Self-Organization and Multi-Agent Reinforcement Learning for Taxi Dispatch

Aamena Ali Ahmed Omran Alshamsi

Master of Science in Information Technology
Faculty of Informatics
The British University in Dubai
February 2009

Abstract

The taxi dispatch problem involves assigning taxis to callers waiting at different locations. An *adjacency-based* dispatch system currently in use by a major taxi company divides the city (in which the system operates) into regional dispatch areas. Each area has fixed designated adjacent areas hand-coded by human experts. When a local area does not have vacant cabs, the system chooses an adjacent area to search. However, such fixed, hand-coded adjacency of areas is not always a good indicator because it does not take into consideration frequent changes in traffic patterns and road structure. This causes dispatch officials to override the system by manually enforcing movement on taxis. In this thesis, I apply two different methods separately to solve the problem: (1) a multiagent self organization technique to dynamically modify the adjacency of dispatch areas (2) a multiagent reinforcement learning method to optimize the dispatch policy for each area. I compare performance of each method with actual data from, and a simulation of, an operational dispatch system. The multiagent self organization technique decreases the total waiting time by up to 25% in comparison with the real system and increases taxi utilization by 20% in comparison with results of the simulation without self-organization. Interestingly, I also discover that human intervention (by either the taxi-dispatch officials or the taxi drivers) to manually overcome the limitations of the existing dispatch system can be counterproductive when used with a self-organizing system. Furthermore, the proposed multiagent reinforcement learning method decreases the total waiting time by up to 33.5% in comparison with the real system.

Acknowledgement

I would like to thank all whome supported me to complete my thesis.

Special Thanks to my supervisors Dr. Sherief Abdallah and Dr. Iyad Rahwan for their support, advice and guideness.

Thanks for AAMAS 2009 reviewers for their constructive comments on the paper (Multiagent Self-organization for a Taxi Dispatch System) that I have submitted in collaboration with Dr. Sherief Abdallah and Dr. Iyad Rahwan to the Eighth International Conference on Autonomous Agents and Multiagent Systems 2009. Chapters 3, 4 and 5 will appear in the proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AA-MAS 2009), Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10 15, 2009, Budapest, Hungary.

I would like to thank my mother, family and friends for their ultimate love and inspiration during the research.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Aamena Ali Ahmed Omran Alshamsi*)

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Introduction

The taxi dispatch problem is the problem of assigning taxis to callers waiting at different locations. The main objective of a taxi dispatch system is to minimize the customers' total waiting time, which is the time needed to dispatch the customer's call and the time it takes the dispatched cab to reach the customer. There are also few secondary objectives that are benefficial to achieve, including maximizing the number of dispatched calls and maximizing the total number of served calls per taxi. One of the approaches to solve this problem (which I refer to as *adjacency-based dispatch*) is to use fixed regional dispatch areas. These regional areas are identified by their Global Positioning System (GPS) locations and obtained by taking into account traffic, street structures and population density. The location of each incoming call or taxi is identified by its corresponding dispatch area.

In theory, one can use a centralized (single) agent to optimize the performance of a taxi dispatch system. In practice, however, a single decision maker will have prohibitively large state-action space to optimize and learn about. Instead, I choose to view the taxi dispatch system as a multi-agent system. Each area is an agent; each call is a task; and each taxi is a resource. The network of the areas interacts to run the dispatch process.

The work proposes two different methods to separately optimize the performance in a taxi dispatch system: (1) multiagent self organization technique (2) multiagent reinforcement learning method. To my knowledge (Related Work in Section 2), none of the previous works applied multiagent reinforcement learning or a self-organization technique to the taxi dispatch problem. I develop a computer simulation of a real taxi dispatch system (initialized using actual data obtained from a major taxi operator) to demonstrate that the proposed approaches outperform the conventional existing system in terms of customer waiting time (respecting customer satisfaction). In addition, Self Organization outperforms the simulation of the existing system in terms of the number of dispatched calls per taxi (directly proportional to fleet utilization). Then, I apply, test and evaluate each approach separately. At each phase, multiple experiments are conducted to demonstrate the efficiency of the incorporated approach in comparison with the existing system and the simulation in prior phases.

1.2 Statement of the Problem

The majority of the previous studies in the taxi dispatch problem did not utilize multiagent systems. They focused on the assignment method aiming at making an assigned taxi reach the customer in the shortest time possible using different parameters such as road networks and real time traffic [7] [10]. Other studies incorporated the satisfaction of drivers in the assignment method [20].

Some of the previous work used Multiagent communication to dispatch taxis to customers [19] [9]. However, the researches focused on the assignment method rather than organizing the network of agents dynamically. Pedro et al (2002) [18] presented an algorithm that used computational geometry techniques to determine adjacency information for agents. Despite that the proposed algorithm has been suggested to be used for a taxi dispatch domain, it doesn't consider some costs associated with the agents such as distance traveled which is directly relevant to the customer satisfaction. Thus, none of the mentioned works optimize the performance of a taxi dispatch system by organizing the network of agents. Moreover, none of the mentioned works apply a multiagent reinforcement learning method to optimize the performance of a taxi dispatch system.

Adjacency-based taxi dispatch systems use fixed areas to search in while receiving calls. Usually, the calls are dispatched to taxis within the call's area. If there are no vacant cabs in the call's area, the system searches in the defined adjacent areas. One of the limitations of such systems that adjacency of areas are adjusted manually. This manual adjustment can be suboptimal because it relies on human judgment and only occurs at rather long intervals. Thus, the manual adjustment doesn't take into consideration changes in traffic and road structures.

1.3 Research Questions

The thesis aims at answering the following questions

1. How can a taxi dispatch system with a number of dynamic factors be simulated?
2. How to apply self-organization techniques to an adjacency-based taxi dispatch system? and does it improve the performance of the system?
3. How to apply a multiagent reinforcement learning method to a taxi dispatch system? and does it improve the performance of the system?

1.4 Contributions

In this section, I outline the main contributions of this thesis. I do so by showing how the work reported in this thesis answers each of the research questions outlined above.

- How can a taxi dispatch system with a number of dynamic factors be simulated?

The goals of this thesis it to apply two different multiagents methods separately to an adjacency-based taxi dispatch system. However, it is impractical to deploy new methods immediately in a real system without carefully studying the new approach in a simulated environment. Thus, a computer simulation of an existing taxi dispatch system (of a major Taxi company in the Middle East) is developed. The simulation uses the same dispatch policy used by the current system and also uses real data in generating calls and computing travel delay. The simulator takes into consideration uncontrollable factors such as the number of incoming calls, traffic, and availability of cabs in an abstract manner that is reasonably accurate yet tractable. The simulator also mimics some behaviors of dispatch parties such as the dispatch officers and the drivers by moving taxis towards congested areas. The taxi dispatch problem can be mapped to a multiagent system as follows. Each area is an agent; each call is a task; and each taxi is a resource. The network of the areas interacts to run the dispatch process.

- How to apply self-organization techniques to an adjacency-based taxi dispatch system? and does it improve the performance of the system?

The existing taxi dispatch system that is simulated uses fixed dispatch areas so that if there are no vacant cabs in the main dispatch area of the incoming call, the system searches for a vacant cab in the adjacent areas. The adjacency of areas is hand coded in a schedule by dispatch experts, but it doesn't take into consideration frequent changes in streets or traffic [1]. Therefore, I apply a self-organization technique to the taxi dispatch problem so that dispatch areas can self-organize themselves during the dispatch process. To my knowledge this is the first work to apply a self-organization technique on a taxi dispatch problem and benchmark it against the performance of an existing operational system showing superior performance. The proposed technique decreases the total waiting time by up to 25% in comparison with the real system and increases taxi utilization by 20% in comparison with results of the simulation without self-organization.

- How to apply a multiagent reinforcement learning method to a taxi dispatch system? and does it improve the performance of the system?

All taxi dispatch systems have an objective in common which is to dispatch the incoming calls in such a way that minimizes the average total waiting time of calls. In a taxi dispatch system that uses fixed dispatch areas, it is difficult to choose which area to search for vacant cabs within in order to assign one of the vacant cabs to an incoming call. Currently, the system uses a simple rule to achieve this: it simply chooses the taxi that has been vacant the longest among taxis available in adjacent areas. However, durations of travel among areas differ according to weekdays versus weekends and peak hours versus other hours. This means that while more taxis may be vacant in a particular adjacent area, those taxis may not be the best choice when it comes to minimizing customer waiting time. Therefore, I apply a multiagent reinforcement learning method to the taxi dispatch problem so that each agent (dispatch area), decides what to do for each incoming call at each state. To my knowledge, this is the first work to apply a multiagent reinforcement learning method on a taxi dispatch problem and benchmark it against the performance of an existing operational system showing superior performance. The proposed technique decreases the total waiting time by up to 30.5% in comparison with the real system.

To summarize, the contributions of this research project are to:

1. Develop a realistic simulator of the taxi dispatch problem that is based on real data acquired from a taxi dispatch company in the middle east and list a number of lessons about modeling the taxi dispatch domain in order to provide key foundation for further researches.
2. Apply a self-organization technique to the developed taxi dispatch simulation and evaluate the performance of the technique.
3. Apply a multiagent reinforcement learning method to the developed taxi dispatch simulation and evaluate the performance of the method.

1.5 Scope

The simulation is based on a real taxi dispatch system, so it follows the same dispatch policy and uses real call logs, fleet distrubution and averaged duration between different pairs of areas. However, the simulation assumes that taxis only receive jobs via calls unlike the real system that accepts passengers pick up from the street without a call. Also, I simulate some behaviors of dispatch parties such as drivers, dispatch offcials and operation officials. However, some behaviors such as driver rejection of jobs and force assignments of jobs are not simulated because they are unpredicted harder to model and predict.

[1] Note that the adjacency schedule used by the system has no direct correspondence with geographical adjacency among areas.

In addition, I assume that there is no precise geographical information about locations of calls or taxis, so locations are identified only by their corresponding pre-defined areas and no Global Positioning System (GPS) coordinates are provided to identify their locations. Therefore, the thesis deals only with an adjacency based dispatch system.

1.6 Organization of the thesis

The remaining of this thesis is organized as follows. Chapter 2 displays the related works to this thesis and presents the concept of Multiagent Systems. Chapter 3 describes the existing taxi dispatch system that is simulated. Chapter 4 describes the developed simulator and how it mimics the existing system, presents the obtained results and evaluates them in comparison with the existing system. Chapter 5 presents the concept of overlay networks and self-organization, describes the proposed self-organization technique, presents the obtained results and evaluates them in comparison with the existing system and the simulation. Chapter 6 presents the multiagent reinforcement method that is incorporated in the simulation, displays the obtained results and evaluates them. At the end, the conclusion chapter summarizes the thesis and presents the future work.

Chapter 2

Literature Survey

The previous work on automating the taxi dispatch problem follows one of the following directions: (1) Assignment Method (2) Multiagent Systems. This section briefly reviews the previous work in both of these directions, with more focus on the multiagent systems directions, because it is more related to my work.

2.1 Assignment Optimization Approaches

Most approaches that have been presented on the problem of establishing a flexible taxi dispatch system focused on the assignment method as the assigned taxi should reach the customer in the shortest time possible. For example, Chung [7] proposed to use the A* search algorithm to calculate the shortest path by adjusting it to accommodate the road network's conditions.

Also, Der-Horng et al [10] proposed a dispatch system whereby the dispatch of taxis is determined by real-time traffic instead of just using the shortest line path to the customer. Another approach proposed by Shrivastra et al [20] uses a fuzzy rule combination approach to solve taxi dispatch problems. Two contradicting rules are satisfied by the proposed approach simultaneously which are "nearest vehicle first"and "the least utilized vehicle first". Similarly, Chen [6] proposed a distributed taxi hailing protocol aiming at assigning tasks to taxis with shortest moving distance toward potential passengers. The protocol reduces both the customer waiting time and the empty cruise time of the taxi.

More importantly, these approaches require the availability of supplementary data such as road structures, or real time traffic information which might be unavailable or up-to-date. Another aspect that is not covered by the mentioned solutions is how to enhance the structure of the taxi network and manage it automatically.

2.2 Multi-Agent Approaches

A multiagent system (MAS) is a system consisting of several autonomous intelligent agents that jointly interact to fulfill the tasks or to maximize a utility [16]. The environment at which agents interact may have some constraints that make agents do not know fully about the world [16]. Multi-agent systems have been utilized in a number of real-world applications to solve problems that are difficult or complex for a single agent or centralized conventional system to solve such as transportation, online trading and traffic management [1].

An agent is a computer system that is autonomous enough to execute actions fulfilling its design goals when interacting with and getting feedback from its environment [24]. An intelligent agent observes an environment and acts accordingly in a way that achieve its goals by using heuristics, knowledge or learning [24].

Some of the previous work used Multiagent communication to dispatch taxis to customers. Seow et al [19] proposed a multiagent architecture, whereby taxis are represented by collaborative agents that in turn negotiate on behalf of taxi drivers for incoming calls. The proposed solution aims at increasing customer satisfaction as well as driver satisfaction. In comparison with an existing systems in Singapore, the proposed system reduced passenger pick-up time and empty taxi cruise time. The proposed architecture shares the limitation of the solutions mentioned in Section 2.1 in terms of its data requirements (e.g. real time GPS locations of taxis). Moreover, unlike the work, it focuses on the assignment method only rather than the structure of the network and it is based on de-centralized networks.

Another interesting approach to the taxi dispatch problem is highlighted by de Weerdt et al [9]. A resource-based planning framework is presented where agents can merge plans by exchanging their resources. They proposed a plan-merging algorithm that is evaluated on a taxi domain. Empirical results showed a decrease of the total distance driven by all taxis if passengers are allowed to share rides. Of course, this approach requires customers to be willing to share rides, and hence has a different focus to the work.

Some researchers have realized that enhancing assignment methods alone does not necessarily make dispatch systems efficient. This is because, the way the network of taxis and tasks is structured affects the assignment method as well. The better the network is organized and structured, the less time is needed for call assignment, passenger pick up time and average travel distance. Sander et al [18] presented a novel algorithm for efficient task allocation that uses computational geometry techniques to determine adjacency information for the agents. The adjacency information help agents to determine which neighboring nodes should be considered in the decision making process as they look for a nearby task to accomplish. Nevertheless, it is stated in the work that the algorithm doesn't consider some costs associated with the agents such as distance traveled which is directly relevant to the customer satisfaction. The main goal of the solution is to maximize the number of dispatched jobs.

Freight transportation is a slightly different problem from the taxi dispatch problem. This is mainly because freight transportation can allow multiple shipments to be bundled together to optimize the utilization of container space. This is equivalent to taxi sharing among passengers, which is not yet practical. Davidsson et al [8] surveyed some agent-based approaches to freight transportation and traffic management and presented a framework for assessing such approaches. Other studies addressed the problem of global transportation scheduling such as the work of Perugini et al [17] which requires many transport organization to co-operate and coordinate to transport partial requests along partial routes to accomplish transport requests. Another related area is the optimization of flexible multi-vehicle pick-up and delivery problems within defined time frame, such as the work of Dorer and Calisti [11]. Similar to the problem of freight transportation, Horn [14] addressed the problem of Demand-responsive transport services offered by taxis or buses aiming at reducing the travel time of the vehicle. This approach addresses only the driver satisfaction and shares the limitation of the work of de Weerdt et al [9] that passengers should be willing to share rides.

Chapter 3

The existing taxi dispatch system

This work relies on the existing dispatch system of a company that is operational in a city with high population density in the Middle East.[1] The existing taxi dispatch system uses fixed regional areas to dispatch calls. The main city is partitioned into 131 areas besides 10 more areas of surrounding cities. The classification of areas depends on population density, traffic, customer demand and other factors related to the dispatch process. Boundaries of areas are identified by GPS coordinates and each area has its own *cab queue* and *call queue*. The *cab queue* contains a list of vacant cabs that are located within an area in a descending order according to the vacancy period and it keeps track of availability of those cabs. It is updated whenever a cab enters the area or leaves it or the availability status changes from "vacant"to "occupied"or "on call"or vice versa. The *call queue* of an area contains a list of waiting calls that originate in that area in a descending order according to the call's waiting time.

3.1 Dispatch Policy

The system uses the following dispatch policy (based on interviews with officials from the taxi company). Whenever a customer calls the dispatch center, his/her credentials, location, time, the number of passengers and preferences are entered into the system. Based on the address provided, the call operator tries to identify the customer's suburb location. Then, the call is placed in the area that contains the defined suburb given that each area contains a number of suburbs. Based on that, the system automatically starts looking for a vacant cab within the selected area. If there are vacant cabs within the selected area, the system assigns the call to the cab that has been vacant the longest. If the call is placed in an area with no vacant cabs available, the system starts looking, after one minute, for the cab that has been vacant the longest within a predefined list of adjacent areas. If there are no vacant cabs in the main area and the adjacent areas, the system waits for a cab to become available in either the main area or the adjacent areas. If the waiting time for any call has exceeded 1 hour, the call is cancelled. The following points summarize the dispatch process:

1. The operator enters the caller's credential in the dispatch system.
2. The operator identifies the suburb of the caller according to provided address.
3. The system identifies the area of the caller based on the given suburb according the list of predefined areas.
4. The system attempts to dispatch the call.
   (a) If there is at least one vacant cab in the caller's area, the system assigns the incoming call to the cab. In case there are more than one vacant cab in area, the call is assigned to the one that has been vacant the longest.

[1] For confidentiality reasons, I am unable to disclose the name of the company at this stage.

(b) If there are no vacant cabs in the main area, the system searches the adjacent areas of the main area (if available) after one minute. If there are vacant cabs within the adjacent areas, the system assigns the call the cab that has been vacant the longest.

(c) If there are no vacant cabs in the main area and the adjacent areas, the system waits for a vacant cab to show up in one of the areas to assign the call to.

5. If the system fails to dispatch the call within one hour, the call is cancelled.

Figure 3.1 summarizes the dispatch process handled by the existing system.

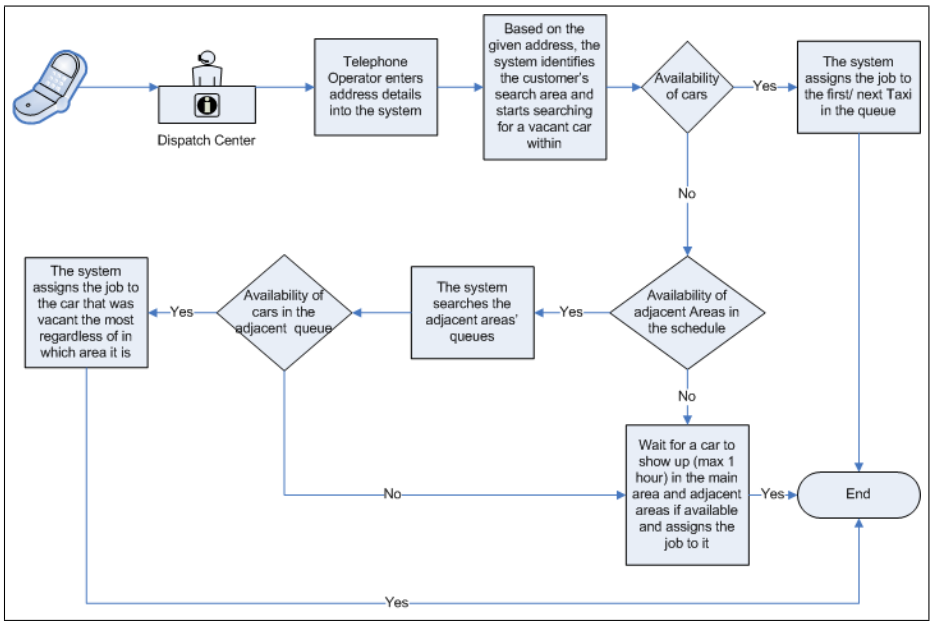

Figure 3.1: Existing System

3.2 Adjacency Schedule

The adjacency of areas is defined in a schedule based on the experience of the dispatch center officials and some statistics of high-demand areas. The schedule is updated regularly (e.g. every few months) manually to take into consideration changes in traffic patterns, so high-traffic areas need adjacent areas. The adjacency of areas neither necessarily reflects physical adjacency nor implies symmetric adjacency of areas in the schedule.

3.3 General Statistics

Table 3.1 displays important statistics during the period 6/8/2007 - 11/9/2007, from 9:00 am to 3:00 pm (in order to avoid rush hours). [2]

These statistics are used to benchmark the simulator against the existing system and later to evaluate the proposed approaches against the existing system.

[2]Fridays and Saturdays which are weekend days in the city are excluded. Thursdays are also excluded because directly precede the weekend.

Parameter	Average in minutes	Standard Deviation
Dispatch Time	5.45	9.8
Pickup Time	14.5	17.2
Total Waiting Time	20	20.3

Table 3.1: General Statitisics

3.4 Limitations of the existing dispatch system

The existing system has a number of limitations that are described in this section. I later show how the approach addresses these limitations.

3.4.1 Customer Satisfaction and Total Income

The system uses a dispatch policy that is biased in favor of the driver's interest rather than the customer's interest. More importantly, the current system does not take into account the profit of the taxi company as a whole: priority is set to cabs that have been vacant the longest rather than minimizing the total waiting time of the customer.

3.4.2 Adjacency of Dispatch Areas

The existing system defines the adjacency of areas manually roughly every few months. This manual adjustment can be suboptimal because it relies on human judgment and only occurs at rather long intervals. The following scenario depicted in Figure 3.2 demonstrates the limitation of exchanging vacant taxi among the predefined adjacent areas. A customer in Area A calls the dispatch center to request a taxi. However, at that time there are no vacant cabs in Area A. Thus, the system started searching for vacant taxis within the adjacent areas as predefined in the adjacency schedule by the officials. Area B, C, D, E and F are defined to be adjacent to Area A. As shown on the map, only Area B and Area C have vacant taxis. The call then should be assigned to the taxi that has been vacant the longest. However, as can be seen from the map the taxi in Area B might take, for example, 45 minutes to reach the customer. Similarly, the taxi in Area C might take 30 minutes which is still too long and might dissatisfy the customer. On the other hand, Area G has a vacant taxi which might take approximately 15 minutes to reach the customer. However, since Area G is not defined to be an adjacent area to Area A, it is not be considered in the search.

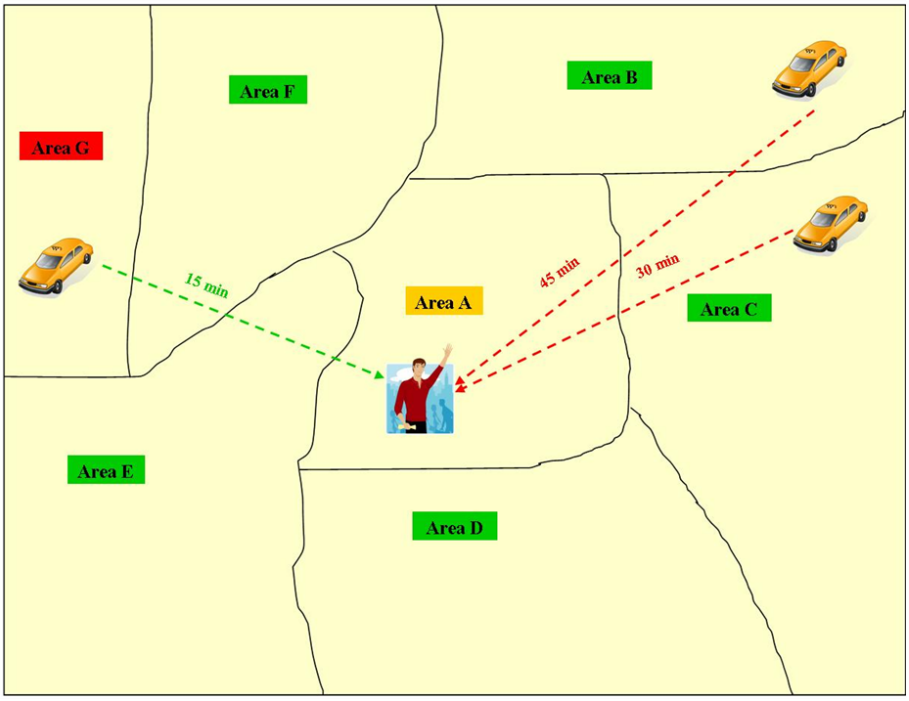

Figure 3.2: Taxi Dispatcher Scenario

Chapter 4

Simulating the existing system

It is impractical to deploy a new approach immediately in the real world without carefully studying the new approach in a simulated environment. The purpose of my dissertation is to make a case for incorporating the multi-agent approach by showing its potential benefit in a simulated environment based on real data.

This chapter describes my simulator. Table 4.1 lists important terms that are used in this section and the rest of the dissertation.

Term	Description
Cab or Car	Taxi
Dispatch Time	How long the system takes to dispatch an incoming call (in minutes)
Pick up Time	how long an assigned cab takes to pickup a customer since the call is dispatched (in minutes)
Total Waiting Time	Pick up time + Dispatch Time
Job or task	Call
Search Areas or Dispatch Areas	Areas predefined by the existing system
Customer or Caller	Potential Passenger
Main Area	Caller's Area
Utilization of a neighbor area	The usefulness of an adjacent area measured by the number of times an area assigns a call to a vacant cab in this neighbor area

Table 4.1: Terms used in the document

The simulator uses the same dispatch policy used by the current system and also uses real data in generating calls and computing travel delay. The simulator takes into consideration uncontrollable factors such as the number of incoming calls, traffic, and availability of cabs in an abstract manner that is reasonably accurate yet tractable (more details later). The simulator starts with a number of cabs distributed among areas. Then, at each time unit (minute), it checks for incoming calls, attempts to dispatch them along with the waiting calls according to the dispatch policy and availability of cabs and moves cabs to their destinations. At the end of a simulation run, various statistics are collected, including taxi utilization and average total waiting time. Appendix A provides more details.

4.1 Real Data

The average and the standard deviation are calculated on some key data that are obtained from the real system. The period of time whereby data is obtained is identified to ensure accuracy since traffic differs at peak hours versus other remaining hours and differs also in working days versus weekends. The period is defined to be from 6/8/2007 till 11/9/2007, where Thursdays, Fridays and Wednesdays are excluded and time is limited to be from 9:00 am till 3:00 pm.

Fleet Distribution The taxi dispatch database contains the number of cabs in each area at each hour of the day. Thus, different distributions are obtained from the database and fed into the simulator. To run the simulator, a number of cabs are initially distributed among areas by identifying the number of cabs at each area. At time 0 of the simulation, all cabs are vacant.

Call Distribution A call log was taken from the dispatch database. I have only used data about successful calls in the simulator, because unsuccessful calls lacked some details. The log contains the following attributes for each call: call time, location (area) and destination (area). This is used to generate tasks in various simulation runs.

Trip Duration It is difficult to calculate the duration of trips in the simulator because many parameters are needed to be taken into account such as traffic, road structures and GPS locations. Instead, the simulator assumes the duration between any pair of areas follows a normal distribution. The average and standard deviation of the duration between each pair of areas is calculated from the real-data. This captures (approximately) many factors that are very difficult to simulate such as traffic and street structures.

Area Adjacency Schedule The latest schedule of area adjacency is used which lists all areas and their corresponding adjacent areas if available.

Dispatch Policy The policy is described earlier in Section 4.2. To run the simulator, one should select a predefined time/day period. Based on the obtained data on the selected period, a number of cabs are distributed among the areas. Then each minute, the simulator checks for incoming calls based on the obtained call log at the predefined period. Meanwhile, the simulator dispatches calls according to the defined dispatch policy, moves cabs toward their destination, keeps track of cabs and call details and moves some vacant cabs to congested areas if needed. At periodic intervals of the simulation (e.g. 300 time units) and at the end of the simulator run, statistics are displayed showing the following:

1. The total number of completed calls, dispatched calls, cancelled calls and waiting calls.
2. The maximum, minimum, average and standard deviation of number of calls served by a taxi.
3. The maximum, minimum, average and standard deviation of calls' dispatch time.
4. The maximum, minimum, average and standard deviation of calls' pick up time.
5. The maximum, minimum, average and standard deviation of calls' total waiting time.

4.2 System Behavior

The existing dispatch process is summarized in algorithm 4.1.

Algorithm 4.1: Dispatch Algorithm

```
Input: Calls and Cabs
Output: Dispatching calls to cabs
foreach incoming call or waiting call i with waiting time ≤ 60 minutes do
    if there is a vacant cab j in the call's area then
        Call i is dispatched to cab j;
    else if waiting time >1 minute then
        if there are vacant cabs l in the adjacent areas of call's i area then
            Call i is dispatched to the cab s that has been vacant the longest among cabs
            l;
        else
            Call i is added to the end of the waiting list
        end
    end
end
```

The dispatch algorithm above does not account for all actual behaviors in the real system. Drivers sometimes work autonomously, driving around and picking up passengers from the street. Drivers also get ad hoc instructions from the dispatch center or the operation section. I simulate the behavior of drivers moving toward congested areas by using four strategies to move cabs that stay vacant to areas that have shortage in vacant taxis. This applies only to movements not controlled by the dispatch center. The strategies have been set according to interviews with dispatch officials since the dispatch process is affected by human interference. I tried to simulate the following main dispatch parties' behaviors: the movement of drivers to congested areas regardless of their current location, instructions provided by some operation officials to send taxis to areas that are *expected* to have high demand, and instructions provided by some dispatch officials to send nearest taxis to areas with high demand. Moreover, different sets of strategies have been used to test the simulator as shown below.

Strategy1 Vacant cabs in other cities moves to predefined destinations;

Strategy2 The cab that is vacant the longest moves to the area with maximum number of waiting calls;

Strategy3 Vacant cabs in other cities moves to areas with maximum number of waiting calls;

Strategy4 Nearest vacant cab moves to area with calls that have being waiting the longest;

The strategies are grouped as follows for testing purpose (Appendix A provides more details):

The strict strategy (Combination 1) consists of strategies 1 and 3. Thus, each vacant cab outside the main city moves to the area with maximum number of waiting calls. Otherwise, the cab moves to the predefined destination according to the cab's location.

The flexible strategy (Combination 2) consists of strategies 1, 2, 3, and 4

4.3 Experimental Results

I have conducted experiments for the simulation of the existing system and used different settings such as distribution of cabs among areas at the first run of the simulator and different movements of vacant cabs. The purpose of the experiments is to compare the performance of the simulator to the existing system The evaluation includes the measurements described in Table 4.2.

The total number of cabs during any time of day or night is approximately 2400 cabs. I selected a fleet distribution with a total of 1622 cabs to all of the conducted experiments. This

Measurement	**Abbrev.**
Averaged dispatch time	ADT
Averaged pick up time	APT
Averaged total waiting time	$ATWT$
Percentage of cancelled calls (Cancellation Rate)	CT
Percentage of waiting calls at a given run (Waiting Rate)	WT
Percentage of served calls (Success Rate)	ST
Average total number of calls served by each cab (Fleet Utilization)	FU
Averaged degree distribution	ADD

Table 4.2: Terms used in the experiments

distribution is captured on 12/08/07 at 9:00 am. I tried also distributions with lower total number of taxis such as 793 (the total number of vacant cabs on the same day and hour) which shows roughly the same performance. Note that not all the trips in the real system are obtained via calls. Some cabs pick up passengers from the street which is not covered by the simulation but implicitly is taken into consideration by reducing the number of available cabs. The simulation is not utilizing the whole fleet, so that some of the cabs do not receive any call during the total runs of the simulation. This, mimics the real scenario when some cabs get busy picking up passengers from the street all the time and receive few or no calls.

To ensure that the developed dispatch simulation uses the exact dispatch policy used by the existing system, I have conducted two experiments using the two different combinations of strategies of moving vacant cabs as described in Section 4.2.

Table 4.3 illustrates the obtained accumulated results after the last run along with the statistics of the real taxi dispatch system.

The two experiments use the following parameters:

- Total number of cabs: **1622**
- Total number of calls: **43348 (Actual Data)**
- Total number of runs: **4504**
- Periodic statistics at each: **300 runs**

The results of the simulation using the strict strategy (Combination 1) are unsatisfactory when compared to the real data. This is expected since in reality vacant taxis attempt to move looking for potential passengers unlike the movement in the simulation using the strict strategy (Combination 1), which allows the return of vacant cabs to the main city only. Thus, the averaged dispatch time and the total waiting time increases significantly. As a result, the cancellation rate increases and the success rate drops. Note that the log that I used contains only successful calls. Therefore, the real system has success rate of 100% for the data I have used.

When the flexible strategy (Combination 2) is used to move vacant cabs in the simulation, the results improve significantly. The number of dispatched calls increases to reach 82.2%. Consequently, the cancellation rate declines sharply to reach 17% of the total calls. Also, the number of waiting calls in the queue at the last run has been reduced. In addition, the fleet utilization is enhanced as a result of applying the flexible strategy (Combination 2) in the simulation.

The average dispatch time and the average total waiting time drop by around 6 minutes, but they are still higher than the real system (by 15 minutes approximately for the dispatch time and 9 minutes for the total waiting time). While surprising at first, it is actually justified because the flexible strategy (Combination 2) of movement of vacant cabs toward congested areas does not take place frequently and is triggered only when the number of waiting calls is

above a threshold so the travel time of the vacant cabs affects the dispatch time of waiting calls. Although I tried to simulate the behavior of the involved parties in the dispatch process, there are still other human factors that are difficult to simulate and predict. In general, the pick up time in the simulation is lower than the existing one. This might be because the simulation is restricted to assign calls to vacant cabs within the main area or the adjacent areas of the main area of the call. While in the real system, officials may force-assign calls to vacant cabs in other areas which increases the travel time of the assigned cab to reach the customer. I will use the simulation with the flexible strategy (Combination 2) to simulate the existing system in the subsequent chapters because it captures (to an extent) the human behavior. It should be noted that at the end the results are based on simulation. While the results I show in subsequent chapters are quite promising, a fair comparison to the existing system is only possible through incorporating the approach into the real live system.

Rule	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A
Comb. 1	26.45 ±25.8	9 ±6.58	35.4 ±26.3	47.9	50.8	1.26	13.55 ±13.2
Comb. 2	20.57 ±21	8.5 ±5.79	29 ±22	17	82.2	0.7	21.9 ±8.47

Table 4.3: Simulation of existing system with different strategies of vacant cab movements

Table 4.4 displays the obtained accumulated results after the last run along with the statistics of the real taxi dispatch system when using a cab distribution with a total of 793 cabs given that the total number of runs is 4504. The average dispatch time and total waiting time (when the strict strategy is used) are a bit higher in comparison when a cab distribution with total of 1622 cabs is used in Section 5.3. This is reasonable because the strategy does not take into consideration the movement of vacant cabs, so using a lower total number of vacant cabs affects the dispatch process negatively. Unlike using the flexible strategy, which utilizes the fleet more efficiently. Consequently, the lower total number of cabs does not affect the dispatch process significantly in comparison when a cab distribution with total of 1622 cabs is used in Section 5.3.

Rule	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A
Comb. 1	32.68 ±24.86	8.9 ±6.56	41.67 ±25.43	58.2	40.38	1.37	22.03 ±20.04
Comb. 2	21.86 ±21.2	8.5 ±5.78	30.3 ±22.1	19.1	79.9	0.007	43.6 ±10.7

Table 4.4: Simulation of existing system with different strategies of vacant cab movements (total number of cabs = 793)

Chapter 5

Self-Organization

At this chapter, a self-organization technique is proposed as a solution to the taxi dispatch problem. The aim is to demonstrate that the proposed technique outperforms the existing system and the simulation in terms of average total waiting time.

This chapter presents an overview of overlay network in general. Then, it describes the proposed self-organization technique that is incorporated into the simulation. After that, it displays and evaluates experimental results conducted at this phase. The results are compared to the results of the existing system and the simulation without self-organization in Section 4.3. Related Works to the proposed self-organization technique are listed. The differences and similarities of the related works to this thesis are highlighted.

5.1 Overlay Networks

An overlay network is a virtual network which is placed on top of one or more networks by adding a layer that changes the properties of the underlying network(s) [22]. Mainly, the connectivity of nodes are changed virtually based on needs. The internet is an example of an overlay network that is built upon the telephone network [22]. Changing the connectivity of nodes could be dynamic to tackle the needs of the applications of the designated network at runtime which is called *Self-Organization*. Self-Organization enables overlay networks to organize and adapt themselves at runtime to reduce communication overhead between nodes [4].

5.2 Self-Organization Technique

The existing dispatch system uses a hard-coded schedule of adjacency for the dispatch purpose. Each area uses its neighbor(s), if it has shortage in taxis while receiving a call, and assigns the call to the cab that has been vacant longest among the adjacent areas (For more details, please refer to Section 4.2). Some areas have no adjacent areas, and others have more than one adjacent area.

Since the adjacency schedule is hand coded and has been set by dispatch experts, it doesn't take into consideration frequent changes in street structures or traffic. I propose in this section a self-organizing approach that automatically adapts the areas' adjacency (starting from the hand-coded adjacency schedule).

Algorithm 5.1 shows the proposed self-organizing mechanism, which works as follows. Each area j that has waiting calls more than a threshold T_w (I use $T_w = 10$ in the experiments) checks its neighborhood. If area j has a neighbor i whose utilization (number of times that the main area j assign its calls to area i) is greater than threshold T_u (the experiments use $T_u = 10$),

it checks area i's neighborhood. The algorithm uses parameter p_o to control the frequency of self-organization: with probability p_o self-organization occurs. If area i has neighbor z that has the maximum utilization (among neighbors of area i) that is greater than threshold T_i (the experiments use $T_i = 10$), then area z is added as a neighbor of area j (if it is not already a neighbor of area j). Furthermore, the area with the maximum number of vacant cabs (the maximum number of vacant cabs should be at least 5 in the experiments) offers to be added to the neighborhood of area j with a certain probability. This strategy is necessary to connect areas that are fully isolated (the hand-coded adjacency schedule contains areas with no neighbors). To avoid excess in number of neighbors of an area, the neighbor with minimum utilization is removed until the number of neighbors reaches threshold N_{max}(I use $N_{max} = 5$ in the experiments).

Algorithm 5.1: Strategy 1 for self-organization

Input: Current Adjacency Schedule
Output: Adjacency Schedule is self-organized
foreach *area i at time n with pending calls* $>= T_w$ **do**
  with probability $p_o = n$, **if** *area i has a neighbor j with utilization* $>= T_u$ *and area j has at least one neighbor z that has the max utilization among neighbors of area j that is* $>= T_i$ *;*
  then
    add area z as a neighbor of area i;
  end
  with probability p_o, let area y be the area with the max number of vacant cabs among all the areas.;
  add area y as a neighbor of area i;
  while *the number of i's neighbors* $> N_{max}$ **do**
    remove the neighbor with minimum utilization;
  end
end

5.3 Experimental Results

The following section compares the proposed self-organizing approach to two benchmarks: the simulated existing system, which provides a fair comparison in terms of modeling human behavior, and the existing system itself, which provides a rough estimate of how the proposed architecture would perform in practice.

The same measurements of performance in Chapter 4.3 located at Table 4.2 are used in the following experiments. I have tried different values of self-organization probability p_o = (0.001, 0.1, 0.4, 0.8). The obtained results are compared to the existing system and the simulating of the existing system. I calculate the averaged degree distribution ADD of the network of areas to verify the stability (convergence) of the approach. The degree of a node in a network is the total number of connections it has to other nodes and the degree distribution is the total distribution of degrees in the whole network [12]. The following experiments use the flexible strategy (Combination 2) to move vacant cabs and use a cab distribution with total of 1662. Table 5.1 illustrates the obtained results. The table also contains the statistics of the existing system for reference given the first two rows here are basically row 1 and 3 in Table 4.3.

Overall, the self-organization is not effective while using the flexible strategy (Combination 2) for moving vacant cabs. There is no enhancement in the results when self-organization is incorporated in the simulation. The average degree distribution has still increased which means the self-organization is taking place but there is no noticeable improvement. This was surprising at first, because one would expect self-organization to improve performance over a fixed organization even by a small margin. Upon careful inspection, I have found that the flexible strategy (Combination 2) causes excessive movement of vacant cabs, which in turn reduces the frequency of assigning calls to vacant cabs in adjacent areas. Although self-organization

P_o	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*	*ADD*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A	1.33 ±1.06
0	20.57 ±21	8.5 ±5.79	29 ±22	17	82.2	0.7	21.9 ±8.4	1.33 ±1.06
0.001	20.22 ±20.7	8.5 ±5.7	28.73 ±21.5	14.9	84.3	0.6	22.49 ±8.4	1.44 ±1.12
0.1	18.56 ±20.7	8.76 ±6.1	27.33 ±21.7	17.9	81.2	0.7	21.68 ±8	1.8 ±1.33
0.4	18.47 ±20.4	8.86 ±6.4	27.33 ±21.3	14.8	84.3	0.7	22.51 ±8.6	1.91 ±1.37
0.8	19.57 ±20.7	8.72 ±6.1	28.29 ±21.8	15.5	83.7	0.7	22.32 ±8.4	1.87 ±1.28

Table 5.1: Accumulated statistics with different probabilities of self-organization using the flexible strategy (Combination2)

process is taking place and adding new neighbors, the process has no effect because areas do not utilize their neighbors. This observation has lead me to step back and try the strict strategy (Combination 1) of moving vacant cabs in the following experiments.

The strict strategy (Combination 1) for Movement of vacant cabs

Table 5.2 compares the self-organizing approach using the strict movement strategy (and under different values of p_o) to of the existing system.

P_o	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*	*ADD*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A	1.33 ±1.06
0	20.5 ±21	8.5 ±5.7	29 ±22	17	82.2	0.7	21.9 ±8.4	1.33 ±1.06
0.001	11.5 ±17.9	11.25 ±9.1	22.82 ±19.5	8.2	91.5	0.16	24.38 ±16.9	1.54 ±1.19
0.1	2.8 ±7.8	12.53 ±10.9	15.39 ±13.5	0.6	99.1	0.16	26.41 ±15	1.69 ±1.24
0.4	2.9 ±8.1	12.23 ±10.5	15.16 ±13.2	0.6	99.1	0.13	26.42 ±14.4	1.7 ±1.31
0.8	2.7 ±8.1	12.3 ±11.1	15.14 ±13.8	0.6	99.2	0.07	26.45 ±15.1	1.7 ±1.31

Table 5.2: Accumulated statistics with different probabilities of self-organization using the strict strategy (Combination1)

In comparison with the simulation without self-organization, when $p_o = 0.001$ the average dispatch time is reduced by around 9 minutes which causes a decline in the total waiting time by 8 minutes on average. However, the pick up time rises slightly because the self-organization technique doesn't take into consideration the duration among areas when adding new neighbors which yields in higher travel time of taxis to pick up customers. With $p_o = 0.001$, the results are still not better than the real system because using a relatively low probability of self-organization, delays the results to converge. Moreover, the averaged degree distribution increases slightly by 0.2. On the other hand, when $p_o = 0.1$ or greater, the system's performance significantly improves across all measures in comparison with both the real system data and the simulation of the real system without organization. Increasing p_o beyond 0.1 does not improve performance tangibly, suggesting the convergence of the self-organizing mechanism. Figure 5.1 plots the averaged dispatch time that is computed every 300 time steps (=300 minutes) for each simulation run to measure performance improvement of self-organization. Initially, the averaged dispatch time starts low then it starts to raise. The reason is that all cabs are vacant at the beginning then, they get busy serving calls which causes the gradual increase of the dispatch

time. Without self organization, the averaged dispatch time increases over time. When p_o of 0.001 is used; the averaged dispatch time reduces gradually after a peak at time 900 to reach the minimum at the last run but results take longer time to converge since self-organization doesn't take place frequently. While for p_o of 0.1 or greater, the averaged dispatch time starts to decline after a slight increase at the beginning. Then, it increases again a bit with some fluctuations due to changes in the availability of cabs and the need to self-organize the network to tackle for new needs.

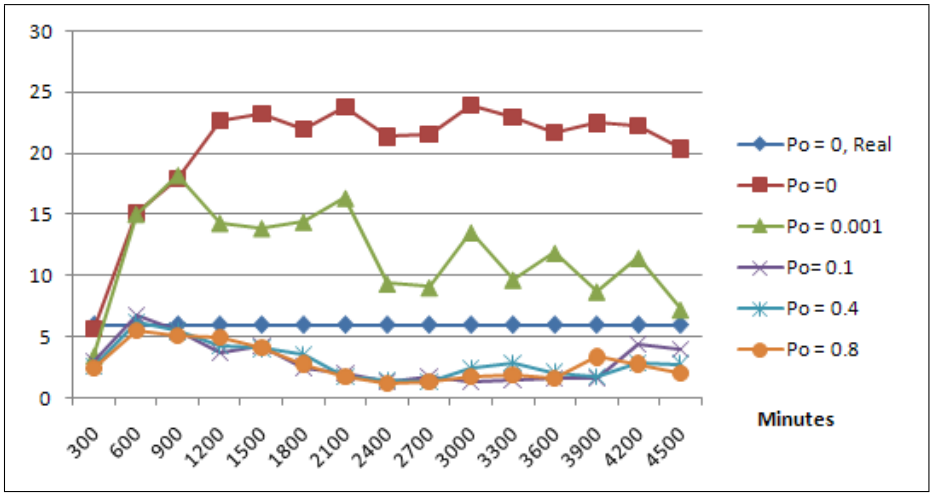

Figure 5.1: Average Dispatch Time for different values of p_o plus the existing system

Figure 5.2 plots the average pick up time computed for each 300 time steps of each simulation run. When self-organization has been incorporated in the simulation, the pick up time increases slightly. When $p_o = 0.001$, the increase is slower in comparison with higher values of p_o since the self-organization does not take place frequently. Note that $p_o = 0$ has the minimum pickup time because it only assigns cabs to local available cabs or immediately adjacent areas.

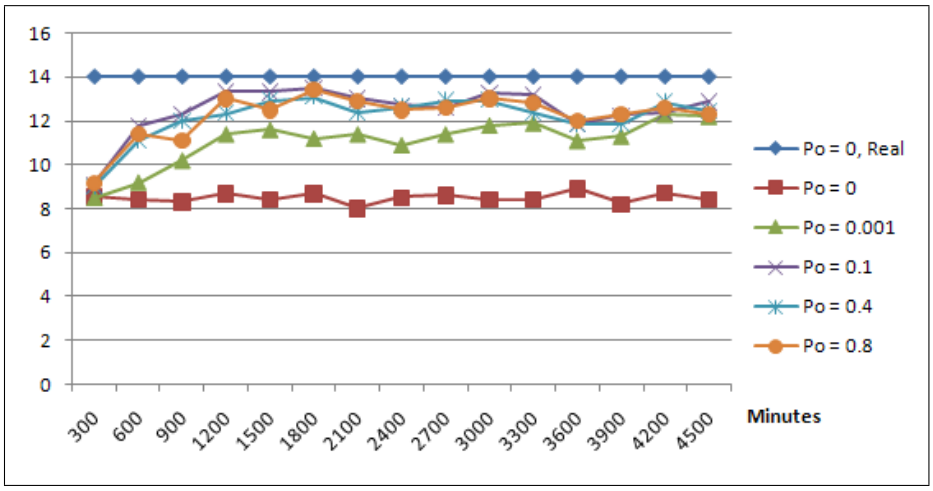

Figure 5.2: Average Pick up Time for different values of p_o plus the existing system

The average total waiting time computed for each 300 time step (Figure 5.3) is very similar to Figure 5.1 since it is dependent on the dispatch time.

One might wonder if the system achieves better performance by simply overconnecting the areas. To investigate this further, I plotted the average *degree distribution* and standard deviation for each 300 time step for the simulation using 0.1 p_o of self-organization in Figure 5.4 to show the changes in the degree distribution. We can see that the degree distribution

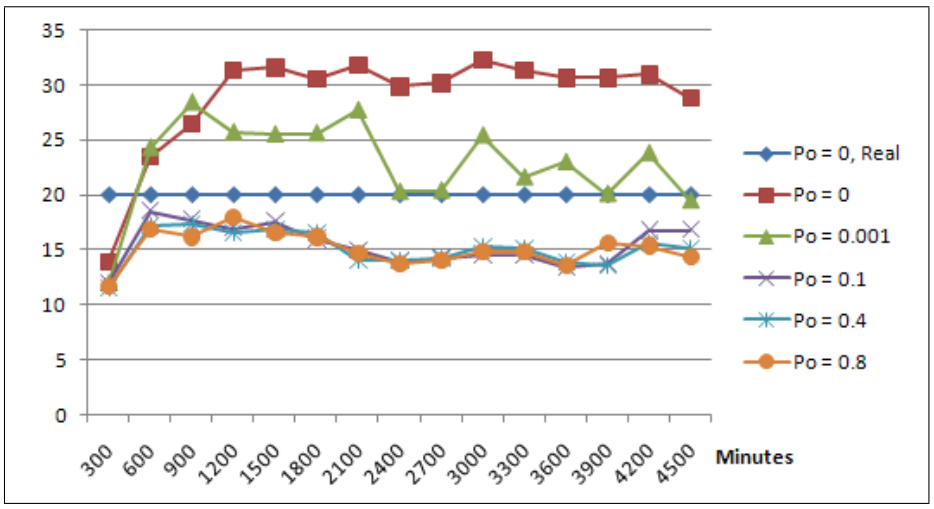

Figure 5.3: Average Total Waiting Time for different values of p_o plus the existing system's one

remains relatively stable, which suggests that the system is not overconnecting areas, but rather improving the quality of the adjacency schedule.

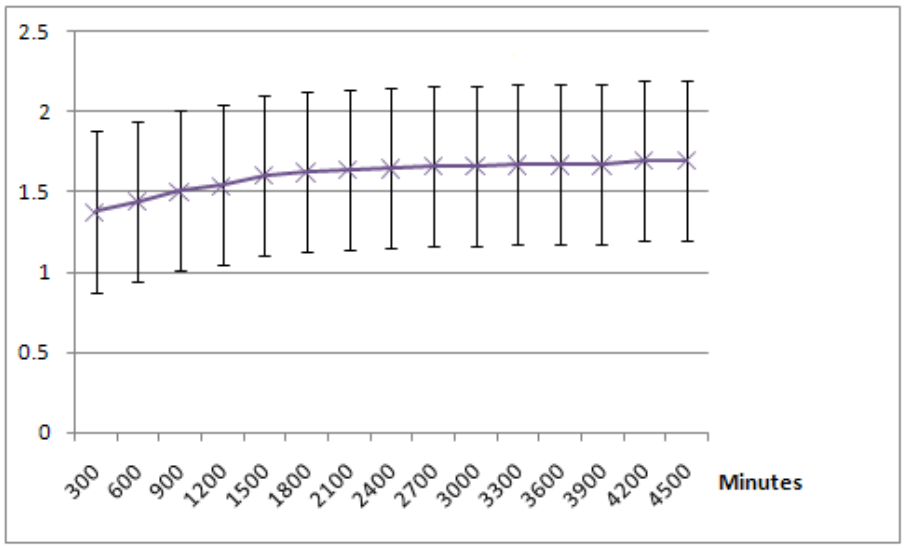

Figure 5.4: Average Degree Distribution for p_o 0.1

Finally, Figure 5.5 plots the number of agents which have a number of neighbors: 0, 1, 2, 3, 4, 5 respectively at both the first time step and the last one for the simulation using p_o of 0.1. The overall degree distribution has not changed significantly. There is a small drop in isolated areas (with 0 neighbors) at the expense of slight increase in areas with 3,4, and 5 neighbors.

Table 5.3 presents the results of applying different probabilities of self-organization to the simulation using the flexible strategy of moving vacant cabs given that the total number of runs is 4504 and the total number of cabs is 793. In comparison with Table 5.1, the results are roughly the same except for a slight increase in the average dispatch time and total waiting time. The increase is reasonable since the total number of cabs is lower. Similarly, the results are not satisfactory, so the strict strategy is used in the following experiments.

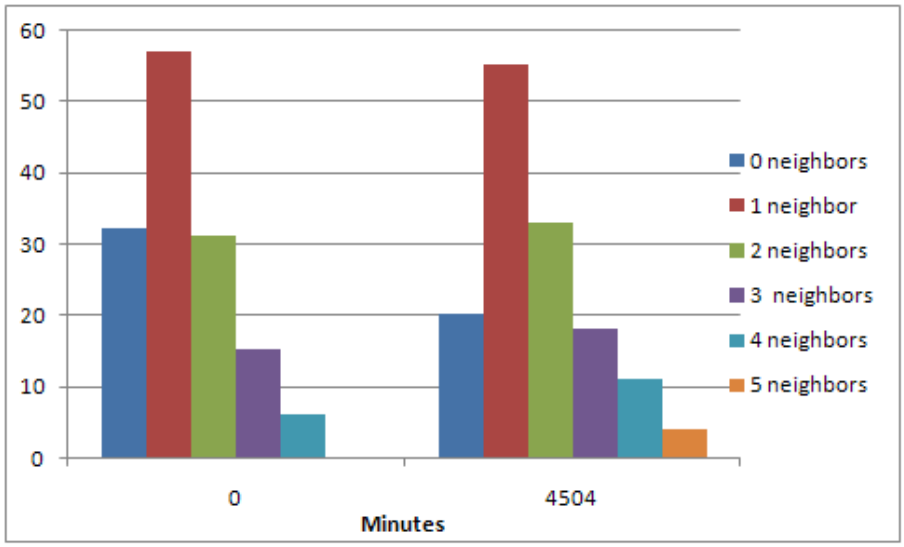

Figure 5.5: Number of agents having 0, 1, 2, 3, 4, 5 neighbors for p_o 0.1

P_o	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*	*ADD*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A	1.33 ±1.06
0	21.86 ±21.2	8.5 ±5.78	30.3 ±22.1	19.1	79.9	0.007	43.6 ±10.7	1.33 ±1.06
0.001	17.9 ±21.04	11.89 ±10.74	29.79 ±23.3	13.93	85.41	0.61	46.56 ±19.15	1.61 ±1.29
0.1	21.73 ±21.17	8.6 ±5.86	30.33 ±22.04	18.77	80.39	0.8	43.87 ±12.7	1.86 ±1.34
0.4	21.57 ±21.4	8.65 ±6.04	30.22 ±22.24	20.74	78.43	0.78	42.79 ±14.72	2.02 ±1.477
0.8	21.61 ±21.3	8.64 ±6.05	30.25 ±22.17	20.21	78.95	0.8	43.07 ±13.54	2.02 ±1.45

Table 5.3: Accumulated statistics with different probabilities of self-organization using the flexible strategy (Combination2)

5.3.1 Strict Strategy

Table 5.4 displays the results of using the strict strategy in the simulation with self-organization given that the total number of runs is 4504 and the total number of cabs is 793. As a result of decreasing the total number of cabs, the average dispatch time and total waiting time increase but still lower in comparison with the existing system and the simulation without self-organization. The average degree distribution fails to converge when using different probabilities of self-organization p_o. This is because self-organization attempts to overcome the low total number of cabs by connecting more dispatch areas. Interestingly, the fleet utilization increased by around 25% because the self-organization is connecting more area pairs. As a result, more cabs are utilized.

Figure 5.6 plots the average dispatch time computed for each 300 time steps of each simulation run. Overall, the average dispatch time is decreasing over time with a bit fluctuation especially when $p_o = 0.001$.

The average pick up time computed for each 300 time steps is plotted in Figure 5.7. Generally, there is no difference when compared with experiments that uses higher total number of cabs.

The changes of the average total waiting time over time are plotted in Figure 5.8. Basically,

P_o	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*	*ADD*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A	1.33 ±1.06
0	32.6 ±24.8	8.95 ±6.39	41.63 ±25.3	58	40.4	0.01	22 ±20.9	1.33 ±1.06
0.001	17.9 ±21.04	11.8 ±10.7	29.7 ±23.3	13.9	85.4	0.6	46.5 ±19.15	1.6 ±1.29
0.1	4.5 ±10.07	12.5 ±10.7	17.02 ±14.6	1.17	98.55	0.24	53.72 ±17.7	1.76 ±1.3
0.4	4.04 ±9.1	12.62 ±10.94	16.67 ±14.18	0.92	98.89	0.14	53.9 ±17.8	1.81 ±1.34
0.8	3.6 ±8.72	11.95 ±9.76	15.56 ±13.16	0.89	98.9	0.085	53.95 ±17.26	1.85 ±1.38

Table 5.4: Accumulated statistics with different probabilities of self-organization using the strict strategy (Combination1)

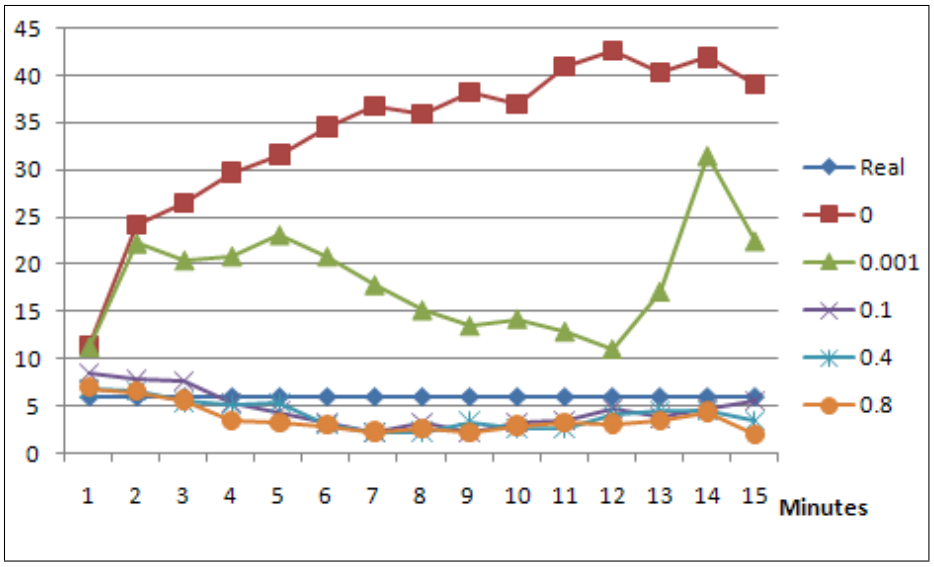

Figure 5.6: Average Dispatch Time for different values of p_o plus the existing system's one (793 cabs)

the behavior is similar to the behavior of average dispatch time over time plotted in Figure 5.6.

The simulation has been run for 9008 time units, to demonstrate the convergence of the average degree distribution of the area network while using a relatively low total number of cabs which is 793. The simulation runs are repeated once. Table 5.5 shows the convergence of the average degree distribution at probability of p_o 0.4. The degree distribution starts to converge at probability of p_o 0.4. However, all other measurements are indifferent from Table 5.4.

The changes of the average degree distribution over time is illustrated in Figure 5.9 for p_o = 0.4. The average degree distribution starts to converge at around 4500 time unit.

Finally, Figure 5.10 plots the number of agents which have a number of neighbors: 0, 1, 2, 3, 4, 5 respectively at both the first time step and the last one for the simulation using p_o of 0.4. The overall degree distribution has changed. There is a small drop in isolated areas (with 0 neighbors) at the expense of slight increase in areas with 3,4, and 5 neighbors.

The results demonstrates that the simulation operates efficiently and approximately performs the same even when a lower total number of cabs is used.

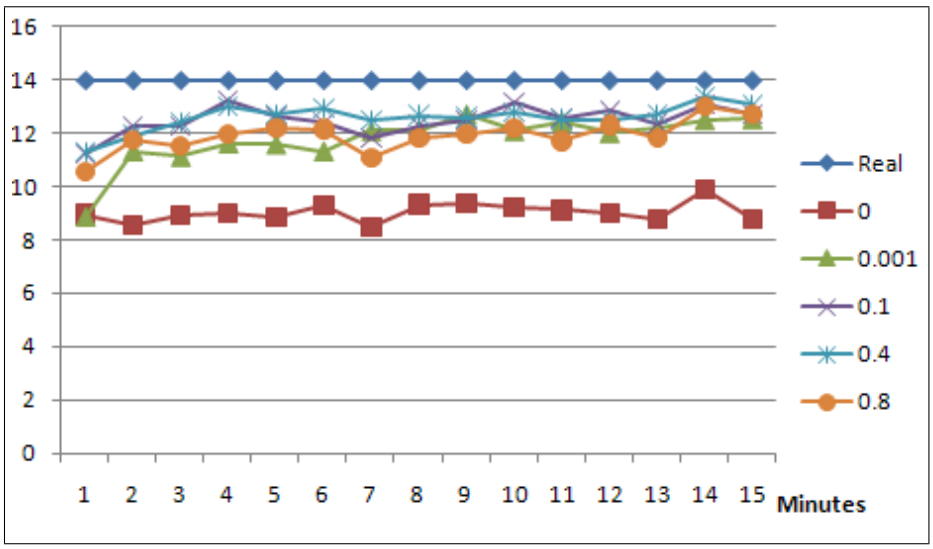

Figure 5.7: Average Pick up Time for different values of p_o plus the existing system's one (793 cabs)

5.4 Related work

The proposed technique is similar to the self-organization technique used by Abdallah and Lesser [2]. The work suggests to optimize the underlying network of agents by allowing agents to add or remove neighbors in a task allocation problem. The self-organization technique is combined with multiagent reinforcement learning to optimize the local policy of each agent in the multiagent system.

Other work presented self-organization techniques to allow forming coalitions or teams of agents. The work of Goldman and Lesser [21] presents a self-organization approach to allow agents to form suitable coalitions via negotiation. The work presents self-organization techniques that is domain dependent unlike the proposed technique in this thesis. Similarly, Gaston and desJardins [13] present a self-organization approach to form teams of agents.

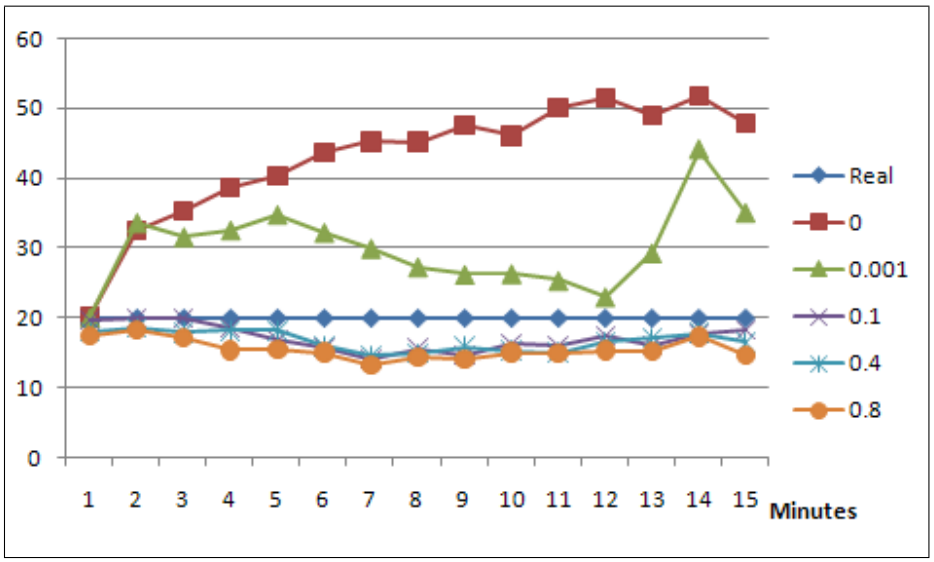

Figure 5.8: Average Total Waiting Time for different values of p_o plus the existing system's one

P_o	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*	*FU*	*ADD*
Real	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A	N/A	1.33 ±1.06
0	36.06 ±24	9.01 ±6.54	45.08 ±24.53	71.5	27.75	0.73	30.3 ±32.8	1.33 ±1.06
0.001	15.78 ±19.1	12.45 ±11.18	28.24 ±21.67	8.45	99.87	0.12	99.81 ±45.82	1.68 ±1.29
0.1	3.14 ±8.12	12.69 ±11.2	15.83 ±13.9	0.67	99.9	0.034	108.37 ±30.22	1.79 ±1.33
0.4	2.97 ±7.7	12.46 ±10.42	15.44 ±12.95	0.52	99.9	0.06	108.5 ±28.8	1.82 ±1.32
0.8	2.9 ±7.96	12.39 ±10.6	15.32 ±13.17	0.64	99.9	0.034	108.4 ±28.2	1.83 ±1.39

Table 5.5: Accumulated statistics with different probabilities of self-organization using the strict strategy (The runs of the simulation are repeated once)

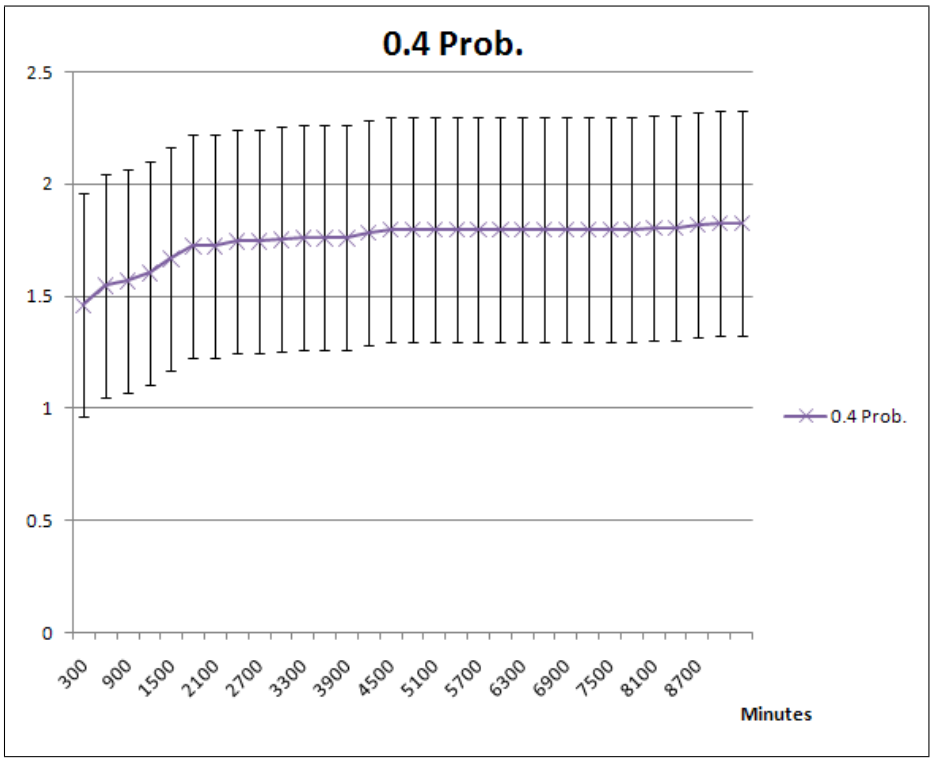

Figure 5.9: Average Degree Distribution for p_o 0.4

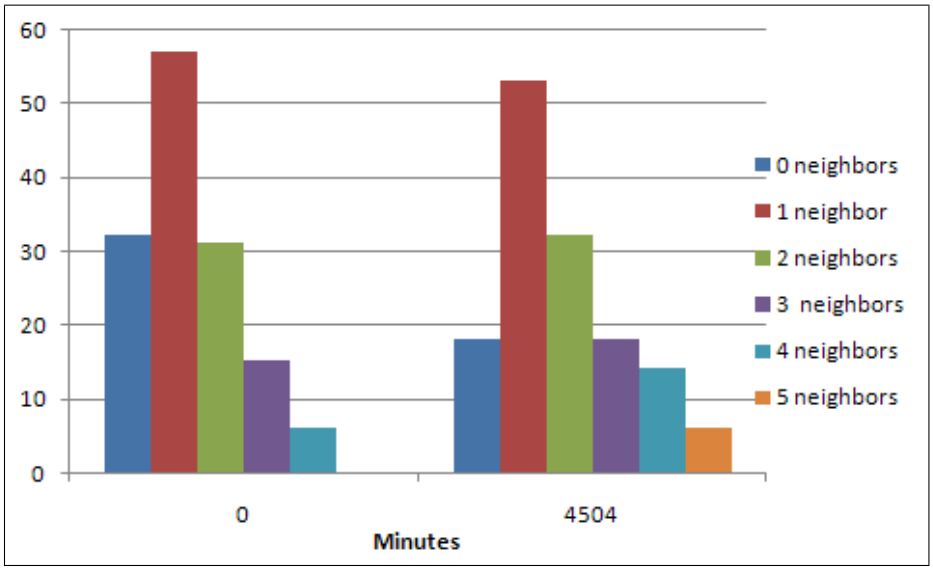

Figure 5.10: Number of agents having 0, 1, 2, 3, 4, 5 neighbors for p_o 0.4

Chapter 6

Multiagent Reinforcement Learning for Adaptive Dispatch Policies

In this chapter, a Multiagent Reinforcement Learning method is applied to the dispatch problem (without self-organization). The method allows adaptive dispatch policies for each agent. The results of the simulation in conjunction with MARL are compared to the existing system, the simulation of the existing system and the simulation with self-organization technique.

This chapter presents an introduction to Markov Decision Process (MDP) and Mulitagent Reinforcement Learning. Then, it describes how Multiagent Reinforcement Learning is incorporated into the simulation using both the original and the updated networks of agents that has resulted from using Self-Organization. The network of areas is assumed to be fixed during the learning process. After that, it displays the experimental results conducted at this phase and evaluates them. Finally, the chapter presents related work to the reinforcement learning method.

6.1 Overview of Markov Decision Process and Multiagent Reinforcement Learning

Markov Decision Process (MDP) is a common method to model the decision process of an agent [23]. MDPs offer a simple precise way of formulating the problem of an intelligent agent interacting with its environment to fulfill a goal through planning or learning [23]. In reference to the description in [23], the MDP framework consists of one or more agents along with their underlying environment. At each time unit t, an agent i observes the environment and identify the current state s_t. Accordingly, the agent selects an action a and executes it. Based on that, the agent gets a feedback from its environment which is represented by the reward r. Also, the state of the environment is updated to be s_{t+1}. At each time unit, the agent updates the probability of choosing a particular action at certain state which is called the policy pi_t. The aim of each agent is to choose the best action at each state to maximize the reward it gets from the environment [23]. Figure 6.1 depicts the Markov decision process. To summarize, the Markov decision process model consists of:

Agent(s) The agent or the agents that execute tasks to fullfil a designated goal defined by the multiagent system they belong to.

State Each state in the state space of each agent $s \in S$.

Action Each action in the action set of each agent $a \in A$.

Reward The reward r that an agent i gets from the environment as a result of executing action a at state s. The reward $r \in \Re$ is a real number that could be positive or negative to represent a penalty.

Discount Factor The discount factor $0 <= \gamma <= 1$ discounts the future rewards.

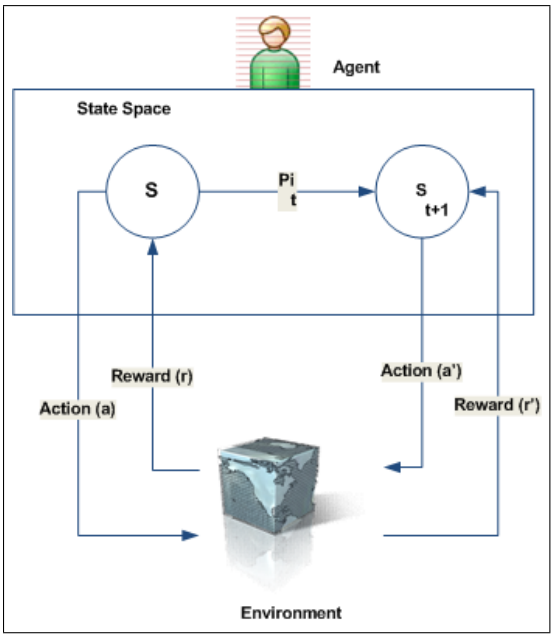

Figure 6.1: Markov Decision Process

The complexity that yields from the interaction between agents in a multiagent system can increase as the number of agents increases [16]. Therefore, machine learning, particularly reinforcement learning can be utilized to solve the scalability and the complexity of this problem by automating the search and optimization process [16]. The problem of multiagent systems can be formulated as a Markov Decision Process (MDP) to in order to apply a multiagent reinforcement method. Learning helps agents to make decisions at each state it encounters by discovering by its own how to maximize utility or fulfill a given task via repeated trials [16]. The process is domain-independent, so it requires minimal information about the environment which overcomes the complexity of some problems [2]. One of the common methods of Multiagent Reinforcement Learning is Q-Learning. It simply learns the reward an agent gets from the world while executing an action at a specific state [15]. The method is described as follows in [15]. Each agent has three data structures for states, actions and combinations of state-action. State-Action data structure Q_i stores the action value of executing each action at a specific state. The data structure Q_i is updated whenever the agent executes an action and accordingly receives a reward from the environment. The cell of action value (s, a) encapsulates the action value of executing action a at state s. The higher the action value (s, a), the more probable that the agent is going to execute action a at state s since the action is getting good reward from the world. Cell (s, a) summarizes what an agent learns via interaction with other agents and with the environment. The Q-Learning algorithm is represented by the following equation.

$$\mathrm{Q}_t(\mathrm{s,a}) = (1\text{-}\alpha)Q(s,a) + \alpha[r + \beta max Q(s,a')] \quad (6.1)$$

In Q-learning, the agent starts with random initial values of Q(s, a) for all s $\in S, a \in$ A. At each time t, the agent chooses an action and observes its reward r. The agent then updates its Q-values based on the mentioned equation where $\alpha \in [0; 1)$ is the learning rate. The learning rate α needs to be reduced over time in order for the learning algorithm to converge [15]. β is the discount factor that is applied to the associated utilization of an action. The value of β is

$\in [0; 1]$

The limitation of MARL is that it requires excessive communication between agents especially when the number of agents is high which slow down the learning process to converge. Thus, a good idea to deal with this problem is to organize agents into an *Overlay Networks* so that each agent communicate only with its neighboring agents [2].

6.2 Implementation of Multiagent Reinforcement Learning

The taxi dispatch problem can be mapped to a Markov decision process as follows.

State The state of an agent consists of the number of vacant cabs and the number of waiting calls in the main area of the agent and each neighbor of the agent if available. In addition, the state includes the task time and location (area). A state consists of the following parameters:

1. Call Location (area)
2. Call Hour
   (a) 0: hour = 9
   (b) 1: hour >9 and hour <14
   (c) 2: hour = 14
3. Subset state of neighbors including self
   (a) Vacant Cabs
      i. 0: calls = 0
      ii. 1: calls >0 and calls <10
      iii. 2: calls between 10 and 50
      iv. 3: calls >50
   (b) Waiting Calls
      i. 0: cabs = 0
      ii. 1: cabs >0 and cabs <5
      iii. 2: cabs between 5 and 10
      iv. 3: cabs >10

Action Each agents has its own set of actions depending on its neighbors. The actions are the following dispatch policies: (Action 0) dispatching the incoming call to the cab that has been vacant the longest in the area with maximum number of vacant cabs (Action 1) dispatching the incoming call to a vacant cab at the main area of the agent (Action 2) waiting and doing nothing (Action 3) dispatching the incoming call to the cab that has been vacant the longest among the neighbors of the agent (if the agent has neighbors) (Action i >3) Routing the incoming call to a neighbor i - 3 (if the agent has at least one neighbor). Action (1) is disabled, if there are no vacant cabs in the main area of the agent. Also, Action (3) is disabled, if there no vacant cabs in the neighbors of the agent (if the agent has at least one neighbor).0

Reward The reward is the total waiting time of a dispatched call in minus because the less the total waiting time, the better the reward. If the agent fails to dispatch an incoming call in 60 time units, the reward is set to -1000 to ensure dispatching incoming call faster.

Agent role

Each agent is responsible for (1) making a decision regarding an incoming call, (2) executing an action, and (3) learning about the executed action.

Process 1 For each incoming task, a decision is made whereby an action is selected to be executed from the available action set of the agent as described in Algorithm 6.1. The agent generates the current state so that the state is added to the agent's state space if it is not already a part of it.

I use decaying ϵ-greedy exploration strategy, where the exploration rate ϵ is initialized to be 0.5. Then, at each time unit t, $\epsilon = \epsilon$ / ((t / 100) + 1. Thus, a random number is generated between 0 and 1. If the generated random number is below ϵ, an action is randomly chosen from the agent's action list. Otherwise, the best action at the current state is chosen to be executed (policy of the current state). Then, the agent adds the combination of the current state and the selected action to its stateAction list if the combination is not already exists. Finally, a SEEKUPDATE message is sent to the dispatcher to get feedback of the associated task.

Algorithm 6.1: Agents decide REQUEST messages

Input: All tasks at time unit i
Output: A decision is made regarding each task
foreach *REQUEST message j received at this cycle* **do**
  Generate current state s;
  if *s doesn't match any state in the state list* **then**
    add State s to the state list of the agent;
    Disable impossible action 1 or action 3 at this state;
  end
  randomNumber = p;
  if $p > \epsilon$ **then**
    Choose the policy of state c;
  else
    Choose an action randomly ;
  end
  Execute the chosen action;
  register for feedback about the task under consideration;
  Generate Current StateAction n using the selected action and the current state ;
  if *n doesnt match any stateAction in the stateAction list* **then**
    add StateAction n to the stateAction list of the agent;
  end
  Send seekUpdate message to get feedback of the task whenever executed;
end

Process 2 The process is described in Algorithm 6.2.

Algorithm 6.2: Execute an action

```
Input: action j and task i
Output: Executing action j
switch chosen action j do
    case Action 0
        The task is dispatched to the cab that has been vacant the longest in the area
        with max number of vacant cabs;
        Learn;
    end
    case Action 1
        The task is dispatched to the cab that has been vacant the longest at self;
        Learn;
    end
    case Action 2
        No Action is taken regarding the task at current time unit;
        Send REQUEST message to self to be delivered at next time unit;
    end
    case Action 3
        The task is dispatched to the cab that has been vacant the longest among the
        neighbors;
        Learn;
    end
    case Action i >3
        The task is routed to neighbor i - 3 to take a decision regarding the task at next
        time unit;
        Send REQUEST message to the neighbor to be delivered at next time unit;
    end
end
```

Process 3 When a task completes, all agents that have been involved in routing the task receive a reward signal = -Total Waiting Time. It should be noted that the reward signal for the same task is different for different agents, because the waiting time includes routing time as well. Agents use the Q-learning algorithm which is best described by the following update equation:

$$Q_t(s,a) = (1\text{-}\alpha)Q(s,a) + \alpha[r + \beta maxQ(s,a')] \quad (6.2)$$

where α is the learning rate and β is the discount factor. The discount factor $\beta = 0.9, \alpha = 0.1$

The state that is resulted from executing an action is generated. Then, the agent updates the corresponding action value. Since the goal of agents is to dispatch calls such that the total waiting time of each call is minimized, the reward of executing action n = - Total Waiting Time. The learning rate ϵ is described earlier and the discount β factor = 0.9. Appendix A provides more details.

6.3 Experimental Results

The performance of the simulation in conjunction with the Mulitagent Reinforcement Learning method is investigated via conducting a number of experiments. The same measurements of performance in Chapter 4.3 located at Table 4.2 are used in these experiments. The following experiments either use the static strategy (Combination1) or the flexible strategy (Combination 2) and run for a total number of 67560 simulation steps. The experiments use the original adjacency schedule of dispatch areas.

Table 6.1 displays the accumulated results of applying the Multiagent Reinforcement Learn-

ing method to the simulation besides the results of the real system, the simulation of the existing system and the simulation with self-organization ($P = 0.1$) for reference. Both strategies of moving vacant cabs are used in the experiments.

Overall, the results are the same with the two strategies of moving vacant cabs because the agents using MARL attempt to achieve the policy at each state regardless of current cabs' locations. Noticeably, the MARL method increases the average pick up time and consequently the average total waiting time is increased. This is expected since agents starts choosing actions randomly at the beginning of the simulation run, so this delays the results to be improved. To illustrate the learning curve of agents, I display the periodic results at each 300 time unit using the strict strategy.

Rule	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*
Real system	5.45 ±9.8	14.5 ±17.2	20 ±20.3	0	100	N/A
Simulation	20.5 ±21	8.5 ±5.7	29 ±22	17	82.2	0.7
Self Org: 0.1	2.8 ±7.8	12.53 ±10.9	15.39 ±13.5	0.6	99.1	0.16
MARL Comb1.	0.56 ±2.2	12.8 ±11.4	13.39 ±11.5	0.00004	99.9	00.00002
MARL Comb2.	0.58 ±2.3	12.7 ±10.9	13.3 ±11.4	0.00006	99.9	00.00003

Table 6.1: Accumulated statistics with MARL using different movement strategies

Overall, MARL method decreases the average total waiting time dramatically by 15.5 minutes in comparison with simulation and by 6.5 minutes in comparison with the existing system. However, the average pick up time increases by 4 minutes in comparison with the simulation because of the new introduced dispatch policies. The policy of dispatching an incoming call to the cab that has been vacant the longest in the area with maximum number of vacant cabs affects the average pick up time because the area might be close or far away from the main area of the call. On the other hand, the policy is useful in case there are no vacant cabs in the main area or the adjacent areas of the call so it reduces the dispatch time. The policy of routing an incoming call to a neighboring area affects the average pick up time as well, because the neighbor might dispatch the call to one of its neighbors which might be distant from the call's location. Nevertheless, the policy is still beneficial in terms of overcoming the shortage of vacant cabs in the main area or the adjacent areas of the call. The success rate has increased to reach approximately 100%.

To study the combination of self-organization and multiagent reinforcement learning in the taxi dispatch system, an experiment is conducted using the updated adjacency schedule obtained by applying the self-organization technique in Chapter 5. Table 6.2 displays the results of applying MARL on self-organized schedule besides the results of applying MARL on the original schedule which is row 3 in Table 6.1.

Rule	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*
Original Schedule	0.56 ±2.2	12.8 ±11.4	13.39 ±11.5	0.00004	99.9	0.00002
Updated Schedule	0.64 ±2	13.37 ±11.1	14 ±11.6	0.00004	99.9	0.00001

Table 6.2: Accumulated statistics with MARL using different adjacency schedule

Obviously, the average total waiting time is higher when using the updated schedule with MARL as well as the average pick up time. Because self-organization connects some distant areas, the pick up time rises. Thus, agents might need more time to decrease the average total waiting time. Also, a need arises to combine self-organization with MARL following the work of Abdallah and Lesser [2]. The combination of the two methods will remove distant neighbors and speedup the learning process.

Following are the periodic statistics for the simulation with MARL using the strict strategy and the original adjacency schedule. Figure 6.2 plots the average dispatch time. Initially, the

averaged dispatch time starts high, then declines because of the penalty that is set for agents when they delay dispatching the call. Although the average dispatch time declines gradually, there are some fluctuations. This is justifiable because agents try to optimize two contradicting factors which is reducing the average pick up time and the dispatch time in the same time. The reduction of both times may be difficult because sometimes there might be no vacant cabs in nearby areas, whereas agents have to dispatch calls at the earliest to avoid cancellation of the calls and consequently getting a penalty.

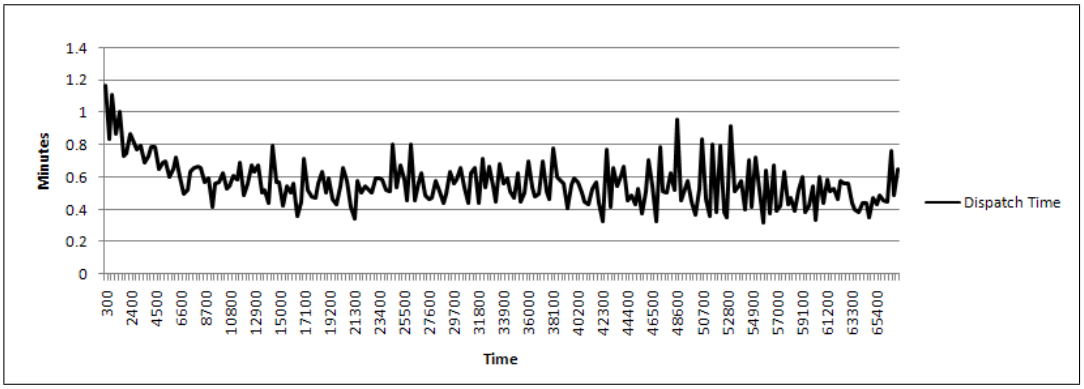

Figure 6.2: Average Dispatch Time for the simulation with MARL

Figure 6.3 plots the average pick up time computed. The average pick up time starts a bit high then it declines gradually in small steps to reach a sort of convergence at time unit 35000 and the average pick up time reaches a minimum of 11.8 minutes. Although the results converge, there some fluctuations in the average pick up time that are caused by the changes in the availability of vacant cabs.

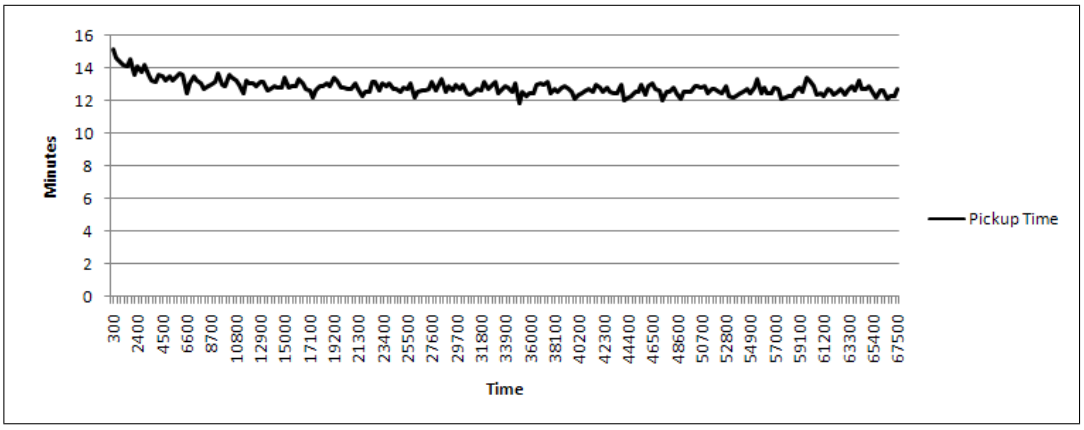

Figure 6.3: Average Pickup Time for the simulation with MARL

The average total waiting time plotted in Figure 6.4. The behavior of the average total time is very similar to Figure 6.3.

To demonstrate the stability of the proposed MARL method, two more experiments have been conducted using the static strategy. The three experiments are displayed in Table 6.3 given that the first row is Row 4 in Table 6.1. The results converge in all of the experiments in Table 6.3.

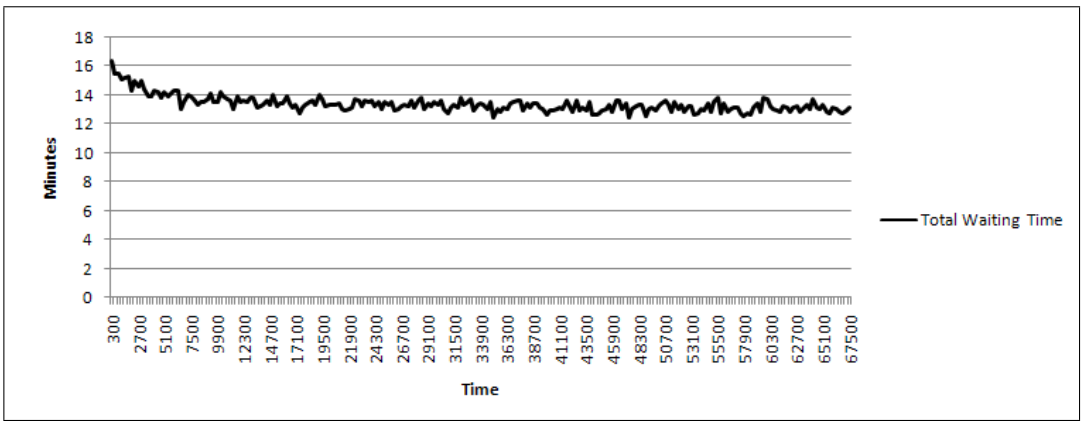

Figure 6.4: Average Total Waiting Time for the simulation with MARL

Rule	*ADT*	*APT*	*ATWT*	*CR*	*SR*	*WR*
Exp1	0.56 ±2.2	12.8 ±11.4	13.39 ±11.5	0.00004	99.9	0.00002
Exp2	0.56 ±2.2	12.74 ±10.9	13.3 ±11.42	0.00004	99.9	0.00003
Exp3	0.58 ±2.3	12.79 ±11	13.37 ±11.53	0.00003	99.9	0.00003

Table 6.3: Accumulated statistics with MARL using the strict strategy (Combination1)

6.4 Related Works

The used reinforcement learning algorithm does not only depend on the agent's actions and states but also neighbors' actions and states. Whereas, the work of Abdallah & Lesser 2008 [3] assumes that agents doesn't observes other agents' states or actions by introducing the Weighted Policy Learner (WPL).

Also, the MARL method uses a variable decaying learning rate to ensure the convergence of results at the end. Bowling and Veloso [5] investigated some learning algorithms to show how they fail to converge. They also proposed a reinforcement learning technique that uses a variable learning rate to overcome limitations of other algorithms.

Chapter 7

Future Work and Conclusion

The taxi dispatch problem involves assigning taxis to callers waiting at different locations. Adjacency-based taxi dispatch systems use fixed areas to search in while receiving calls. Usually, the calls are dispatched to taxis within the call's area. If there are no vacant cabs in the call's area, the system searches in the deffined adjacent areas.

To apply two different multiagents methods separately to an adjacency-based taxi dispatch system, a computer simulation of an existing taxi dispatch system, of a major Taxi company in the Middle East, is developed. The simulation uses the same dispatch policy used by the current system and also uses real data in generating calls and computing travel delay. The simulator also mimics some behaviors of dispatch parties such as the dispatch officers and the drivers by moving taxis towards congested areas. The simulation is bench-marked against the real system in terms of customer waiting time (respecting customer satisfaction). The significance of the simulation extends beyond the immediate needs of this thesis, and can be used to evaluate other methods in the future.

The thesis proposes a multiagent self-organization technique to optimize the adjacency list of the taxi dispatch system. The effeciency of the technique has been demonstrated by simulating an existing taxi dispatch system and incorporating the self-organization technique, which allows agent to add or remove other agents from their neighborhood. Agents are dispatch areas, tasks are calls and resources that fulfill the tasks are taxis. Experimental results show that the proposed self-organization technique decreases the total waiting time by up to 25% in comparison with the real system and increases taxi utilization by 20% in comparison with results of the simulation without self-organization. The dissertation also proposes the first multiagent reinforcement learning solution to the taxi dispatch problem. Experimental results show that the proposed MARL method decreases the total waiting time by 33.5% in comparison with the existing system and by more than 54% in comparison with the simulated system.

One avenue of future research is improving the simulator itself, in order to model its behavior more accurately. Jobs obtained by passenger pickup from streets (without calls) to be added to the simulation using real data. Also, the simulation is suggested to be adapted to deal with Global Positioning System (GPS) locations.

I am currently working on (gradually) implementing the proposed approaches in the real live taxi dispatch system. I am also investigating the use of multiagent reinforcement learning in combination with the self-organizing technique, following the work of Abdallah and Lesser [2] to decrease the total waiting time further. Another interesting point is investigating driver fairness, so that drivers roughly receive the same number of calls.

Another avenue of future research is exploring other MARL and self-organization algorithms, and comparing those with the specific algorithms I used in this this thesis.

Bibliography

[1] Adaptive technologies inc. http://www.adaptiveinc.com/Research/glossary.php. Glossary.

[2] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 172–179, Honolulu, 2007. IFAAMAS.

[3] S. Abdallah and V. Lesser. A Multiagent Reinforcement Learning Algorithm with Non-linear Dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008.

[4] S. Apel and K. Böhm. Self-organization in overlay networks. In *Proceedings of 1st CAISE05 Workshop on Adaptive and Self-Managing Enterprise Applications*, Porto, Portugal, 2005.

[5] M. Bowling and M. M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215 – 250, 2002.

[6] C.-H. Chen. Distributed taxi hailing protocol in vehicular ad-hoc networks. Master's thesis, Submitted to the Department of Computer Science and Information Engineering at the Central National University in Taiwan for the degree of Master on 2008-07-23, 2008.

[7] L. C. Chung. GPS taxi dispatch system based on A* shortest path algorithm. Master's thesis, Submitted to the Department of Transportation and Logistics at Malaysia University of Science and Technology (MUST) in partial fulfillment of the requirements for the degree of Master of Science in Transportation and Logistics, 2005.

[8] P. Davidsson, L. Henesey, L. Ramstedt, J. Trnquist, and F. Wenstedt. Agent-based approaches to transport logistics. In F. Klügl, A. Bazzan, and S. Ossowski, editors, *Applications of Agent Technology in Traffic and Transportation Book*, Whitestein Series in Software Agent Technologies. Birkhäuser Basel, Basel, Switzerland, 2005.

[9] M. M. de Weerdt, R. P. van der Krogt, and C. Witteveen. Resource based multi agent plan merging: framework and application. In *Proceedings of the 22nd Annual Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig)*, pages 218–233, 2003.

[10] L. Der-Horng, H. Wang, R. L. Cheu, and S. H. Teo. A taxi dispatch system based on current demands and real-time traffic conditions. In *Transportation Research Record*, pages 193–200. National Research Council, Washington, 2004.

[11] K. Dorer and M. Calisti. An adaptive approach to dynamic transport optimization. In F. Klügl, A. Bazzan, and S. Ossowski, editors, *Applications of Agent Technology in Traffic and Transportation Book*, Whitestein Series in Software Agent Technologies, pages 33–49. Birkhäuser Basel, Basel, Switzerland, 2005.

[12] S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks. In *Advances in Physics*, pages 1079–1187, 2002.

[13] M. E. Gaston and M. Desjardins. Agent-organized networks for dynamic team formation. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 230–237, New York, NY, USA, 2005. Association for Computing Machinery (ACM Press).

[14] M. E. T. Horn. Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation research. Part C*, 2002.

[15] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250. Morgan Kaufmann, 1998.

[16] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[17] D. Perugini, D. Lambert, L. Sterling, and A. Pearce. Provisional agreement protocol for global transportation scheduling. In *Workshop on agents in traffic and transportation held in conjunction with the International Conference on Autonomous Agents and Multi Agent Systems*. 2004.

[18] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *AAMAS 02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1191–1198. Association for Computing Machinery (ACM Press), 2002.

[19] K. T. Seow, N. H. Dang, and D.-H. Lee. Towards an automated multiagent taxi-dispatch system. In *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference*, pages 1045–1050, Scottsdale, Arizona, USA, 2007.

[20] M. Shrivastra, P. K. Chande, A. S. Monga, and H. Kashiwagi. Taxi dispatch: A fuzzy rule approach. In *IEEE Conference on Intelligent Transportation Systems*, pages 978–982, Piscataway NJ, 1997.

[21] M. Sims, C. Goldman, and V. Lesser. `Self-Organization through` Bottom-up Coalition Formation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, pages 867–874, Melbourne, AUS, July 2003. Association for Computing Machinery (ACM Press).

[22] I. Stoica. Overlay networks. Available at http://www.cs.virginia.edu/ cs757/slidespdf/757-09-overlay.pdf. UC Berkeley.

[23] R. S. Sutton. On the significance of markov decision processes. In *7th International Conference on Artificial Neural Networks (ICANN)*, pages 273–282. Springer, 1997.

[24] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2002.

Appendix A

Implementation

This chapter provides technical details of the implementation of the taxi dispatch simulation. Also, the chapter elaborates more about the design details of the simulation described in earlier chapters. The limitations of the system are also highlighted at the end of the chapter. The code is based on Whatrain Taxi Dispatch Simulator available at http://www.whatrain.com/taxi/ and the simulator of Dr. Sherief Abdallah available at http://dis.cs.umass.edu/ shario/dtap.

A.1 Main System Architecture

The proposed taxi dispatch system has been developed in Java. The developed application is single threaded and retrieves data from a Microsoft Access Database. Thus, the main two layers of the system is the data layer and the application layer itself. There is no client interface because the interaction of the client is limited to defining the parameters and viewing results. The basic architecture of the system is illustrated in Figure A.1. According to the contributions of the thesis, there are two versions of the simulation: (1) the simulation of the existing dispatch system with self-organization option (2) the simulation with multiagent reinforcement learning method.

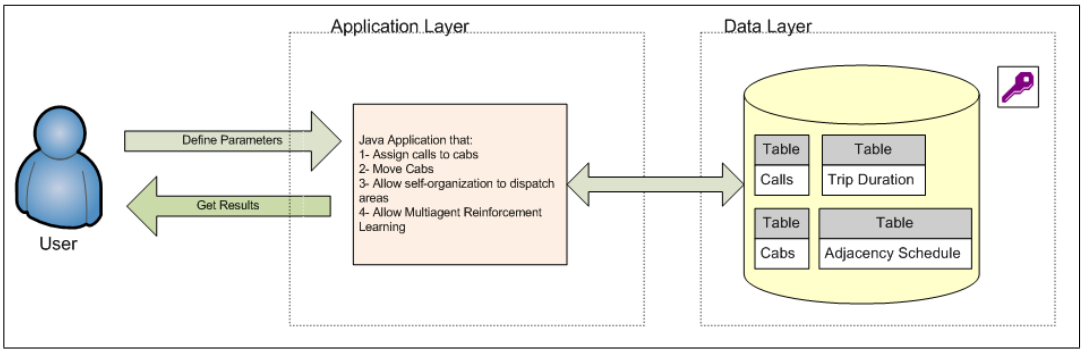

Figure A.1: System Architecture

The application follows the object oriented programming paradigm. Main objects of the system are represented by classes that interacts to carry on the main activities of the system.

A.2 Simulation with Self-Organization option

Figure A.2 summarizes the simulator job. The sub-processes (in gray color) that take place at each simulation run are described in details in the following sub-sections. First, the simulator

retrieves the number of cabs at each area and place them in a list. Second, the user specifies the total number of times steps, so the run counter is initialized to zero. At each time step, the simulator retrieves the number of calls per area, the simulator attempts to dispatch the calls, and the cabs are moved toward their destinations.

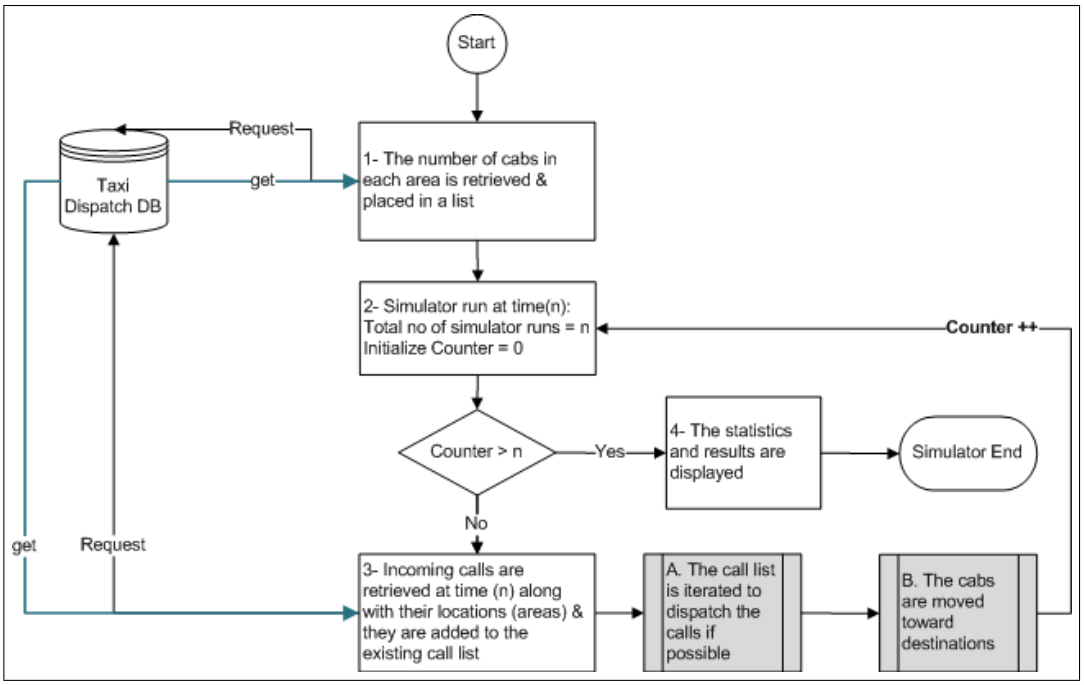

Figure A.2: Simulator Design

A.2.1 Dispatch Process

The dispatch process is already described in Chapter 4. When call i is dispatched to cab j at time n, the following events take place:

1. Cab j destination is changed to be call i location
2. The status of cab j is changed to "On Call "
3. The status of call i is changed to "Dispatched "
4. Call i is removed from the active call list
5. Call i is added to the dispatched call list
6. The dispatch time of call i = time n - call i time
7. Duration to pick up the customer is calculated (**a predefined process**)
8. Duration to destination is calculated (**a predefined process**)
9. Service time at which the customer is picked up = time n + duration between cab i location and call j location
10. Drop off time at which the customer is reached his/her destination = Service time + duration between call j location and call j destination
11. Pick up time of call j = service time - call i time
12. Total waiting time of call j = pick up time of call j + dispatch time of call j

The pre-defined process of calculation of duration between each pair of areas is described as follows.

Duration Calculation

Durations between some pairs of areas are not available in the obtained data because there might be no trips between those pairs of areas. While, in the simulation a need might arise to know the missing duration to move some vacant cabs. For example, suppose that duration between area b and area a at hour h_i is not available in the obtained data. Whereas the simulator assigns a call in area a to a vacant cab at the adjacent area b, so the duration between area b and area a must be calculated. In this case the simulator checks the availability of the duration between area a and area b in the preceding hour to use it. Otherwise, the simulator tries to find the closest duration. The process is summarized in the Table A.1 to calculate the duration between area a and area b at hour h. Once the simulator finds the duration at any point, it stops searching and assigns the obtained value to the duration. Otherwise, the simulator checks the availability of the next value.

Point	Value
1	The average and standard deviation of duration between area a and area b at hour h
2	The average and standard deviation of duration between area a and area b at hour h - 1
3	The average and standard deviation of duration between area a and area b at hour h + 1
4	The average and standard deviation of duration between area a and area b
5	The symmetric average and standard deviation of duration between area b and area a
6	the duration is assumed to be 23 minutes and the standard deviation is zero

Table A.1: Duration Calculation

When the simulator assign a value to the duration, a Gaussian distribution is generated using the mean and standard deviation. Then, a random number is generated using the distribution to be re-assigned to the duration.

A.2.2 Movement of cabs

All cabs that have destinations set for them are moved toward these destinations. Thus for cabs that are "on call"or "occupied"or "moving"and the current time unit n = estimated arrival time to destination, the status is updated as follows in Algorithm A.1:

Algorithm A.1: Cab Movements

```
Input: all cabs
Output: movement of cabs
foreach cab i at time n do
    switch cab i status do
        case on Call and arrival time = n
            cab i picked up calls ++;
            on call = false;
            occupied = true;
            call.pickedUp = true;
            location = call.location;
            Destination = call.destination;
        case occupied and arrival time = n
            cab i picked up calls −;
            cab i served call ++;
            available = true;
            occupied = false;
            call.completed = true;
            call.pickedUp = false;
            location = destination;
            vacant since = n;
        case moving and arrival time = n
            location = destination;
            moving = false
        end
    end
end
```

A.2.3 Movement of vacant cabs

Strategy1(Vacant cabs in other cities move to predefined destinations) Since the dispatch system accepts jobs only in city a, cabs become idle and vacant outside city a if they drop off passengers outside city a. Therefore, they should return to city a to be able to receive jobs. Determination of which area in city a a cab shall go, is hard coded and obtained by running queries over the database log. Thus, the chosen areas relatively have high number of incoming calls, adjacent areas, and located near the vacant cabs.

- If the cab is at city b, the destination is set to area 87.
- If the cab is at city c or city d, the destination is set to area 64.
- If the cab is at north cities, the destination is set to area 11.

Strategy2 (The cab that is vacant the longest move to the area with maximum number of waiting calls) At each time unit, the cab that has been vacant the longest moves to the area with maximum number of waiting cabs if the number of vacant cabs moving toward the area is less than a threshold.

Strategy3 (Vacant cabs in other cities move to areas with max number of waiting calls)

Strategy4 (Nearest vacant cab move to area with calls that have being waiting the longest) At each time unit, the nearest vacant cab moves to the area with waiting calls that have the highest waiting time if the number of vacant cabs moving toward the area is less than a threshold.

The strategies are grouped as follows for testing purpose:

The strict strategy consists of Strategy1 and Startegy3 and described in details in algorithm A.2.

Algorithm A.2: Rule 2

```
Input: some vacant cabs located outside city a
Output: all vacant cabs in city a
foreach vacant cab h located outside city a do
    if there is an area i in city a with max number of waiting calls then
        move cab h to area i;
    else
        switch cab h location do
            case cab h at area a
                The destination of cab h is set to area 87
            case cab h at area b or c
                The destination of cab h is set to area 64
            case cab h at north cities
                the destination of cab h is set to area 11
        end
    end
end
```

The flexible strategy consists of Strategies 1,2,3,4.

A.2.4 Parameters Configuration

The following parameters should be specified prior to running the simulation. Based on defined values of parameters, the simulation operates.

1. Cab Distribution
2. Total Number of Runs
3. Number of runs at which periodic statistics are collected
4. Probability at which Self-Organization technique is run
5. Strategy of movement of vacant cabs
6. The option of running self-organization
7. The option of repeating the simulation runs

A.3 Simulation with MARL

The simulation at this phase consists of two main classes: the dispatcher and the agents. The agent class consists of 141 instances representing the dispatch areas. The two classes interact to run the learning process along with the dispatch process. In order to incorporate MARL, the simulation is adapted to be decentralized. The simulation consists of two main roles: the dispatcher and the agents. The processes handled by the dispatcher role is described as follows.

Dispatcher Role

The dispatcher is responsible of (1) receiving calls each time unit, (2) assigning calls to agents, (3) moving cabs to destinations, (4) dispatching calls to cabs according to actions chosen by corresponding agents, (5) providing feedback of dispatched calls to concerned agents for learning purpose and (6) displaying statistics at the last run of the simulation.

Process 1 This process is already described in Chapter 4.

Process 2 Each incoming call is routed to the agent of the call's area to make a decision.

Process 3 This process is already described in Chapter 4.

Process 4 This process is dependent on the action that an agent executes regarding a call. If the agent chooses to dispatch a call locally. The call is dispatched to the cab that has been vacant the longest in the agent's area. The dispatch process is described in details in Section 4.2.

Process 5 Whenever a call is dispatched, agents that have executed actions regarding this call are informed of the total waiting time as it reflects the reward of the corresponding action.

Process 6 This process is already described in Chapter 4.

The simulation contains some supplementary classes: Call, Cab, Message, Action, State, and State-Action. The following UML diagram of the system describes in details the simulation with MARL. Figure A.3 summarizes the interaction between all classes in the simulation.

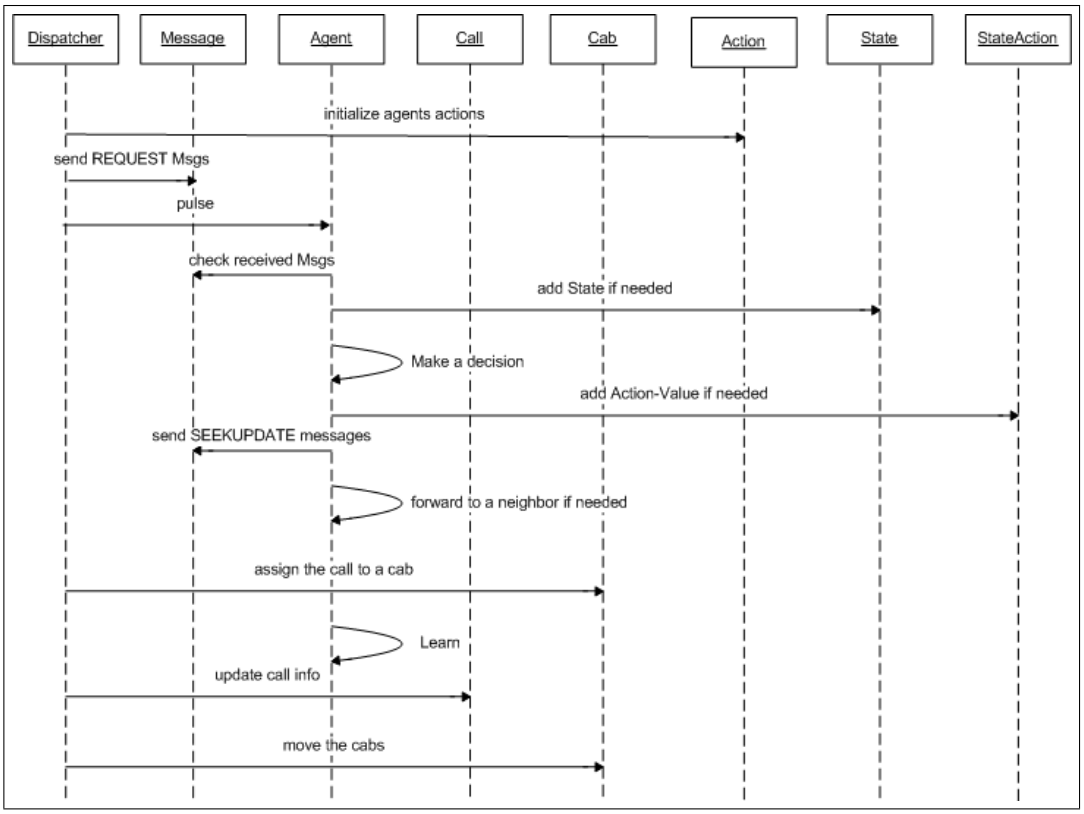

Figure A.3: Sequence Diagram

A.4 Database Structure

The database has been developed using Microsoft Access due to some reasons such as the ease of portability and configuration with JAVA. The database consists of four tables:

1. Trip Duration
2. Cab Distribution
3. Call Distribution
4. Area Adjacency Schedule

A.5 Technical Limitations

The simulation is not multi-threaded as existing dispatch systems. Instead, it synchronizes with an Access database to retrieve calls. Also, the simulation doesn't deal with GPS information. Other limitation is that the simulation doesn't have graphical interface to the user.