Data Engineering, 2013.

VITA

Daniel Ayala

Email: dayalar@gmail.com

Education

- Ph.D. Computer Science, University of Illinois at Chicago, 2017.

  Advisor: Dr. Ouri Wolfson

- M.C.S. Computer Science, University of Illinois at Urbana-Champaign, 2008.

- B.S. Computer Science, Universidad de Puerto Rico - Recinto de Río Piedras, 2003.

Research Interests

Mobile Data Management, Intelligent Transportation Systems, Algorithms for improving Urban Transportation Systems

Employment

- ASSISTANT PROFESSOR

  Department of Computer and Mathematical Sciences

  Lewis University

  January 2015 – present

- SENIOR RESEARCHER

  HERE Inc. (Nokia)

Part of a Research and Analytics group in the Automotive Cloud Services Division of the Connected Driving department

July 2014–December 2014

- RESEARCH ASSISTANT & NSF-IGERT FELLOW

  Computational Transportation Science Program

  Department of Computer Science, UIC

  advised by Dr. Ouri Wolfson

  August 2007–July 2014.

- COMPUTER & BIOINFORMATICS CONSULTANT

  High Performance Computing facility

  University of Puerto Rico

  October 2004–July 2007

Teaching

Lewis Univ.

- 70-471: Machine Learning, Fall 2015 (undergraduate level)
- 13-310: Discrete Mathematics, Fall 2015 (undergraduate level)
- 13-511: Concepts of Statistics I,

  Spring 2015 and Fall 2015 (graduate level course for Data Science masters students)

- 70-460: Programming Languages, Spring 2015 (undergraduate level)

  UIC

- CS 581: Spatial Database Management Systems,

  Spring 2014, co-taught with Dr. Ouri Wolfson

  (graduate level course).

- CS 480: Database Systems,

  Spring 2013, co-taught with Dr. Ouri Wolfson

  (senior undergraduate and graduate level course).

Advising & Mentoring

- Masters students (mentored along with Dr. Ouri Wolfson at UIC):

  Pavan Kumar Reddy Jaya (2013) - *Parking Navigator App for Android Smartphones*

  Vijay Bhojwani (2013) - *Parking Space Prediction Model*

Publications

Working Papers

1. Daniel Ayala, Ouri Wolfson, Bhaskar DasGupta, Jie Lin, and Bo Xu, *Spatio-temporal Matching for Transportation Applications*, in preparation

Peer-reviewed Conference Papers

1. Yunjie Zhao, Daniel Ayala, Dongwook Jang, Gavril Giurgiu, *Smart Recommendation For Drivers: A Refueling Application*, Intelligent Transportation Systems World Congress 2015, Bordeaux, France, October 2015.

2. Qing Guo, Ouri Wolfson, Daniel Ayala, *A Framework on Spatio-Temporal Resource Search*, Proceedings of the 11th IEEE International Wireless Communications & Mobile Computing Conference (IWCMC 2015), Dubrovnik, Croatia, August 2015.
3. Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar DasGupta and Jie Lin, *Pricing of Parking for Congestion Reduction*, 20th International Conference on Advances in Geografic Information Systems (ACM GIS 2012), November 6-9, 2012, Redondo Beach, California.
4. Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar DasGupta and Jie Lin, *Spatio-temporal Matching: Algorithms for Transportation Applications*, 20th International Conference on Advances in Geografic Information Systems (ACM GIS 2012), November 6-9, 2012, Redondo Beach, California.
5. Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar DasGupta and Jie Lin, *Stability of Marriage and Vehicular Parking*, 2nd International Workshop on Matching Under Preferences (MATCH-UP 2012), Budapest, Hungary, July 19-20, 2012.
6. Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar DasGupta and Jie Lin, *Parking in Competitive Settings: A Gravitational Approach*, 13th International Conference on Mobile Data Management (IEEE MDM), , Bengaluru, India, July 23-26, 2012.
7. Daniel Ayala, Ouri Wolfson, Bo Xu, Bhaskar Dasgupta and Jie Lin, *Parking Slot Assignment Games*, 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 299-308, 2011.

8. Daniel Ayala, Jie Lin, Ouri Wolfson, Naphtali Rishe, and Masaaki Tanizaki, *Communication Reduction for Floating Car Data-based Traffic Information Systems*, Proc. of the The Second International Conference on Advanced Geographic Information Systems, Applications, and Services (GeoProcessing2010), St. Maarten, Netherlands Antilles, Feb. 2010, pp. 44-51 (Best Papers Award).

Patents

- Daniel Ayala, Bhaskar DasGupta, Jie Lin, Ouri Wolfson, Bo Xu, *System and Methods for Improved Selection of a Resource among Available Resources*, US patent publication number US20140122190 A1, May 2014.

Presentations

Invited Presentations

- *Parking Slot Assignment Games*, at Universidad de Puerto Rico - Recinto de Río Piedras, San Juan, PR on March 10, 2014.
- *Parking Slot Assignment Games*, at Lewis University, Romeoville, IL on February 19, 2014.
- *Parking in Competitive Settings: A Gravitational Approach*, at Nokia, Chicago, IL on September 13, 2012.

Conference and Workshop Presentations

- *Pricing of Parking for Congestion Reduction*, at 20th International Conference on Advances in Geografic Information Systems (ACM GIS 2012), November 6-9, 2012, Redondo Beach, California.

- *Spatio-temporal Matching: Algorithms for Transportation Applications*, at 20th International Conference on Advances in Geografic Information Systems (ACM GIS 2012), November 6-9, 2012, Redondo Beach, California (poster presentation).
- *Stability of Marriage and Vehicular Parking*, at 2nd International Workshop on Matching Under Preferences (MATCH-UP 2012), July 19-20, 2012, Budapest, Hungary.
- *Parking in Competitive Settings: A Gravitational Approach*, at 13th International Conference on Mobile Data Management (IEEE MDM), July 23-26, 2012, Bengaluru, India.
- *Parking Slot Assignment Games*, at 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, November 2011.
- *Communication Reduction for Floating Car Data-based Traffic Information Systems*, at Proc. of the The Second International Conference on Advanced Geographic Information Systems, Applications, and Services (GeoProcessing2010), St. Maarten, Netherlands Antilles, Feb. 2010.

Honors, Awards, & Fellowships

- NSF-IGERT Fellowship in Computational Transportation Science
  University of Illinois at Chicago, 2007–2014
- National Collegiate Computer Science Award, U.S. Achievement Academy, 2002–2003
- National Collegiate Mathematics Award, U.S. Achievement Academy, 2003
- The National Dean's List, 2001–2003
- University of Puerto Rico Computer Science NSF Fellowship, 2001

References

- Available upon request