Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Petr Mezek**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Multi-agent job allocation mechanism for courier services**

Guidelines:

1. Familiarize yourself with job allocation in delivery services.
2. Formally specify the problem of job allocation in courier services.
3. Design a multi-agent mechanism for job allocation in courier services.
4. Implement the designed allocation mechanism.
5. Evaluate the properties of the allocation mechanism on a simulation model.

Bibliography/Sources:

[1] Bo Chen and H.H. Cheng. A review of the applications of agent technology in traffic and transportation systems. Intelligent Transportation Systems, IEEE Transactions on, 11(2):485-497, June 2010.
[2] Paul Davidsson, Lawrence Henesey, Linda Ramstedt, Johanna Törnquist, and Fredrik Wernstedt. An analysis of agent-based approaches to transport logistics. Transportation Research Part C: Emerging Technologies, 13(4):255-271, 2005.
[3] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. European Journal of Operational Research, 225(1):1-11, 2013.
[4] Gianpaolo Ghiani, Emanuele Manni, Antonella Quaranta, and Chefi Triki. Anticipatory algorithms for same-day courier dispatching. Transportation Research Part E: Logistics and Transportation Review, 45(1):96-106, 2009.

Diploma Thesis Supervisor: Ing. Michal Jakob, Ph.D.

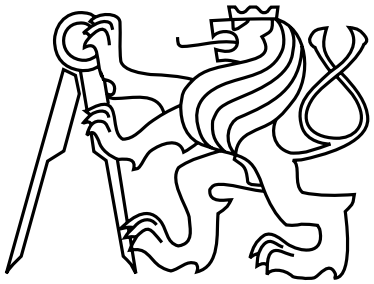
Valid until the end of the summer semester of academic year 2015/2016

doc. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, October 31, 2014

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Diploma thesis

Multi-agent job allocation mechanism for courier services

Bc. Petr Mezek

Supervisor: Ing. Michal Jakob, Ph.D.

Study Programme: Open informatics

Field of Study: Artificial intelligence

Aknowledgements

First, I would like to thank my supervisor Ing. Michal Jakob, Ph.D. for his time and valuable advises during the creation of this work. Second, I would like to thank Emanuel Buzek for a language assistance. And finally, I would like to thank my family and friends for their support.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions[14] about ethical principles for writing academic thesis.

Prague, 29. 12. 2014 .....

Abstract

These days delivery companies try to deliver a consignment to a customer as soon as possible. They focus especially on a same day delivery and immediate delivery service, therefore dynamic vehicle problem (D-VRP) begins to be a part of logistics planning systems.

This thesis presents the existing variants of D-VRP including solution methods. The variant studied by this work involves heterogeneous fleet, limited capacity, time windows, arbitrary pickup and delivery location without a necessity of visiting depot in a dynamic environment. The dynamic environment is defined by time dependent traffic model and consignment's generation in the real time.

The problem is formalized as an optimization task. A multi-agent system including planning algorithm was designed as a solution to assign new consignments to couriers.

The proposed model is transformed into a multi-agent simulation and evaluated on the real data. The evaluation results show that the multi-agent system is able to serve to human dispatchers as the real support system. The measured rate of reasonable assignment of consignment is up to 96%.

Abstrakt

V současnosti je patrná snaha přepravních společností o rychlejší doručení k zákazníkovi. Klade se důraz zejména na dodání v den objednání či dokonce na okamžité doručení. Do logistických plánovacích systémů se touto cestou dostává plánování tras vozidel v reálném čase, tzv. dynamic vehicle routing problem (D-VRP).

Práce představuje existující varianty D-VRP a možnosti jejich řešení. Pro podrobnější zkoumání byla vybrána varianta s heterogenním vozovým parkem, omezenou kapacitou, časovými okny, libovolným místem vyzvednutí a doručení v plně dynamickém prostředí (online generování nových zakázek a dopravní komplikace závislé na čase).

Problém je formulován jako optimalizační úloha. Pro řešení problému je navržen multiagentní systém (včetně plánovacího algoritmu), který přiřazuje nově příchozí zásilky kurýrům.

Představený model je nakonec převeden do multiagentní simulace. Na základě provedených experimentů s reálnými daty se zjišťuje, že výsledný systém je možné aplikovat v praxi jako podpůrný systém pro lidské operátory. Naměřená úspěšnost smysluplného přiřazení zásilky dosahuje až 96%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

These days many people expect service delivery immediately or at least as soon as possible from the moment they decide. This statement has extra validity on a field of logistics. There is clearly a change from next day delivery to the same day delivery. As an example, we can take a look at of Amazon [1] about immediately delivering by drons in near future. Some examples of dron operational deployment are available at the time of writing this study[2].

However, an immediate reaction and in fact a necessity of online planning raise a new challenge in front of delivery companies and developers of their IT systems. A shift from classical (and for many year extensively studied) static planning to its dynamic variant is in very early stage and additional research must be done.

Couriers companies providing same day delivery or even immediate delivery usually solve a planning problem by skilled human dispatchers. Nevertheless, human cognitive skills are limited, and no one can talk about properties of plans created by dispatchers. Dispatchers base plans mostly on their intuition and experiences. Although this approach is working and these companies prosper, what will happen when demand for these services increases dramatically and the size of planning domain exceeds dispatcher's cognitive skills? Will they still be able to create reasonable plans? Will they handle such pressure? Is it even possible to simply add more operators and solve the problem by working together as a team? These questions are open ones, but we can't wait with answers till delivery companies simply crack up.

We learned from the past examples that if we want continuous growth and improvement of service quality, it is often necessary to replace humans by machines or at least take away (automatized) part of human work.[3] Nevertheless, the research of dynamic logistics[4] is not currently sufficiently mature to create an expert system capable of full replacement of human operators. On the other hand, the development is mature enough to provide

[1] Amazon, an international leader in online commerce reveals its vision of dron delivery. <http://en.wikipedia.org/wiki/Amazon_Prime_Air>

[2] Gernam logistics company DHL starts its dron delivery in September 2014. <http://www.livescience.com/48032-dhl-drone-delivery-service.html>

[3] The most significant replacement of humans by machines was done during the industrial revolution. <http://en.wikipedia.org/wiki/Industrial_Revolution>

[4] By a term dynamic logistics we mean a solution of vehicle routing problem in real time with variable input data and also dynamic planning environment.

supporting system that will make operator's work easier and increasing throughput of the whole system.

Construction of such expert system is not an easy task and there does not exists universal approach for solving it. The goal of this thesis is to try to find some reasonable solution based on a multi-agent simulation. If we propose some system, a next task is to determine how good it is.

1.1 Thesis outline

First, we present a wide overview of related work in the Chapter 2. From this chapter we gain an inspiration to a problem formalization (Chapter 3) and especially to a design our multi-agent system. An extensive description of negotiation protocol and agents themselves is situated in Chapter 4.

Second, we have implemented proposed multi-agent system (implementation description has a dedicated Chapter 5) including an algorithm solving static case of vehicle routing (Chapter 4). Once the MAS was implemented, we run simulations with almost one thousand configurations and obtained valuable data.

Finally, we evaluate all measured data in Chapter 6 and realize that our system has a very good chance to fulfil a role of supporting system.

Chapter 2

Related work

Vehicle routing problem is a combinatorial problem defined for more than 40 year and consists in designing the optimal set of routes for fleet of vehicles in order to serve a given set of customers [5]. There are a various types of VRPs and a general difference among them is in the set of constrains like capacity of vehicles, time windows, personal contrains of each vehicle (length of shift [27], some vehicles cannot go to certain streets, etc.) or an extensions of basic definition about allowing more depots, customer requests are not only collected into depot but they should be delivered as well, and many others.

In this review we will mainly talk about the dynamic case of VRP. Let us recall the difference between static and dynamic case. In the static (a classical one) VRP, we have a problem instance where all the input data is known in the beginning and it is immutable. On the other hand, dynamic VPR (D-VRP) has only a part of data (or even none) known in the beginning and the data is revealing in time. An example of D-VRP is shown on Figure 2.1. The static VRP is a well known NP-hard problem and its dynamic variant is even more complex, that is why it is one of the most challenging combinatorial optimization task. In this chapter we introduce an area of suitable usage of D-VRP, we present a review of possible ways of D-VRP solving and finally, we discuss a problem of comparison in dynamic VRP.

[5] http://www.bernabe.dorronsoro.es/vrp/

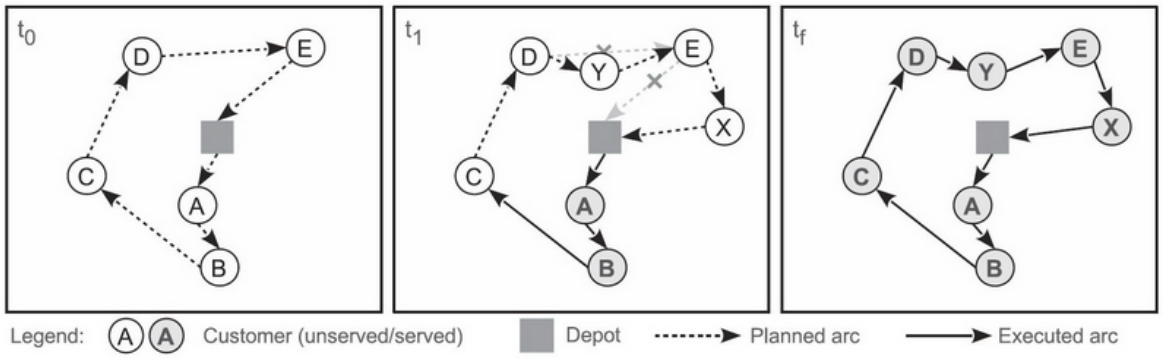


Figure 2.1: Example of dynamic vehicle routing [21]

2.1 Dynamic vehicle routing (D-VRP)

The traditional way of solving dynamic vehicle routing problem is through experienced human dispatchers. Here is clear that the level of dispatcher's experiences is critical for finding a good solution. Problems from real world are usually very complex and they have limited cognitive skills [3], that is why we cannot obtain an optimal solution by ourselves. So companies profits are not as high as they could be and the opportunity for intelligent systems is very high. [21][27]

Typical application of D-VRP is in city logistics [3][5] - courier services in urban areas. Couriers are sent to customers' locations to pickup packages and to deliver them in a short time (exact time window depends on the service level paid by the customer). Couriers can consolidate these requests into a route and satisfy several customers at once. Companies providing services of immediate often have a heterogeneous fleet including bicycles, motorbikes, cars and vans. Then the problem is to dynamically route couriers and to take into account not only existing requests, requests' type, pickup and delivery location but also traffic conditions and varying travel times as well. [21]

The first reference about dynamic VRP comes from 1977 [33], where authors studied single vehicle DARP and proposed algorithm simply inserts new request into the current route [21]. In the most cases customers want to pick up their parcel right now and deliver it as soon as possible. This concept is known as immediate delivery and it was introduced in year 1980 [23]. One could expect that after more than thirty years of extensive research everything has to be already done but the problem is so difficult that applications deployed in the real world are, even today, very rare. We can say that time invested into searching for better decision is paid by lower reactiveness of the whole system, so the importance of achieving a good decision as soon as possible is crucial.

From the functional point of view, systems solving dynamic VRP could be classified into three classes [21][4]:

- **decision support systems** - gives suggestions of possible solution to human dispatchers and the final assignment and responsibility for service quality is on human. [3]
- **partially automated systems** - perform automatic assignment of requests to vehicles but unexpected situations have to be solved by human dispatchers. A comprehensively described example can be found in [27] including process and companies diagrams.
- **fully automated systems** - human dispatchers are no longer necessary and the whole process (dispatching of couriers, rerouting, reactions on unexpected situations such as vehicle break down, etc.) is covered by an intelligent system. An extensive review of requirements on an intelligent dispatching system can be found in [3] and [27].

2.2 Solution methods

Solution methods for D-VRP are ranging from linear programming, through metaheuristics to multi-agent systems. Each method is suitable for certain type of D-VRP and they deal with the dynamism in different ways.

One of the earliest study of dynamic re-dispatching of vehicles was done in late 90's [24] (notice the beginning of GPS in 1995[6]). They examined several behaviour strategies of vehicles (diversion of current route to pickup next request, waiting till new request appears, etc.) and the authors discovered that waiting time for service (time between creating the request and picking it up) could be decreased significantly. Distance travelled by empty vehicles could be decreased by up to 25%.

2.2.1 Periodic re-optimalizing

Companies dealing with D-VRP usually have some customers' requests from the previous day and the requests can be used to create the initial route state per day (by running static variant of VRP solver). Authors of [22][15] presented an algorithm based on insertion of new requests into already existing routes. If there exists an empty vehicle, it gets the new request, otherwise 2-opt heuristics is used for optimizing the routes and for selecting the most suitable vehicle. This technique could of course lead into a state, when the original solution was optimal and by inserting requests into current routes we create a suboptimal one [21]. However, the advantage is a very fast reaction time (new solution in less than 1 second) and works surprisingly well [3]. Interesting feature of this approach is update of arrival time during insertion phase. Vehicles can react on changes in the traffic situation (traffic jam, car accident, etc.) and keep customers informed about the time of arrival. When the re-assignment is performed, the original vehicle remains also assigned to ensure that customer will be server on time (resulting in possibility of double service).

To obtain a good solution is time expensive. Some systems can provide a solution immediately and then optimize the solution every fixed time period. This technique is called pro-active and it is studied in [6]. Proposed solutions are modified dynamically to minimize travelled distance or to get vehicles closer to areas with highest probability of revealing new request. Probability distribution is based on historical data, therefore the optimal position of vehicle is determined by k-means algorithm in time-space. Average computation time is nearly one minute for instance with up to 12 vehicles and 150 customer request per day.

2.2.2 Heuristics

Heuristics are the basic stones of almost all algorithms solving D-VRP in reasonable time. Authors of [8] came up witch comparison of 6 heuristics.

- *construct* - rebuild the whole solution every time (new request comes)
- *construct+* - same as construct but created solution is improved by neighbourhood search based on ejection chains
- *insert* - insert a new request on the best position in the route
- *insert+* - same as insert but improved in the same way as construct+
- *adaptive descent* - based on adaptive memory where the best known solutions so far, are stored and then try the best value is found

[6] http://www.gps.gov

- *tabu search* - starts from a point from the adaptive memory (never twice from the same point) and stops at the first local optimum

Best results are achieved by a tabu search, but the tabu search is approximately 10x slower than simple insert+ being able to get interesting results as well (about 7% behind tabu search). Multiprocessor version was introduced and time savings are not linear at all. For example decrease only about 17% of time on 16 processors. Testing instances were up to 12 vehicles and 30 requests per hour.

2.2.3 Stochastic forecasting

Sometimes the delivery is possible in more than one day (we have a longer time horizon) and because of that we have a slightly different problem - a multi period VRP. We should decide which orders will be served which day and to forecast demand in future to build optimal or at least balanced routes for each day. This problem is presented in [32], solved by heuristics neighbourhood search and tested on real data in Sweden. Another prediction model could work with phantom orders (a virtual orders generated from historical data) and plan with them. When new real order appears, it can be included into existing route, because vehicle is close to its location [27]. Data mining is done through transforming geographic area to a grid and time into time slices. In such a state space, we can simply mark areas with high probability of new order revealing. In other words an ideal situation is to foresee demand in next time and used it for planning [3].

Efficiency of stochastic forecasting is demonstrated by authors of [9]. They have compared a purely reactive method (re-optimization is done only on so far observed data) and stochastic forecasting (based on examination both historical and current data). The method using forecasting outperformed the second one by up to 69%. Vehicle utilization is even decreased. On the other hand, it takes more than 100x longer to obtain solution and the reaction time is changed from seconds to minutes. Nevertheless, in case of high density of real requests the benefit of foreseeing decreases and both approaches are approximately equal [7].

2.2.4 Auctions

All discussed solutions so far are not fast enough for use in real world - typical size of courier service company fleet in a middle sized city (e.g. Prague with 1,5 millions inhabitants) is 100 vehicles and several thousands of customer requests. Supposing linear complexity it will take minutes for assigning a new customer request. A way out of this trap is parallelism and especially electronic auctions [3]. The most often used is Vicrey auction [31], where the lowest bid wins and pays the second price. The bidders are revealing their real prices [7][26]. This is a very promising concept for solving D-VRP, so let us discussed it in more details.

Suppose a competitive environment - more couriers companies try to win a contract [16] or couriers compete among each other to be allowed to satisfy customer request [7]. When a new request appears, the new auction is taking place and the solution is obtained by updating its route. We use the following auction terminology:

- **combinatorial auction** - subject of each auction is a set of items and bidders are bidding for a subset of offering itemset. This auction type is very close to real-life situations, but unfortunately determination of winner is NP problem [29][26].

- **Vicrey's auction** - as we already mentioned above, the winner pays price of second lowest bid and all the bidder reveals their true prices. Nevertheless, there is one even more important property for intelligent logistics system - required amounth of communication. The amount of communication is very low, in fact only two messages from the auctioneer to the bidders and one message in opposite direction per auction. [17]

- **sequential auction** - suppose the likelihood of two auctions taking place at the same time to be zero [17] and the bidders have sufficient computational power to create a bid before another auction appears [7]. This assumption is a great simplification of implementation and even decreasing the problem complexity (recall NP complexity of combinatorial auction), but we have to face other problems. What if several requests really appear in the same moment then we have to somehow order them. Two different orderings can assign a request to different courier and affects the solution's quality.

- **decommitment** is a technique helping to escape from local optimum, where system because of auction ordering get stucked [30]. Courier may for certain penalty refuse already assigned task and therefore obtain a better (more effective) solution. These decommitted tasks are then re-auctioned to be assigned to a new courier [16].

Although the complexity of combinatorial auction is height, there are successful deployments of this approach. For example, a courier company in 1995 reported a savings of 13% by implementing a multi-round combinatorial auction - including about 800 requests. But expansion of combinatorial auction did not happen, maybe because there is no guarantee of equilibrium. [29]

Vicrey's auction is the most often used auction type in logistics together with sequential auction assumption [16][30]. There are numerous variations trying to deal with local optimum. For example [16] introduced an opportunity cost included into a bid construction. The authors consider the opportunity as follows: will this job lead to the area with high possibility of other jobs? Even a combination of oportunity cost and decommitment strategy is discussed and the result shows that the usage of both techniques gives the best results. Authors compared the results with ILP[7] based central solver and it was up to 100x times slower (and in real word scenarios unusable) then auction approach, on the other hand, their solution achieved "only" 53% of ILP solver quality.

Decommitment is a very powerful technique and it is theoretically proven that it improves a pareto optimal boarder of the solution space [30]. We can measure the efficiency of the decommitment on the total real profit (profit minus costs) of potential company. For illustration there is a scheme of transportation system using decommitment on a Figure 2.2. One can protest that the increase of profit must mean decrease of a service quality but this is not necessary true. Results show that only 0,2% of decommited requests were delivered late (under specific conditions and scenarios presented in [30]) and a concept of hidden decommitment can be used - decommitted requests are re-auctioned in parallel with the main auction and conditions of winner determination are more strict.

The most suitable situations for decommitment are:

[7] Integer linear programming

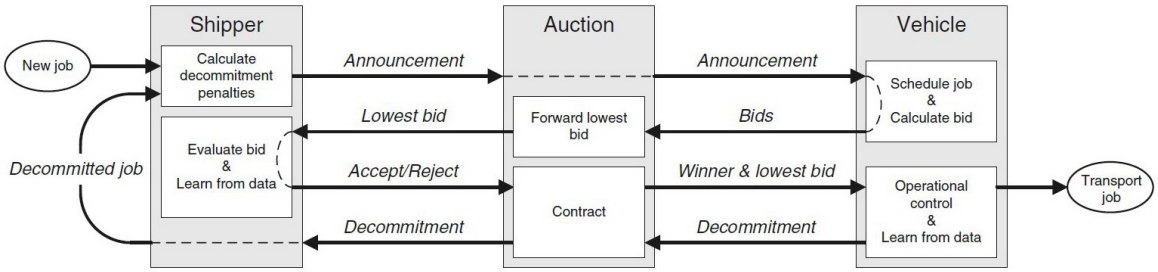


Figure 2.2: Example of inteligent logistics system scheme using decommitment [16]

- vehicle is close to its capacity limit
- number of vehicles is large
- environment is highly dynamic

where the first case is best for route optimization. Others ensure enough possibilities for assignment of decommitted request to other vehicle. [30][16]

2.2.5 Multiagent systems

We have already presented that dynamic VRP is in its nature a decentralized system (each vehicle should be autonomous, collaborative and reactive [2]) and that is why it is almost ideal scenario for using cooperative multiagent system (MAS). Instead of usual centralized planning [2] we are now talking about a decentralized one [25] and one of the biggest benefits is obvious - parallelism [13][14][2]. But parallelization is not the only one benefit. There are many others:

- ability to overcome uncertainty in dynamic environment - accept new customer requests, alter current routes, etc.
- system is flexible and able to react on unexpected events [2][17] - vehicle break down, traffic jam, etc.
- each vehicle can have a different preferences and constraints - working hours, minimal travel speed, different traffic rules, etc.
- system can deal with partial or noisy data [4] - e.g. agent can predict its location in case GPS signal is lost
- system is stable [17] - new requests do not destroy completely current schedule and only small alterations are made - imagine a situation when single one customer request initiate reassignment of the whole fleet and the amount of necessary communication

Until 2005 there was only one fully deployed MAS solution which served only as decision support system. Most projects are in the phase of conceptual proposal (30%) or in a phase of

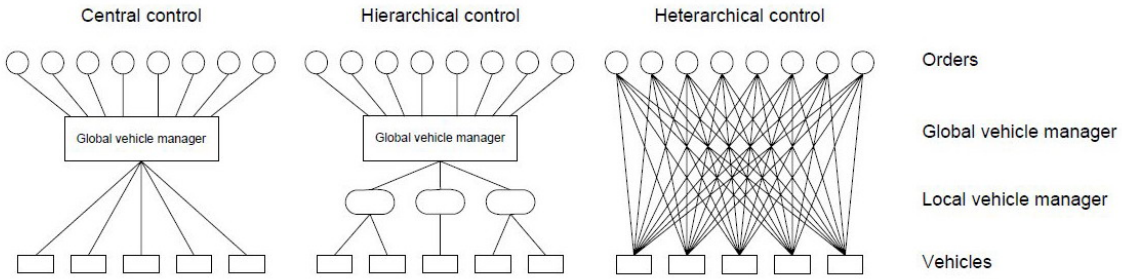


Figure 2.3: Hierarchical structures of MAS in logistics [17]

simulation with artificial or limited data [4]. It seems that operation research community does not focus on MAS but rather on, for decades studied, centralized approaches [11]. Maybe because of lack (or complexity) of comparison with state of the art centralized algorithms. We briefly discuss problem of the comparison in the last part of the chapter.

One of the first applications of MAS in D-VRP comes from 1999 where Solomon's insertion heuristic [25] was used. The performance was not of the best knows solvers of that time but it was close - only 6% further. The possibility of parallelism is introduced, exactly the possibility of optimization more route simultaneously. Solomon's insertion heuristics has quadratic complexity, so a speed up could be really significant. Nevertheless, main contribution is the concept of dynamic world, because all commercial solutions of that time were based on a static world assumption.[13]

A combination of agent based system and local search can be found in [14], where MAS is used for local improvements. Note that this study makes a static world assumption and it determines the minimal number of vehicles count able to satisfy all customers. In real world scenarios we have a fleet of fixed size and we want to maximize service level, minimize cost, etc. Presented algorithm just replaced part of centralized approach with the multi-agent. The real power of the multi-agent system remains unutilized.

Another option how to use MAS for solving D-VRP is a competition (an auction). Vehicles can be modelled as differently behaving agents (different goals, constrains, etc.). Each vehicle constructs an uniform bid and the bid a key property to simply unify a diverse fleet. This approach is used in [11] where the main focus is on choosing the right heuristics for local route optimization and on comparison with current state of the art centralized solvers. Examined were the following heuristics:

- travel time savings (TTS)
- slacknes savings (SLS)

where SLS outperformed TTS. The SLS is more difficult to compute and we pay for the better solution with time.

In the beginning, the authors achieved an average error 2.4% in comparison with the best known result (Solomon's and Homberger's benchmarks were used). Nevertheless the multi-agent approach outperformed state of the art algorithms in required runtime. The results

are even improved in the following study [12]. They achieved the best known results in 90% of instances with relative error 0.3%, even though the negotiation algorithm is relatively simple. The main contribution is in the introduction of possibility of backtracking which can be viewed as a type of decommitment [30]. The authors demonstrate that a multi-agent system is able to solve VRP as well as centralized solutions.

A multi-agent system can be organized into various structures, e.g. each customer request is modelled as separate agent or fleet of agents, etc. Several structures are depicted on Figure 2.3.

Experiments show [17] that structures are without the hierarchical controls better and, in fact, easier. On the other hand, there has to be some mechanism ensuring to obtain a solution as good as possible. The following solutions are feasible [30]:

- *trade among agents* - an empty agent broadcasts if there is a free task (not starting service yet) near its location. Trading makes the task distribution more uniform and average vehicle utilization increases.
- *postponing of finished auctions* - if the latest pickup time of customer's request is not in very close future, we could repeat the auction later. In other words, we do not fix the vehicles routes till it is really necessary. Therefore the overall auction quantity is higher and performance of the whole system decreases. Concept of postponing is similar as decommitment.

2.3 Difficulties with comparison of D-VRP

The basic problem is that there are no standardized benchmarks for dynamic VRP [21]. Nevertheless, many authors use well known Solomon's [13][14] and Hamberger's [12][11] benchmarks for static VRP. Here we have to ask, how worthy these comparisons are, because these benchmarks are testing a relaxation of dynamic vehicle routing problem. Will the result be similarly good in the dynamic environment? This is an open question.

In case of MAS the situation is even more complicated, because each agent behaves differently and each system can optimize a different objective function (maximizing company profit, service level or minimize utilization of vehicles, etc.). That is why the comparison across studies is very rare [4]. It seems that possibility of MAS adaptation to almost every situation is also its disadvantage, because one can hardly say, how the system will behave in real world. A multi-agent simulation are mostly used to revealing of system behaviour [4].

Chapter 3

Problem formalization

This chapter presents a definition and formalization of the problem solved by this thesis. From the classical point of view it is a quite general case of vehicle routing problem. Let us take a look at the features which are taken into account: different pickup and delivery locations, so the vehicle need not travel through a depot to deliver a consignment, time windows at both locations (pickup and delivery) and vehicle capacity together with consignment sizes. The last feature, and the most advanced one is dynamism. The dynamism is employed in the following sense: requests on transportation are appearing through the time and properties of the transportation are time dependent as well.

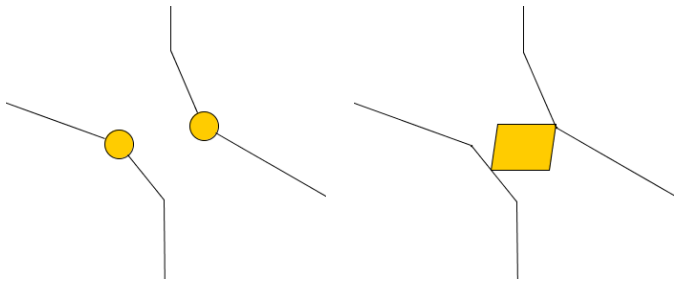
First of all we have to define the environment which will be represented as a transportation graph. Then we will describe an instance of the problem - a courier fleet with jobs. Also a definition of a solution is presented in both minimalistic and classical form. It is not sufficient to provide only some solution of a problem instance without saying how good the solution is, therefore the last section is dedicated to evaluation criteria and a definition of a criteria function of proposed optimization task.

3.1 Transportation graph

Let $G = (V, E)$ be the transportation graph, where V is a set of all vertices and E is a set of all edges. Than let $m_t : E \times T \to X, X = \{f : \mathbb{R}^+ \to \mathbb{R}\}$ be a mapping of edges in time to a distribution function of expected travel time. The distribution function is defined on positive real numbers, because of zero or even negative traveling time makes no sense. An example of suitable distribution function for modelling traveling time is a beta distribution or a beta distribution of the second kind. Let $m_d : E \to \mathbb{R}^+$ be a mapping of edges to the length of the shortest path from the initial vertex of the edge to the destination one.

A **vertex** v is an exact location in the city and it is situated at every crossroad or road deviation. This definition is suitable for representation of places on streets, but disallows to represent whole building. One building has usually multiple entrances from mutually disconnected streets. Both variations are illustrated on Figure 5.4.

An **edge** $e \in E, e = \{u, v, S\}$ represents the shortest path between vertices u and v. Each edge has an associated set S containing all vehicle types being able to travel along it. With that we can represent pedestrian zones, pathways for bikes or motorways for fast vehicles



(a) Vertices as places on streets (b) Vertices as buildings

Figure 3.1: Difference between vertices definition

only. Thank to the mapping m_t we can define a function $\chi(e, v, t_i, \gamma) = \{{}^{i}s_e^{\gamma}, {}^{i}l_e^{\gamma}, {}^{i}w_e^{\gamma}, \psi\}$ to obtain information about the expected traveling time, where $e \in E$ is an edge of the graph G, $v \in S$ is a vehicle type, $t_i \in T$ is the time i, γ holds for a confidence of desired travel time interval[8] and the output values are the following: ${}^{i}s_e^{\gamma}$ is the shortest traveling time across the edge e with beginning in the time i with desired confidence γ, ${}^{i}l_e^{\gamma}$ is the latest traveling time, ${}^{i}w_e^{\gamma}$ is the most probable traveling time and ψ represents the distribution function of expected travel time across the edge e in the time i.

3.2 Couriers fleet

Let $M = \{m_1, m_2, \ldots, m_n\}$ be a set of n couriers, where $m_1, m_2, \ldots, m_n$ are particular couriers. The courier i is a 7-tuple $m_i = \{\delta_i, \varphi_i, \tau_i^p, \tau_i^d, \tau_i^e, \hat{c}_i, v_i\}$ where

- $\delta_i : V \to \mathbb{R}^+$ is a function defining a personal preferences of courier m_i to any location, where V is a set of all vertices of the transportation graph G. Values from interval $(0, 1)$ are the positive preference, interval $(1, \infty)$ is the negative preference and value 1 stands for a neutral element.
- $\varphi_i : E \to \langle 1, \infty)$ is a penalty function of a movement across a particular edge $e \in E$ from the transportation graph G. Value equal to 1 means no penalty. The penalty function is suitable to model for example terrain, damaged routes, narrow streets, etc.
- $\tau_i^p \in \mathbb{R}^+$ is a constant denoting required time to pick up a consignment by the courier m_i.
- $\tau_i^d \in \mathbb{R}^+$ is a constant denoting required time to deliver a consignment by the courier m_i.
- $\tau_i^e \in \mathbb{R}^+$ is an amount of effort required on one time unit of service. It usually depends on a vehicle type of courier. Imagine a difference in fuel consumption of a huge van and a motorbike.

[8] The interval of travel time is defined by $\langle {}^{i}s_e^{\gamma}, {}^{i}l_e^{\gamma} \rangle$ and the confidence expresses a probability that a random value X will be from this interval.

- $\hat{c}_i \in \mathbb{R}^+$ is the maximal capacity that courier m_i is able to carry from one place to another.
- $v_i \in S$ is a vehicle type used by courier m_i

3.3 Jobs

Let $J = \{j_1, j_2, \ldots, j_m\}$ be a set o m jobs[9] defining an instance of the dynamic vehicle routing problem. Each job j_i is 8-tuple $j_i = \{u_i, v_i, c_i, p_i^e, p_i^l, d_i^e, d_i^l, t_i^o\}$, where

- $u_i \in V$ is an origin location of the job j_i
- $v_i \in V$ is a destination location of the job j_i
- $c_i \in \mathbb{R}^+$ is transported capacity (consignment size) required by the job j_i
- $p_i^e \in \mathbb{N}$ is a time of the earliest possible pickup. Domain of the variable is $\mathbb{N}$, because we assume discrete time.
- $p_i^l \in \mathbb{N}$ is a time of the latest possible pickup.
- $d_i^e \in \mathbb{N}$ is a time of the earliest possible delivery.
- $d_i^l \in \mathbb{N}$ is a time of the latest possible delivery.
- $t_i^o \in \mathbb{N}$ is an ordering time (a creational time) of the job j_i

3.4 Solution

A solution of dynamic VRP is to assign incoming jobs (customers requests) to the available resources (couriers). Let assume, that J_t^i is a set of jobs assigned to the courier m_i in time t. The solution consists of couriers' plans. Let $X_t = \{x_t^1, x_t^2, \ldots, x_t^n\}$ be a set of n couriers' plans in time t, where indexes $1, 2, \ldots, n$ correspond to indexes in the set of couriers M.The plan for the courier m_i in the time t is the following sequence

$$x_t^i = (y_k)_{k=1}^l | l = 2 \cdot |J_t^i|, \exists j_m \in J_t^i, y_k = \begin{cases} \omega_p(j_m) \\ \omega_d(j_m) \end{cases} \tag{3.1}$$

where l is a length of the plan x_t^i, $\omega_p(j_m)$ is an operation pickup of job j_m and $\omega_d(j_m)$ is an operation delivery of job j_m. This minimalistic notation is fully sufficient, because it is possible to create couriers routes from plans, jobs and the transportation graph.

[9] A job is a representation of customer request on transportation by the courier's company.

3.5 Route

A solution notation presented in the Section 3.4 contains all necessary information for construction of final couriers' routes. On the other hand, the most of metrics and properties can be measured or computed only from complete routes. Let us describe a construction of the route from a solution in time t.

Let $R_t = \{r_t^1, r_t^2, \ldots, r_t^n\}$ be a set of routes in time t for couriers $1, 2, \ldots, n$. Than let $X_t = \{x_t^1, x_t^2, \ldots, x_t^n\}$ be a set of plans in the same time t. Let define a mapping from a set V (all vertices of the transportation graph G) into a set of couriers actions:

$$\forall m_i \in M, a_t^i : V \to \{\{\omega_p(j_m), \omega_d(j_m), \varnothing\} | j_m \in J_t^i, \omega_p(j_m) \in x_t^i, \omega_d(j_m) \in x_t^i\} \tag{3.2}$$

Now we have to construct a sequence of vertices representing couriers stops for pickup or delivery of consignments. But the first element (the zero element) of the sequence must be a location of courier m_i in time t. Let denote this location by $u_t^i \in V$. So the sequence definition is the following:

$$V_t^i = (w_j)_{j=0}^k | k = 2 \cdot |J_t^i|, w_j = \begin{cases} u_t^i & j = 0 \\ u_l & x_{t,j}^i = \omega_p(j_l) \\ v_l & x_{t,j}^i = \omega_d(j_l) \end{cases} \tag{3.3}$$

where u_l is an origin location of job j_l being placed at j-th position in the plan x_t^i and v_l is a destination location of the same job.

The final route for courier m_i in time t is a sequence of edges E_t^i from the transportation graph G, where these edges are building the shortest path from a vertex $V_{t,0}^i$ to the last element of sequence V_t^i through all vertices being places in the sequence V_t^i.

3.6 Evaluation criteria

We must define some metric on a route for comparing the routes to each other. The first metric is called an effort and it expresses how much work has to be invested into serving the route by particular courier. The second metric is called a level of timely service and it is expressed by the smallest probability of timely arrival to service vertex across the whole route. Then we put these metrics together and define a metric called route value. In the last section we present the problem as an optimization task.

3.6.1 Effort to serve route

The effort is computed per route r_t^i in time t for courier m_i, therefore we will consider in the following definitions these indexes (both t and i) as fixed constants for better readability of the text. Let make an assumption that an interesting part of the whole route is from the time t to the future. So the time t can be considered as the current time.

Before we will continue in the definition of the effort, it is good to define so called selectors[10], because the definition will be with selectors more compact and also easier.

- selector of destination vertex of an edge is defined by the Equation 3.4:

$$\beta_e(e^i_{t,j}) = v | e^i_{t,j} = \{u, v, S\}, u, v \in V \tag{3.4}$$

- selector of index of a job assigned to courier m_i in time t and the job is pickup at the vertex v. The selector is defined by the following equation:

$$\beta_p(v, i, t) = \begin{cases} k & \exists j_k \in J^i_t : a^i_t(v) = \omega_p(j_k) \\ \varnothing & otherwise \end{cases} \tag{3.5}$$

  where i is an index of courier and t is time.

- selector of the index of a job assigned to courier m_i in time t and the job is delivered at the vertex v. The selector is defined by the following equation:

$$\beta_d(v, i, t) = \begin{cases} k & \exists j_k \in J^i_t : a^i_t(v) = \omega_d(j_k) \\ \varnothing & otherwise \end{cases} \tag{3.6}$$

  where i is an index of courier and t is time.

The route r^i_t is represented by the sequence of edges $E^i_t = (e^i_{t,j})^k_{j=1}$ (see Section 3.5 for more details) and for each edge $e^i_{t,j}$ we define the following list of functions:

- α_τ is a function computing required time to service at the destination location of some edge e by courier m_i. The function is defined by Equation 3.7.

$$\alpha_\tau(e^i_{t,j}, i, t) = \begin{cases} \tau^p_i & \beta_p(\beta_e(e^i_{t,j}), i, t) \neq \varnothing \\ \tau^d_i & \beta_d(\beta_e(e^i_{t,j}), i, t) \neq \varnothing \\ 0 & otherwise \end{cases} \tag{3.7}$$

- α_c is a recursive function determining couriers load at the given edge $e^i_{t,j}$ (Equation 3.8).

$$\alpha_c(e^i_{t,j}, i, t) = \begin{cases} c^t_i & j = 1 \\ \alpha_c(e^i_{t,j-1}, i, t) + c_k | \{k = \beta_p(\beta_e(e^i_{t,j}), i, t)\} & k \neq \varnothing \\ \alpha_c(e^i_{t,j-1}, i, t) - c_k | \{k = \beta_d(\beta_e(e^i_{t,j}), i, t)\} & k \neq \varnothing \end{cases} \tag{3.8}$$

  where c^t_i is current load of the courier m_i in the time t.

[10] Selector is an function selecting one particular value from a complicated data structure. The selectors are marked by β

- α_w is a function determining expected waiting time at the end of the given edge $e^i_{t,j}$ (Equation 3.9).

$$\alpha_c(e^i_{t,j}, i, t, \gamma) = \begin{cases} max(p^e_k - \alpha_e(e^i_{t,j}, i, t, \gamma), 0) | \{k = \beta_p(\beta_e(e^i_{t,j}), i, t)\} & k \neq \varnothing \\ max(d^e_k - \alpha_e(e^i_{t,j}, i, t, \gamma), 0) | \{k = \beta_d(\beta_e(e^i_{t,j}), i, t)\} & k \neq \varnothing \\ 0 & otherwise \end{cases} \tag{3.9}$$

  where p^e_k is the earliest possible pickup of job the j_k and d^e_k is the earliest possible delivery of job the j_k.

- α_t returns the most probable traveling time across the given edge at the specified time t (Equation 3.10).

$$\alpha_t(e^i_{t,j}, i, t, \gamma) = {}^t w^\gamma_z | z = e^i_{t,j}, \{{}^t s^\gamma_z, {}^t l^\gamma_z, {}^t w^\gamma_z, \psi\} = \chi(z, v_i, t, \gamma) \tag{3.10}$$

- α_e is expected time of arrival to the destination vertex of of given edge $e^i_{t,j}$ (Equation 3.11).

$$\alpha_e(e^i_{t,j}, i, t, \gamma) = \begin{cases} 0 & j = 1 \\ t_d + \alpha_t(e^i_{t,j}, i, t_d, \gamma) | \{t_d = \alpha_d(e^i_{t,j-1}, i, t, \gamma)\} & otherwise \end{cases} \tag{3.11}$$

  where t_d is the departure time from the previous edge.

- α_d is expected departure time from the destination vertex of the given edge $e^i_{t,j}$ (Equation 3.12).

$$\alpha_d(e^i_{t,j}, i, t, \gamma) = \begin{cases} t & j = 1 \\ \alpha_e(e^i_{t,j-1}, i, t, \gamma) + \alpha_w(e^i_{t,j}, i, t, \gamma) + \alpha_\tau(e^i_{t,j}, i, t) & otherwise \end{cases} \tag{3.12}$$

Now we defined all necessary supporting functions and we can present a formula for computing an effort for the given route. The route is defined by a sequence of edges as we already mentioned. The formula 3.13 shows how to compute effort of the edge sequence E^i_t.

$$\epsilon_{E^i_t} = \sum_{j=1}^{|E^i_t|} \tau^e_i \cdot (\alpha_t(e^i_{t,j}, i, t, \gamma) + \alpha_w(e^i_{t,j}, i, t, \gamma) + \alpha_\tau(e^i_{t,j}, i, t)) \cdot \frac{\hat{c}_i \cdot \delta_i(\beta_e(e^i_{t,j})) \cdot \varphi_i(e^i_{t,j})}{\alpha_c(e^i_{t,j}, i, t, \gamma) + \bar{c} \cdot \hat{c}_i} \tag{3.13}$$

where $\bar{c}$ is a constant of capacity weight influencing an impact of courier's capacity on the effort.

3.6.2 Route level of timely service

The level of timely service is computed per route and the courier in the specific time, therefore we fixed indexes i and t in the following computation. The level of timely service is equal to the lowest value of probability that the courier will arrive to a service location

in time. First, we have to define a function computing the latest expected time of arrival to the destination vertex of the given edge (see Equation 3.16).

$$\alpha_l(e^i_{t,j}, i, t, \gamma) = {}^t l^\gamma_z | \{z = e^i_{t,j}, \{{}^t s^\gamma_z, {}^t l^\gamma_z, {}^t w^\gamma_z, \psi\} = \chi(z, v_i, t, \gamma)\} \tag{3.14}$$

$$\alpha_\psi(e^i_{t,j}, i, t, \gamma) = \psi | \{z = e^i_{t,j}, \{{}^t s^\gamma_z, {}^t l^\gamma_z, {}^t w^\gamma_z, \psi\} = \chi(z, v_i, t, \gamma)\} \tag{3.15}$$

$$\alpha_x(e^i_{t,j}, i, t, \gamma) = \begin{cases} 0 & j = 1 \\ t_d + \alpha_l(e^i_{t,j}, i, t_d, \gamma) | \{t_d = \alpha_d(e^i_{t,j-1}, i, t, \gamma)\} & otherwise \end{cases} \tag{3.16}$$

Now we can define a probability, that courier will arrive to a location in time - see Formula 3.17.

$$p_{e^i_{t,j}} = \begin{cases} 1 - \int_{p^l_k}^{q} \psi(x)dx | \{k = \beta_p(\beta_e(e^i_{t,j}), q = \alpha_x(e^i_{t,j}, i, t, \gamma), i, t) \\ \qquad , \psi = \alpha_\psi(e^i_{t,j}, i, \alpha_d(e^i_{t,j-1}, i, t, \gamma), \gamma)\} & q > p^l_k, k \neq \varnothing \\ 1 - \int_{d^l_k}^{q} \psi(x)dx | \{k = \beta_d(\beta_e(e^i_{t,j}), q = \alpha_x(e^i_{t,j}, i, t, \gamma), i, t) \\ \qquad , \psi = \alpha_\psi(e^i_{t,j}, i, \alpha_d(e^i_{t,j-1}, i, t, \gamma), \gamma)\} & q > d^l_k, k \neq \varnothing \\ 1 & otherwise \end{cases} \tag{3.17}$$

where ψ is a distribution function of expected travel time valid in the time of departure from the previous edge. Finally, we can define a Formula 3.18 for computing a level of timely service on given route.

$$L^i_t = \min_{e^i_{t,j} \in E^i_t} (p_{e^i_{t,j}}) \tag{3.18}$$

3.6.3 Value of the route

The value of the effort gives us an information how good the route is, but the system is dynamic and therefore we need some comparison with route's quality from the previous moment. Let us call this information an additional effort saying whether the required effort for serving altered route is higher or lower.

Let asume time t and time t', where t' is a direct predecessor of the time t. Let r^i_t and $r^i_{t'}$ be the routes of the courier m_i in times t and t'. The value of additional effort is defined by the Equation 3.19.

$$\hat{\epsilon}^i_{t',t} = \begin{cases} max(\epsilon_{E^i_t} - \epsilon_{E^i_{t'}}, 1) & r^i_{t'} \neq \varnothing \\ 0.5 \cdot \epsilon_{E^i_t} & otherwise \end{cases} \tag{3.19}$$

Now we can observe a courier route development through the time. In other words, we are able to say how good the courier's route is. And also we are able to say which level of timely service the route has. Let compound an additional effort with level of timely service into single route value $\theta_{r_t^i}$ defined by the following equation:

$$\theta_{r_t^i} = \begin{cases} \hat{\epsilon}^i_{t',t} \cdot (1 + (1 - L^i_t)) \cdot L_w & L^i_t > L_{min} \\ \infty & otherwise \end{cases} \tag{3.20}$$

where L_w is a weight of level of timely service in the final value and L_{min} is a threshold of required level of timely service.

3.6.4 Optimalization criterium

Once we have defined a metric describing a development of a route in time, we can use an additional effort to describe a behaviour of the whole system. An overall effort ϵ represents a metric of the whole system through a time. Let define the overall effort formally:

$$\epsilon = \sum_{t \in T} \sum_{i=1}^{|M|} \hat{\epsilon}^i_{t',t} \tag{3.21}$$

The overall effort is a unit of work, therefore the goal of the system is to preserve it as low as possible. The best solution of dynamic vehicle routing problem is the solution with the lowest value of overall effort. The dynamic VRP can be defined as an optimization task with the criteria function described by Equation 3.20.

$$min(\epsilon) \tag{3.22}$$

Chapter 4

Solution

We got familiar with dynamic vehicle routing problem in the Chapter 2 then we formally defined dynamic VRP variation solved in this study. Now we can focus on a description of a proposed solution. In the very first part of the chapter we discuss the problem of dynamic VRP from different perspectives. Then we presents reasons why we choose the multi-agent system with communication protocol based on a contract net protocol [28].

The communication protocol contains messages, therefore it is necessary to define a contract for messaging between agents. Also data structures holding both agent's knowledge and believes are described in detail in the second section of this chapter.

Once we define the data structures, we can describe how to operate with them by concrete algorithms (content of the third section). During proposing a solution we have found out that the problem of dynamic vehicle routing is dividable into many subproblems. The subproblems are introduced and defined in a section 4.4 and a concrete planning algorithm solving these smaller problems is presented in the last section of the chapter.

4.1 Protocol

Computers cannot usually compute with continuous time[11], therefore we can talk about a state of the system in one particular time (let denote it t) and about the system's state in the following time moment (let denote it t'). Under discrete time assumption the dynamism is hidden in transition from one system state to the another one. The system state is defined by a set of couriers plans (definition in Chapter 3). The transition from one state to another one can be caused only by an assignment of new consignment to a courier. Together with requirement on system stability[12] we can see that difference between two system states is only in alteration of one courier's plan. The question is which plan should be altered and how.

[11] There are some approaches how to deal with continuous time, but computation with the continuous time is very difficult and require usage of different software equipment than we intend to use (see Chapter 5 for more details).

[12] System stability: new requests does not completely destroy the current schedule and only small alterations are made. Imagine a situation when single one customer request initiate reassignment of the whole fleet and the amount of necessary communication.

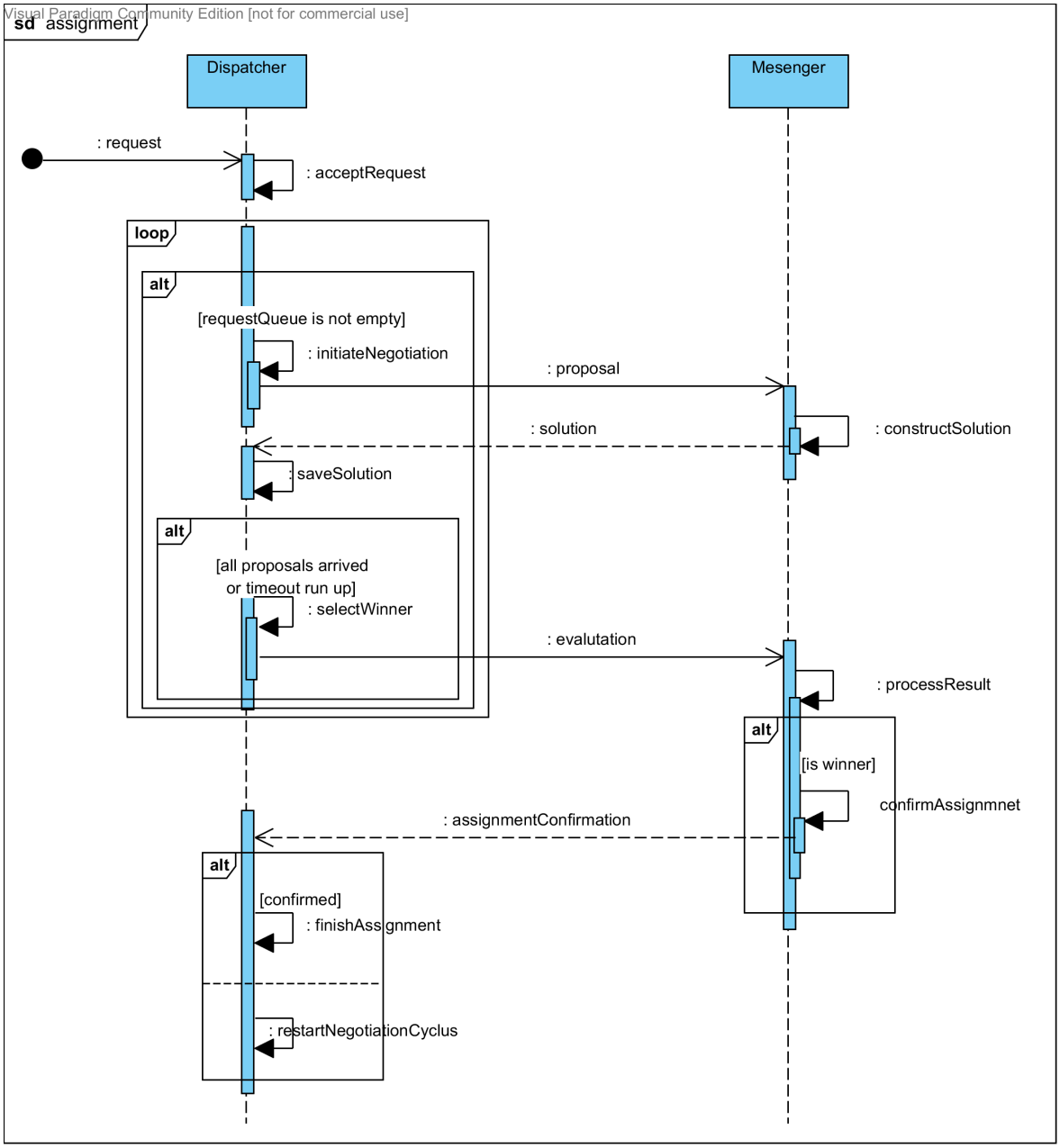


Figure 4.1: Sequential diagram of the communication protocol

The dynamic VRP can be divided into many subproblems:

- an alteration of all current courier's plans
- the decision which alteration is the best

The plan alteration is in fact a routing problem on a courier's level and therefore it is possible to split the whole instance by some manager to workers. So we can talk about that some entities solve smaller but different problems. Because of this decomposition we model the

proposed problem by multi-agent system, where the manager is called a dispatcher agent and the workers will be known as messenger agents.

Before we move forward, there is one important decision to be made. Agents can either collaborate or compete with each other. The competitive model of this problem leads to an auction mechanism and it is widely used in this field of study (see Chapter 2 for various examples). The collaborative approach is usually described as contract net protocol and we are talking about optimization problem in this case. It is hard to say which approach is better, because each has its advantages and disadvantages as well. We have decided to choose the collaborative model, because it is closer to our environment - dispatching inside one company providing courier services.

The contract net protocol in our case works as follows: a dispatcher receives a customer request to be served (it is a problem instance to be solved at particular time t) and he delegates it to messengers. They somehow solve it and return their solution proposals back to the dispatcher. The final step is to choose the best possible solution and to apply it. On the Figure 4.1 we present a communication and negotiation protocol used in our multi-agent system. As you can see, this protocol corresponds with the idea of the contract net protocol presented above.

In the following sections we discuss in detail all parts of communication protocol including transferred messages and necessary algorithms for solution construction. Let us remind that data structure are represented by labels above communication lines in the Diagram 4.1. Algorithms are represented by labels next to loops. This notation should help you with orientation and understanding.

4.2 Data structures

A multi-agent system is dependent on communication between agents (an exact communication protocol is depicted on Diagram 4.1), but these messages must have some fixed contract. Without that it is impossible to understand what agents are trying to say to each other. Definition of this contract is the first part of this section.

Next parts are about a description of data structures used by agents as context. These contexts can be view as agents knowledge and beliefs, because this data are the main source of information for agent's decisions making.

The most important structure is a **request** - a 6-tuple holding all data about customers requirements. It consists of the following variables:

- id - an unique identifier of the request
- origin - variable containing geographic data about origin of customer request. These data are GPS coordinates or a node in the planning graph.
- destination - variable containing data about destination - the same data type as origin.
- pickupTimeWindow - a tuple consisting of two dates determining a pickup window (the earliest and the latest possible pickup).

- deliveryTimeWindow - a tuple consisting of two dates determining a delivery window (the earliest and the latest possible delivery).
- size - float denoting the amount of required transportation capacity

Message announcing a new negotiation carries a **proposal**. The proposal is a tuple holding dispatcher identifier and the request offered in this auction. The messenger sends a **solution** as the reaction to the proposal. The solution is a 4-tuple composed of:

- messengerId - an unique identifier of messenger
- additionalEffort - float value of required additional effort of the messenger to be able to fulfil the proposed request
- L - level of timely service, e.q. the minimal probability that a time window specified by customers will be satisfied
- requestId - an unique identifier of the request competed in this auction

When the winner of the negotiation cycle is selected, a message containing **evaluation** informs messenger about the auction result. It is a 4-tuple covered messengerId, requestId, secondValue and the result itself (binary value equals to true for a winner of the auction and false otherwise). The winner has to reply to the dispatcher if it accepts an assignment and sends a message with **assignmentConfirmation** - a triplet composed of:

- requestId - an unique identifier of the customer request
- value - float value of effort for service
- accept - binary value containing true in case of assignment acceptance and false otherwise

4.2.1 Context properties of dispatcher

Each dispatcher must have some information about the world and the knowledge are saved in the following variables and data structures. The main structure is a **RequestQueue** where all customers' requests are stored and the requests are sorted according to the latest time of possible pickup of each request. **Messengers** is an array of all available messengers being able to serve some consignment. Technically speaking it is an array of tuples. Each tuple consists of the following data:

- messengerId - an unique identifier of messenger
- capacity - the greatest possible package size the messenger is able to carry/deliver.

RequestsInProcess is a data structure holding all customers' requests removed from RequestQueue and not finally assigned to any messenger. It is an array of requests (6-tuple structure described above). Each received solution must be stored for further evaluation and this storage is called **RecievedSolutions**. It is a mapping of request identifier (string) to an array of solutions obtained during negotiation.

Total effort represents an overall effort to serve all competed customers requests.

4.2.2 Context variables of a messenger

Similarly to the dispatcher, the messenger has its knowledge base called context. Each messenger has its basic characterictics, a **messengerId** (an unique identifier) and **availableCapacity** (float value representing the maximal ability of transportation). Apart from characteristics there are a state variables as well. **CurrentLocation** is a geopoint variable denoting messenger's position, timestamp **lastAcceptation** of the last valid assignment of consignment to this messenger and value of **effort**. The effort is a decimal number and it represents messenger's pseudo salary for courier services.

The knowledge base is represented by **solutionsHistory**. It is an array of triplets composed of:

- solution - data structure holding all information about messenger's offer in the negotiation cycle. Complete description can be found above.
- currentTime - timestamp
- route - computed optimal path being realized in case of victory. The structure is defined in section 3.5.

4.3 Algorithms

Once we have the definition of all necessary data structures than it is possible to describe all mechanisms working with these structures. All algorithms presented in this section has labels equal to the labels in the Diagram 4.1, therefore one can easily link the diagram with concrete pseudocodes. Let us describe the negotiation protocol.

First of all a dispatcher has to accept a request. This request is coming from the environment itself. The request is depicted on picture 4.1 as found message. The dispatcher has to save this request (the exact structure of request is described in the Section 4.2) into a request queue (Algorithm 1). Customer requests are drawn according to their priority of a priority queue.

A dispatcher process accepted requests (independently on other activities) in an infinite loop (see loop section in Diagram 4.1). If the request queue is not empty, a new negotiation cycle is initiated. The most urgent request is removed from the queue and the request is placed among other requests being processed. Suitable messengers (only those messengers

Algorithm 1: acceptRequest

```
1 acceptRequest(request, requestQueue)
  // add request into the request queue
2 requestQueue.push(request)
```

Algorithm 2: initiateNegotiation

```
1 initiateNegotiation(requestsQueue, availableMessengers, requestsInProcess)
  // remove and get a head of queue
2 request = requestQueue.remove()
  // mark request in process
3 requestsInProcess.add(request)
  // select messenger with sufficient capacity
4 suitableMessengers = filter(request.size, availableMessengers)
  // prepare proposal
5 proposal = {dispatcherId, request}
  // send proposal to each selected messenger
6 foreach messenger in suitableMessengers do
7 |   callForProposal(proposal,messenger)
8 end
```

Algorithm 3: saveSolution

```
1 saveSolution(request, solution, receivedSolutions)
  // get array of received solutions in this negotiation cyclus
2 solutions = receivedSolutions.get(request.id)
  // register new solution
3 solutions.add(solution)
  // save updated array
4 receivedSolutions.put(request.id, solutions)
```

with capacity at least as big as request requires) are then selected. Sending proposals to selected messengers is the last part of the negotiation cycle initialization - Algorithm 2.

When a messenger receives a proposal, it is able to construct a solution and make an offer to the dispatcher. Finding the solution is the main and the most complex part of the whole multi-agent system, it is described in its own Section 4.5. The created solution is sent back to the dispatcher where the solution must be saved for future evaluation. Saving itself is illustrated by Algorithm 3.

Once all solutions arrive or the time limit expires, it is necessary to determine a winner. The evaluation process is described in Section 3.6.3. Since in our model the value of two or more solutions can be equal, the next evaluation criteria is the level of timely service. Winner determination is illustrated by Algorithm 5.

When the dispatcher announces a winner, the negotiation cycle is not finished yet. The winner has to confirm or denied its assignment and the rejection can result even in restart of the whole negotiation cycle. Procession of the result consists of two steps: messenger's

Algorithm 4: processResult

```
1  processResult(evaluation, context);
   // check if this messenger is a winner or not
2  if evaluation.result is true then
       // yes, messenger won
3      lastAcceptedWork = context.lastAcceptation
       // select creational time of the solution from history
4      historyTuple = filter(context.solutionsHistory, evaluation.requestId)
5      solutionCreationTime = historyTuple.currentTime
       // check if the offer is still valid
6      if lastAcceptedWork < solutionCreationTime then
           // confirm asignemnt
           // update current route
7          currentRoute = update(historyTuple.route, context.currentLocation)
8          context.route = currentRoute
           // increase messenger's effort
9          context.effort += evaluation.value
           // update time of last assignment
10         context.lastAcceptation = currentTime
           // send positive confirmation
11         send({evaluation.requestId, evaluation.value, TRUE})
12     else
           // no, this messenger accepted some assignemt in the meantime
           // send negative confirmation
13         send({evaluation.requestId, null, FALSE})
14     end
15 else
       // no, messenger's offer wasn't the best one
16 end
```

confirmation and dispatcher's reaction.

If the winner finds out that its solution has been the best one, it is necessary to check whether an offer is still valid. The winner compares its latest time of accepted assignment

Algorithm 5: selectWinner

```
1  selectWinner(request, receivedSolutions)
   // get array of all received solutions
2  solutions = receivedSolutions.get(request.id)
   // if there is no solution
3  if solutions is empty then
       // request has to be postponed and evaluate again later
4      postpone(request)
       // remove request from an array
5      requestsInProgress.remove(request)
       // add request back into the request queue
6      requestsQueue.push(request)
       // interrupt selection procedure
7      return
8  end
   // evaluate solutions
9  evaluate(solutions)
   // sort solutions by value in ascending order
10 sort(solutions)
   // if there are more solutions with the lowest value
11 if value of first and second element is equal then
       // select solutions with the lowest value
12     lowestSolutions = filter(solutions)
       // sort these solutions according to a change of timely service
13     sort(lowestSolutions)
       // get first element - the winner
14     winner = lowestSolutions.first
       // save the lowest value as the second one
15     secondValue = winner.value
16 else
       // get the first element - the winner
17     winner = solutions.first
       // get second value
18     secondValue = solutions.second.value
19 end
   // send result to the winner
20 cyclusComplete(winner.messengerId, request.id, secondValue, true)
   // send result to all auction participants except winner
21 foreach solution in solutions without winner do
22     cyclusComplete({solution.messengerId, request.id, secondValue, false})
23 end
```

with a timestamp of solution's construction. If the winner got other assignment in the meantime, current negotiation cycle is invalid, because the messenger can no longer guarantee what it proposed. In case of the assignment is valid the messenger will reroute and increase its amount of effort. Described behaviour is shown in the Algorithm 4.

In case of successful confirmation the dispatcher removes the request from a set of requests being in progress and increases its value of total effort (Algorithm 6). Otherwise the request is put back into the request queue what is, in fact, restart of the negotiation (Algorithm 7).

Algorithm 6: finishAssignment

```
1 finishAssignment(confirmation, requestsInProcess, totalEffort);
  // find request among requests in progress
2 request = select(requestsInProgress,confirmation.requestId)
  // remove request from an array
3 requestsInProgress.remove(request)
  // increase total cost
4 totalEffort += confirmation.effort
```

Algorithm 7: restartNegotiationCyclus

```
1 restartNegotiationCyclus(confirmation, requestsInProcess, requestQueue);
  // find request among requests in progress
2 request = select(requestsInPrograss,confirmation.requestId)
  // remove request from an array
3 requestsInProgress.remove(request)
  // add request back into the requests queue
4 requestsQueue.push(request)
```

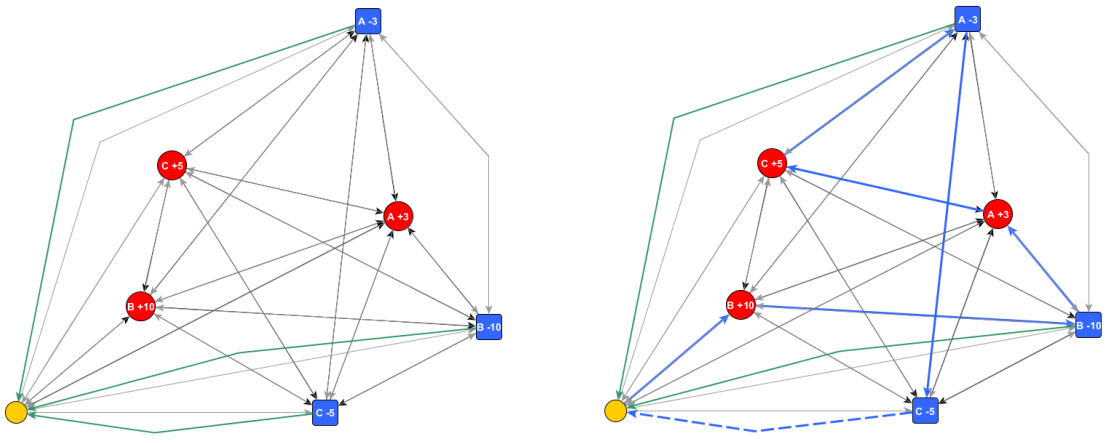
4.4 Solution construction

Solution construction consists basically of two main parts. First, planning the route of messenger and second, computation of the solution value. Other parts of the solution construction are relatively simple, because of processing of input, saving the data for further work and preparing the output structure are just a matter of implementation. So let us focus on the **planning part**.

Each messenger has a set of already assigned customer requests (it can be empty but it is not a problem) denote its size as n. There is one more request proposed by the dispatcher. So we have a set of $n+1$ requests but each customer request has its origin and its destination location. Moreover we have to deal with the current location of the messenger. So there are $(n+1)*2+1$ locations in total. Each location is defined by the time window in which the messenger has to visit it. The delivery location has one additional constrain - messenger must visit the location after it visits an appropriate pickup location.

A capacity is the next characteristics of this planning environment. Each pickup location increases the current load of the messenger and the delivery one decreases it. The overall capacity of the messenger is limited and it can not be exceeded at any time. This planning/routing problem can be classified as paired pickup and delivery traveling salesman problem with time windows and capacity [20]. But the TSP instance is defined by start and end in the same node and this is not our case.

Figure 4.2(a) shows an example with three customer's requests (pickup location is red circle, delivery is blue square and current messenger's location is a yellow circle). Required transportation capacity of each location is represented by a number next to the request identifier. For instance B +10 means an increase of load by 10. A tranformation of our case (messenger doesn't finish in the origin location) is done by edges with zero traveling time (the green ones in the picture) from delivery locations to the current one. This modification ensures ending of each route in some delivery location. All other edges have costs equal to estimated travel time of the shortest path between nodes.



(a) Complete graph for TSP

(b) Example of solution

Figure 4.2: Ilustration of complete planning space

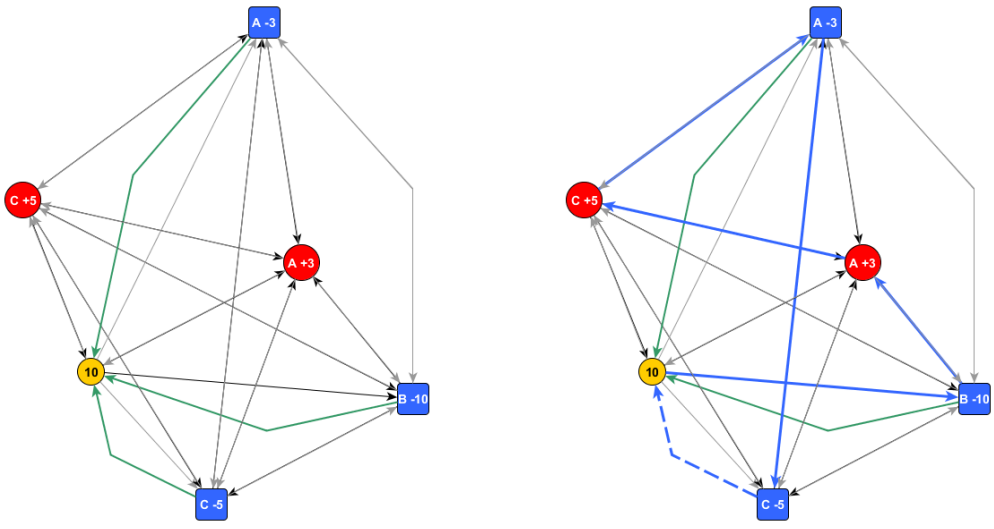
Let say that the messenger's capacity is ten then the blue path on the Figure 4.3 is one of the possible routes. Dashed edge from delivery location C represents a virtual path not traveled by the messenger.

A proposal with the customer request can appear even when messenger has some request already picked up, messenger's load is not empty and not all delivery locations have pickup constraint (a consignment must be picked up before delivery). This situation is illustrated on Figure 4.3(a) where number in the current location denotes the initial load. Figure 4.3(b) shows one possible solution. An algorithm for solving proposed travelling salesman problem is placed into section 4.5, because the algorithm requires deeper discussion together with extensive pseudocode.

From the nature of problem (many constrains have to be satisfied) a solution need not exist. In this case the messenger cannot fulfil the desired requirements and its further participation in the negotiation makes no sense.

After solving the planning part we can focus on the **computation of the value of the solution** according to Equation 3.13. Knowing the solution value we can compute value of additional effort required to serve proposed request (the Equation 3.19). But not only value of additional effort is required, it is necessary to determine the level of timely service as well (the Equation 3.18).

After description of the main parts we can finally move to the last one, the whole construction **procedure overview** (see algorithm 8 for more details). First we have to merge a set of all already assigned locations including their constrains (time windows, ...) with current location. Then we use the historical data to reconstruct the current route. It is done simply by removing so far serviced parts of the route and by adding a current location at the beginning of the route.



(a) Complete graph for TSP

(b) Example of solution

Figure 4.3: Ilustration of partial planning space

Once we merge a set of all locations with new request than the input data for the planning part is complete. Because of the reasons described above it is necessary to check whether the solution is valid. If not, we terminate the procedure without any solution. In the other case it is possible to compute solutions/routes values according to equations 3.13 and 3.19. Determine the level of timely service and finally the solution message is created and ready to be sent. But before the sending, created solution is saved to the messenger' solutions' history together with the current time and the proposed route.

Algorithm 8: Solution construction

```
1  constructSolution(proposal, context);
   // union set of currently assigned request with current location
2  locations = {extractLocations(context.route) ∪ context.currentLocation}
   // reconstruct current route from the previous one
3  currentRoute = reconstructRoute(context.route, context.currentLocation)
   // create a route with new request
4  route = solve({locations ∪ extractLocations(proposal.request)})
   // if solution is a valid one
5  if route is valid solution then
       // compute solution value of current path
6      v_l = computeSolutionValue(currentRoute, context, context.availableCapacity)
       // compute solution value of new path
7      v = computeSolutionValue(route, context, context.availableCapacity)
       // compute price of the proposal
8      if currentRoute is not empty then
9          additionalEffort = max(v - v_l,1)
10     else
11         additionalEffort = v/2
12     end
       // compute level of timely service
13     L = computeLevelTimelyOfService(route)
       // create a solution
14     solution = {context.messengerId, additionalEffort, L, proposal.id}
       // save the solution and the route for future usage
15     context.solutionsHistory.add(solution, route, currentTime})
       // send the solution back to the dispatcher
16     send(solution, proposal.dispatcherId)
17 end
```

4.5 Route creation

Many possible approaches of extending an existing route by a new location have been introduced in Section 2.2. But these approaches do not deal with the variable traveling times through the time space. Let us assume the usage of *insert heuristics*. The insert heuristic is in the original approach ([8]) very fast and simple, but in our dynamic environment we have to recompute all edges of the route to find out if the insertion is valid. Even a very small

deviation done by insertion of new location can cause significant changes in the estimated times of arrival. Imagine a situation of crossing a bridge just in time before the traffic peak begins. These several minutes mean, for example, a multiplication of the original traveling time.

Because of the reasons stated above we have decided to implement *construct heuristics* which involves construction of the entire route in every alteration of courier's plan.

The construction of the route is done by search algorithm based on depth first search. If an edge is invalid (maximum allowed capacity is exceeded or time window constrain is not satisfied) we prune the current search branch. Search itself is an expensive operation and in our case when we have to repeat the calculation of traveling time. The traffic model can return different values for different time moments (value of transit duration is dependent on the previous parts of the route). For this reason we have decided to stop the search when the first valid route is found. So our algorithm does not search for an optimal solution, but it is trying to provide a valid solution in a reasonable time.

In order to further decrease the response time, the next optimization is the introduction of limit of maximal quantity of planning points[13]. This limitation prevents to spend too much computation time and it can also be considered as the balancing element of the whole assignment algorithm. When a messenger has assigned many consignments, this limitation disqualifies him from a further competition till he delivers (or picks up) some consignment. On the other hand, this limitation increases a probability that win messengers with only a few consignment.

From the point of view of the logistics company it is suitable to allow a certain tolerance of late arrival to the customer. It is always better to arrive there few minutes later than never. So we have introduced a parameter called late arrival tolerance which is added to the latest time of pickup and delivery. Therefore the planning algorithm is more benevolent, nevertheless, it does not mean that a messenger will necessary arrive to the destination location late. We are using a probability distribution and an estimation of the most probable traveling time, but this duration can be shorter in reality.

We have presented all necessary properties of the planning algorithm. You can find how the algorithm actually looks like in the following section.

4.5.1 Algorithm

The planning algorithm begins from the initial state. The initial state is represented by the current messenger's location, current load and the maximal capacity being able to be transported by the messenger. From this information we create the first and fixed element of the future route. The last part of the initialization is a check that the maximal number of locations is not exceeded.

When the route is initiated, we can call the expand procedure on it (see Algorithm 9), which is the main part of the search algorithm. First, it is necessary to extract the last element (the last location in the route) and a set of locations (so called open set) being not

[13]We call the limit of maximal quantity of planning points also as planning horizon, plan length or even as planning size in the following text.

used so far. If this set is empty, the route contains all required locations and we have a valid solution. The solution is saves and the current branch is terminated.

In all other cases (there is no complete route yet) we have to try to expand the route. The expansion can be done by some element coming from the open set. So let iterate through the open set and try to insert location into the route. The algorithm stops when the first valid route is found. Therefore there is a check whether a valid route exists at the beginning of the loop. If the complete route exists, the expansion is terminated together with the rest of the algorithm. Otherwise we try to insert next element into the route and check, whether this placement is valid or not (decision about validity is described further in this section). The valid element is put into the route and then it the route is expanded again.

The validity check (see Algorithm 10) of a placement an element into the route consists of the following rules:

Algorithm 9: Route expansion

```
1  expandRoute(route, locations, lateArrivalTolerance)
   // get last element of given route
2  lastElement = route.getLast()
   // get location being not in the route yet
3  openSet = createOpenSet(route, locations)
   // is all locations are placed in the route
4  if openSet is empty then
   |  // than we add current route among the complete ones
5  |  completeRoutes.add(route)
   |  // and terminate current search branch
6  |  return
7  end
   // route is not complete, try to expand it
8  foreach location in openSet do
   |  // condition to stop whole search when the first
   |  // solution is found
9  |  if completeRoutes is not empty then
10 |  |  return
11 |  end
   |  // try to place current location at the end of the route
12 |  element = createElement(lastElement, location, route)
   |  // if this placement is not valid
13 |  if isValid(element, route, lateArrivalTolerance) = false then
14 |  |  continue
15 |  end
   |  // ok, element is valid, place it into the route
16 |  route.add(element)
   |  // and try to further expansion
17 |  expand(route, locations)
18 end
```

- *Precondition*: if the route element is a delivery one, this consignment has to be already picked up.
- *Capacity*: the total load of a messenger cannot exceed its maximum capacity.
- *Arrival*: the estimated time of arrival must be earlier than the latest one allowed by the customer or the estimated time of arrival must be within the late arrival tolerance.

If the rules stated above are satisfied, the element can be placed into the route, otherwise we have reached a bad branch and it is necessary to backtrack the search.

The validity check procedure uses an estimate of the arrival time. This time is computed according to Equation 3.11 and obtaining a parameter value is the main reason, why we have chosen this algorithm and why the planning is so time expensive.

Algorithm 10: Validation of route element

```
1  isValid(element, route, lateArrivalTolerance)
   // check if the route contains required precondition
   // e.g.  consignment delivery has to later than pickup
2  if element precondition is not null and route does not contain it then
3  |  return false
4  end
   // check the maximal capacity of messenger
5  if route.maxCapacity < element.capacity then
6  |  return false
7  end
   // check if the messenger will arrive on time
8  if element.eta > element.latestArrival + lateArrivalTolerance then
9  |  return false
10 end
   // all conditions are satisfies
11 return true
```

Chapter 5

Implementation

An implementation of proposed multi-agent system is based on several frameworks developed in the agent technology center[14] at Faculty of electrical engeneering of Czech Technical University in Prague. A core of the whole implementation is a general project A-Lite[15] and its extension to complex city simulation AgentPolis[16]. A MobilityTestBed[17] is a next additional extension of these frameworks allowing easy evaluation and benchmark of transportation problems. We name our project as LogisticsTestBed, because it is an extenstion of the MobilityTestBed and it focuses on the logistics problems, especially on dynamic vehicle routing problem studied by this work. A visualization of dependencies among project is shown on Figure 5.1 and the projects are written in JAVA. A content of this chapter is a brief touch to used frameworks and description of the interesting parts of the LogisticsTestBed.

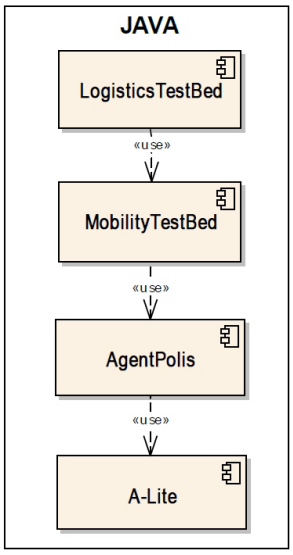


Figure 5.1: Dependency of LogisticsTestBed on other frameworks

[14] Agent technology center - <http://agents.felk.cvut.cz>
[15] A-lite, general simulation platform - <http://jones.felk.cvut.cz/redmine/projects/alite/>
[16] AgentPolis, simulation framework - <http://agentpolis.com/>
[17] MobilityTestBed - <https://github.com/agents4its/mobilitytestbed>

5.1 Overview of used frameworks

The frameworks will be introduced only briefly. Deeper description can be found for example in a bachelor's project of my colleague [19].

5.1.1 A-lite

A goal of a general multi-agent simulation platform A-Lite is to provide highly modular and variable sets of functionality defined by a simple API. It has not defined architectures for various multi-agent simulations, but it provides many reusable elements for building any possible multi-agent system. Aside of already mentioned API, the A-lite provides even a visualization interface. So everyone can observe a system behaviour in the real time.

5.1.2 AgentPolis

The AgentPolis is a platform for multi-agent simulations taking place in the cities [20]. It is based on the A-Lite framework and its very early development began in 2010 in cooperation with IBM[18]. That time it carried a name UrbanSim. It went through a significant development and nowadays it has many applications, for example we can list projects as UrbanCrime [19], FareEvasion[19], MobilityTestBed, SharedTaxi [18] and others.

5.1.3 MobilityTestBed

The MobilityTestBed is a simulation tool for evaluation of dynamic dial a ride problems (DARP). There are build-in several scenarios of DAPR and everyone can easily implement their own algorithm and evaluate it. The MobilityTestBed originates from 2013 where it was developed within a diploma thesis [1]. Than it was revised and incrementally developed till the current state [10].

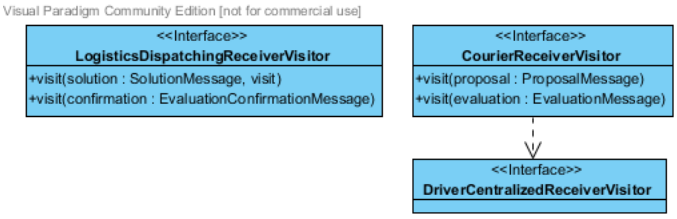
5.2 LogisticsTestBed implementation

Even though we use the MobilityTestBed as an environment for our implementation, we must change some of the core parts. The biggest difference between the MobilityTestBed and our LogisticsTestBed lies in a communication protocol. A multi-agent system solving the dynamic vehicle routing problem has almost completely different interaction among agents than MAS for dynamic DARP. So, the most interesting implementation parts are a model of rush hours and the implementation of communication. The model of rush hours is also discussed as a traffic model.

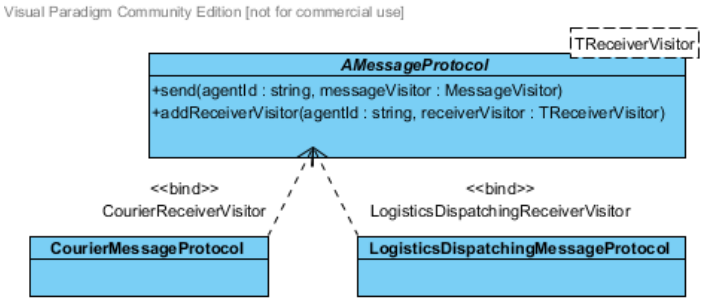
A speed of a algorithm is usually very sensitive on selected data structures. A description of the selected data structures is placed in one of the following sections. Last but not least we optimized LogisticsTestBed on easy and distributed experiments' evaluation. Description of the optimization can be worthy and it may inspire someone in his own work.

[18] IBM - <http://www.ibm.com/us/en/>

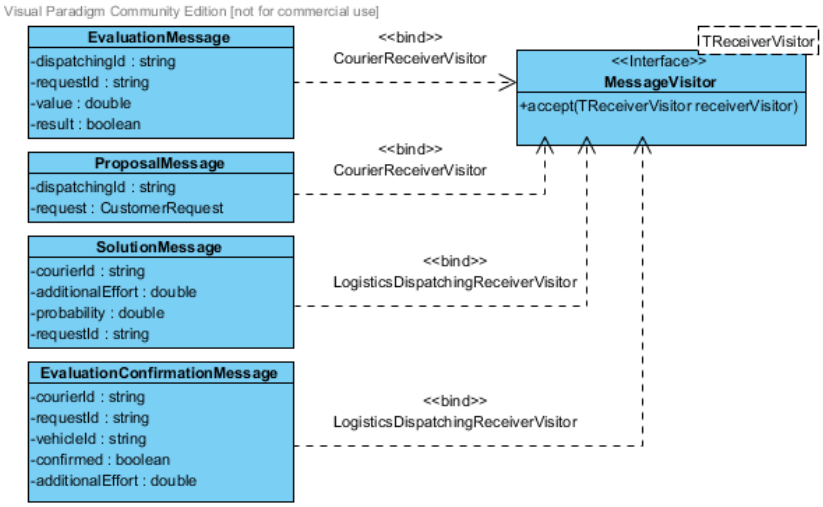
[19] FareEvasion project - <http://jones.felk.cvut.cz/redmine/projects/agentpolis/wiki/Fareevasion>



(a) Interfaces for receiving messages



(b) Protocols for sending messages



(c) Definition of messages

Figure 5.2: Model of basic parts for communication between agents

5.2.1 Communication between agents

We try to describe a communication protocol used in MobilityTestBed in general but with demonstration on our project. First, we have to describe basic elements required to communication:

- Someone has to send a message. The sender is an instance of agent logic (a descendant of class AgentLogic) in the MobilityTestbed.

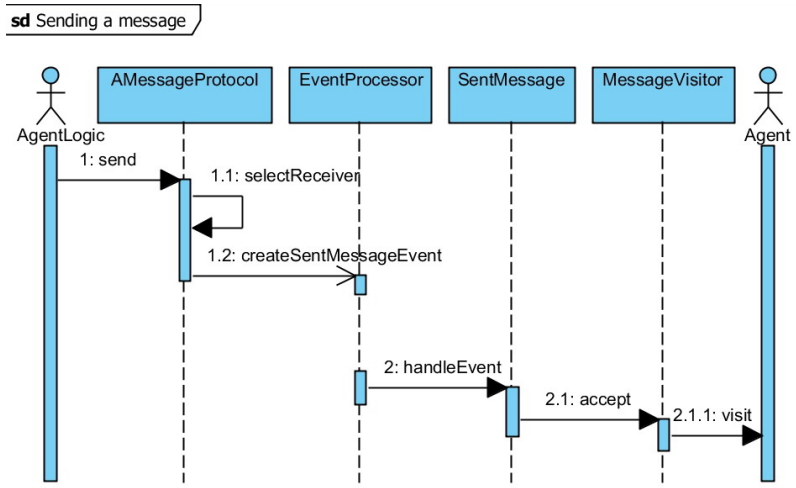


Figure 5.3: Sequential diagram of communication between agents

- Messages must be sent through some channel and this channel is called MessageProtocol in the MobilityTestbed. On the Figure 5.2(b) we see an abstract class AMessageProtocol providing basic methods such as addReceiverVisitor and send. The first method allows to register a recepient of messages. The second one sends the message. We implement two protocols in our LogisticsTestbed: CourierMessageProtocol (a channel for messaging to couriers agents) and LogisticsDispatchingMessageProtocol (for messaging to dispatching agent).

- The agents must be ready for incoming data and the Figure 5.2(c) shows structure of the additional messages in the LogisticsTestbed. Each message implements an interface MessageVisitor for a possibility to be sent through the simulation core.

- Sent messages should be delivered to the recipient defined by an interface. All courier agents implement CourierReceiverVisitor (pay attention on generic specification in Diagrams 5.2(b) and 5.2(c)) and the dispatching agents implement LogisticsDispatchingReceiverVisitor.

Now we introduce all basic elements taking part in communication protocol. Let us describe a communication sequence. The communication sequence is illustrated on Diagram 5.3. At the very beginning the sender must create a message and send it through a MessageProtocol. The MessageProtocol is responsible for selecting an appropriate agent and creation of an event for the simulation core. The communication is done through the event to ensure that a message will arrive on time. When the message should be delivered, the SentMessage object invokes accept method on sent message. It causes the final delivery to the desired agent.

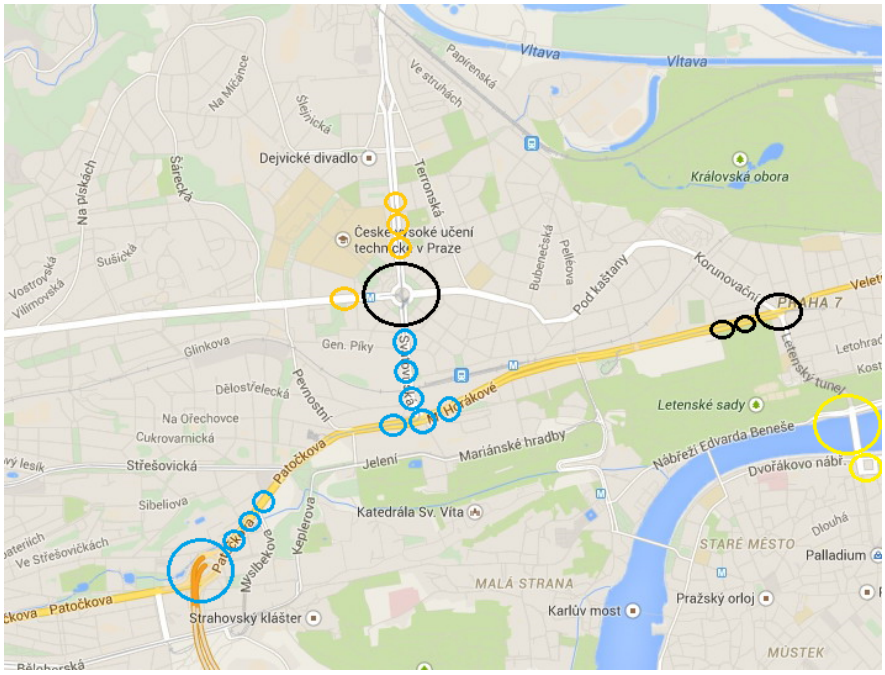
5.2.2 Rush hours model

A traffic is very complex system and everyone who study it, has to build some traffic model. Without the model is a serious problem how to interpret results. Of course, we can

always say that our experiments were done under ideal or so called laboratory conditions. On the other hand, the conclusion of these experiments would not be as valuable as it could be.

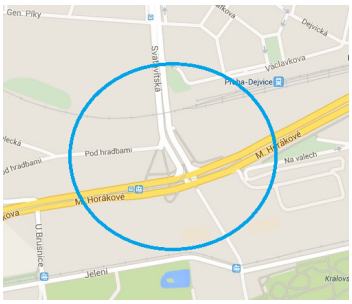
Our traffic model is based on observations and statistical data, that's why its accuracy is directly dependent on the quality of provided information. Let us describe our model. It consists of 3-tuples and for each of them it is possible to define the traffic speed at geographic location (defined by a circle of arbitrary centre and radius) in particular time window. By this definition we can model for example a traffic jam with very high precision. For instance in the middle of Ječná street between 9:00am and 9:10am is usually huge traffic moving only 5 km per hour.

Someone can propose a slowdown defined by a ratio of vehicle's maximal speed, because for example motorbikes can go between cars and move faster than others. On the other hand, streets in the city centre are narrow and this manoeuvre could be dangerous. Next

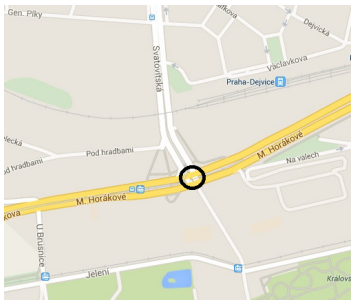


(a) Overview

(b) Beginning of problem



(c) Rush hour



(d) Problem is disappearing

Figure 5.4: Ilustration of rush hours model

alteration of our definition could be usage of probability to determine whether this traffic problem will exist or how big the slowdown should be.

On the figure number 5.4(a) we can find an example of rush hours model. For simplification we use only three time windows and these time windows are distinguished by different colours. For instance blue circle means time window between 8:30am and 9:00am, the yellow one is from 9:00am to 9:15am and the black one from 9:20am to the midday. Our model can be used not only on static description, but we can model a development of traffic problem as well. See figures 5.4(b),5.4(c) and 5.4(d) where the first one shows beginning of the problem, the second one is the real rush hour and finally the last one illustrates a subsident of the traffic problem. A representation in our implementation is as follows: each circle from Figure 5.4(a) is the instance of class RushHourElement and a set of RushHourElement is checked during each estimation of expected travel time. If the element is active, the expected travel time is adjusted according to the RushHourElement definition.

5.2.3 Data structures in planning algorithm

The planning algorithm was proposed in Section 4.5. It is a search algorithm and these algorithms basically holds data in two data structures. The first one is input data and the second one is an open list of possible expansion elements of a current state. For both of these structure we choose implementation of Set in JAVA, especially HashSet. The main reason for this choose is the simplicity of a final code and a possibility to use reliable methods as contains, isEmpty and add. We realize that this structure do not guarantee an order during iterating through it and implementation based on arrays can be significantly faster.

On the other hand, the proposed algorithm is simple and also a very slow. So our recommendation for the future work is to replace it by something more sophisticated instead of speed up the current implementation.

5.2.4 Optimalization for easy evaluation

An easy evaluation of many experiments is a very important property of each testbed. The LogisticsTestbed supports configuration loading from a remote server. It is possible to specify the server URL and then load the configuration from given JSON file. The structure of the JSON file should corresponds to a class ExperimentsParams.

When the experiment ends, the program can make a http request on specified URL. The query parameters has the same names as properties of the class ExperimentsResults. Thank to the loading and the saving to the remote server, we can easily run multiple simulation instances to speed up experiment's evaluation.

The support for executing given system call is the next usable feature. The system call is expected as the second input parameter of the simulation. So, one can for example initiate the next simulation run when the current one finishes.

We used both optimizations and that is why wa can run many evaluations on seven computers.

Chapter 6

Experiments

In the previous chapters we formalized the dynamic vehicle routing problem. We have proposed and implemented a multi-agent simulation based on these knowledge. Now we can run experiments in various scenarios and study a behaviour of the system. The main purpose of the chapter is a description of the scenarios.

First, we will define an experiments' environment and all necessary settings. Second, we evaluate the experiments and finally, we will try to estimate dependencies among parameters and metrics of the multi-agent system.

6.1 Experiment settings

Each experiment consists of several parts. The experiment has to be run somewhere (an environment), the experiment has to run on something (an input data) with particular settings (parameters of the MAS) The experiment provides some output (values of defined metrics) in the end. The section presents definition and setting of all these parts.

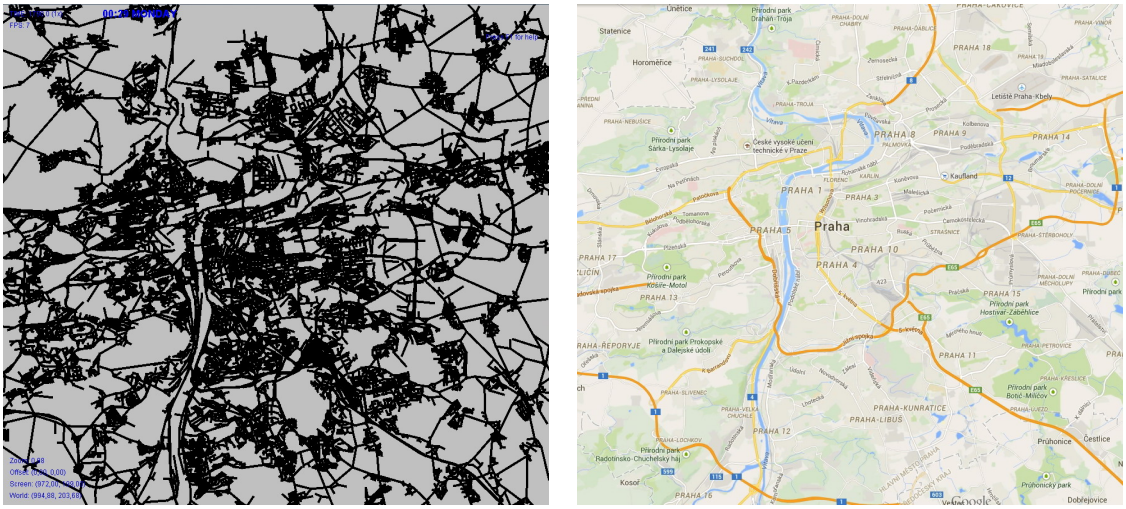
6.1.1 Environment

All experiments are taking place in Prague and the location size is approximately 21 x 17 kilometres [20]. An OpenStreetMaps[21] was used as the map background and we focus primary on the routes for vehicles (cycloroutes, pedestrian and touristic routes are omitted). Even with this selection the final graph after a transformation contains 28 074 nodes. Both area and the number of nodes suggest that the planning space is very big and choose of space of such size is quite ambitious. The environment area is shown on the Figure 6.1(a) in the simulation framework and the Figure 6.8(d) presents environment view in Google Maps.

The model of rush hours was defined in Chapter 4 and we use it also in the experiments. However, because of computation power we have to limit quantity of locations and define only a few traffic problems. On the other hand, we increase a radius of problematic locations and the final model more less corresponds to the main traffic problems in Prague (of course

[20] GPS coordinates of top left most corner are 50.1613961N, 14.2914678E and 49.9930828N, 14.6508797E for the bottom right corner.

[21] OpenStreetMaps: <http://www.openstreetmaps.org>



(a) Overview (b) Beginning of problem

Figure 6.1: Experiment location

the model is not complete and we neither try to build it). The traffic model illustration is depicted on Figure 6.2.

6.1.2 Experiments input data

The input data are often a crucial part of the logistics studies. When you do not have the real data, it is necessary to generate the data artificially but what data should be generated? How to distribute the data density? The generation is a very difficult task and results of these experiments are not as valuable as experiments done on the real data. We luckily have an access to the real data of a courier company in Prague. On the other hand, the obtained data are very big: several thousands of consignments served by nearly 200 couriers per day. The instance of such size exceeds our computational abilities (simulation framework runs only in single thread and simulation will take several days). So, we decided to extract only a subset of the input data according to a following key:

- First, we select a set of couriers such that we preserve a diversity of the fleet. So there are couriers with vans, pickups, ordinary cars, motorbikes and bikes as well, in the final set. According to the vehicle types we set their capacities (bikes with motorbikes have capacity equal to 5, cars have 15, pickup 30 and big vans 60).
- Second, we choose a day period as a couriers' shift and the shift is from 8:00am to 14:00am. Everything should be delivered till the end of the shift.
- Finally we found consignments served by the selected couriers in the given time period. The final consignments selection has 233 items.

There are additional information in the input data. So we can work with real time of pickup and delivery, therefore it is possible to measure a deviation from service level reached

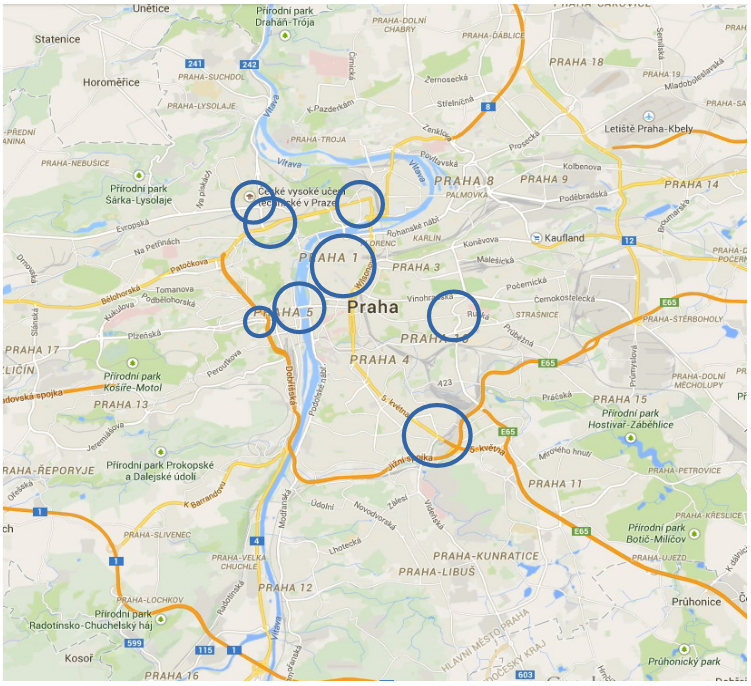


Figure 6.2: Illustration of the rush hours model

in the reality. Because all of given consignments were assigned only to selected couriers and these couriers did not serve any other consignments, we can say that our MAS does the same job as skilled human dispatchers.

We prepared two additional sets of couriers to show if 35 couriers is the best quantity or not. The first set contains 25 couriers and the second one contains 45 couriers. The consignments data remains the same for all experiments.

6.1.3 Scenario parameters

A scale of the input parameters is very wide as we have already indicated in the previous chapters. We must decide according to our intuition and experience and limit the number of possible parameter combinations. Some of them have fixed value all the time and others are varying only in several values. Even though we evaluated almost one thousand different configurations, exactly 974.

In Section 3.1 we talked about a distribution function of the expected travel time between two locations. For experiments we choose a beta distribution [22] to model the expected travel time. The shape of beta distribution corresponds with the time distribution pretty well. Especially when alpha parameter is equal to 2 and beta parameter is equal to 3.

The following table (Table 6.1) gives us an overview of all parameters with their domains and references where they are defined.

[22] Definition of beta distribution: <http://en.wikipedia.org/wiki/Beta_distribution>

Parameter	Definition	Domain
Consignments quantity	6.1.2	{233}
Couriers quantity	6.1.2	{25,35,45}
Alpha value of beta distribution	3.1, 6.1.3	{2}
Beta value of beta distribution	3.1, 6.1.3	{3}
Late arrival tolerance	4.5	{300, 750, 1000} s
Threshold of timely service	3.6.3	{0.5, 0.7, 0.9}
Limit of locations in planning	4.5	{7,10,13}
Time estimation confidence	3.1	{0.4, 0.6, 0.8, 0.9}
Weight of capacity in part. effort formula	3.6.3	{0.4, 0.65, 0.9}
Timely service weight in solution eval. formula	3.6.3	{0.4}
Negotiation interval	4.3	{10} s
Beginning of a shift	6.1.2	{8 am}
End of a shift	6.1.2	{14 am}

Table 6.1: Overview of scenario parameters including their domains

6.1.4 Metrics

For the experiments evaluation we use all available metrics. Complete list including their definitions can be found at the end of this section. Values of the metrics related to the couriers or the consignments are available in an extended form (by the extended form we mean a set of the following values: lower bound, upper bound, average, median and also all data). The metrics in extended form are marked by asterisk (*).

The experiment configurations were run 5 times to eliminate an influence of random noise and to obtain as precise data as possible (with respect to available computation resources). We are convinced that 5 repetitions are sufficient, because there is no apriory randomness - the input data are fixed and nothing is generated randomly. Nevertheless, the implementation in JAVA uses unsorted data structures [23], the proposed algorithm is neither complete nor optimal one and it can result in slightly different plans and assignments in each run.

We define 10 metrics:

- **real time of algorithm** is a duration of running time (in seconds) of all computing parts. By computing parts we mean decision process done by the dispatching agent and the planning solved by courier's agents.
- **real time of simulation** is a value of real time of algorithm plus all additional time required on initialization, communication among agents, interactions with environment, etc.
- **total traveled distance** - total number of kilometres travelled by all messengers including traveling without parcel.

[23] In documentation of HashMap and HashSet (<http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html>) used in the simulation can found the following:

> This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

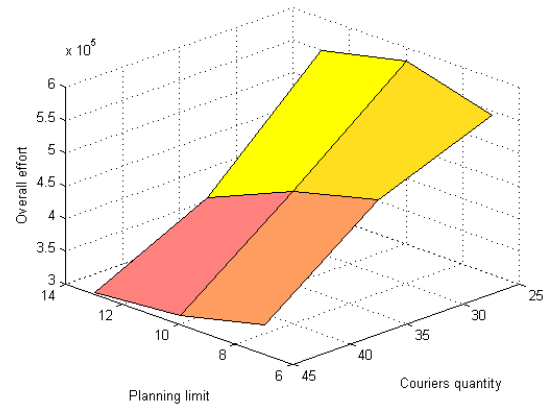
- the existence of the real input allows us to measure a deviation of the real data and the artificial ones. We define **deviation from real delivery*** which is an absolute value of difference between times of deliveries.
- **deviation from real pickup*** is the very same metric as deviation from real delivery, but this value is measured at the pickup location.
- an **overall effort** is the optimization criterion of the entire system. The overall effort is computed as the sum of all partial additional efforts required to deliver the assigned parcels by the messenger.
- From the service level perspective it is very important to say how many consignment will be deliver. In other words how big is the probability that a courier to the customer. This metric is called **ratio of served consignments** and it is computed as a quantity of served consignments (when a messenger transports a consignment from the pickup location to the destination, regardless of arrival time) divided by the total number of consignments.
- **waiting time at pickup location*** is a time period between the latest pickup time allowed be the customer and a real messenger's arrival. If the messenger arrives earlier then the value of the metric is equal to zero.
- **waiting time at delivery location*** - same computation as waiting time at pickup, but at the delivery location
- comparison of **courier's workload*** is a relatively difficult task, because we have a heterogeneous fleet with various vehicle types. So we need to introduce a concept of time slots. One time slot is a capacity of one capacity unit available for one minute. Therefore we are able to unify and compare utilization of messenger's on bikes with messenger's driving vans. Each messenger has its total amount of time slots - it is courier's maximal capacity multiplied by the number of minutes when the messenger is working - and we can measured real usage of the slots. The final value of the metric is a ratio between total amount and used quantity of time slots.

6.2 Results

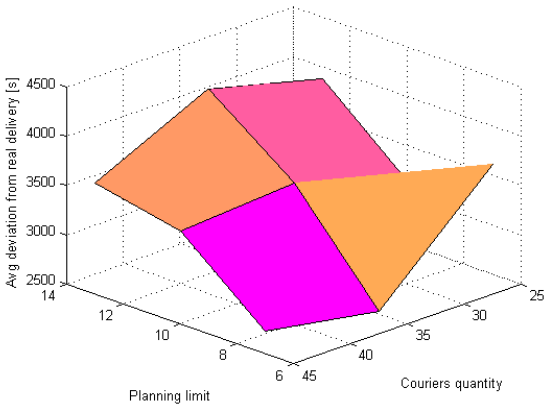
An evaluation of relationship among input parameters and metric's values is one of the most important parts of the study. The evaluation can discover behaviour of the proposed multi-agent simulation and even properties of the whole problem. The goal of this section is to analyse all achieved measurements (the data from 4870 simulation's runs) and try to explain a behaviour of the system.

6.2.1 Metrics dependency on couriers quantity and planning limit

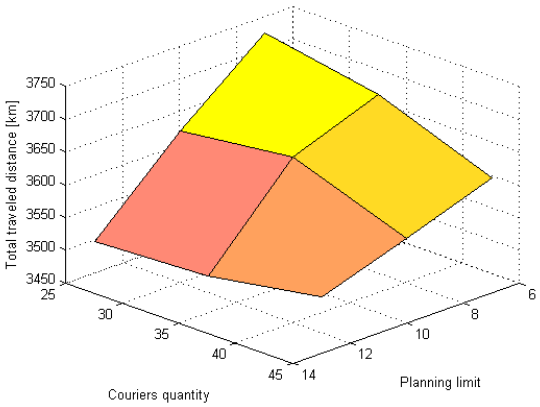
The first experiment focuses on the basic metrics and their dependency on the couriers quantity and the limit of maximal locations being allowed in the planning phase of the algorithm. We choose the couriers quantity and the planning limit parameters, because their influence on the output data is the biggest one. We study primarily metric's dependency on these parameters in other experiments as well. We have to set up a set of parameters for



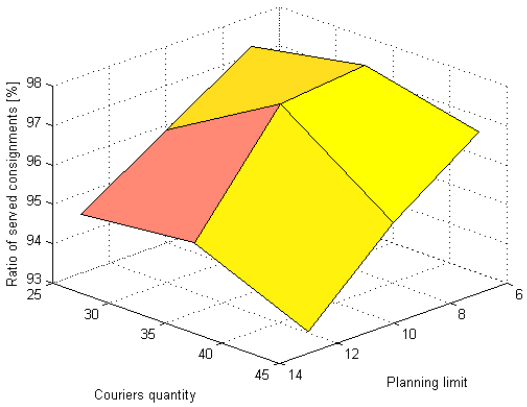
(a) Overall effort



(b) Average deviation from real delivery time



(c) Total travelled distance



(d) Ratio of served consignments

Figure 6.3: Dependency of basic metrics on the couriers quantity and the planning limit

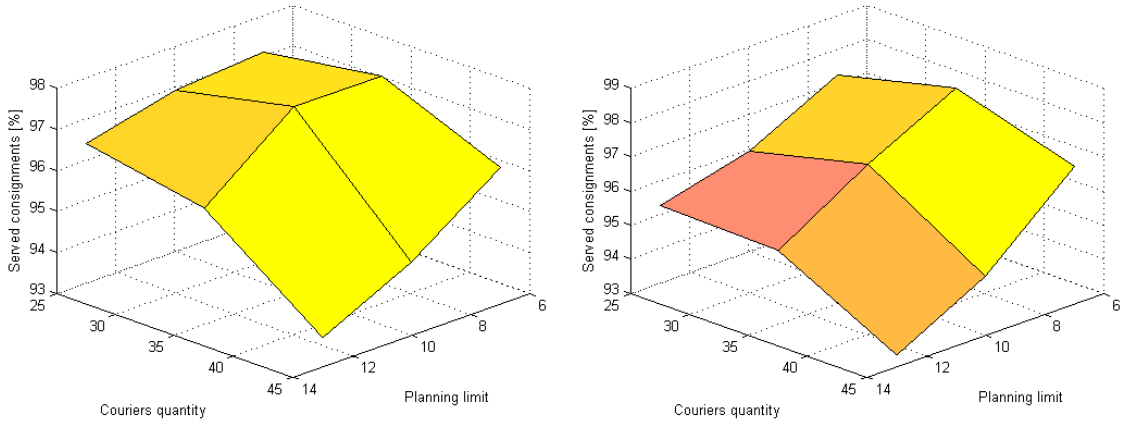
each experiment (in fact we extract a subset of the measured data) to obtain a correct result for an analysis. So we set all parameters to the one particular value except of the courier quantity and the planning limit (see Table 6.2 for complete settings).

List of selected basic metrics is the following:

- overall effort necessary to deliver all consignments
- average deviation from the real delivery time, because the delivery time is the most important property for the courier company.
- total traveled distance
- ratio of served consignments

One can expect by intuition, that the best result should be achieved by using as many couriers as possible together with the biggest planning limit. Let us discover if obtained values presented in Graphs 6.3 match to the assumption.

A trend of the necessary effort to serve all consignments (Graph 6.3(a)) corresponds to our assumption: the best values are obtained for max. quantity of the couriers and the



(a) Threshold of timely service 0.5 (b) Threshold of timely service 0.9

Figure 6.4: Dependency of successful service on the couriers quantity and the plannig limit

highest planning limit. It is caused by long couriers plans containing small detours, because the shortest travelling time is one of the optimization criteria of the system. A value of the necessary effort increases with decreasing couriers' quantity, because couriers have to travel also to the consignments in bigger distance.

An average deviation from the real delivery gains quite high values. We can see its values along one hour in the Graph 6.3(b). This significant difference is caused by using traffic model being far from the reality and creating different plans than human dispatchers. Nevertheless, it is interesting that decreasing planning limit makes the difference smaller. But we cannot say for sure, that this is a positive trend, because metric's value is an absolute value and greater planning limit can cause both earlier delivery and bigger difference. To deliver consignment as soon as possible is of course very desirable property.

A total traveled distance (Graph 6.3(c)) behaves according to our assumption - the lowest and the best value is reached for as many couriers as possible and the biggest planning limit. We suppose that reasons of this behaviour are the same as we have presented for the overall effort.

The last main metric, the success rate of consignment assignment (Graph 6.3(d)), behaves exactly the other way around than our assumption. We can see that greater planning means smaller flexibility of assignment and the new consignment cannot be served by none courier without a violation of the current plan. The plan violation is of course forbidden by the MAS. In view of the fact that the ratio of successful service is crucial, it has dedicated its own Section 6.2.2 for deeper analysis.

6.2.2 Analysis of successful service metric

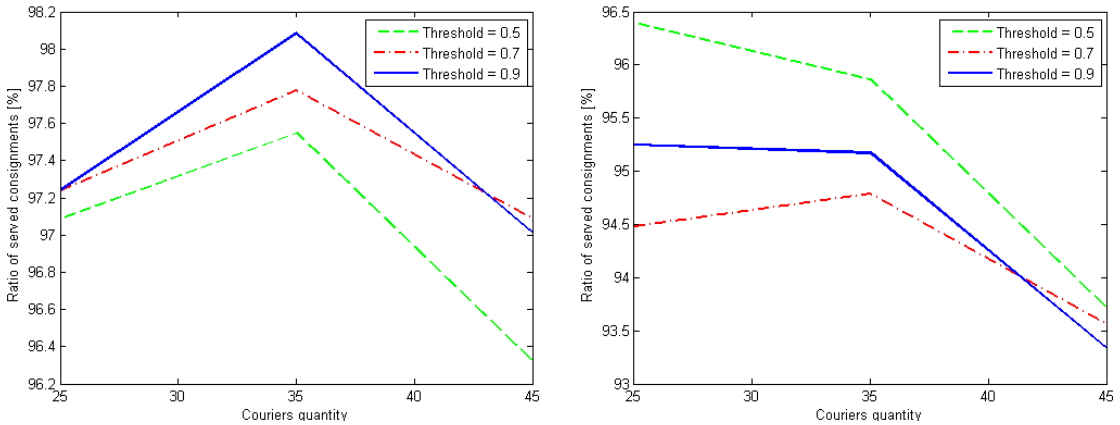
Next experiment is focused on the metric of successful service (e.g. how big is a chance that a consignment will be assigned to a courier). We will evaluate mainly a dependency on the plan size of each courier and the threshold level of timely pickup or delivery. Other parameters have very small influence on this metric and they remain fixed (see Table 6.3 for complete setting).

Parameter	Domain
Consignments quantity	233
Couriers quantity	{25,35,45}
Alpha value of beta distribution	2
Beta value of beta distribution	3
Late arrival tolerance	300 s
Threshold of timely service	0.7
Limit of locations in planning	{7,10,13}
Time estimation confidence	0.4
Weight of capacity in part. effort formula	0.4
Timely service weight in solution eval. formula	0.4
Negotiation interval	10 s
Beginning of a shift	8 am
End of a shift	14 am

Table 6.2: Input parameters for evaluation of dependency between basic metrics and couriers quantity and planning limit

There is clear trend on the Graph 6.4 that long plans have negative impact on metric's values. If we additionally require also high threshold of the timely service then our system reach a state with the lowest values (Graph 6.4(b)). It is caused by a very problematic addition of the new consignment into the current couriers' plans.

An increase of the required quality of the service (the parameter threshold of timely service) can influence negotiation in such a way that system's flexibility increases as well. This is done especially for 35 couriers in Graph 6.5(a). But this statement is valid only for short plans, because for long plans setting the system behaves exactly the other way around (Graph 6.5(b)).



(a) Planning limit 7

(b) Planning limit 13

Figure 6.5: Dependency of successful service on threshold of timely service

Parameter	Domain
Consignments quantity	233
Couriers quantity	{25,35,45}
Alpha value of beta distribution	2
Beta value of beta distribution	3
Late arrival tolerance	300 s
Threshold of timely service	{0.5, 0.7, 0.9}
Limit of locations in planning	{7,10,13}
Time estimation confidence	0.4
Weight of capacity in part. effort formula	0.4
Timely service weight in solution eval. formula	0.4
Negotiation interval	10 s
Beginning of a shift	8 am
End of a shift	14 am

Table 6.3: Input parameters for analysis of successful service ratio

As we already indicated above, it is suitable to use short plans together with high threshold of timely pickup or delivery to increase percentage of served consignments.

6.2.3 Analysis of waiting at pickup and delivery

Waiting is an unpleasant situation and it shows how the courier company is reliable. All measured data of waiting time are displayed on graphs 6.6. Let us take a deeper look on the waiting time behaviour, but before it we must mention that complete experiment setting can be found in Table 6.4.

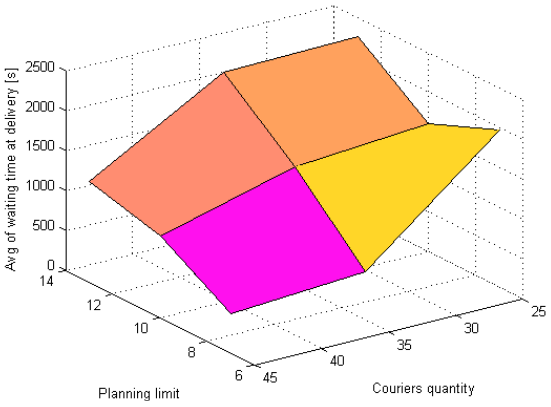
It is possible to see a significant difference between the short plans and the long ones. For example (see Graph 6.6(a)) in case of 35 couriers the average waiting time at delivery for the longest plans is almost 4 times greater in comparison with the shortest ones. A planning uses the most probable traveling time and these high values show its difference to a reality. On the other hand, median values are lower than average, so the bigger part of couriers is serving in time and the rest of them deliver late (compare values from Graphs 6.6(a) and 6.6(b)).

A comparison of waiting time at pickup and the delivery is very interesting. Couriers pick up consignments in time almost all the time (see median values on Graph 6.6(d)), because the time windows at pickup are wide and the pickup locations are served earlier. The chance of the plan violation is smaller than at the delivery location.

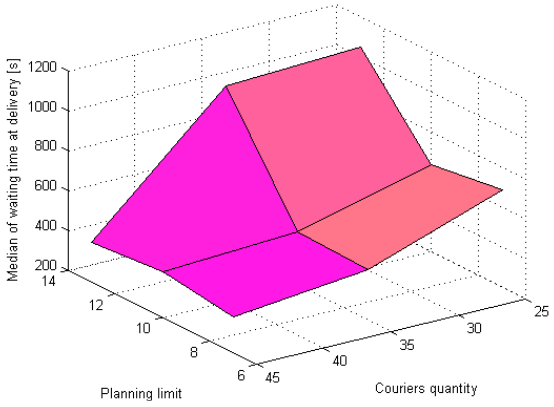
Best results are reached if we use as many couriers as possible together with short plans. Similar results are described in the previous experiment.

Parameter	Domain
Consignments quantity	233
Couriers quantity	{25,35,45}
Aplha value of beta distribution	2
Beta value of beta distribution	3
Late arrival tolerance	300 s
Threshold of timely service	0.7
Limit of locations in planning	{7,10,13}
Time estimation confidence	0.4
Weight of capacity in part. effort formula	0.4
Timely service weight in solution eval. formula	0.4
Negotiation interval	10 s
Beginning of a shift	8 am
End of a shift	14 am

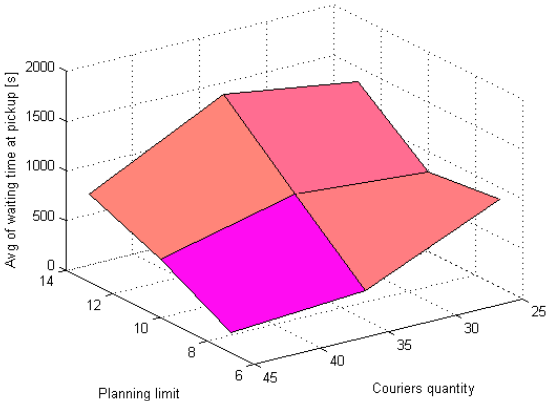
Table 6.4: Input parameters for analysis of waiting time



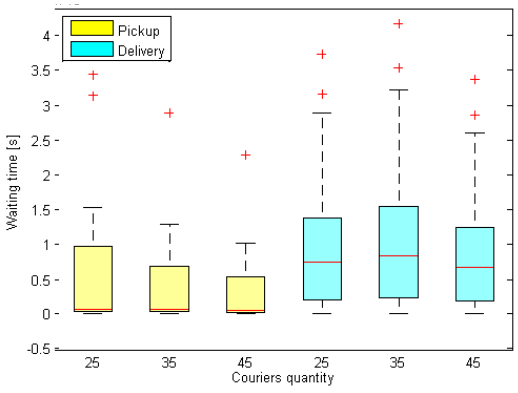
(a) Average waiting time at delivery



(b) Median waiting time at delivery



(c) Average waiting time at pickup



(d) Boxplots of waiting time

Figure 6.6: Dependency of waiting time on couriers quantity and planning limit

Parameter	Domain
Consignments quantity	233
Couriers quantity	{25,35,45}
Alpha value of beta distribution	2
Beta value of beta distribution	3
Late arrival tolerance	300 s
Threshold of timely service	0.7
Limit of locations in planning	{7,10,13}
Time estimation confidence	0.4
Weight of capacity in part. effort formula	0.9
Timely service weight in solution eval. formula	0.4
Negotiation interval	10 s
Beginning of a shift	8 am
End of a shift	14 am

Table 6.5: Input parameters for analysis of couriers load

6.2.4 Analysis of courier's workload

Couriers' workload is the last metric chosen for an individual analysis. Experiment's setting is shown in Table 6.5. A basic property of courier's workload is the variance, so some couriers have no consignment through the whole shift and some of them have always many consignments. It seems a little bit unfair but algorithm prefers bigger load in front of the smaller one. The short plans works as balancing factor, because they limit a number of maximal consignments served by one courier. Graph 6.8(c) shows this balancing for the planning limit equal to 7 - the line is almost steady. To be fair we must notice that huge decrease of couriers workload in configuration for 45 couriers and planning limit 13 will be caused mainly by low percentage of assigned consignments (see Section 6.2.2).

There is a group of boxplots on Figure 6.7. We can see again a big variance and situation when some courier are always almost full utilized. In general, we can say that metric of courier's workload behaves according to our intuition: the quantity of consignment remains constant, couriers load decreases with the increasing quantity of couriers.

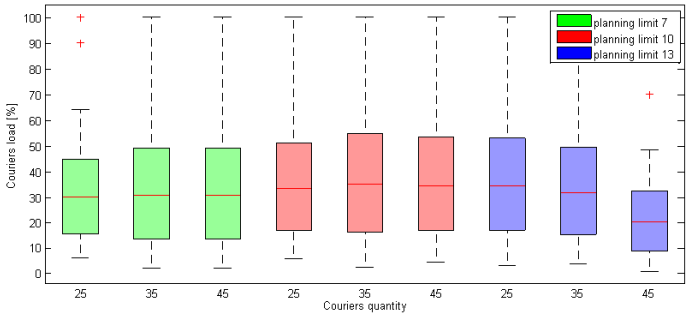
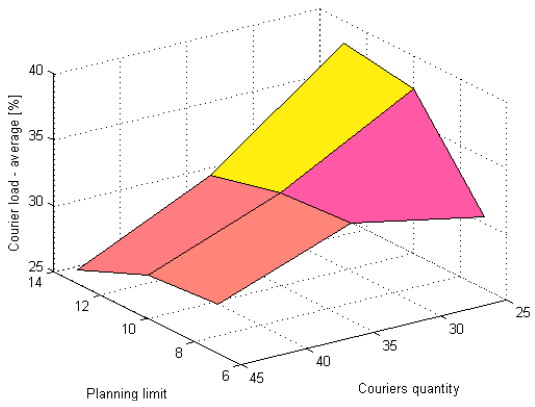
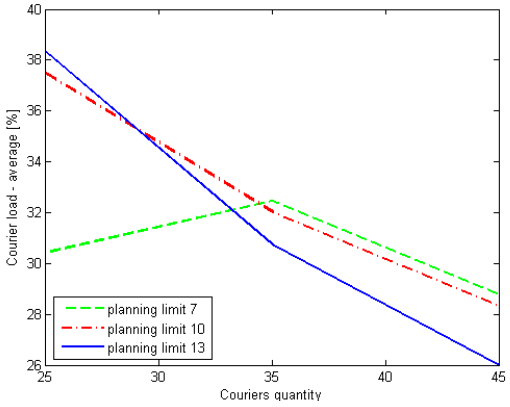


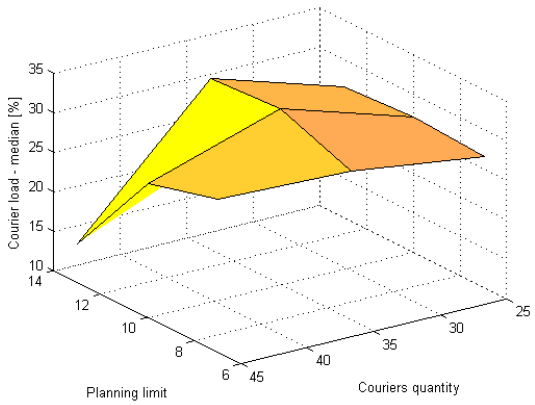
Figure 6.7: Boxplot graph of couriers load



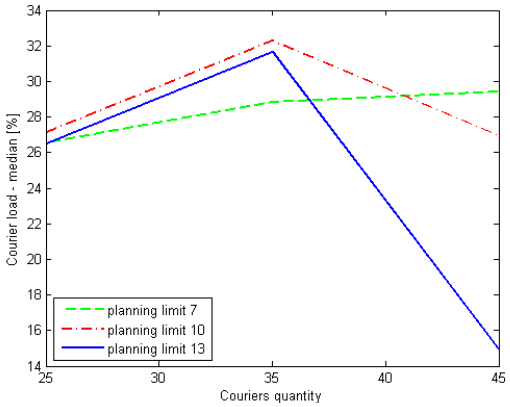
(a) Average 3d graph



(b) Average



(c) Median - 3d graph



(d) Median

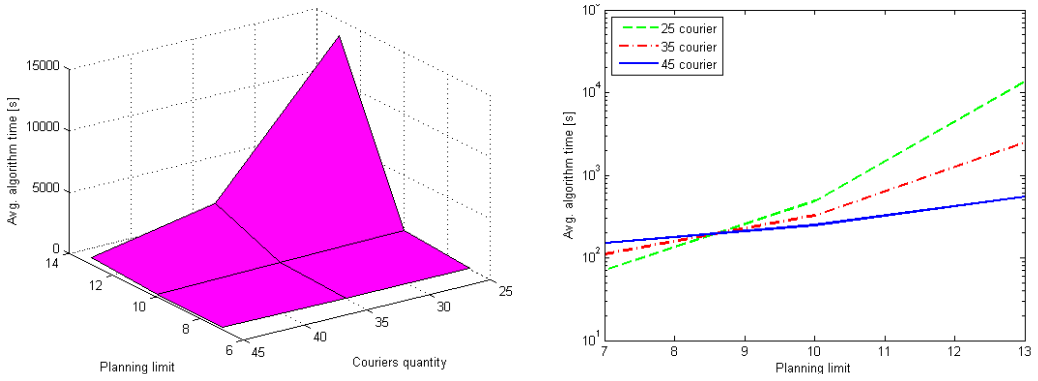
Figure 6.8: Dependency of courier's load on couriers quantity and planning limit

6.2.5 Computational performance

Experiments were performed on seven computers in total and even though it takes many days of computing to obtain all data. The total computational time spended by the simulation runs is 2828 hours (117.8 days). Thank to the possibility of executing multiple simulations in parallel we shrink that time to 'only' 10 days of non-stop computing. List of all used computers is shown in table 6.6. For comparison of simulation time we selected measured data from one particular machine.

Each individual simulation configurations takes different period of time and we have tried to find some relationships. The planning limit parameter has the biggest influence on the algorithm duration[24] as you can easily see on Graph 6.9(a). It is very important to remind that the highest peak has a combination of big planning limit and small couriers' quantity. The second Graph 6.9(b) shows development of the algorithm time on the logarithmical scale.

[24] Algorithm duration is a time spended only on procedures presented in the Section 4.3. Time required on messaging between agents, movement across transportation graph etc. is not included in the algorithm time.



(a) Overview of algorithm duration (b) Alg. duration on logarithmical scale

Figure 6.9: Algorithm duration

Computer	Specification	Computer	Specification
CPU RAM OS	Intel Core i3-370M, 2.40Ghz 8 GB Windows 7 HomePremium	CPU RAM OS	Intel Core i3-2100T, 2.50Ghz 4 GB Windows 7 HomePremium
CPU RAM OS	AMD Phenom II X2 545, 3.0 GHz 12 GB Windows 7 Profesional	CPU RAM OS	Intel Core i5-4210U, 2.40Ghz 8 GB Windows 8.1
CPU RAM OS	Intel Core i5-3317U, 2.60Ghz 6 GB Windows 7 HomePremium	CPU RAM OS	Intel Core i7-920, 2.66Ghz 8 GB Windows 7 Profesional
CPU RAM OS	Intel Core i7-4770K, 4.2Ghz 16 GB Ubuntu 13.04		

Table 6.6: Computers configurations used for experiment's evaluation

6.3 Evaluation summary

Experiments and their evaluation reveal potential system behaviour. We summarize the most interesting results in this section.

- The lowest values of effort required to serve all consignment are reached by using as many couriers as possible in combination with the maximal computational power. The very same behaviour was observed at the total traveled distance.
- Usage of long plans leads to the small system flexibility causing low percentage of the assigned consignments.

- If the system has a flexible setting (especially the short plans) the high level of threshold of timely service can cause more appropriate planning. Thank to it for example increase a percentage of assigned consignments.
- Proposed system constructs such plans that some couriers arrive to destination location quite late. So it does not always guarantee timely service.
- The system has big variance in values of courier's workload. Some couriers are fully utilized almost all the time and contrarily some couriers have no assignment in the whole shift. We can decrease this variance by using short plans.
- By decision to use small planning horizon we can reach a very big computational throughput. Therefore even large instances seem to be solvable in reasonable time.

Chapter 7

Conclusion

The goal of this work was to find out if it is possible to create an automatic system for dynamic vehicle routing problem and to implement such automatic system into process currently solved by skilled human dispatchers. We have examined both the area of general vehicle routing problem and concrete works studied dynamic assignment to couriers. With obtained knowledge we have proposed a solution based on a contract net protocol. We have formalized its individual parts relating to our problem and key decision parts we have described mathematically (e.q. how to compute an exact values of decision variables). After it we have applied a model of contract net protocol into a multi-agent system and its negotiation protocol was discussed in detail.

A determination of local planning problem (at a level of courier's agents) was a very important moment of our work, because we have realized the possibility to solve 'only' static case of vehicle routing. We have used simple greedy search algorithm to solve this relatively easy task (we consider it as easy, because it is widely studied and our instances were small), because this is only a minor part of the whole work.

The final model has been integrated into a simulation framework DarpTestBed and exhaustively evaluated (data computation spended thousands of hours of the CPU time) on a real data of courier company residing in Prague. The results are very encouraging, because it is possible to decide automatically and reasonable up to 96-98% of consignment's assignments. However, there are also consignments being delivered late in the final plans. Therefore our system is suitable as a support system and the final decision should still remain on human dispatchers. Nevertheless, it is a very promising result, because even half-automated system is far more effective and reliable than the manual one. Moreover we can discuss the properties of the proposed plans and alternatively say how far the solution is from the optimum (when someone finds an optimal solution of dynamic vehicle routing).

The proposed system has very high potential in production deployment, where it can help companies to cross over a higher demand for same day or immediate delivery services. It can also foreshadow the first step to a future fully automatized assignment systems.

Bibliography

[1] Lukáš Čanda. Simulační testbed pro algoritmy poptávkové přepravy. 2013.

[2] Bo Chen and H.H. Cheng. A review of the applications of agent technology in traffic and transportation systems. *Intelligent Transportation Systems, IEEE Transactions on*, 11(2):485–497, June 2010.

[3] Teodor Gabriel Crainic, Michel Gendreau, and Jean-Yves Potvin. Intelligent freight-transportation systems: Assessment and the contribution of operations research. *Transportation Research Part C: Emerging Technologies*, 17(6):541 – 557, 2009.

[4] Paul Davidsson, Lawrence Henesey, Linda Ramstedt, Johanna Törnquist, and Fredrik Wernstedt. An analysis of agent-based approaches to transport logistics. *Transportation Research Part C: Emerging Technologies*, 13(4):255 – 271, 2005. Agents in Traffic and Transportation: Exploring Autonomy in Logistics, Management, Simulation, and Cooperative Driving.

[5] Jan Fabian Ehmke. City logistics. In *Integration of Information and Optimization Models for Routing in City Logistics*, volume 177 of *International Series in Operations Research Management Science*, pages 9–22. Springer US, 2012.

[6] Francesco Ferrucci, Stefan Bock, and Michel Gendreau. A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research*, 225(1):130 – 141, 2013.

[7] Miguel Andres Figliozzi, Hani S. Mahmassani, and Patrick Jaillet. Competitive performance assessment of dynamic vehicle routing technologies using sequential auctions. *Transportation Research Record: Journal of the Transportation Research Board*, Volume 1882:10 – 18, 2004.

[8] Michel Gendreau, François Guertin, Jean-Yves Potvin, and René Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157 – 174, 2006.

[9] Gianpaolo Ghiani, Emanuele Manni, Antonella Quaranta, and Chefi Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96 – 106, 2009.

[10] Michal Jakob and Michal Čertický. Simulation testbed for the accelerated development of autonomic mobility systems. *1st International Systems Competition on Autonomic Features and Technologies for Road Traffic Flow Modelling and Control Systems*, 2013.

[11] Petr Kalina, Jiří Vokřínek, and Vladimír Mařík. The art of negotiation: Developing efficient agent-based algorithms for solving vehicle routing problem with time windows. 8062:187–198, 2013.

[12] Petr Kalina, Jiří Vokřínek, and Vladimír Mařík. An efficient route minimization algorithm for the vehicle routing problem with time windows based on agent negotiation. 8291:149–164, 2013.

[13] Robert Kohout and Kutluhan Erol. In-time agent-based vehicle routing with a stochastic improvement heuristic. In *11th Conference on Innovative Applications of Artificial Intelligence*. AAAI/MIT Press, 1999.

[14] Hon Wai Leong and Ming Liu. A multi-agent algorithm for vehicle routing problem with time window. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 106–111, New York, NY, USA, 2006. ACM.

[15] Sandro Lorini, Jean-Yves Potvin, and Nicolas Zufferey. Online vehicle routing and scheduling with dynamic travel times. *Computers Operations Research*, 38(7):1086 – 1090, 2011.

[16] Martijn Mes, Matthieu van der Heijden, and Peter Schuur. Interaction between intelligent agent strategies for real-time transportation planning. *Central European Journal of Operations Research*, pages 1–22, September 2011.

[17] Martijn Mes, Matthieu van der Heijden, and Aart van Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. (161), 2005.

[18] Petr Mezek. Agentní simulace taxi spolujízdy. *České vysoké učení technické v Praze*, 2011.

[19] Marek Modrý. Agent-based modelling of urban crime. *České vysoké učení technické v Praze*, 2011.

[20] SophieN. Parragh, KarlF. Doerner, and RichardF. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

[21] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1 – 11, 2013.

[22] Jean-Yves Potvin, Ying Xu, and Ilham Benyahia. Vehicle routing and scheduling with dynamic travel times. *Computers Operations Research*, 33(4):1129 – 1137, 2006. Part Special Issue: Optimization Days 2003 Part Special Issue: Optimization Days 2003.

[23] Harilaos N Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. page 130, 1980.

[24] Amelia Clare Regan, Hani S. Mahmassani, and Patrick Jaillet. Real-time information for improved efficiency of commercial vehicle operations. 1198.

[25] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, December 2010.

[26] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA, 2008.

[27] Alan Slater. Specification for a dynamic vehicle routing and scheduling system. *International Journal of Transport Management*, 1(1):29 – 40, 2002.

[28] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104–1113, Dec 1980.

[29] Jiongjiong Song and Amelia Regan. Approximation algorithms for the bid construction problem in combinatorial auctions for the procurement of freight transportation contracts. *Transportation Research Part B: Methodological*, 39(10):914 – 933, 2005.

[30] P.J. 't Hoen and J.A. La Poutré. A decommitment strategy in a competitive multi-agent transportation setting. In *Setting", Proceedings of AAMAS 2003*, pages 1010–1011. ACM Press, 2004.

[31] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.

[32] Min Wen, Jean-François Cordeau, Gilbert Laporte, and Jesper Larsen. The dynamic multi-period vehicle routing problem. *Computers Operations Research*, 37(9):1615 – 1623, 2010.

[33] N.H.M. Wilson, N.J. Colvin, Massachusetts Institute of Technology. Center for Transportation Studies, United States. Urban Mass Transportation Administration, and Rochester Genesee Regional Transit Authority. *Computer control of the Rochester dial-a-ride system*. Massachusetts Institute of Technology, Center for Transportation Studies, 1977.

Appendix A

DVD contents

- implementation/
  - demo/
    - experiments/ - directory holding experiment's setting. A file scenario.groovy is the main configuration file of each experiment
    - logistics_testbed_lib/ - required libraries of the runnable example
    - demo.bat - script to run example on windows
    - logistics_testbed.jar - runnable example of simulation
  - source_codes/
    - experiments/
    - extensions/
      - logisticstestbed/ - source codes of LogisticsTestBed application
    - mobilitytestbed/
    - mobilitytestbedanalyzer/
    - mobilitytestbeddbtokmlexporter/
    - mobilitytestbedgenerators/
- text/
  - eclipse_project/
  - PETR_MEZEK_DP.pdf
- readme.txt