Operational Transport Planning in a Multi-Agent Setting

Jonne Zutt

CVS revision: 1.540.
CVS date: 2010/04/05 20:01:21 (UCS).
Ran through LaTeX at 17-09-10 17:54.

Comics and cover illustration: Pure Art, Nathan van der Stoep
Cover design: Joke Herstel, *Wenk*

Operational Transport Planning in a Multi-Agent Setting

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College van Promoties,
in het openbaar te verdedigen op weekdag *dd* maand 2010 om *hh*:*mm* uur
door Jonne ZUTT
ingenieur in de Informatica
geboren te Warmenhuizen

Dit proefschrift is goedgekeurd door de promotor:

Prof.dr. C. Witteveen

Copromotor: Prof.dr.ir. A.J.C. van Gemund

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. H.J. Promotor	Technische Universiteit Delft, promotor
Prof.dr. J-J.Ch. Promotor	Universiteit Utrecht, promotor
Dr. C. Co-promotor	Technische Universiteit Delft, toegevoegd promotor
Prof.dr.ir. A.N. Other	Technische Universiteit Delft
Prof.dr.ir. A.N. Other	Technische Universiteit Delft
Prof.dr.ir. A.N. Other	Technische Universiteit Delft
Dr.ir. G. Uest	Other Universiteit

TRAIL-Thesis Series T2010/*x*, The Netherlands TRAIL Research School

Published and distributed by: Jonne Zutt
E-mail: jonne@zutt.org

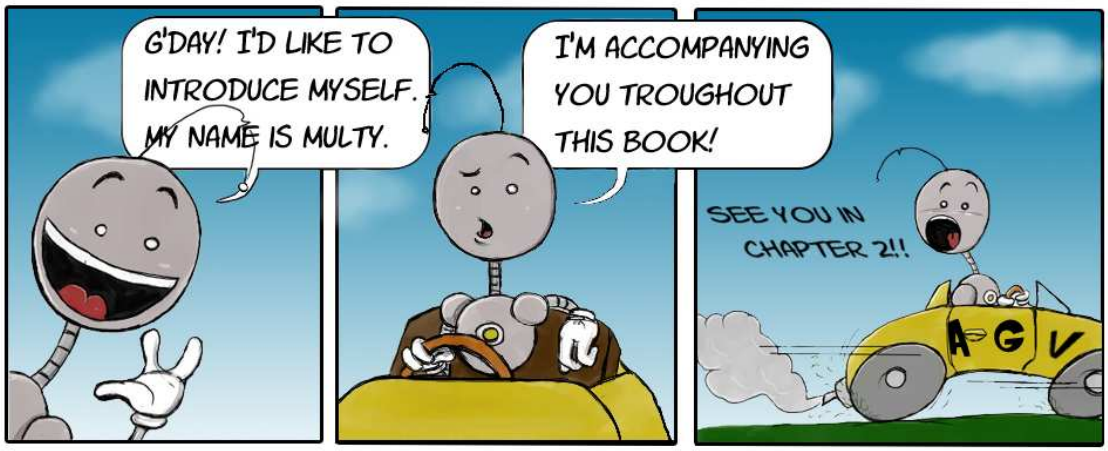
ISBN: 90-*xxx-xxx-x*

Keywords: Research subject, another keyword resource allocation

Printed in The Netherlands

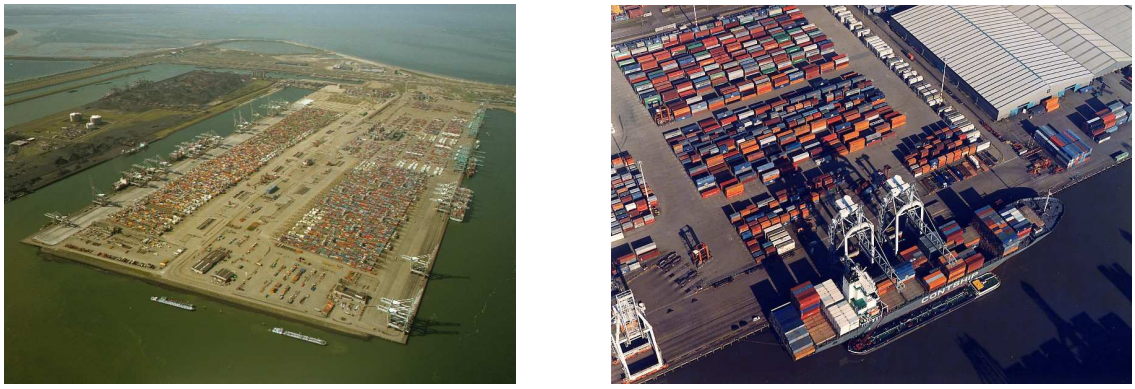
Chapter 1

Introduction



Transportation is among the strongest growing activities in our society. Reasons for the increase of transportation are legion. Transportation costs decrease while quality increases (e.g., airplanes), leading to ever-growing demand for transportation. Not only the growing number of people, but also behavioral changes lead to increasing demand for transportation. For example, existence of the Internet sets up easy world-wide communication for organizations leading to globalization. Also, the growth of industry and the large amount of supply on the market demands for bigger sized enterprises that have to organize their transportation efficiently. At the same time, the growth of transportation renders our transportation infrastructure, like roads, railways, harbors and airports, a scarce resource. Typically, however, plans to increase the transportation infrastructure either take a long time to realize or are simply not realizable at all. As a result, there is an incredible amount of effort in optimizing the process of both public and freight transportation using the current transportation infrastructure as efficiently as possible.

One way to optimize the current use of transportation infrastructure is complete automation of transportation. This development started small by, for example, carrying heavy materials at Corus by robots(Withers and Rijnsdorp, 1978), but is on the rise. In



(a) Aerial view.

(b) Quay cranes unloading a ship.

Figure 1.1: European Container Terminals (ECT).

1988, European Container Terminals (ECT) realized the first ever robotized container terminal that caught world-wide attention (Konings et al., 2009). Automated Guided Vehicles (AGVs) transport freight containers between quay cranes and stocking yard back and forth. These AGVs are completely controlled and steered by computer programs and drive around 24 hours a day, using the existing infrastructure quite efficiently. Nowadays, there are approximately 200 AGVs operational at ECT. Challenges at ECT are to know at all times where containers are exactly located (often techniques with radio frequency identification numbers (RFID) are used to accomplish this), how to stack these containers in the stock yard (if you would need a container soon, you would not want to stack a few others on top of it) and in general how to process the freight (loading or unloading ships, trains, and trucks) as fast as possible (see Figure 1.1).

At the moment of writing, plans are developed to create an underground logistic system (OLS) at Schiphol international airport. This pipe-line connects the flower market of Aalsmeer, Schiphol airport, and a new rail-terminal in Hoofddorp to each other. Feasibility studies on this project are not very positive on short terms, though estimate that such systems become more and more important in the future(Verbraeck and Versteegt, 2001).

Also in public transportation we see the birth of automation. The idea of automated driving dates back about 70 years, when General Motors presented a vision of *driverless* vehicles moved under automated control at the 1939 World's Fairs in New York (Rodney K. Lay, 1996). According to Cheon (2001), researchers began to consider potential uses of computers in traffic management. The fully automated highway was initially examined by General Motors during the late 1970s. Due to the advances in computing technologies, microelectronics, and sensors in the 1980s, the University of California Partners for Advanced Transit and Highways (PATH) program has carried out significant research and development efforts in highway automation since the 1980s. Furthermore, in 1994 the U.S. Department of Transportation launched the National Automated Highway

System Consortium (NAHSC) to fund long-term research on Automated Highway Systems. These techniques could ensure a more efficient use of (high)ways by allowing cars to operate in so-called platoons, driving at a very close distance from each other, thereby increasing the capacity of highways.

Europe does not stay behind in automated transportation. For example, in Paris, underground railway is partly automated. Control of subways is relatively easy due to driving on rails and no interference with other (human) traffic.

The automobile industry in the late nineties introduced *Adaptive Cruise Control.* This system attempts to increase the comfort of a driver by automatically adjusting speed to the direct successor based on distance sensors. Due to the introduction of vehicle-to-vehicle communication, the system could be improved to react faster and more smoothly. There is still a lot of ongoing research in the area of *Co-operative Adaptive Cruise Control*, for instance on the effect of this system on the traffic throughput (van Arem et al., 2006; Laumonier et al., 2006).

Although these automation processes are promising, success of this new technology is heavily depending on public acceptance. Furthermore, there are difficulties here of ethical grounds, e.g., who is to blame if a collision occurs? On the other hand, there are technological advances that can more easily be accepted by the public, like (dynamic) route navigation systems that, for now, could advise only and perhaps later take over human control.

Automation in transportation does not need to be restricted to the automation of the real-time control of (groups of) vehicles. Often, careful planning of transportation processes is as important in increasing the capacity of the infrastructure. The classical approach in automating transportation processes, however, has focused on planning and real-time control as two separate areas. In this thesis, first of all, we will propose an integrated approach to automation of transportation processes by discussing a so-called *context-aware* planning approach, where planning takes into account not only the route of a single vehicle, but the already planned routes of other vehicles as well. In this way, ideally, if the plan is executed there is no further need for real-time control or conflict-resolution, since everything has been taken care for in the context-aware planning phase.

Typically, a contex-aware planning approach takes existing plans into account by constructing a route plan for an individual vehicle. Such a route plan consists of a sequence of (time limited) reservations of parts of the infrastructure, such that conflict-free execution of the plan can be ensured if every one is able to execute his plan correctly.

In this thesis we first of all contribute to the context-aware planning approach by improving the efficiency of existing systems in a significant way and by empirically investigating alternative ways for determining infrastructure reservations, for example, by not using a fixed first-come-first-serve allocation scheme, but also allowing for more sophisticated allocation schemes where the priority of vehicles is taken into account.

Our second main contribution is in the investigation of the role of incidents on the efficiency of automated transportation planning. Often in realistic scenarios there are certain events that cannot be anticipated in advance and have a negative influence on the planned activities leading to the necessity of replanning. We call these events *incidents* and taking incidents into account *incident management*. Robust planning and replanning techniques are required in situations where incidents occur regularly. To give some examples, for an AGV terminal, incidents include malfunctioning communication with a vehicle, engine problems, collisions. Also, changes in transportation requests – even adding new transportation requests – can be viewed as incidents. Planning methods that suffer from performance degradation when the number of disruptions increases render useless in many realistic transport planning applications.

In this thesis we therefore focus on the performance of route planning methods when there are incidents and compare context-aware approaches with classical route-planning approaches.

The basic scientific problem area we are contributing to in this thesis is known as the *pickup and delivery problem*. To give the reader a better idea of this problem area, its development and our contribution, in the sequel of this chapter we will first describe the basic ingredients of thel pickup and delivery transport planning problem Then we present the main challenges in this domain and give an example. Subsequently, our approach, research questions and contributions are presented.

1.1 Pickup and delivery transport planning

In pickup and delivery transportation planning, multiple actors (agents) each have to construct a transportation plan for their vehicle and to ensure that their set of *transportation requests* is correctly executed. A transportation request is a customer's request to deliver freight (or passengers) from a pick-up location (the current location of the freight to be picked up) to the delivery location (the destination of the freight).

Additionally, the customer provides several constraints on how he or she desires the transportation request to be executed. The pickup and delivery must preferably be executed within corresponding *time-windows*; a time-window is an interval of time. Furthermore, the customer specifies a *reward* function for the executor of the transportation request based on these time-windows and the actual time at which the executor will perform the pick-up and delivery events. This reward function will obviously be maximized if the time-windows for loading and unloading are not violated.

The executor of the request owns a *transport resource*. A transport resource is a mobile entity like an automated guided vehicle (AGV), a truck, an airplane, a boat, or a taxi cab. This transport resource can move around through the *transport network*. The transport network defines both locations – at least all of the pick-up and delivery locations

encountered in transportation requests – that can contain transport resources as well as connections between locations. The transport network and resources together are referred to as the *infrastructure*.

The allocation of transportation requests to executors is called *task assignment*. Task assignment can be aided by *auctioneers* that use auction protocols to determine which executor should be assigned which transportation request(s). Task allocation, and also how the individual executors determine the order in which they complete their tasks, is considered to take place at the strategical level and, therefore, considered outside the scope of this thesis. We simply assume each transport resource has a visiting sequence of locations and by traversing this visiting sequence it will correctly execute its transportation requests.

Usually, there is a common transportation network where the vehicles are executing their plans. Due to limited capacities of this network, these individual transportation plans might easily interfere with each other. Therefore, we need to find efficient plans for the individual vehicles, but also need to avoid conflicts with other vehicles during execution of these plans.

Once one or more transportation requests are assigned to an executor, this executor must create an initial *plan* to execute its assigned transportation requests. Such a plan consists of a *route*, a *schedule*, *loading*, and *unloading* information. The route is a sequence of locations that are connected in the given transport network. Of course, this route must visit all pick-up and delivery locations of the assigned transportation requests. A schedule is a sequence of time points that specifies at which time the transport resource plans to travel to the next location. Finally, the loading and unloading information specifies when freight (or passengers) is loaded and unloaded from the transport resource.

There are several aspects that influence the *efficiency* of plans. First, the executors are responsible for executing the transportation requests that are assigned to them. More specific, the reward function is a measure for the efficiency of this request's execution. Second, what are the costs of the executor? We can distinct between fixed and variable costs. Fixed costs are costs for owning or renting the transport resource. Variable costs are costs for having a driver for the transport resource. Possibly, one might want to distinct between different states like (un)loading, driving, waiting for other agents, or being idle.If searching for not only feasible[1] plans, but also efficient plans, one must define a certain cost model that incorporates these notions.

The final aspect of pickup and delivery planning we need to consider is the occurrence of *incidents*. Incidents are events that can disrupt the regular operation of a system, in this case composed of a transportation infrastructure (the network), vehicles, and transportation tasks. Such incidents may occur due to sudden changes to transportation tasks or

[1]By feasible plans we mean a plan that is possible to execute, visits all pick-up and delivery locations of assigned transportation requests and loads and unloads all freight or passengers correctly.

due to malfunctioning of one or more individual resources (errors at the infrastructure resource or transportation resource level).

Although a lot of research has been done in the area of pickup and delivery transportation planning, there are still several challenges remaining. Those are the topic of the next section.

1.1.1 Challenges

Important issues in the transportation domain are the guide-path design (transport network), vehicle scheduling, idle-vehicle positioning, vehicle maintenance (e.g., battery management for AGVs), vehicle routing and conflict resolution (Le-Anh and de Koster, 2004). Our focus is on the short-time online operational planning level. That is, we abstract from low-level control specific aspects like braking, accelerating, and steering together with vehicle properties required to work with these control actions. Strategic issues, such as models to estimate the best number of vehicles or to optimize the guide-path, are also not included.

We consider the most important challenges in pickup and delivery transporation planning to be:

- *Scalability*: due to more widely usage and success,
- *Robustness*: complete automation, no human control,
- More insight in the specific *factors* that make a problem difficult,
- Insight in what *information* is necessary for a planner to perform well.

These challenges form the source of our research and resulted in the contributions described in Section 1.5. The next section describes a concrete example of pickup and delivery transportation.

1.2 Approach

The classical solution to the operational pickup and delivery transportation problem is, in order to reduce its complexity, to separate route planning from run-time conflict resolution. First, each actor plans an optimal route to execute its transportation plan and during execution conflict resolution is used to ensure to remove actual capacity conflicts. This approach has several disadvantages: sometimes it will result in gridlocks[2] and the completion time of the transportation requests are in general not predictable in advance.

[2] A gridlock is a traffic jam so bad that no movement is possible.

The example of the previous section, however, clearly showed that vehicles (or airplanes) have to consider the plans of other vehicles in the network to reach a good performance. More advanced approaches try to separate between route planning and conflict resolution during planning time. With this approach route planning is still done without considering potential conflicts, but the completion time of transportation requests is better predictable at the time the transportation plans are executed. It is, however, questionable how far this separation (of route planning and conflict resolution) harms the quality of the final transportation plans. It is of course possible that some conflicts can be avoided by the vehicles by making detours.

Other researchers adopt an integrated approach, where conflict resolution is integrated with route planning (context-aware routing). With this approach it is guaranteed that vehicles find the optimum plan, i.e., the fastest possible way to reach their destination, given that prior reservations of other vehicles do not change anymore. The best known result is that of Kim and Tanchoco (1991), which has a high computational complexity.

We will design a new framework for multi-agent transportation planning where we distinguish transportation agents and infrastructure agents. Transportation agents make transportation plans, while infrastructure agents make reservation plans for infrastructural resources (lanes, crossings). In making their transportation plans, transportation agents query infrastructure agents about the availability of parts of the infrastructure they need. This extends the approach of Kim and Tanchoco (1991) by allowing infrastructure agents to use other reservation policies than a simple first-come-first serve and, e.g., to take into account priorities of agents. With this flexible framework infrastructure agents can optimize the throughput of the transport network, while the transport agents strive to maximize their own performance.

To see the effect on the performance of the context-aware routing approach versus approaches that separate routing from conflict resolution several experiments will be presented in this thesis. In synthetic problem instances where the important factors, such as the transport network, set of vehicles and the set of transportation requests, have carefully been varied the difference in performance will be measured empirically.

Critics of the context-aware routing approach claim that one might doubt whether it is useful to invest this much time in finding plans which can be destroyed by a few incidents (and cause a re-planning). Would the classical approach perform equally well in incident-rich circumstances? One important aspect to consider is that with the context-aware approach the infrastructure agents know the reservations of all vehicles. Therefore, if an incident occurs, they know exactly which vehicles might be affected by the incident and the infrastructure agents inform the transport agents about delays and unavailability of parts of the infrastructure, enabling them to replan their routes.

Hence, another important factor in the experiments is the level of incidents. The effect of incidents on the performance of the system will be empirically evaluated for the

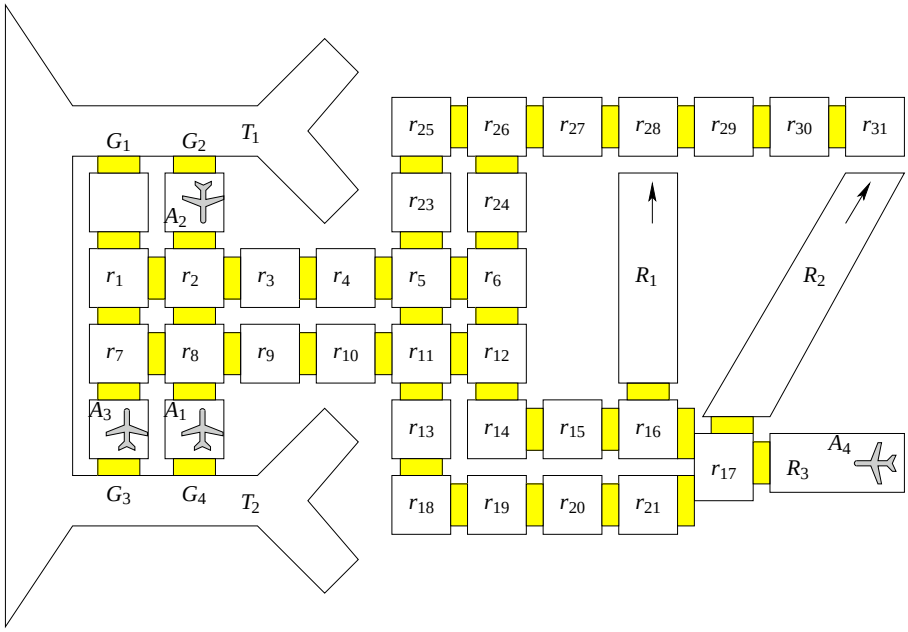


Figure 1.2: An imaginary airport network example.

different transport planning approaches.

1.3 Example

This section describes some of the challenges (automated) systems are faced in pickup and delivery transportation. Figure 1.2 shows a simple example of an imaginary airport, where airplanes have to taxi from runways to gates and back. The example includes four airplanes A_1, A_2, A_3 and A_4, two terminals T_1 and T_2, four gates G_1, G_2, G_3 and G_4, three runways R_1, R_2 and R_3, and the remaining resources have been named $r_1, \ldots, r_{31}$. We assume that all locations have a capacity of 1; hence, no two airplanes can have overlapping reservations for any of the infrastructure resources at any time. The time to traverse a resource is also assumed to be the same, say 1, for all resources.

The airplanes are numbered in the order they had contact with the air traffic controllers. Therefore, air traffic control first received and checked the plan of airplane A_1, then communicated with airplane A_2 followed by A_3 and, finally, airplane A_3 announced its plan. Each time an airplane informs air traffic control about its plan, the latter checks whether the plan is valid with respect to prior reservations of the other airplanes and then makes reservations for the new plan.

The first thing we can learn from this example is that, if two airplanes travel a completely opposite route – say airplane A_2 travels to runway R_3 and airplane A_4 does the opposite – a deadlock is unavoidable if the airplanes start traversing simultanesouly and

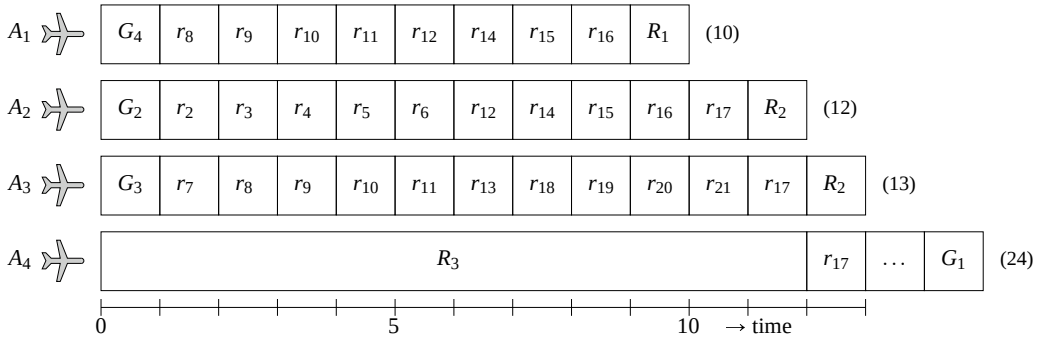


Figure 1.3: An example plan for the four airplanes. Vertically, the airplanes are listed and the horizontal axis represents time. The total costs of this plan are $10+12+13+24=59$.

do not alter their route. However, we can improve on this situation by having each airplane create reservations that cannot be violated by the other airplanes.

The first airplane to construct a plan is A_1. Airplane A_1 wants to take off on runway R_1. Because there are no reservations of other airplanes at the moment, airplane A_1 can execute the route $(G_4, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{14}, r_{15}, r_{16}, R_1)$ immediately and as fast as possible. Airplane A_2 is not delayed by airplane A_1 and can go right behind it. The same holds for airplane A_3, who wants to take off at runway R_2, while both A_1 and A_2 have non-conflicting reservations. For airplane A_3, who wants to take off at runway R_2 as well, there are two possible plans that have the same costs. First, it can travel along the resources r_{10}, r_{11}, r_{12}, and r_{14}. In this case, it has to wait one time unit before it can enter resource r_{12}, because airplane A_2 is in r_{12} at that moment. Second, airplane A_3 can also travel along the resources r_{10}, r_{11}, r_{13}, r_{18}. The resulting plan is one resource longer, but there is no waiting time; hence, the time it enters the runway is the same. Let us say A_3 chooses the plan without waiting time. Then, Figure 1.3 illustrates these plans.

Just before touch down on runway R_3, airplane A_4 contacts air traffic control. It turns out airplane A_4 must taxi to gate G_1. There are two possible choices: the upper route $(R_3, r_{17}, r_{16}, \ldots)$ or the lower route $(R_3, r_{17}, r_{21}, \ldots)$. In both cases airplane A_4 wil encounter an airplane coming from the opposite direction (all resources have unit capacity), which means it cannot enter resource r_{17} until the other airplanes have finished using it. This results in a big waiting time for airplane A_4.

All airplanes constructed a valid plan, which reaches their destination resources as fast as possible, given the prior reservations. But is this plan the best possible plan for the airplanes? No, consider the plan illustrated in Figure 1.4. In this particular situation where airplanes travel opposite routes, it is usually better if from one side they take the one (e.g., upper) route, while from the other side the airplanes take the other route.

We can separate between two different approaches to improve the plan illustrated by Figure 1.3. The first is to *reschedule* one or more airplanes. This means the routes of

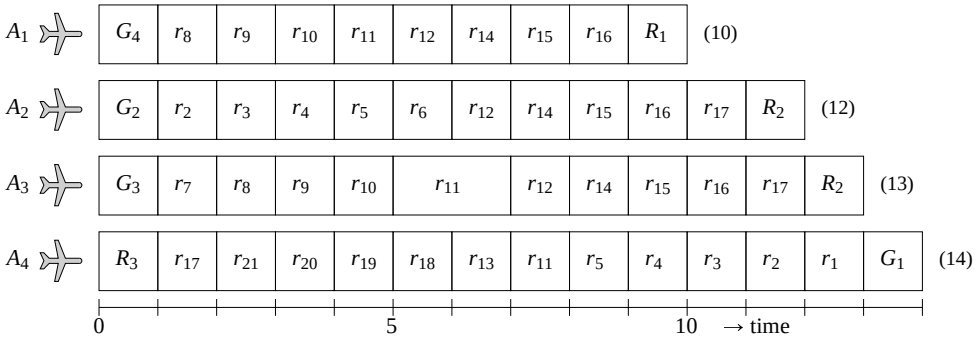


Figure 1.4: The total costs of this plan are $10 + 12 + 13 + 14 = 49$.

the airplanes remain unchanged, but the times at which they enter and exit resources is changed. In this example, rescheduling cannot result in the plan illustrated by Figure 1.4.

The other approach is to *reroute* one or more airplanes. Rerouting both changes the times at which the airplanes enter and exit resources, as well as the sequences of resources that form the routes of the airplanes. Obviously, with rerouting the optimal plan can be reached. The challenge here is the enormous amount of possibilities if the number of airplanes grows.

One possible approach to deal with this is to use *heuristics*, for instance to determine the order in which the airplanes search a shortest path. In this example, a good choice for such a heuristic might be to give priority to airplanes (for creating reservations), which have (approximate) opposite source/destination locations to airplanes that already created their reservations. The use of such a heuristic reduces the amount of possibilities that have to be considered, but sometimes, of course, prevent the discovery of the optimal plan.

The next section describes our approach to solve large pickup and delivery transportation planning problems. Using such an approach computers can support the air traffic controllers to make the right decisions.

1.4 Research questions

The research presented in this thesis addresses routing, scheduling, and conflict-resolution in the transportation planning domain; our first goal is to test and compare planning methods that separate route planning and conflict resolution with integrated planning methods. Furthermore, experiments will show the performance of the different infrastructure agent policies in the flexible framework.

The second goal is to find out whether the context-aware approach is also *robust;* we test and compare the different approaches under increasing density of disruptions like malfunctioning resources.

During the project an agent-based transport planning simulation tool called TRAPLAS

is developed to be able to do experiments and a benchmark set is constructed for pick-up and delivery transport planning problems.

1.5 Research contributions

With hindsight we list the main contributions of this thesis, grouped into categories:

Framework

- Distinguish between transport agents (making transportation plans) and infrastructure agents (making reservation plans for individual infrastructure resources).
- Resource management model for transportation planning.
- More general definition of a conflict not assuming unit capacity resources.

Methods

- Shortest-path algorithm taking into account both forbidden time-windows and an intermediate visiting sequence (ordered sequence of locations to be visited).
- Iterative traffic-aware dynamic (re)routing and scheduling methods that guarantee conflict-free plans.
- Incident management techniques in routing and scheduling.

Experiments

- Benchmark set for the transport planning problem together with a simulation environment that can plan, replan, and execute to gain experimental data. The sets of transportation requests were generated in such a way that the optimal solution is known (and an lowerbound for merged instances, which have increasing request load).
- Insight in what sort of information is important for the (re)planner while (re)planning pick-up delivery transportation under influence of incidents.
- What are the important factors in transportation that make a problem difficult to a (re)planner or that make a big difference in the results?

This research started as a follow up of a master's thesis research (Zutt, 2001) that has been demonstrated at the Belgium-Netherlands Artificial Intelligence Conference (Zutt and de Weerdt, 2000). The system architecture was first published in (Aronson et al., 2002a,b). These papers deal with the separation between the strategic, tactical and operational layer of a transport planning system and the individual activities that belong to these layers. As a joint work with Roman van der Krogt and Leon Aronson, both a paper on the tactical level and a separate paper on the operational level were published (Zutt et al., 2002; van der Krogt et al., 2002).

Some work on diagnosis has been carried out (Bos et al., 2002). However, this is mainly omitted from this thesis due to the choice of adopting a bottom-up approach. If an incident occurs, the system is signaled from where the disruption occurred and, hence, it is assumed that all information concerning the incident is known exactly.

In (Valk et al., 2001a,b,c) methods for coordination in the logistic domain were presented accompanied by some approximation results. The transport planning model, as used throughout this thesis, is first described in (Zutt and Witteveen, 2004). Apart from the model this paper includes the first planning methods, among others, the approach suggested by Hatzack and Nebel to transform route planning for a fleet of vehicles to a variant of Job Shop Scheduling. In (de Weerdt et al., 2003b), the first collaboration results are presented. The latest publication (Zutt et al., 2009) is a compilation of this thesis as a book chapter.

Of course, the transport planning simulation tool TRAPLAS, with all its extensions, is freely available under the GNU General Public License (GPL) at `http://traplas.sourceforge.net`. Also, a three-dimensional visualization toolkit, especially developed for TRAPLAS, is available at `http://traplasviz.sourceforge.net`. TRAPLASVIZ is based on OpenSceneGraph, an open source high performance three-dimensional graphics toolkit.

1.6 Outline of thesis

Chapter 2 describes a family of pickup and delivery problems and the state-of-the-art solution techniques, accompanied by benchmark results, to solve those problems.

Subsequently, Chapter 3 describes our transportation planning framework. Its most important aspect is the separation between transport agents and infrastructure agents, which makes it more flexible: transport agents optimize the execution of their own transportation requests, while infrastructure agents optimize the troughput of the transport network. This chapter also provides definitions for what we consider a conflict exactly and describes how to compute an individual reservation to access an infrastructure resource for a vehicle in such a way that it does not conflict with any existing reservation.

Chapter 4 presents our newly developed planning and scheduling methods. A traffic-aware shortest-path planning method is described; this is like regular shortest-path algorithms, e.g., the famous Dijkstra algorithm, extended with the awareness of the plans of other vehicles. Then, the planning methods are described that transport agents use to create plans to execute the tasks that were assigned to them. Extensions to the methods are presented that can deal with situations in which incidents occur.

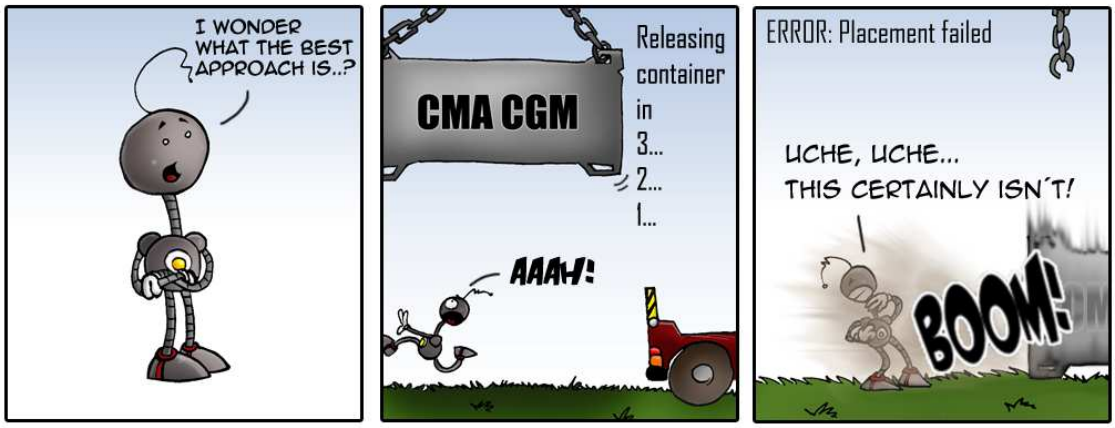
In Chapter 5 describes our experiments. For this purpose a transportation planning simulator, called TRAPLAS, is developed, which is built on top of the Pamela Run-Time Library of van Gemund (1994). The construction of a benchmark set of pick-up and delivery transportation planning problem instances – satisfying our transportation model – is given. The simulator together with the benchmark set have been made publicly available as a SourceForge project (named TRAPLAS). The experiments were efficiently executed on the supercomputer (the Distributed ASCI Supercomputer, DAS-2).

The performance of different planning methods is tested and compared while varying the transport network structure, the number of transportation requests and the number of transport agents. The same experiments are repeated while increasing the level of incidents to test the robustness of the planning methods. Methods that performance best under normal circumstances, might be inferior in case of disruptions that disturb normal plan execution. Throughout these chapters figures are presented to illustrate the scalability of the approaches – in other words, can the planning methods still be used if the system grows (e.g., bigger transport networks, higher request loads, more agents, more incidents).

Appendix A is included as a guide for the reader about the notation used throughout this thesis. In Appendix D it is proven that the transport planning problem is at least as hard to solve as several other famous problems like Satisfiability, the Traveling Salesman Problem (TSP) and others. This supports the development of planning methods that, instead of searching for the best possible (or optimal) solution, are satisfied with searching for approximations of these optimal solutions.

Chapter 2

Pickup and delivery transportation



This chapter describes well-known problems with state-of-the-art solution techniques in the field of pickup and delivery transportation. Although research has progressed very well in the field of pickup and delivery transportation, it cannot easily be determined which of the many different approaches supersedes the others in performance. Attempts have been made by organising competitions at conferences and by providing benchmarks, but hard conclusions cannot be drawn at this point.

Besides the many similarities between the theoretical pickup and delivery transportation problems and their realistic counterparts, there are also two important differences that this chapter points out. First, in August 2006, news reported that the port in Rotterdam could not keep the pace with its competitors (Benneker, 2006; ANP, 2006). Besides lack of space the reason for this was a failing ICT. No matter how well the organisation is structured and how good the planners perform, overall performance depends on the weakest link. Therefore, one must also act with competence in case some (sub)systems are malfunctioning and pre-computed plans must be modified.

Second, the exact structure of the transport network is often abstracted from. Instead of constructing a detailed route for each vehicle, it is assumed that each location is con-

nected to each other location. Implementing this in practice results in a system that has no control over congestion. To minimize congestion, and to maximize throughput of the transport network, space and the individual routes of the vehicles must be modeled in more detail. Basic shortest-path algorithm to determine a shortest route from a source to a destination location, without taking into account other vehicles, do not suffice. The same is required to avoid collissions and guarantee safety constraints, such as a safety distance between vehicles.

2.1 Classical pickup and delivery problems

The general point of interest of this thesis is pickup and delivery transportation problems. In traditional pickup and delivery problems, vehicles have to transport freight from a source to a destination location without transshipment in any of the intermediate locations. Furthermore, it is assumed that all transportation tasks are known in advance.

The first problem to be discussed is the General Pickup and Delivery Problem (Savelsbergh and Sol, 1995). In the GPDP, a fleet of vehicles is given and each vehicles has a specified loading capacity, start location and end location. Furthermore, there is a set of transportation requests and each request consists of a source and destination location. A solution to the GPDP is a set of routes for the vehicles that satisfies the set of transortation requests without any transshipment.

Three specializations of the GPDP that have been extensively studied are the *Pickup and Delivery Problem* (PDP), the *Dial-a-Ride Problem* (DARP) and the *Vehicle Routing Problem* (VRP). In the PDP each transportation request specifies a single source location and a single destination location. Also, all vehicles depart from a single location and they have to return there - this location is referred to as the *depot*. The DARP is a PDP where the load of all transportation requests equals 1, namely people instead of freight is being transported (for example, the taxi-cab domain). Besides that, in the DARP it is obligatory that all transportation requests are executed, whereas for the other problems mentioned here this not always a hard constraint (Savelsbergh and Sol, 1995). The objective in the DARP is often modeled from the point of view of the customer (e.g., minimizing waiting time or travel time of the customer), as opposed to the point of view of the system (e.g., minimize the makespan or total fuel consumption) for the other problems. The VRP is a PDP with either all source locations or all destination locations equal to the depot.

Researchers like Cordeau and Laporte (2003); Cordeau et al. (2004, 2005); Sammarra et al. (2006); Berbeglia et al. (2006); Savelsbergh and Sol (1995) all provided definitions for the aforementioned problems, usually formulated as mixed-integer programming problems. Mixed-integer programming problems are linear programming problems where some of the variables must be integer. Although linear programming problems can be solved efficiently, mixed-integer programming problems cannot (unless

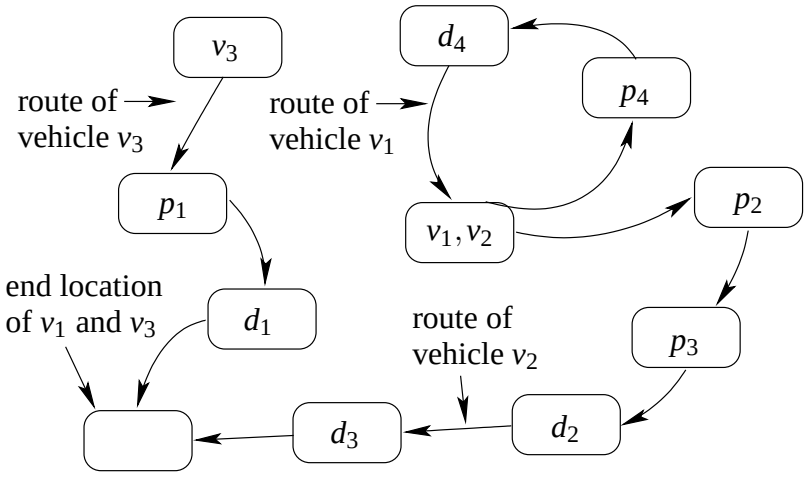


Figure 2.1: General Pickup and Delivery Problem (GPDP).

$P = NP$). All variants of the GPDP discussed in this section are known to be NP-hard.

In the GPDP tasks must be assigned to vehicles and routes must be computed for the vehicles that execute the tasks assigned to it. In this section the GPDP and variants are described in detail. Furthermore, mixed integer programs developed by Savelsbergh and Sol (1995) are also given, because in this form the GPDP and variants are usually encountered in literature and they give a detailed definition.

Remark 2.1 To understand the mixed integer programs, for the GPDP given in Figure 2.2, some notation needs to be introduced. Let N be the set of transportation requests to be executed. For each transportation request $i \in N$, a vector of load quantities $\overline{q}_i$ indicates what has to be transported from locations in the set of origins $N_i^+ \subset N$ to the set of destinations $N_i^- \subset N$. For origin $s \in N$ this quantity $q_{i,s}$ is a positive load to be transported to destination locations that occur at other elements of $\overline{q}_i$. It must always hold that $\sum_{j \in N_i^+} q_{i,j} = -\sum_{j \in N_i^-} -q_{i,j}$ (for each transportation request $i \in N$ the sum of loading quantities equals the sum of unloading quantities). The set $N^+ = \cup_{i \in N} N_i^+$ is defined as the set of all origins, and, likewise, the set $N^- = \cup_{i \in N} N_i^-$ for the set of all destinations. Their union forms the set $V = N^- \cup N^+$.

The set M is the set of all vehicles, where each vehicle $k \in M$ has a capacity $Q_k \in \mathbb{N}$, a start location k^+, and an end location k^-. $M^+ = \{k^+ : k \in M\}$ is the set of start locations of the vehicles and $M^- = \{k^- : k \in M\}$ the set of end locations. These together form the set $W = M^+ \cup M^-$. The transport network has vertex set $V \cup W$ and for all $i, j \in V \cup W$ the distance from i to j is denoted by d_{ij}, t_{ij} is the travel time, and c_{ij} the travel costs. Dwell times, such as the length of time cargo remains at a location before being loaded onto a ship or the amount of time a train spends in the station with its doors open, can be incorporated in these travel times.

The GPDP, as illustrated in Figure 2.1, is the problem of how to divide the transportation

requests in N over the available vehicles in M and to construct a route $Rt_k = (v_1, v_2, \ldots, v_n)$ for each vehicle $k \in M$, where $v_i \in V \cup W$. For each vehicle $k \in M$, this route must start in k^+ ($v_1 = k^+$), end in k^- ($v_n = k^-$), and it must visit all source and destination locations $N_i^+ \cup N_i^-$ of transportation request $i \in N$ assigned to this vehicle exactly once, and no other locations. Furthermore, the source locations of the transportation requests must obviously be visited before the destination, but also at no point along the route the capacity Q_k of the vehicle can be exceeded. Finding an arbitrary plan, i.e., a set of routes, that satisfies all of this is trivial, just start with an arbitrary partition of the transportation requests as an assignment to the vehicles and put these in an arbitrary sequence (source directly preceding destination). The difficult part is the optimization: to find a route for each of the vehicles such that, for instance, total cost is minimized or total profit is maximized.

Figure 2.2 illustrates the General Pickup and Delivery Problem formulated as a mathematical programming problem. Here, four additional variables play a role: assignment variable $z_i^k \in \{0,1\}$ equals 1 if transportation request $i \in N$ is assigned to vehicle $k \in M$, movement variable $x_{ij}^k \in \{0,1\}$ equals 1 if vehicle $k \in M$ travels from location $i \in V \cup k^+$ to location $j \in V \cup k^-$. The departure time at vertex $i \in V \cup W$ is specified by D_i and the the current load of the vehicle arriving at vertex i is represented by y_i.

Constraint 2.1, 2.2, and 2.12 correspond to the task assignment. Constraint 2.1 specifies that each transportation request is assigned to exactly one vehicle, Constraint 2.2 ensures that a vehicle only enters and leaves a vertex exactly once when the request is assigned to the vehicle ($z_i^k = 1$). Constraints 2.3, 2.4, and 2.5 specify the start location, start time and end location for the vehicles. Constraint 2.6 ensures that each pickup occurs before the corresponding delivery, Constraint 2.7 makes sure the traversal times are taken into account correctly. Constraint 2.8 says all vehicles start empty, Constraint 2.9 makes sure that vehicles are never overloaded, and Constraint 2.10 ensures that a vertex is completely processed, with respect to loading and unloading, when visited.

The objective function determines what is being optimized. For a single vehicle, there are objective functions that minimize the completion time, the travel time, travel distance, or customer inconvenience. For multiple vehicles, the number of vehicles used could be minimized or the summed profits can be maximized. The actual object function that is to be used depends heavily on the (real-life) problem that is being modeled.

As the name suggests, the General Pickup and Delivery Problem serves as a generalization of the Pickup and Delivery Problem, but also for the Vehicle Routing Problem and Dial-a-Ride Problem. All three of these problems can easily be defined as specializations of the GPDP. To start with, the *Pickup and Delivery Problem*, see Figure 2.3, is a GPDP in which each transportation request specifies a single source location, a single destination location, and all vehicles depart from and return to a special location called the *depot*. Such as depot represents the location where for instance all vehicles are parked and where all drivers start their day and return to in the evening. Figure 2.4 shows a mixed integer

Maximize objectives

Subject to

$$\sum_{k \in M} z_i^k = 1 \qquad \forall i \in N, \tag{2.1}$$

$$\sum_{j \in V \cup W} x_{lj}^k = \sum_{j \in V \cup W} x_{jl}^k = z_i^k \qquad \forall i \in N, l \in N_i^+ \cup N_i^-, k \in M, \tag{2.2}$$

$$\sum_{j \in V \cup \{k^-\}} x_{k^+ j}^k = 1 \qquad \forall k \in M, \tag{2.3}$$

$$\sum_{i \in V \cup \{k^+\}} x_{ik^-}^k = 1 \qquad \forall k \in M, \tag{2.4}$$

$$D_{k^+} = 0 \qquad \forall k \in M, \tag{2.5}$$

$$D_p \leqslant D_q \qquad \forall i \in N, p \in N_i^+, q \in N_i^-, \tag{2.6}$$

$$x_{ij}^k = 1 \Rightarrow D_i + t_{ij} \leqslant D_j \qquad \forall i, j \in V \cup W, k \in M, \tag{2.7}$$

$$y_{k^+} = 0 \qquad \forall k \in M, \tag{2.8}$$

$$y_l \leqslant \sum_{k \in M} Q_k z_i^k \qquad \forall i \in N, l \in N_i^+ \cup N_i^-, \tag{2.9}$$

$$x_{ij}^k = 1 \Rightarrow y_i + q_i = y_j \qquad \forall i, j \in V \cup W, k \in M, \tag{2.10}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall i, j \in V \cup W, k \in M, \tag{2.11}$$

$$z_i^k \in \{0,1\} \qquad \forall i \in N, k \in M, \tag{2.12}$$

$$D_i \geqslant 0 \qquad \forall i \in V \cup W, \tag{2.13}$$

$$y_i \geqslant 0 \qquad \forall i \in V \cup W. \tag{2.14}$$

Figure 2.2: Mixed integer program for the GPDP (Savelsbergh and Sol, 1995). Note that $x_{ij}^k = 1 \Rightarrow D_i + t_{ij} \leqslant D_j$ can be written as $x_{ij}^k (D_i + t_{ij}) \leqslant D_j$, but is written like this for clarity (same holds for Constraint 2.10).

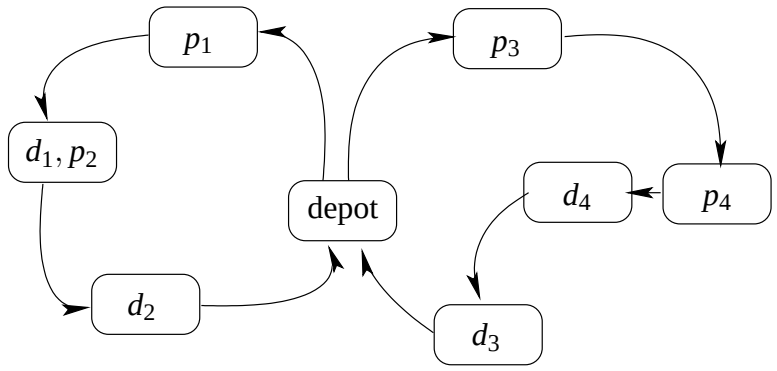


Figure 2.3: Pickup and Delivery Problem (PDP).

Maximize objectives
Subject to
All GPDP constraints, (2.15)
$|W| = 1$, (2.16)
$|N_i^+| = |N_i^-| = 1 \qquad \forall i \in N.$ (2.17)

Figure 2.4: Mixed integer program for the PDP (Savelsbergh and Sol, 1995).

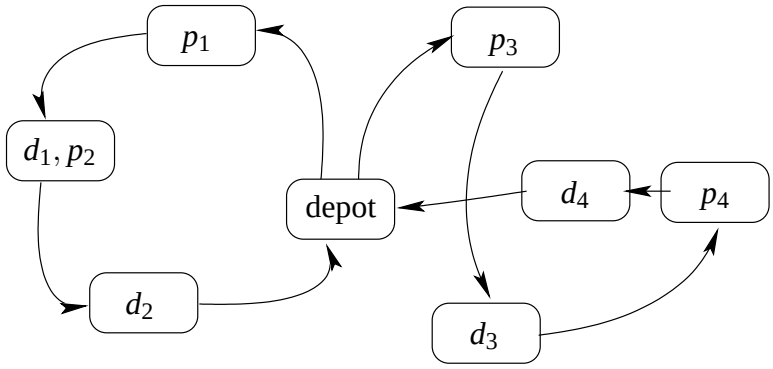


Figure 2.5: In the Dial-a-Ride Problem (DARP) the loads to be transported have a volume equal to 1 (passengers instead of freight).

program for the PDP in which Constraint 2.17 specifies that the number of pickup and the number of delivery locations equals 1 and Constraint 2.16 enforces all vehicles to start and end at a single special location called the depot.

The *Dial-a-Ride Problem*, see Figure 2.5, is a PDP in which all loads have a constant volume equal to 1. It arises in contexts where passengers are transported instead of freight. The DARP distinguishes itself from the GPDP by focusing on customer inconvenience.

Maximize objectives
Subject to
All GPDP constraints, (2.18)
$|W| = 1$, (2.19)
$|N_i^+| = |N_i^-| = 1 \qquad \forall i \in N,$ (2.20)
$q_{ij} = 1 \qquad \forall i \in N, j \in N_i^+ \cup N_i^-.$ (2.21)

Figure 2.6: Mixed integer program for the DARP (Savelsbergh and Sol, 1995).

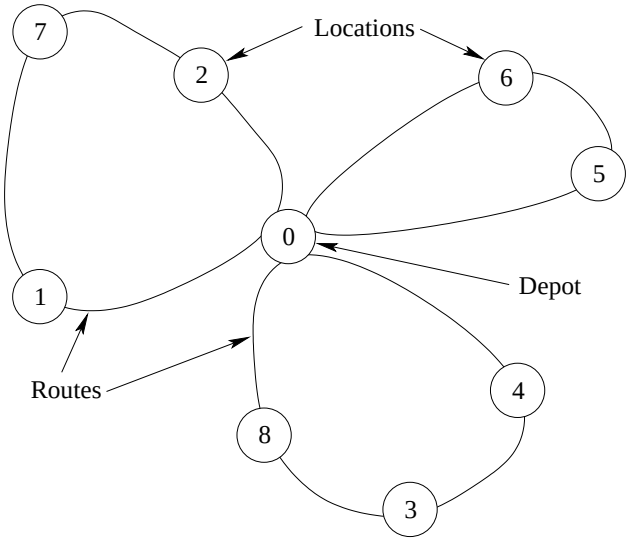


Figure 2.7: Vehicle Routing Problem with nine locations, among which one depot, and three vehicles. The solution is a set of three routes, one for each vehicle, such that each vehicle starts and ends in the depot and all locations are visited by exactly one vehicle.

$$
\begin{aligned}
&\text{Maximize objectives} && && \\
&\text{Subject to} && && \\
&\text{All GPDP constraints,} && && (2.22)\\
&|W| = 1, && && (2.23)\\
&|N_i^+| = |N_i^-| = 1 && \forall i \in N, && (2.24)\\
&N_i^+ = W \vee N_i^- = W && \forall i \in N. && (2.25)
\end{aligned}
$$

Figure 2.8: Mixed integer program for the VRP (Savelsbergh and Sol, 1995).

Often, additional constraints (or modifications to the optimization criterion) are present to limit *(i)* the amount of waiting time, *(ii)* the time spend by a customer in the vehicle, and *(iii)* deviations from desired departure or arrival times. Figure 2.6 shows a mixed integer program for the DARP very similar to the PDP, but with the additional Constraint 2.21 specifying that all loads have a volume equal to 1.

The *Vehicle Routing Problem*, see Figure 2.7, is a PDP in which either all origins of the transportation requests are the depot, or all destinations of the transportation requests are the depot. The VRP is very popular for instance for problems with supplying warehouses. The depot is then a distribution center from which a set of warehouses has to be supplied. Figure 2.8 shows a mixed integer program for the VRP in which Constraint 2.25 specifies that for all requests $i \in N$ either the pickup location N_i^+ or the destination location N_i^-

must be equal to the depot.

The pickup and delivery problems described above are thoroughly investigated by many researchers. They are so interesting because many real-life problems can be formulated using these pickup and delivery problem variants as a starting point; GPDP is a relaxed version of many real-life pickup and delivery problems. Furthermore, there exist many planning tools that can solve GPDP and variants and, therefore, it was possible to create benchmark sets that provide detailed comparison of the performance of these different planning tools. After considering a practical extension to GPDP, namely the concept of time, these planning tools will be described.

2.1.1 Time

In the early 1980s it was noticed that time played an important role in most practical pickup and delivery problems. Customers usually enforce, or at least prefer, the loading and unloading of freight to occur within so-called *time-windows*. A time-window is just an interval of time. Of course, the original problem definition already had *precedence constraints* stating that each pickup event must precede the corresponding delivery event in time, but these time-windows further restrict the loading and unloading events to take place in specified time-windows. Using these time-windows it became possible to model when packages to be transported were available, or when trucks were allowed to supply supermarkets (for example, parking space for the trucks might only be available during the morning).

A time-window can be *restrictive or non-restrictive* (Mitrović-Minić, 1998), which is also referred to *hard* versus *soft* constraints respectively. If time-windows are restrictive, then a plan that violates any of the given time-windows is not feasible. In that case, arriving at a customer before the time-window opens results in extra waiting time and arriving to late results in complete failure of the transportation request. In case of soft constraints, loading or unloading too early or too late results in less profit or a greater penalty for the responsible agent.

Mitrović-Minić (1998) mention that the notion of time can both complicate or simplify the problem. It complicates the problem, because finding a feasible plan becomes much more difficult. Without considering time any arbitrary assignment of transportation requests to vehicles with arbitrary routes would have been feasible. If the time-windows are *restrictive*, finding a feasible plan already is an NP-hard problem. On the other hand, there are cases where time simplifies the problem. Finding an optimal plan with restrictive time-windows can be easier, as due to all the time constraints, the search space becomes much smaller. The actual situation depends on the exact problem instance at hand, as both for soft and hard constraints the optimization problem is *NP*-hard.

2.1.2 Solution methods

The number of papers and even survey papers in the field of pickup and delivery transportation problems is enormous. The results of solution methods are not easy to compare. Not only because there are many problem variants, but also because there are different categories of solution approaches, each with their own goals. Furthermore, often the solution methods are considered too complex for thorough analysis and the quality of these methods is measured by running them on large and publicly available benchmark sets. These benchmark sets are very valuable, although unfortunately good comparison between solution methods seems non-existing. This section compares, as far as possible, exact approaches, heuristic and meta-heuristic approaches to pickup and delivery transportation problem variants.

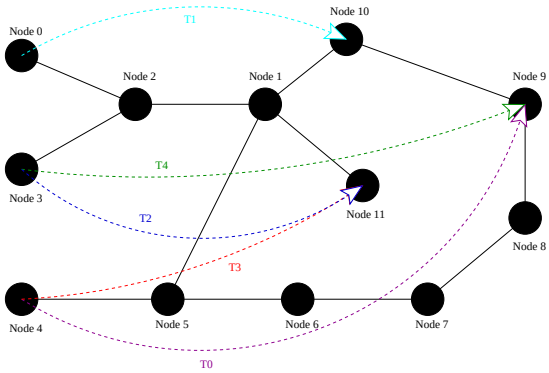
Exact approaches

Exact approaches are guaranteed to find the optimal solution. They search all possible alternatives except for those that could be proven not to be optimal.

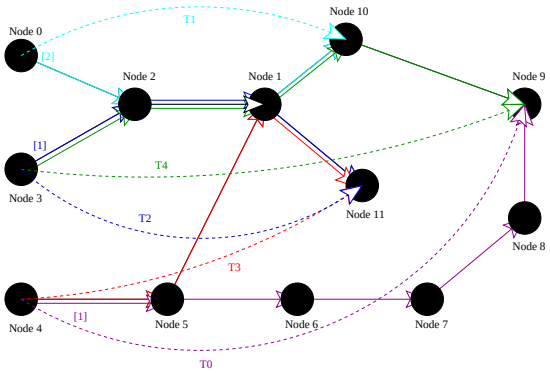
Psaraftis (1980) developed an $\mathcal{O}(|O|^2 3^{|O|})$ algorithm for the DARP, where each customer had to be served immediately, using forward dynamic programming, where $|O|$ is the number of transportation requests. Because it was written to serve each customer immediately, it can be used without modification for the PDP with a single vehicle. In 1983, this dynamic programming method was further developed such that it could also be applied to the DARP with time-windows (Psaraftis, 1983). Several years later the approach of Psaraftis was improved, when Desrosiers, Dumas and Soumis (1986) improved the forward dynamic programming by using an effective state elimination criterion, especially effective when the time-windows are tight and vehicle capacities small.

Desrosiers et al. (1986) used a branch and bound tree approach for the multiple vehicle PDP with time-windows. The branch and bound technique is also applied to the VRP by Fisher (1994). Later, Desrosiers et al. (1991) developed a new exact method based on the Dantzig-Wolfe decomposition or column generation, embedded in a branch-and-bound tree. Dantzig-Wolfe decomposition is a technique for solving large-scale linear programming problems. The idea of Desrosiers et al. was to use the strength of this decomposition technique to end up with a very small branch-and-bound tree.

There are also AI methods for solving GPDP, which were not directly developed with the intention to apply them to GPDP. The STRIPS and PDDL languages (see Appendix H) are able to represent problem instances in a variety of planning domains, among which a logistic domain called TRUCKS. There have been several editions of planning competitions (International Planning Competition (IPC) hosted at the ICAPS conference in 2007), where special-purpose and general-purpose planners compete with each other on several planning domains represented by STRIPS or PDDL. It is the aim of the IPC to analyse



(a) Problem instance consisting of five source-destination pairs for five vehicles.



(b) Conflict-free routes indicated by the arrows in the color of the corresponding transportation request. Numbers in square brackets denote waiting times.

Figure 2.9: SATPlanning package.

the current state of the art in planning for a variety of domains.

Among the IPC competitors was SATPLAN (Kautz and Selman, 1992, 1999) that finished first in 2004 and 2005 (together with MAXPLAN) in the optimal planning track. Kautz and Selman developed efficient reductions to Satisfiability and then used dedicated Satisfiability solvers to find optimal transportation plans. Figure 2.9 illustrates the SATPlanning package, developed by Broekens, Van Rantwijk, Zutt et al. in 2000, which follows the idea of Kautz and Selman. Figure 2.9(a), a PDP instances, where the transformation requests are already assigned to the vehicles, is transformed into a Satisfiability instance. This instances is solved by a dedicated Satisfiability solver and then transformed back into Figure 2.9(b). After personal communication Kautz commented it is hard to give numbers on the performance of these planners and, therefore, the performance is illustrated by showing the results of these planners on the logistic domain that was part of the IPC 2005.

Figure 2.10 illustrates the performance of the IPC 2005 competitors on the TRUCKS domain. In the TRUCKS domain, trucks have to move packages between locations under certain spatial constraints and delivering deadlines. Figure 2.10 shows the results of the competing planners on 30 different problem instances in the TRUCKS domain. On the left side are the planners that search for the optimal solution, on the right planners that aim for satisfying solutions[1]. The instances consist of 3 to 7 locations with 6 to 42 connections, 3 to 20 packages, and 3 to 7 trucks; from problem instance 1 to problem instances 30 they are increasing in size. The optimal planners only found solutions for the 10 smallest instances. The satisfying planners, although producing nearly-optimal solutions on these

[1]The planners tagged with ipc04 are reference planners, those were the winners of the IPC 2004.

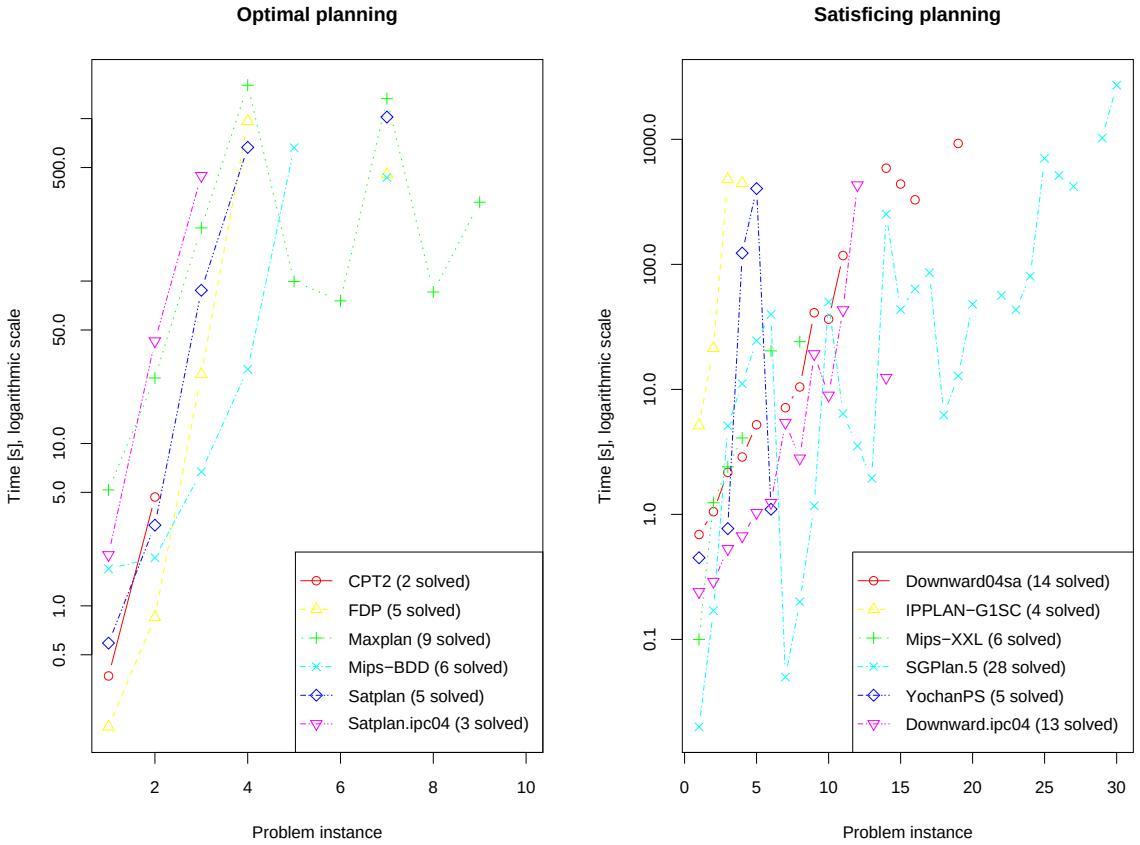


Figure 2.10: Results of the propositional TRUCKS domain at the International Planning Competition 2005 (IPC-2005).

instances, take quite a lot of time, which emphasizes the importance of special-purpose planners for logistic domains.

The general impression of using exact approaches for pickup and delivery problems is that they are able to solve problem instances with less than dozens of transportation requests. That is why many researchers chose to develop heuristic approaches that do not guarantee to find the best possible solution, but are able to find nearly-optimal solutions for problem instances with thousands of transportation requests.

Heuristic approaches

Heuristic approach are solution methods in which a *heuristic*[2] is used. A heuristic is a way to approximate the optimal solution, not by searching the complete search space, but considering only a part of it. For example, a heuristic can be that two transportation tasks that have to be pickup up (or delivered) at the same location are always assigned to

[2]The name heuristic originates from the famous exclamation “Eureka!” (I have found it!) of Archimedes.

the same vehicle, without further consideration. Such a heuristic obviously speeds up the algorithm, but not necessarily always find the optimal solution.

Heuristic approaches are again divided into two categories: regular heuristics and meta-heuristics. *Regular heuristics* are usually domain specific guides that direct the search towards a local optimum and then return the best encountered solution. *Meta-heuristics* are high-level methods that control the execution of a simpler heuristic, in a hopefully efficient way. They are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when it is not practical to implement such a method.

In the 1970s several heuristic approaches are developed, that can again be categorized in *decomposition*, *insertion*, and *local search* methods. An example of a decomposition heuristics is the two-phase method called cluster-first, route-second (Fisher and Jaikumar, 1981). This heuristic, for example, decomposes the problem by clustering locations that are within a circle (i.e., that are near to each other) and only then constructs a route along these locations. The strategy of insertion heuristics (Jaw et al., 1986) is to start with an empty plan, and then sequentially inserting new transportation requests that seem to fit well into the existing plan. Local search methods, studied by Savelsbergh (1990), start with suboptimal but feasible plans that are continuously improved until some local optimum is reached.

Meta-heuristic approaches

The following are examples of meta-heuristic approaches that have been successfully applied to pickup and delivery problems. It is not clear which of the meta-heuristic approaches performs best, but the largest known problem instances are solved due to meta-heuristic approaches. These instances contain thousands of transportation requests executed by dozens of vehicles. The exact results of different methods to well-known benchmarking sets (Solomon, Christofides, Elliot, and Toth) can be found on the VRP Web.

Tabu search Gendreau et al. (1998) and Gendreau et al. (1999) analyzed meta-heuristic approaches for both the dynamic VRP with time-windows and the dynamic VRP with time-window and pickup and delivery. Their strategy is to solve the dynamic problem as a sequence of static problems. The purpose of their papers is to test different heuristics with *Adaptive Tabu Search*. Tabu search is a local search method that examines the neighborhood in order to move to the best neighbor; solutions that were recently examined are forbidden, or tabu, for a certain number of iterations. Their conclusions were that *(i)* adaptive heuristics outperform others, *(ii)* the difference between adaptive and non-adaptive heuristics is bigger in less dynamic situations (otherwise there was not enough

computation time to find improvements), and *(iii)* the dynamic case has a more complex objective function than needed in the static case.

Simulated annealing Simulated annealing is based on the annealing of solids, the process to heat and then cool solids, which is usually done to soften them. It is a local search method developed in an attempt to improve local search when this gets stuck in a local optimum. Normally, for a candidate solution that is being investigated, only changes are allowed that move towards the local optimum (i.e., changes that improve the candidate solution). Simulated annealing does the same, but with a certain (small) probability, it allows the candidate solution to move into some (random) other direction. Chiang and Russell (1996) describes several approaches to VRP using simulated annealing.

Ant systems Ant Colony Optimization is a population-based, general search technique for the solution of difficult combinatorial problems which is inspired by the pheromone trail laying behavior of real ant colonies. The first Ant System was developed by Dorigo et al. (1991). It had encouraging initial results on the Traveling Salesman Problem, but was inferior to the state-of-the-art methods at that time.

Ant-based routing, more generally referred to as swarm-based routing, has been applied to load balancing in telecommunications networks by Schoonderwoerd et al. (1997), as a control mechanism for communications networks by Di Caro and Dorigo (1998), for combinatorial and continuous optimization by Rubinstein (1999), and later as route finding strategy, for example by Bjarne E. Helvik (2001) or by Di Caro et al. (2004). These systems are still in use and in active development.

Evolutionary algorithms Genetic algorithms have also been applied to the VRP. A genetic algorithm is an adaptive heuristic search method based on population genetics (Bräysy, 2001). Generation after generation new individuals, which are the candidate solutions usually represented by bit strings, are being created. In a recombination phase, different individuals are used to create new ones, by combining chromosomes of the parents, hopefully improving on the best solution so far. Furthermore, mutation randomly modifies genes of a single individual to ensure genetic diversity of the individuals. Evolutionary algorithms are still being developed, e.g., in the EvoVRP project (Pereira et al., 2002; Tavares et al., 2003).

Several exact and heuristic approaches to pickup and delivery problem variants have been described. All of these are still being further developed and it is not easy to compare their performance. Table 2.11, 2.12, and 2.13 show results presented at the VRP web. Here only the benchmarks are listed for which the corresponding solution methods are given. It seems that evolutionary algorithms perform best for these three benchmark sets.

Taillard instance (number of customers)	Best known solution [total distance driven]	Method
tai75a (75)	1618.36	Parallel iterative search heuristic
tai75c (75)	1291.01	Parallel iterative search heuristic
tai75d (75)	1365.42	Parallel iterative search heuristic
tai100a (100)	2041.34	Ant systems
tai100b (100)	1940.61	Evolutionary algorithms
tai100c (100)	1406.20	Ant systems
tai100d (100)	1581.25	Ant systems
tai150a (150)	3055.23	Parallel iterative search heuristic
tai150b (150)	2656.47	Ant systems
tai150c (150)	2341.84	Parallel iterative search heuristic
tai150d (150)	2645.39	Parallel iterative search heuristic
tai385 (385)	24431.44	Local search

Table 2.11: Benchmark results on the VRP for the Taillard instances, available at the VRP Web.

Li et al. (2005) provide another large-scale VRP benchmark in which their own heuristic method, variable-length neighbor list record-to-record travel, performs best followed by granular tabu search.

There are, however, some important shortcomings that have to be overcome for these methods to be useful in practice. On the one hand, congestion has been ignored. The routes of the vehicles are no more than a visiting sequence, i.e., an ordering of locations to be visited by the vehicles. When the exact routes are determined, unpleasant surprises might occur with respect to congestion. If many vehicles take a similar route, bottlenecks might arise in the transport network. On the other hand, so far it is assumed that all transportation tasks are known in advance. And also, the plans get executed exactly according to plan, without any failures (e.g., vehicle breakdown).

The following two sections cover these issues. The first is *context-aware routing*, in which the vehicles take into account the plans of other vehicles and, hence, advances the possibilities to avoid potential bottlenecks or congestion in the transport network. The second is *incident management*, which is about taking into account disturbances and failures (including changes to or newly arriving tasks). The above methods can still provide a useful starting point, i.e., an assignment of transportation requests to the vehicles and, for each vehicle, an ordering in which to execute the transportation requests. Other techniques must be used to maintain robustness and feasibility of these plans.

Note that conflicts are not incidents. Conflicts arise as predictable problems due to planning (or the execution thereof), while incidents are unpredictable events by definition.

Golden instance (number of customers)	Best known solution [total distance driven]	Method
1 (240)	5627.54	Evolutionary algorithms
2 (320)	8447.92	Tabu search
3 (400)	11036.23	Tabu search
4 (480)	13624.52	Tabu search
5 (200)	6460.98	Tabu search
6 (280)	8412.88	Tabu search
7 (360)	10195.56	Tabu search
8 (440)	11663.55	Evolutionary algorithms
9 (255)	583.39	Evolutionary algorithms
10 (323)	742.03	Local search
11 (399)	918.45	Evolutionary algorithms
12 (483)	1107.19	Evolutionary algorithms
13 (252)	859.11	Evolutionary algorithms
14 (320)	1081.31	Evolutionary algorithms
15 (396)	1345.23	Evolutionary algorithms
16 (480)	1622.69	Evolutionary algorithms
17 (240)	707.79	Evolutionary algorithms
18 (300)	998.73	Evolutionary algorithms
19 (360)	1366.86	Evolutionary algorithms
20 (420)	1821.15	Evolutionary algorithms

Table 2.12: Benchmark results on the VRP for the Golden and Van Breedam instances, available at the VRP Web.

This does not mean the techniques to deal with both do not overlap. In both cases, at the operational level, fast replanning decisions are necessary to recover the plans of the involved vehicles.

2.2 Conflict resolution

The classical problems together with their variants described in the previous sections consider the routes of the vehicles at an abstract level. In fact, only the order in which pickup and delivery locations (or customers in the Vehicle Routing Problem) are visited is considered a solution to the problem. If the ratio of the number of vehicles to the size of the transport network is relatively large, which is usually the case at AGV terminals for example, the traversal time for a certain route is highly influenced by the vehicle load along this route. In such cases a more detailed model of the infrastructure and the routes

Van Breedam instance (number of customers)	Best known solution [total distance driven]	Method
1 (100)	1106	Evolutionary algorithms
2 (100)	1506	Evolutionary algorithms
3 (100)	1751	Evolutionary algorithms
4 (100)	1470	Evolutionary algorithms
5 (100)	950	Evolutionary algorithms
6 (100)	969	Evolutionary algorithms
7 (100)	1032	Descent heuristic
8 (100)	1067	Descent heuristic
9 (100)	1690	Evolutionary algorithms
10 (100)	1026	Evolutionary algorithms
11 (100)	1028	Evolutionary algorithms
12 (100)	1616	Descent heuristic
13 (100)	983	Descent heuristic
14 (100)	2337	Descent heuristic
15 (100)	1083	Descent heuristic

Table 2.13: Benchmark results on the VRP for the Van Breedam instances, available at the VRP Web.

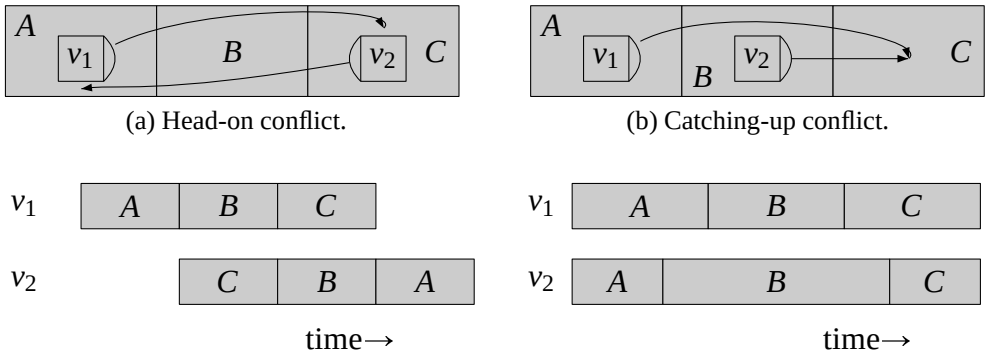
of the vehicles is required.

The sequel of this chapter focuses on planning for a fleet of vehicles and execution of these plans on transport networks, where the locations have limited capacities. The approaches are divided into three different categories: *(i)* approaches that prevent conflicts from occurring, *(ii)* approaches that solve conflicts during execution of the plans (or after a route has been chosen), and *(iii)* approaches that solve conflicts during the (route) planning phase. First, the next section describes reservations and the two main types of conflicts.

2.2.1 Reservations and conflicts

When agents created plans to execute their assigned transportation tasks, they would like to have a way to ensure that these plans remain feasible until the end. To make that possible, the other agents have to know about this plan and, hence, the agent has to make *reservations* for the resources in its plan. These reservations must be publicly available, but, if desired, they can be anonymous.

A situation where a plan cannot be executed together with the plans of other agents, due to safety constraints or other type of constraints, is denoted a *conflict*. The concept of conflict-free shortest-time AGV routing was first introduced by Broadbent et al.



(a) Head-on conflict.

(b) Catching-up conflict.

(c) Plans with head-on conflict in *B* and *C*.

(d) Plans with catching-up conflict in *B*.

Figure 2.14: Head-on and catching-up conflicts.

(1985). Kim and Tanchoco (1991) make a distinction between two type of conflicts, which is followed for instance by Maza and Castagna (2005): the *head-on* conflict and the *catching-up* conflict. In Figure 2.14(a) and 2.14(c) the head-on conflict is illustrated. This occurs when two agents plan to drive right through each other. In other words, when trying to execute their planning, a frontal collision occurs. Then, in Figure 2.14(b) and 2.14(d) the catching-up conflict is shown. This occurs when one agent overtakes another agent on the same lane. This cannot happen without the two agents occupy the same space at some point in time. Note that these conflicts are defined assuming that resources have unit capacity. Taghaboni and Tanchoco (1988) model bidirectional lanes as multiple single unidirectional lanes and hence eliminated head-on conflicts on lanes by design. Möhring et al. (2004) take the physical dimensions of the AGV into account: an AGV traveling along one arc may *spill over* onto a neighboring arc. To avoid conflicts, the authors associate a polygon with each arc to represent the area that an AGV uses when traveling along the arc, and they prohibit the simultaneous use of two arcs if their polygons intersect.

Agents that want to take into account other agents while planning can make use of context-aware planning methods. These methods can be categorized by the phase in which they detect and solve conflicts. First, methods are available to prevent conflicts by restraining the behavior of the agents. Second, one can ignore conflicts up to the latest moment at which the plans are being executed and solve conflicts as they occur. And third, there are methods that use look-ahead to solve future conflicts that can be detected by carefully examining the plans of the agents.

2.2.2 Preventing conflicts

If all agents are controlled by a centralistic program, this program can take into account all interactions between the agents and strive for the optimal plans for all agents. This is not

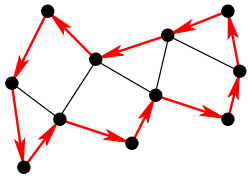


Figure 2.15: Hamiltonian cycle in a transport network. Agents have to move according to the red arrows and are not allowed to rest. Conflicts can never occur. Each (goal) location is reached in a number of steps smaller than the number of locations. There cannot be more agents than the length of the Hamiltonian cycle.

useful in practice, because such an approach is not scalable to bigger systems, it can only be used for toy problems. Another approach is to let each agent plan for itself only. These self-interested agents can deal with conflicts when they occur, e.g., by contacting a central unit for conflict resolution or by negotiating with the other agents that are involved.

Shoham and Tennenholtz (1995) argue that both of these approaches are undesirable when applied to context-aware routing. Either because centralistic approaches are not scalable, or because negotiation is a costly time-consuming process. They suggest the use of *social laws*, possibly combined with negotiation to solve conflicts, in the latter case as an attempt to minimize the amount of communication required.

Social laws are certain rules, such as traffic laws, that can be used to prevent any conflicts to occur in a multiple agent system at the cost of some performance. Also, social laws can prevent as much as possible, not necessarily all, conflicts while still gaining nearly optimal performance. Shoham and Tennenholtz (1995) applied this in the domain of mobile robots. An example they give is to construct an Hamiltonian cycle through the transport network, in their case an $n \times n$-grid, and a traffic law that forces all agents to follow this cycle and disallow them to rest. In that case, conflicts simply cannot occur (assuming they all travel at the same speed). The performance is not very well, however. It takes $\mathcal{O}(n^2)$ steps to reach a goal location on the $n \times n$-grid. Note that the rows or columns in these grid networks can, for example, model lanes in a supermarket. This traffic law can be applied to non-grid transport networks as well, see Figure 2.15.

After this simple example they propose a somewhat more complicated set of social laws that achieves a better performance on grid networks. This is done by putting a coarser grid on top of the original grid network. The coarser grid can then help to avoid having to follow a Hamiltonian cycle for a long time. For these traffic laws, given that there exists a plan of t steps in the system with only a single agent, the number of steps to reach a goal on the $n \times n$-grid is at most $t + 2n + o(n)$. The main drawback of this approach, however, is that it must be assumed that the number of agents m is fairly limited, i.e., $m = \mathcal{O}(\sqrt{n})$. For systems where $m = \mathcal{O}(n)$ Shoham and Tennenholtz (1995) present a final set of traffic laws that allows an agent to reach each arbitrary goal within $4n$ steps.

To conclude, social laws can really be helpful as a means to minimize the number of conflicts that need to be solved, but one can often not solely rely on them as that would imply a serious limitation on the number of agents allowed in the system.

2.2.3 Solving conflicts during execution

Another way of dealing with conflicts is to ignore them until the very last moment, namely until the plans are being executed and the conflict is present. The advantage of this approach is that a minimal amount of planning is required, as one simply does not take conflicts into account in the initial planning. The downside of this is that the quality of the resulting plans might be inferior. However, at least in incident-rich environments, it is questionable whether solving conflicts in advance is of any use. Later in this thesis experiments will provide empirical evidence with respect to these questions.

In this section two examples of approaches where conflicts are solved during execution are given. The first example is AgileFrames, where agents are controlled by static scripts that tell the agents what to do. The other example is a basic routing approach, based on a shortest-path algorithm. In this approach, the agents simply traverse a shortest path to their destination location, and solve conflicts along their ways as they occur.

2.2.3.1 AgileFrames

An example of an approach that solves conflicts during execution is *AgileFrames* (Lindeijer, 2003; Lindeijer and Evers, 1999; de Feijter, 2006), a logistic modeling framework that is designed with flexibility, adaptability, scalability and communication abilities in mind. Agents execute static scripts to move through a transport network modeled using resources. Such a script is a sequence of claims and releases of these resources. Conflicts are solved during execution, at the time they occur, and are ignored during planning.

The routing scripts are a small program that move an agent from a specified source to a specified destination resource. In case the scripts are static and non-preemptive (the agents cannot be interrupted while executing the script), it is not possible to ensure deadlock-free execution, unless there are special restrictions (for example, all agents follow a Hamiltonian cycle as described in Section 2.2.2).

AgileFrames is inspired by the airport of Amsterdam and new container terminals at the port of Rotterdam. It models the infrastructure with resources that have capacity. Agents are routed through this infrastructure by use of static scripts, that can be proven to ensure minimum safety distances between the agents.

AgileFrames consists of several components, namely Services, Traces and Forces. *Services*, an acronym of Service coordination and engineering system, defines the client-system interface and contains the logistic planning and scheduling. *Forces* provides means for the functioning, the control, and the feedback of the real operations. Although there

```
Ticket ticS = null;                  ▷ declares a Ticket variable reference.
ticS = new PrimT(S);                 ▷ creates a primal Ticket on semaphore S.
ticS.insist();          ▷ waiting request on ticket ticS, resembles Dijkstra's wait().
doA.exec();               ▷ initiates action for this actor and wait for its termination.
ticS.free();          ▷ return of the capacity, resembles Dijkstra's signal-operation.
```

Figure 2.16: Traces script fragment for a one-direction track, obtained from Lindeijer and Evers (1999).

thus is a special component for the logistic planning and scheduling, the bulk of work in the AgileFrames project is not on planning but on the final component named Traces.

Traces, short for traffic control and engineering system, concerns the control of conflicting use of shared facilities such as traffic infrastructure and is designed to meet the agility requirements and to handle high traffic intensities at any scale. Locations in the traffic infrastructure are modeled by resources and agents do not communicate with each other. Semaphores project the overload of these resources by agents.

In Traces, the routes of agents are programmed using scripts that are written in the Java language. See Figure 2.16 for an example. In this code fragment, semaphore *S* represents the freely available capacity of an infrastructure resource (e.g., a piece of a road). The ticket created in the script is a means to place a request on this semaphore.

The ticket in this example was a primitive ticket `PrimT`. That is a simple ticket just waiting for access on the semaphore. In Traces, there exists other types of tickets. For example `SelT`, a select-ticket. With a select-ticket, an agent can get access to one out of multiple different semaphores, whichever has free capacity available first. Also, there is a collective ticket `ColT`, which can be used to get credit from multiple semaphores simultaneously.

Priorities can be handled in different ways, the default being First-In-First-Out. An overview is given in Table 2.17. Furthermore, a semaphore guards multiple so-called *access lines*. Access lines are a means to prioritize tickets. Again, there are several mechanism to prioritize tickets. There is a `BasicSem` for the case a semaphore has only one single access line. There is a `RankedSem`, where the access lines are ordered and higher ranked access lines always take precedence over lower ranked access lines. A `CyclicSem` schedules the access lines in a cyclic way, which `CyclixSem` also does, but the latter makes sure access is exclusive, i.e., an agent only gets access when there are no agents on other access lines that got access to the semaphore already. Finally, there is a `VariSem` type of semaphore that can use user-programmed procedures.

AgileFrames claims to be more than a simulation tool. Its developers refer to it as an *operating system*. The argument is that it can control realistic agents in a real setting. The Traces component is rich and well documented. About the Services (especially referring to the planning and scheduling methods) and Forces components less information seems

Queue discipline	Explanation
First-In-First-Out	The first agent that claims, goes first.
Smallest claim first	Similar to *shortest processing time first* from production logistics.
Last-In-Last-Out	Similar to a stack, the last agent takes precedence.
Randomly selected	Uniform-randomly over all (actually, the first so many tickets as there was capacity remaining) tickets.

Table 2.17: Traces priorities.

to be available. Implementation in a laboratory demonstrates its ability to control real-life autonomous guided vehicles. Due to the absence of robust (re)planning and scheduling methods, a deadlock-free situation cannot be guaranteed without special restrictions to the transport network and the behavior of the agents. Also, no research has been done on its ability to recover from incidents.

2.2.3.2 Basic routing

If there were no other agents and no incidents, a rational agent would drive a shortest path from its current location to its destination location. Hence, the basis of route planning is a simple shortest-path planning algorithm. Like here, shortest-path planners are often used as part of a more complex question and must often be done in real-time. Although the variety of shortest-path planners usually provide the same results (i.e., they return a path of the same minimal length), it is important to make a well considered choice as this can significantly speed up a system, which is important for real-time applications.

Simple shortest-path algorithms The first stage in the traditional approach is to generate shortest paths along the pickup and delivery locations of the agent. To compute these paths the agent uses a basic shortest path algorithm, such as Dijkstra (1959). In this section, the best-first search algortihm A* is also described, becase it is more similar to the context-aware routing algorithm described later.

Algorithm 2.1 describes the A* algorithm. The A* is a well-known best-first search algorithm in Artificial Intelligence (Hart et al., 1968; Russell and Norvig, 1995). It maintains a priority queue Q that contains partial solutions. The queue is organized such that the best partial candidate solution is at front; hence the name *best-first*. The estimated value of a candidate solution is the sum of the costs of the partial path made so far and a heuristic function that estimates the costs required to complete the partial solution. This heuristic function ensures that the A* algorithm expands as few nodes as possible, making it the algorithm with the least number of iterations. Satisfying some restrictions on the heuristic, the A* algorithm is *complete* in the sense that if a solution exists, it will always

Algorithm 2.1 A* shortest-path algorithm.

1: **function** A*($v \in V, s, d \in N, t \in \mathbb{R}^+$)
2: **Pre:** Vehicle v, source s, destination d, and departure time t.
3: **Post:** Shortest route and schedule from s to d for vehicle v starting at time t.
4: $Q \leftarrow \{s\}$
5: $\forall n \in N \backslash \{s\} : l(n) \leftarrow \infty$
6: $l(s) \leftarrow t$ ▷ initialization
7: **while** $Q \neq \varnothing$ **do**
8: $n \leftarrow \text{argmin}_{q \in Q}\, l(q) + h(q)$ ▷ select most-promising candidate
9: $Q \leftarrow Q \backslash \{n\}$
10: **if** $n = d$ **then**
11: **return** route and schedule computed from labels ▷ ready
12: **end if**
13: **for all** $(n, n') \in \{(n, n'') \in E : l(n'') > l(n) + traveltime((,v,,)n, l(n))\}$ **do**
14: $l(n') \leftarrow l(n) + traveltime(v, n, l(n))$ ▷ expanding successors
15: $Q \leftarrow Q \cup \{n'\}$
16: **end for**
17: **end while**
18: **return** no route possible
19: **end function**

find the shortest (optimal) path from source to destination.

For basic shortest-path planning, the A* algorithm is not the best option. It requires fewer iterations than most other algorithms, however, the time required per iteration relatively long. Post (2004) presents an overview of several faster shortest-path algorithms. These algorithms do not maintain a sorted queue Q. Instead, they visit resources more than once, resulting in more, but faster, iterations. Zhan (1997); Zhan and Noon (1998, 2000); Cherkassky et al. (1994) take a closer look at the data types used to implement the algorithm, which is proven to be important. Finally, Cherkassky et al. (1994) notes that the graph representation used is also crucial to the performance of the algorithms. They showed that the best graph representation to use is Forward Star representation.

Algorithm 2.1 presents the A* algorithm, because it can best be compared to more advanced shortest-path algorithms later in this chapter. Function $traveltime((,v,,)n,t)$ computes the minimal traversal time for vehicle $v \in V$ to traverse location $n \in N$ when starting at time t. It takes into account the maximum allowed driving speed at this resource, the maximum driving speed of the transport resource and the distance of the infrastructure resource.

Operational conflict resolution After each agent computed a shortest path to execute its transportation requests, it is of course likely there are some conflicts. Sometimes, more

agents plan to claim a resource at the same time than is possible given the capacity of the resource.

To overcome this problem the basic routing approach uses operational conflict resolution. This form of conflict resolution is similar to the use of traffic rules, such as traffic arriving from right precedes, or intelligent traffic lights. Each time a conflict is detected during the operational phase, a heuristic is used to prioritize the agents for entering the popular infrastructure resource.

Many different alternatives are available to implement such resource usage rules. A valuable overview on scheduling problems by Morton and Pentico (1993) is used to support the selection of heuristics that is used in Chapter 5.

Morton and Pentico (1993) distinguish between (advanced) *dispatch* heuristics and *release* heuristics. Dispatch heuristics schedule forward in time at each choice point (when timing, routing is done, etc.) by calculating priority values according to some rule and the highest priority is chosen. *Advanced* denotes that due time problems and critical resources are forecasted and taken into account a priori. Their overview, though modified to the transportation domain, of dispatch heuristics includes:

- Critical ratio, uses the ratio of required lead time to current slack to determine the priority of a task, [3]
- (Weighted) COVERT[4], if the slack is much greater than the lead time, the priority is 0; it rises linearly, up to a certain maximum for this task, as the slack goes to zero,
- (Weighted) early/tardy, compares the (weighted) sum of earliness and tardiness of all tasks,
- SCHED-STAR, if the slack is negative, the job is sure to be tardy and has full priority; priority decreases exponentially with the number of average processing time lenghts available until its due time.

Executing tasks immediately when they arrive in the system is not always the best thing to do. It might improve the performance to release tasks somewhat later, as other tasks could arrive that should be dealt with first. Morton and Pentico (1993) acknowledge this and list the following release heuristics:

- Immediate release, provided as a benchmark,
- Average queue time release, releases a task to an agent at the due time of the task minus the amount of time the agent is idle,

[3] Lead time is the estimated time required to complete a task (e.g., deliver the freight) from the moment the task is known to the system. Hence, this time includes planning, waiting, etcetera.

[4] COVERT is short for Cost over Time.

- Queue length release, estimates a release time to an agent based on the number of tasks that are already assigned to this agent.

Garrido et al. (2000) describes the notion of *slack probability*, defined as:

$$P_s(O_i) = 1 - \frac{\delta_i}{\phi_i - \sigma_i + 1},$$

where the duration of operation O_i is δ_i, and its earliest finish time and earliest start time are σ_i and ϕ_i respectively. Note that slack is defined in Operational Research as $\phi_i - \sigma_i$, the amount of time the operation can be delayed without delaying anything else. If the operation has only one possible start time, then $\phi_i - \sigma_i$ is about equal to the duration δ_i and $P_s(O_i)$ is approximately 0. Otherwise, the more slack there is, the more $P_s(O_i)$ approaches to 1. An operation is more conflictive than another if its slack probability is minor.

2.2.4 Solving conflicts in advance

Another approach when it comes to context-aware planning and scheduling is the idea to transform the search for a conflict-free plan to Job Shop Scheduling with blocking (Hatzack and Nebel, 2001). In this approach, the problem is split into two phases. In the first phase, a route is determined for each agent. In the second phase, one by one the agents create a schedule, which determines the times at which the subsequent resources in the route are claimed. An appropriate delay is inserted into the schedule whenever a conflict is detected. Hence, conflicts are solved after determining the route (as opposed to the previously mentioned time-window graph routing method), but prior to the execution. An advantage of this approach is that heuristics known to perform well for Job Shop Scheduling with blocking can immediately be applied to finding conflict-free routes for a set of agents.

Also, *time-window graph routing* (Kim and Tanchoco, 1991) fits in this category. Time-window graph routing is an approach that integrates free-path routing (as opposed to fixed-path routing, where a fixed path is followed from source to destination) with conflict-resolution. The agents make public reservations for their plans on a first-come-first-served basis and, hence, performance (especially individual, but also for the total system) depends on the order in which the agents plan.

2.2.4.1 Two phases approach

The two phases approach separates the process of determining a route along the pickup and delivery locations and computing the schedule times, i.e., the times at which each of the locations will be visited. In the first phase, a route is determined for each agent. In the second phase, one by one the agents create a schedule that does not conflict with

previously computed schedules. At the end of the second phase, all agents have a conflict-free plan and, if the plans are executed exactly as specified, deadlocks cannot occur.

An important difference with the previous approach is that conflict-resolution is applied before the individual plans are executed. In fact, this conflict-resolution can be considered as a kind of plan repair, modifying individual plans if they are in conflict. For example, Broadbent et al. (1985) employ a simple shortest-path algorithm to find a set of initial routes. In case of catching-up conflicts, some agents are slowed down; for head-on conflicts, an alternative route is found that does not make use of the road at which the conflict occurred. Broadbent's algorithm can be used both on unidirectional and bidirectional infrastructures, but in the latter case it need not find the optimal solution.

The approach proposed by Hatzack and Nebel (2001) also can be considered as a two-phase approach to this problem. In the first phase, the individual routes are chosen, which they assume to be fixed. Then for the second phase, they pointed out a correspondence between computing conflict-free schedules for the agents and a particular scheduling variant called *Job Shop Scheduling with blocking*. In this second phase it is ensured that the constraints imposed by the resources are satisfied. To describe this correspondence the Job Shop Scheduling with blocking problem must be defined.

Scheduling is concerned with the optimal allocation of a set R of scarce resources to a set of activities (jobs) J over time. Each job $j \in J$ requires some specific set $R_j \subseteq R$ of resources and for each resource $r \in R_j$ the duration $t_{r,j}$ needed for j to use r is specified. Typically, each resource r can be used only by one job j at the same time. A solution to such a problem is a *schedule*, i.e., an allocation of intervals $[\sigma_{i,k}, \phi_{i,k})$ to each job $j_i \in J$ for using resource $r_{i,k}$ such that the constraints (non-overlapping and minimal duration) are satisfied. In a *job shop* scheduling problem each job j_i consists of a *sequence* of k_i operations $o_{i,1}, \ldots, o_{i,k_i}$, where operation $o_{i,j}$ needs resource $r_{i,j} \in R$ for $p_{i,j}$ time units, with $r_{i,j} \neq r_{i,j+1}$ for $i = 1, \ldots, k_{j-1}$. *Blocking* means that a job j continues to claim resource r after processing, if the next resource r' it needs is not available. During that time, no other job can use resource r. Definition 2.2 formally defines the Job shop scheduling with blocking problem.

Definition 2.2 (Job shop scheduling with blocking) Given a set J of jobs and a set R of resources, find a schedule, i.e., $[\sigma_{i,k}, \phi_{i,k})$ to each job $j_i \in J$ for using resource $r_{i,k}$ that is a solution to the following optimization problem.

Maximize objectives

Subject to

$$\sigma_{i,j} \geqslant \text{release\_time}(i,j), \tag{2.26}$$

$$\phi_{i,j} - \sigma_{i,j} \geqslant p_{i,j}, \tag{2.27}$$

$$\phi_{i,j} = \sigma_{i,j+1}, \tag{2.28}$$

$$\mu_{i,j} = \mu_{r,s} \Rightarrow o_{i,j} = o_{r,s} \vee [\sigma_{i,j}, \phi_{i,j}] \cap [\sigma_{r,s}, \phi_{r,s}] = \varnothing, \tag{2.29}$$

$$\mu_{i,j} = \mu_{r,s+1} \wedge \mu_{i,j+1} = \mu_{r,s} \Rightarrow \phi_{i,j} \neq \sigma_{r,s+1}. \tag{2.30}$$

In Definition 2.2 for Job Shop Scheduling with blocking Constraint 2.26 ensures that an operation is not scheduling before the job it belongs to is released. With Constraint 2.27 no operation $o_{i,j}$ is scheduling in an interval $[\sigma_{i,j}, \phi_{i,j}]$ smaller that its processing time $p_{i,j}$. Constraint 2.28 specifies that a job always claims a machine (the blocking property). Then the latter two constraints are to prevent conflicts. Constraint 2.29 ensures that two different operations scheduled on the same machine do not have overlapping time intervals and Constraint 2.30, which is according to Hatzack and Nebel a new constraint in scheduling literature, prevents a deadlock situation where two jobs with opposite machine routing face each other.

Now the similarity with the transportation problem discussed above is clear: let the jobs correspond to agents $a \in A$ that have to execute a route Rt_a as a sequence of operations. Each operation (r_i, t_i) in fact is a request for using the resource r_i during the time interval $[t_i, t_{i+1})$. A feasible conflict-free schedule is a *set of agent schedules* $\{Sd_a\}_{a \in A}$ that is conflict-free. Scheduling heuristics for job-shop scheduling problems, therefore, can be used to compute agent schedules in which resources are claimed by at most one agent at a time thereby avoiding any resource conflicts from a given set $\{Rt_a\}_{a \in A}$ of agent routes.

In their paper Hatzack and Nebel applied a fast delay minimizing heuristic to obtain such a set of agent schedules. This heuristic incrementally inserts jobs/operations in a first-come-first served manner into the schedule. Besides this makespan minimizing heuristic many other heuristics can be used, for instance one that considers the profits that come with the task.

From job shop scheduling to route planning

Besides the suggestion to transform the fleet routing problem to Job shop scheduling with blocking, Hatzack and Nebel also described Algorithm 2.2 as an example implementation of their idea. This fast job-shop scheduling heuristic computes a route and a schedule for an agent that takes into account the current reservations of all other agents. It is

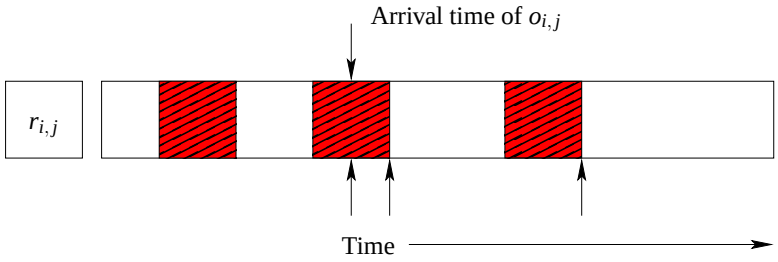


Figure 2.18: The set of potential start times $\Sigma_{i,j}$ considered for operation $o_{i,j}$ is the earliest possible time together with all later end times of reservations of other agents.

then applied to all agents sequentially, in first-come-first-served order (this is done by the Autocontroller). It is neither optimal (for the fleet of agents) nor polynomial-time (even for a single agent), but in their experiments the algorithm produced reasonably good results, and performed very efficiently.

The AutoController procedure in Algorithm 2.2 shows this sequential scheduling of agents. Procedure ScheduleActivity[5] is concerned with the scheduling of each individual agent. Figure 2.18 illustrates the start times that are attempted for each operation $o_{i,j}$ on resource $r_{i,j}$. The earliest possible start time $\sigma^*_{i,j}$ is the finish time $\phi_{i,j-1}$ of the preceding operation or the release time of job j_i if $j = 1$. Now, the set of finish times Φ_{μ_j} of all operations scheduled at machine μ_j, analogous to infrastructure resource μ_j in the transportation domain, can be defined as $\Phi_{\mu_j} = \{\phi_{r,s} : o_{r,s} \in S_{\mu_j}\}$, where S_{μ_j} is the schedule of all operations at machine μ_j. The set of starting times considered, as illustrated by Figure 2.18, is then $\Sigma_{i,j} = \{\sigma^*_{i,j}\} \cup \{\phi \in \Phi_{\mu_j} : \phi > \sigma^*_{i,j}\}$.

The predicate $Insertable(S, o_{i,j}, \sigma)$, used by Algorithm 2.2, is true if and only if *(i)* in the given schedule S no other agent has a reservation for resource $r_{i,j}$ that overlaps interval $[\sigma, \sigma + p_{i,j}]$, *(ii)* the agent can wait at the previous resource $r_{i,j-1}$ also during $[\phi_{i,j-1}, \sigma]$, and *(iii)* there is no head-on conflict with another agent (i.e., no exchange of machine with any other operation at time σ).

The last attempted start time is beyond the last reservation of all agents, which clearly indicates that the algorithm always terminates (that is, assuming that an agent can always wait in its current location). The idea is that the current operation $o_{i,j}$ is scheduled at time $\sigma_{i,j}$ and then the rest of the operations of job j_i are tried to be scheduled by using a recursive call. If this call succeeds, the procedure is done and returns success; otherwise, the next start time is attempted and a new recursive call is required.

No detailed results are presented in Hatzack and Nebel (2001). The authors claim it is suitable for very fast approximations, to do fast time simulations and the performance is similar to that of humans. Their concluding remarks include that, although the al-

[5]The original version of Line 15 of Algorithm 2.2 (Hatzack and Nebel, 2001) contains a typo that is corrected here.

Algorithm 2.2 Hatzack and Nebel's routing algorithm.

```
1:  procedure AUTOCONTROLLER(j_1, ..., j_n)
2:      Pre: (j_1, ..., j_n) is a sequence of jobs where j_i = (o_{i,1}, ..., o_{i,k_i}).
3:      Post: All jobs in (j_1, ..., j_n) are scheduled and free of conflicts.
4:      S ← ∅
5:      for all j_i ∈ {j_1, ..., j_n} do
6:          SCHEDULEACTIVITY(S, o_{i,1})
7:      end for
8:      return S                                   ▷ returns feasible schedule S
9:  end procedure

10: procedure SCHEDULEACTIVITY(S, o_{i,j})
11:     Pre: Up to operation o_{i,j-1} schedule S is a conflict-free schedule.
12:     Post: Schedules operation o_{i,j} and beyond into schedule S.
13:     if j ⩽ k_i then
14:         inserted = false
15:         Σ_{i,j} ← {σ*_{i,j}} ∪ {φ ∈ Φ_{μ_{i,j}} | φ > σ*_{i,j}}   ▷ compute potential start times for o_{i,j}
16:         while Σ_{i,j} ≠ ∅ ∧ ¬inserted do
17:             σ ← min Σ_{i,j}                    ▷ get next potential start time
18:             if INSERTABLE(S, o_{i,j}, σ) then
19:                 σ_{i,j} ← σ; φ_{i,j} ← σ_{i,j} + τ_{i,j}   ▷ assign start/end time to o_{i,j}
20:                 if j > 1 then
21:                     φ_{i,j-1} ← σ_{i,j}        ▷ adapt end time of preceeding operation
22:                 end if
23:                 inserted ← SCHEDULEACTIVITY(S, o_{i,j+1})   ▷ continue recursion
24:                 if ¬inserted then
25:                     S ← S\{o_{i,j}}
26:                     if j > 1 then
27:                         φ_{i,j-1} ← σ_{i,j-1} + τ_{i,j-1}   ▷ reset end of preceeding operation
28:                     end if
29:                 end if
30:             end if
31:         end while
32:     else
33:         inserted ← true                        ▷ last operation of task j_i has been inserted
34:     end if
35:     return inserted
36: end procedure
```

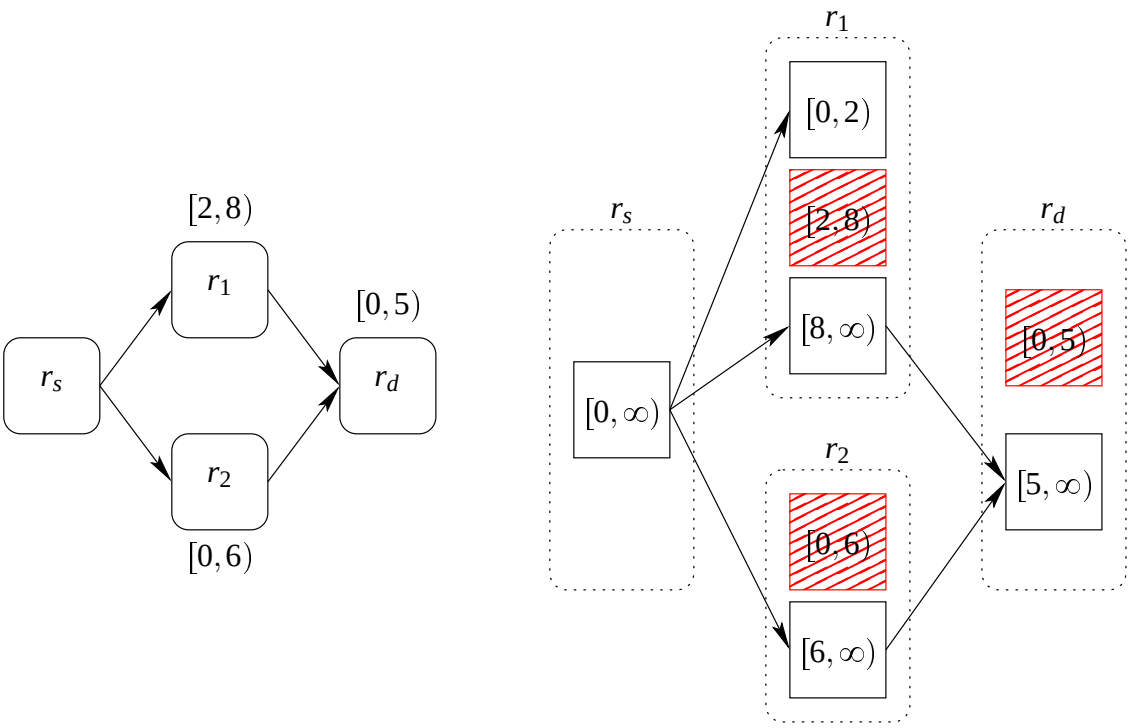
gorithm involves backtracking, it hardly never occurs. The algorithm can however be shown to have exponential time complexity. The authors have been asked for their data or implementation, in order to reproduce their experiments. However, due to disclosure agreements they were not able to publish the problem instances they used nor their exact implementation.

2.2.4.2 Context-aware routing

There are also approaches that aim at the integration of the route planning and the conflict-resolution process. *Context awareness* refers to the fact that an agent has to be aware of the consequences of the route planning by other agents since his individually optimal route choice might be seriously affected by the route choices of other agents.

Typically, context-aware routing approaches consider all possible routes from a source to a destination location while considering reservations of other vehicles. For example, the algorithm proposed by Huang et al. (1993) finds a path through the (graph of) free time-windows on the resources, rather than directly through the graph of resources. Huang's algorithm is optimal both for unidirectional and bidirectional networks, but it assumes unit capacity for all resources. Fujii et al. (1989) combine the search through free time-windows with a heuristic that calculates the shortest path from the current resource to the destination resource, assuming no other traffic. The solution method proposed should result in an optimal, polynomial-time algorithm, but the description of the algorithm is not entirely correct. Additionally, the authors do not provide any complexity analysis of the algorithm. The work of Kim and Tanchoco (1991) is similar to the work of Fujii et al., but their treatment of the problem and the analysis of their algorithm is more comprehensive. Kim and Tanchoco's algorithm finds the (individually) optimal solution for both uni- and bidirectional networks, and they give an $\mathcal{O}(n^4v^2)$ time complexity for their algorithm, where n is the number of agents in the system, and v is the number of resources in the infrastructure network. Due to this relatively high run-time complexity (especially given the limited computational resources in the early 1990s), Taghaboni-Dutta and Tanchoco (1995) developed an approximation algorithm that decides at every intersection to which resource to go next, based on the estimated traffic density of the resources from the current intersection to the destination. The authors show a small loss of plan quality, but they claim that the algorithm consumes significantly fewer computational resources; however, they do not quantify the run-time complexity of the approximation algorithm, nor do they present any CPU cost comparisons.

The idea of time-window graph routing is, instead of routing through locations in a graph as a basic shortest-path algorithm would, to route through a free time-window graph (see Figure 2.19). For each location, a set of disjoint free time-windows is computed from all known reservations that exactly specifies the time-windows at which the load of the



(a) Transport network. Arcs represent connections between locations. The time intervals are reservations of other agents.

(b) Free time-window graph. The red intervals are reservations by other agents. Connections specify the reachability relation between free time-windows.

Figure 2.19: Transport network and free time-window graph.

location is smaller than its capacity. Furthermore, a reachability relation must be defined that specifies which free time-windows at one location can be reached from which free time-windows at another location (similar to arcs in a normal graph). Applying a basic shortest-path algorithm on this time-window graph results in a context-aware plan for the original instance. This plan is optimal, given the reservations of all other agents do not change.

Let us look at the example depicted in Figure 2.19. The task is to route from source location r_s to destination location r_d starting at time $t = 0$, either by traveling via location r_1 or via location r_2 and taking into account the specified reservations of other agents. At first sight, it seems quicker to traverse via location r_1, because location r_2 is already reserved up to time $t = 6$. However, the journey cannot continue then because of the reservation $[0,5)$ in location r_d and the agent is not allowed to wait in location r_1 due to the reservation $[2,8)$ over there and, hence, the agent must wait in resource r_s until time $t = 8$ if it desires to travel the upper route. The free time-window graph gives somewhat more information. There is only an arc from location r_1 to location r_d from the free time-window $[8,\infty)$, so one can immediately infer that the upper route has costs greater than 8

and the lower route is quicker (assuming the all locations have equal traversal times). The connection from r_s to the first free time-window $[0,2)$ in r_1 is in fact useless (and can be discarded) if this free time-window has no outgoing arcs, except if location r_1 is the goal location.

This context-aware method can be used by all agents in a sequence resulting in a set of conflict-free plans. That this leads to sub-optimal performance for the total system is no surprise, as the final plans depend on the order in which the agents planned; in general, planning earlier leads to a more efficient plan for each individual agent, but the method does not specify in which order the agents should plan to optimize the performance of the total system. Of course, there exist methods that attempt to improve on this arbitrary ordering of when agents plan. These methods can also benefit from using time-window graph routing to quickly create a conflict-free plan.

The execution of transportation plans often differs from the constructed plans. This is because of unforeseen events, such as container ships arriving late, orders being cancelled, or modeling inaccuracies. The next section deals with these unforeseen events, which we call *incidents*.

2.3 Incident management

An event that can potentially render a current plan infeasible, because it was not anticipated into advance usually due to a malfunctioning system or faulty component of this system, is referred to as an *incident*.

The event of a system or some of its components being faulty for a period of time (the repair time) is referred to as an *incident*. *Incident management* is a field of research that attempts to take into account incidents and develops systems where performance degrades as few as possible when the number of incidents increases.

In this section a nice experiment by Beamon (1998c) is presented that illustrates why reliability must be taken into account in early stages (already in the design phase of a system). The term *performability* is used to denote that performance and reliability are measured simultaneously. After the necessity of performability is illustrated, known methods are described to deal with incidents.

2.3.1 Performability

Research on the performance of AGV systems often had the underlying assumption that all the components are going to last life long. In reality, these components are not completely reliable and are subject to failures over a period of time. To give an example, road intersections generally reduce the reliability of the system by adding potential sources of

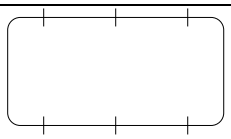
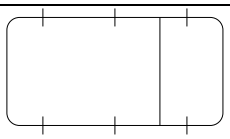
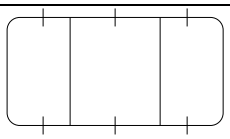
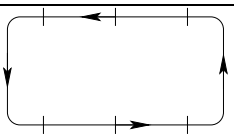
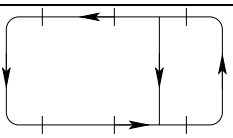
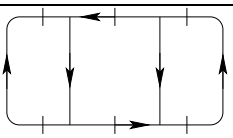
	(A) No shortcuts	(B) One shortcut	(C) Two shortcuts
Layout			
Flow Path Optimization Model			
TTD	544	488	484
FTP	8.7%	10.9%	14.0%
FDU Index	63.1	62.7	80.3

Table 2.20: More shortcuts do lead to a decrease in total travel distance (TTD). However, the unreliability measure (FTP) has increased due to the additional cross-overs. The flow-distance-unreliability index (FDU) is minimized for layout B (Beamon, 1998c).

failure. The reason for this is that guide path requires some mechanical branching to enable vehicles to choose their destination and the fact that intersections allow for collisions to occur. Studies have shown that incorporating reliability analysis in the design of guide paths increases the performance of the system (Beamon, 1998b,c,a).

Beamon (1998c) describes guide paths for an AGV system, see Table 2.20, where layout A has no shortcuts, layout B has a single shortcut, and layout C with two shortcuts. The second row of the table depicts the optimal path directions based on the Flow Path Optimization Model (Kaspi and Tanchoco, 1990). Note that the exact set of pickup and delivery requests together with the used distances of the edges in the network are also mentioned in Beamon (1998c), but are omitted here.

The total travel distance (TTD) is a lower bound on the total travel costs computed as the sum of distances from pickup to delivery of all transportation requests. The total flow that is not delivered correctly is given by the failure trip percentage (FTP), which is defined the percentage of flow from pickup to delivery that fails due to unreliable components in the infrastructure.

The total travel distance is minimized on layout C with two shortcuts. At the same time, the unreliability measure is maximized for this guide path. Hence there is a trade-off between reliability and the objective function. Beamon (1998c) suggests the use a flow-distance-unreliability index (FDU) that is minimized for layout B in the example. The flow-distance-unreliability index is defined as the sum of the flow, distance and unreliability measure for all pickup and delivery pairs.

2.3.2 Incidents

There are several different sources of incidents to be distinguished. Incidents can refer to failure, such as communication failure between AGVs and a central planning system, break-down of a mobile entity (engine failure) or failures in the transport network (e.g., due to traffic accidents). Other incidents refer to modifications of transportation requests. Even the arrival of new transportation requests can very well be regarded as incidents, because the current planning is rendered infeasible for not taking into account these new transportation requests. In an unpublished manuscript, Davenport and Beck (2000) describe a similar list of causes for uncertainty.

Incidents that refer to failure are normally represented by *(i)* the malfunctioning object, *(ii)* an interval of time specifying the time of failure and the repair time, and *(iii)* the impact or severity of the incident. For mobile entities the severity can be represented by a percentage denoting the percentage of the normal traversal speed that is in effect during the incident.

Modification or new transportation requests are usually taken into account by using online planners, i.e., planning systems that have a certain dynamism that enables them to react to these changes and adapt the plans accordingly. Failing communication limits the ability to cooperate and assign or re-assign new transportation requests. Sometimes a decrease of speed of the mobile entities is also enforced during these communication incidents.

2.3.3 Incident management methods

Due to the experiment of Beamon flow path optimization models were adjusted to take into account reliability during guide-path design. But also during planning and execution, methods have been developed to take into account disturbances. The real world is not so stable, many disruption and task modifications occur, leading to the necessity of incident management methods. Incident management methods can be distinguished into *pro-active* and *reactive* methods. Pro-active methods attempt to create robust schedules, while reactive methods recover from incidents at the moment they occur.

One pro-active approach to incident management is to generate robust schedules that are able to absorb a certain amount of disruptions without the need for replanning. Gao (1995) describes a technique called *temporal protection*. For each location and vehicle historical statistics are maintained about its reliability. Then, according to this data the duration of actions, such as drive, load, etc., are extended with some temporal slack.

Davenport et al. (2001) developed a superior method called *time-window slack* or *focused time-window slack*. Instead of hiding the temporal slack in the durations of actions the time-window slack method modifies the problem definition a little, such that the scheduling algorithm can reason about the slack. For instance, the scheduling can

sometimes be more flexible by shifting the temporal slack here and there. For the focused time-window slack method, the temporal slack depends on when the activity is scheduled. Activities that are scheduled when the probability of an incident is greater, e.g., approaching the wear-out period, get more slack added.

Van der Krogt (2005) presents reactive plan repair methods for both single-agent and multi-agent systems. His focus was on repairing plans without computing them from scratch after an incident occurs and requiring as few changes as possible such that commitments to other parties are little affected. After extending the Action Resource Formalism (ARF, see de Weerdt et al. (2003a)) with gaps and incidents (a gap in a plan can result from an incident), refinement planning approaches are surveyed and developed. This approach makes use of a library with plans. At the moment a plan rendered infeasible, there is a plan with a gap. The plan library is searched for elements that can be used to fill up the gap.

Another example of reactive methods can be found in the field of robot path planning. Stentz (1994); Koenig and Likhachev (2002) worked on robot path planning in partially unknown environments. The *Lifelong Planning A** and *D** variants are similar to the well-known best-first search A* algorithm, but try to improve on this by avoiding having to start from scratch when a small disturbance occurs. For example, while a robot approaches towards its goal location it obtains new information from sensing its local environment. This data might change the travel costs (e.g., suddenly it detects a wall). In that case, a basic shortest-path algorithm would have to restart from scratch. The D* or LPA* algorithms are able to locally propagate this change in costs, using stored data from the previous computation, and are potentially faster in computing the new optimal shortest path.

This section described the importance of taking reliability into account, the classical approaches to pickup and delivery problems do not suffice in practice. There is a trade-off in costs between having a reliable operation with sub-optimal performance (when there are no failures) and an unreliable operation that is cheap, but where performance degrades in case of incidents. At the very least, system designers have to be aware that there often are components in the system that can be malfunctioning at a certain point in time. Several aforementioned techniques for taking incidents into account are available for this purpose.

2.4 Summary

This chapter presented an overview on classical pickup and delivery transportation problems and described several of the many different solution techniques. While evaluating these techniques, it becomes apparent that there are two important problems that are not covered.

First, the classical approaches often abstract from route planning by using a distance

matrix, which specifies the distance between all pairs of locations. The context-aware approach models the transport network in greater detail, such that bottlenecks in the transport network can be identified (many reservations for the same resource) and congestion can be minimized. But the existing context-aware approaches are computationally expensive and, hence, cannot be applied to large systems. We will improve on this aspect of context-aware routing as well as its flexibility.

Second, it is usually assumed the components of a system function flawlessly. Section 2.3 elaborates on the importance of taking into account reliability and robustness. In this thesis, incidents play an important role.

The next chapter describes a framework for pickup and delivery transportation, which also includes non unit-capacity resources and, hence, a more general definition of a conflict.

Chapter 3

A framework for multi-agent transport planning

This chapter proposes a new framework for pickup and delivery transportation that can be applied when mobile entities carry out transportation requests. Transportation plans have to be constructed to ensure that these requests are correctly and efficiently executed and their deadlines are met.

Usually, there is a common transportation network where these plans are executed. Due to limited capacities of this network, these individual transportation plans might interfere with each other. An example application of the framework presented in this chapter is modern material handling systems, which make more and more use of autonomous guided vehicles in manufacturing plants, warehouses, distribution centers, and terminals.

The framework distinguishes transportation agents and infrastructure agents. Transportation agents make transportation plans, while infrastructure agents make reservation plans for infrastructure resources (lanes, crossings). In making their transportation plans, transportation agents query infrastructure agents about the availability of parts of the infrastructure they need. This extends the approach of Kim and Tanchoco (1991) by allow-

ing infrastructure agents to use other reservation policies than a simple first-come-first-serve policy and, e.g., to take into account priorities of agents. Moreover, this approach can also be used to deal with incidents by informing transportation agents about delays and unavailability of parts of the infrastructure, enabling them to replan their routes.

This chapter is organized as follows. At the beginning the model is described, which consists of the transport network, infrastructure and transport resources (the mobile entities), requests, plans, and incidents. Then, the invariants of this model are described, followed by the requirements that define when a transportation plan is feasible. After that, a set of mobile entities is considered and the exact meaning of a conflict in our framework is introduced. Subsequently, a section is devoted to agents. The responsibilities of the infrastructure agents and transport agents are described. Finally, performance indicators are presented that can be used to measure the performance of the mobile entities (i.e., the quality of the transportation plans) and the system as a whole.

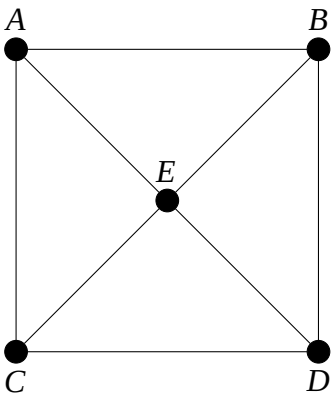
3.1 Ingredients

In this section the main ingredients of the transportation model are described. First, the *transport network* is modeled using *infrastructure resources* that have several properties (e.g., capacity, distance, maximum speed). Another type of resources are *transport resources*. Transport resources, which have a maximum driving speed and loading capacity, are the mobile entities moving around though the transport network. Together the infrastructure resources and transport resources form the set of resources in the model.

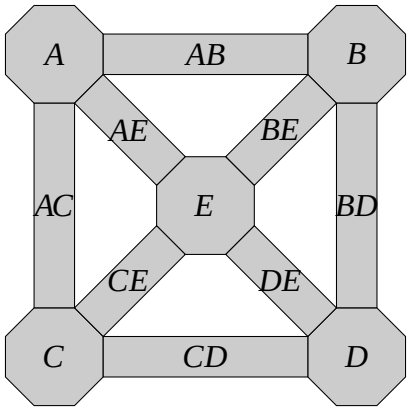
Second, there are the transportation *requests* (also referred to as *tasks*) that represent a customer request for transporting a freight (or perhaps a passenger) from a source to a destination location. The customer specifies a time-window (i.e., an interval in time) for both the pick-up and the delivery event. Furthermore, a reward function is given for each transportation request, that defines the reward for the responsible agent. If the agent succeeds in executing the pick-up and delivery event inside the specified time-windows, the reward for this agent is typically maximized. Violation of one of these time-windows decreases the reward as specified by the reward function.

Third, an important aspect of our model are *incidents*. There can be many different types of incidents, among others, customers that change or retract transportation requests, unexpected traffic jams (predictable traffic jams are not considered to be incidents, because they can be taken into account during planning), vehicle break-down, communication failure, etc. In the model it is specified what types of incidents are considered in this thesis.

The following section describes the transport network and the infrastructure and transport resources.



(a) Graph representation using nodes and edges. $G = (V,E)$ where $V = \{A,B,C,D,E\}$ and $E = \{\{A,B\}, \{A,C\}, \{B,D\}, \{C,D\}, \{A,E\}, \{B,E\}, \{C,E\}, \{D,E\}\}$.



(b) A resource for each node and edge. $R = \{A, B, C, D, E, AB, AC, BD, CD, AE, BE, CE, DE\}$ and $E = \{\{A, AB\}, \{A, AC\}, \{A, AE\}, \{B, BD\}, \{B, BE\}, \{C, CD\}, \{C, CE\}, \{D, DE\}, \{E, AE\}, \{E, BE\}, \{E, CE\}, \{E, DE\}\}$.

Figure 3.1: Graph representation versus resource-based representation of a transport network.

3.1.1 Transport network and resources

Following Hatzack and Nebel (2001) we make use of a non-classic resource-based graph representation of the transport network by using infrastructure resources. This simplifies, for example, the modeling of intersections. Infrastructure resources represent roads, road segments, part of an intersection, parking space, etc. Figure 3.1a illustrates a classical graph representation using a graph $G = (V,E)$ of nodes V and edges E. Figure 3.1b shows the resource-based representation $G_R = (R^{inf}, E_R)$ with locations R^{inf} and connections E_R. Each node $v \in V$ and each edge $e \in E$ correspond to a resource r_v and r_e in the resource-based representation. Furthermore, for each edge $e = \{v_1, v_2\} \in E$ the resource-based network has connections $\{r_{v_1}, r_e\}, \{r_e, r_{v_2}\} \in E_R$. In the resource-based representation a vehicle always resides in the infrastructure resource – representing space – that it occupies. These resources have several properties like travel distance, maximum allowed traversal speed, maximum load, etc, while the edges just define the adjacency relation and have no properties of its own.

The *transport network*, or infrastructure, represents how the mobile entities (the transport resources) can move around. Transport network $I = (R, E_R, k^{inf}, k^{tr}, d^{inf}, s^{inf}, s^{tr})$ is a tuple consisting of a set of resources R, a directed connectivity relation E_R (defining which resources are neighbors), capacity functions k^{tr} and k^{inf}, distance function d^{inf}, and maximum speed functions s^{tr} and s^{inf}.

The set $R = R^{inf} \cup R^{tr}$ of resources is decomposed into the set of *infrastructure* re-

sources R^{inf} and the set of *transport* resources R^{tr}. The infrastructure resources represent space that can be occupied by the transport resources. A transport resource $v \in R^{tr}$ represents a mobile entity, e.g., a vehicle that can move around through the transport network. The directed *connectivity* relation $E_R \subseteq R^{inf} \times R^{inf}$ defines which infrastructure resources a transport resource can traverse to from a given infrastructure resource. The *distance* function $d^{inf} : R^{inf} \to \mathbb{R}^+$ gives, for each infrastructure resource $r \in R^{inf}$, the positive non-zero distance $d^{inf}(r) > 0$ it takes to traverse infrastructure resource r.

For all infrastructure resources $r \in R^{inf}$, the *capacity* function $k^{inf} : R^{inf} \to \mathbb{N}$ specifies the number $k^{inf}(r)$ of transport resources that can use resource r simultaneously. In other words, for each infrastructure resource $r \in R^{inf}$, at any point in time $k^{inf}(r)$ is the maximum number of transport resources in resource r. For transport resources, there is a *capacity* function $k^{inf} : R^{tr} \to \mathbb{N}$, where $k^{inf}(r)$ is the load capacity of resource $r \in R^{tr}$ in terms of freight (see the next section).

The *Speed* function $s^{inf} : R^{inf} \to \mathbb{R}$ specifies that $s^{inf}(r)$ is the maximum possible driving speed at infrastructure resource $r \in R^{inf}$ irrelevant to which transport resource is traversing infrastructure resource r. For a transport resource $v \in R^{tr}$, the *speed* function[1] $s^{tr} : R^{tr} \to \mathbb{R}$ specifies the maximum driving speed $s^{tr}(v)$.

In the following section the transportation requests, which form the workload for the system, is described.

3.1.2 Transportation requests

Transportation *requests* (also refered to as orders or *tasks*) represent the workload for the system. The set of transportation requests is denoted by O. Each order $o_j \in O$ is a six-tuple $o_j = (f_j, s_j, \tau_j^s, d_j, \tau_j^d, \pi_j)$ denoting the request to pick up freight $f_j \in F$ (or perhaps a passenger) with volume $vol(f_j) \in \mathbb{N}$ at a certain source location $s_j \in R^{inf}$ within a specified time-window $\tau_j^s = [t_{j,1}^s, t_{j,2}^s]$ and to deliver it at a specified destination location $d_j \in R^{inf}$ within a time-window $\tau_j^d = [t_{j,1}^d, t_{j,2}^d]$.

The lowerbound and upperbound of a time-window τ, as well as other time points in our model, are modeled as real values in $\mathbb{R}$ extended with positive and negative infinity. The set $T = \mathbb{R} \cup \{-\infty, \infty\}$ contains all possible time points. Positive infinity is added to be able to specify, for instance, that a pickup or delivery request has no deadline. The set $W = T \times T$ represents all possible time-windows[2].

The time required to load and unload freight $f_j \in F$ is denoted $\delta_l(f_j) \in T$ and $\delta_u(f_j) \in T$, respectively. Assuming that transport resource $v \in R^{tr}$ has loaded a subset $O'_v \subseteq O_v$ of the requests assigned to it, the capacity constraint of a transport resource can now be

[1] In our model the speed and capacity of an infrastructure resource do not depend on the number of transport resources present.

[2] A time-window $\tau = [t_1, t_2] \in W$, $t_1, t_2 \in T$, for which $t_1 > t_2$, is said to be an invalid time-window.

specified as $\sum_{o_j \in O'_v} f_j \leqslant k^{tr}(v)$ – the sum of all currently loaded freight is smaller than the capacity of the transport resource – always holds.

Time values $t^s_{j,1}, t^s_{j,2}, t^d_{j,1}, t^d_{j,2} \in T$ are continuous, and the time-windows must be meaningful, i.e., it is possible to load within the loading time-window, $t^s_{j,2} \geqslant t^s_{j,1} + \delta_l(f_j)$, it is possible to unload within the unloading time-window, $t^d_{j,2} > t^d_{j,1} + \delta_u(f_j)$, and (neglecting driving time for the moment) it must be possible to unload after the minimal loading time, $t^d_{j,2} \geqslant t^s_{j,1} + \delta_l(f_j) + \delta_u(f_j)$. An infinite time value indicates it does not matter how early $(-\infty)$ or how late $(+\infty)$ the good is picked up or delivered.

Associated with each request o_j there is a reward function $\pi_j : W \times W \to \mathbb{R}$. If the actual pick-up and delivery time-windows are $\breve{\tau}^s_j$ and $\breve{\tau}^d_j$ respectively, $\pi_j(\breve{\tau}^s_j, \breve{\tau}^d_j)$ is maximized if the request is executed within its time-windows, i.e., $\breve{\tau}^s_j$ is during τ^s_j and $\breve{\tau}^d_j$ is during τ^d_j, and will typically be smaller if one or both of the time-windows of the transportation request are violated[3]. Both loading or unloading too early and loading or unloading too late typically decreases the reward the agent receives for executing the transportation request.

Finally, there is a load function $L_v : O_v \to T$ and an unload function $U_v : O_v \to T$ that, for each transportation request $o \in O_v$ in the set of transportation requests planned for execution by transport resource $v \in R^{tr}$, specifies the time at which the pickup and delivery takes place, $L_v(o)$ and $U_v(o)$ respectively. For all transportation requests to be executed, it must hold that, for all $o \in O_v$, $U_v(o) > L_v(o) \geqslant t_0$, where t_0 is the starting time.

There are more restrictions to what a correct plan can look like, which will be described later in Section 3.2. The next section describes the next component of our model, which is the transportation plan of a transport resource.

3.1.3 Agents and plans

The framework distinguishes transport agents and infrastructure agents. This section introduces both of these agents and their plans as ingredients of our framework. After the requirements with respect to conflicts are defined, Section 3.3 revisits infrastructure and transport agents and describes how the traversal time and reservations are computed exactly.

3.1.3.1 Transport agents

Each transport resource is controlled by one transport agent. This transport agent creates a plan for the transport resource, which successfully executes all the assigned transportation requests as efficiently as possible.

[3] Allen's interval algebra (Allen, 1983) defines that a time-interval τ' is during time-interval τ if and only if time-window τ' does not start before time-window τ and does not end at a later time.

A transport resource $v \in R^{tr}$ has a transportation plan that specifies the intended actions of the transport resource. Such a plan consists of the route the transport resource planned to traverse as well as the schedule information, which specifies at what time each of the locations (infrastructure resources) will be occupied by the transport resource.

The route $Rt_v = (r_{v,1}, r_{v,2}, \ldots, r_{v,N_v})$ for transport resource $v \in R^{tr}$ through infrastructure I is represented as a sequence of N_v infrastructure resources such that resources $r_{v,i}$ and $r_{v,i+1}$ are connected to each other, i.e., $(r_{v,i}, r_{v,i+1}) \in E_R$ for $1 \leqslant i < N_v$. Accompanying this route Rt_v, the schedule Sd_v of transport resource $v \in R^{tr}$ provides information on when each of these resources in Rt_v are claimed. A schedule $Sd_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,N_v})$ is a sequence of time points, where $t_{v,i}$ specifies the time transport resource $v \in R^{tr}$ claims resource $r_{v,i}$. This implies that transport resource v uses $r_{v,i}$ during the time-window $[t_{v,i}, t_{v,i+1})$ for $1 \leqslant i < N_v$ and uses resource r_{v,N_v} during time-window $[t_{v,N_v}, \infty)$. Obviously, at any time, the route and schedule have the same length, i.e., $\forall v \in R^{tr} : |Rt_v| = |Sd_v| = N_v$.

3.1.3.2 Infrastructure agents

The infrastructure agents are the road managers in a traffic network. They ensure the safety of (a part of) the network and their goal is to optimize the throughput of the network. The infrastructure agents determine which reservations are, and which are not, allowed for the transport agents. Different policies can be used to prioritize multiple transport resources, which want to enter the same infrastructure resource at the same time (these will later be described in Section 4.2.4).

The infrastructure agent maintains a set of reservations made by for the transport resources. Each time a transport agents wants to reserve the transportation plan it computed or re-computed, this set of reservation is changed. Set $Q(r) \subseteq R^{tr} \times W$ is the set of transport resource and time-window pairs stored at infrastructure resource $r \in R^{inf}$. If $\langle v, [t_1, t_2) \rangle \in Q(r)$, this means that vehicle $v \in R^{tr}$ has reserved access to infrastructure resource $r \in R^{inf}$ during the time-window starting at time t_1 and ending at time t_2.

To be able to specify later what exactly is a conflict between reservations of transport resources, we also introduce the following *claim* function. Using the plan representation described above, the claim function specifies in which infrastructure resource the transport resource $v \in R^{tr}$ has a reservation at time t and is defined as:

$$claim(v,t) = r \quad \Leftrightarrow \quad \exists r_{v,i} \in Rt_v : r = r_{v,i} \wedge t \in [t_{v,i}, t_{v,i+1}). \tag{3.1}$$

Often transportation plans are not executed according to the transportation plans that were initially computed. Because models are never perfect, the actions of the agents might not have the exact effect as described in the model. In our case, we want to test the robustness of the transport planning methods if such a situation occurs. That is why we modeled incidents, which is the topic of the next section.

3.1.4 Incidents

Incidents are events that disrupt regular plan execution and generally require replanning. There are several types of incidents, which are best categorized by the entity that they influence. These are *(i)* the agents, *(ii)* the transportation requests, *(iii)* the transport resources, *(iv)* and the infrastructure resources. To the first category belong communication failure problems. In AGV systems, often wireless communication is used between the AGVs, which can very well be subject to communication failure. In the second category, changes to transportation requests (e.g., a customer being late), but also the arrival of new transportation requests can be modeled as incidents. The latter two categories are related to (partial) resource failure, of both infrastructure resources as well as transport resources. Since, in the experiments presented in this thesis, only the third and fourth category of incidents play a role, the focus of this section is also on these types of incidents.

Communication failure If a communication failure incident occurs, the affected agent is not able to communicate with any other agents during the specified time interval. That means the agent might have to fall back to simpler planning methods for which it does not need to communicate with others. A communication failure incident (a, τ) in the set of incidents $\mathcal{I}$ specifies that agent $a \in A$ is not able to communicate with other agents during time-window $\tau \in W$.

Resource failure (speed) Resource failure indicates that a certain resource – either an infrastructure resource or a transport resource – does not function properly during a given interval in time. A resource failure incident (t_r, r, i, τ) in the set of incidents $\mathcal{I}$ is a tuple consisting of the time t_r at which the incident is announced to the agents, the resource $r \in R$ the incident operates on, an impact value $0 \leqslant i < 1$ indicating the severity of the incident, and a time-window $\tau \in W$ during which the incident is effective. The duration of τ is often denoted the *repair* time of the resource. If the incident operates on a transport resource, i.e., $r \in R^{tr}$, the vehicle's maximum speed is multiplied with $(1-i)$ during time-window τ. If the incident operates on an infrastructure resource, $r \in R^{inf}$, the maximum allowed speed of the resource is temporary multiplied with $(1-i)$.

Resource failure (capacity) If the resource is a cargo unit (place where the freight is loaded), the maximum loading capacity is multiplied with $(1-i)$ (where again the impact value i is between 0 and 1); this only affects new loading operation, not currently loaded transportation requests.

Now the components have been described it is possible to represent the transportation plans for all of the transport resources. Not all possible transportation plans for a transport resource are allowed and, therefore, the next section lists the invariants and requirements

on plans and sets of plans of the transport resources.

3.2 Invariants and requirements

This section describes the invariants to restrict to the problem setting and the validity requirements, first for a single transportation plan and then for the plans of a set of transport resources.

On the one hand, the invariants simplify the specification of the planning methods in Chapter 4. On the other hand, they restrict the transportation problem considered (such as the assumption stating that transshipment is not considered).

The requirements are needed to specify what is a feasible transportation plan. The transportation resource must, of course, be able to execute the plan and, by executing the plan, it must successfully complete the transportation requests. Furthermore, there are additional requirements for a set of transportation plans to ensure that it is also possible and safe to execute the set of plans together. The next section describes the invariants of the framework.

3.2.1 Framework invariants

Transport resources must claim exactly one infrastructure resource at all times (no ghost resources). Furthermore, each transport resource is given a start and destination location with sufficient capacity. This assumption prevents agents to be in a location where they bother other agents, but have no goal for themselves to leave this location (and, hence, this assumption simplifies the planning methods).

1. During the lifetime of each transport resource, exactly one infrastructure resource must always be claimed.

2. Each transport resource is initially located in an infrastructure resource $r \in R^{inf}$ with sufficient capacity, i.e., $k^{inf}(r) \geqslant |A|$. Such an infrastructure resource r is referred to as a parking space.

3. Each transport resource also ends in an infrastructure resource $r \in R^{inf}$ with sufficient capacity, i.e., $k^{inf}(r) \geqslant |A|$.

4. *Transshipment* is not possible. A task assigned to an agent can still be reassigned to another agent up to the moment the freight is loaded. When the freight has been loaded this agent is responsible for a correct execution of the particular task.

5. The storing of the reservations belonging to the transportation plan of a transport resource is assumed to be an *atomic* operation (i.e., no problem will arise that an

agent is searching a plan while reservations are being modified by another agent resulting in invalid transportation plans).

The next section describes the requirements for a transportation plan to be feasible and the requirements for a set of transportation plans to be safely executable together.

3.2.2 Framework requirements

In this section first the requirements of a transportation plan for a single transport resource are described. A transportation plan is called *feasible* if it meets the requirements described here. Subsequently, a set of transportation plans for multiple transport resources is considered. Not all combinations of feasible transportation plans can safely be executed together. The notion of a conflict is described and the requirements are listed for a set of transportation plans to be free of conflicts.

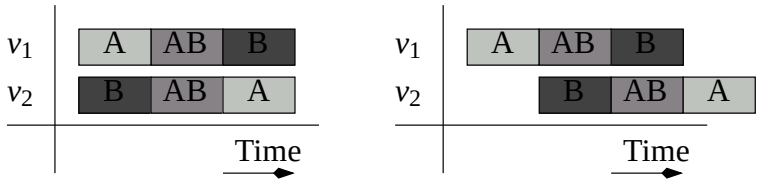
3.2.2.1 Requirements of a transportation plan

A *feasible* transportation plan can be executed by the transport resource and it will successfully complete all transportation requests that have been assigned to the transport resource.

Among others this means that all adjacent resources in the route of the transport resource must be neighbors in the transport network and all transportation requests assigned to the transport resource must be planned for correct execution. Definition 3.1 specifies when a plan is feasible.

Definition 3.1 A *feasible* plan $P_v = (v, Rt_v, Sd_v, L_v, U_v)$ for a transport resource $v \in R^{tr}$ is a plan that correctly takes into account all of the information, e.g., incidents, that is known at the time the plan was computed. A feasible plan P_v consists of a route $Rt_v = (r_{v,1}, \ldots, r_{v,N_v})$, a schedule $Sd_v = (t_{v,1}, \ldots, t_{v,N_v})$, load $L_v : O_v \rightarrow T$, and unload $U_v : O_v \rightarrow T$ information for transport resource $v \in R^{tr}$, for which the following must hold:

- The route and schedule have the same length, $|Rt_v| = |Sd_v| = N_v$,
- The first resource is claimed at the current time t, i.e., $claim(v,t) = r_1 \wedge t \in [t_1, t_2)$,
- The last resource is claimed as long as the transport resource exists, typically $t_{N_v+1} = \infty$,
- All infrastructure resources adjacent in route Rt_v must be neighbors in the transport network: $\forall i \in [1, N_v) : (r_{v,i}, r_{v,i+1}) \in E_R$,
- All loading actions must be performed in the infrastructure resource that was specified by the customer: $\forall o_i \in O_v, \exists j \in [1, N_v) : L_v(o_i) \in [t_{v,j}, t_{v,j+1}) \wedge r_{v,j} = s_i$, and likewise for unloading: $\forall o_i \in O_v, \exists j \in [1, N_v) : U_v(o_i) \in [t_{v,j}, t_{v,j+1}) \wedge r_{v,j} = d_i$, and



(a) Time-windows for both claims on resource AB overlap.

(b) Instantaneous exchange of resource AB and B between the transport resources.

Figure 3.2: Examples of the two types of conflicts. There are three resources with capacity 1, i.e., $R^{inf} = \{A, B, AB\}$ and $\forall r \in R^{inf} : k^{inf}(r) = 1$. Both conflicts would disappear if the capacities were 2.

- Unloading takes place after loading a request: $\forall o_i \in O_v : U_v(o_i) > L_v(o_i)$.

Instead of considering only a single transportation plan, in the following section the transportation plans of all transport resources together are taken into account.

3.2.2.2 Requirements of a set of plans

In order to consider when a set of transportation plans can safely be executed at the same time, the notion of a conflict will be considered. If a conflict between two or more transportation plans is present, this means the transportation plans cannot be executed together. We assume that the individual transportation plans are already feasible as discussed in the previous section and now consider the notion of conflicts.

Hatzack and Nebel (2001) suggest how to model conflicts between the plans for different transport resources. In their paper, each infrastructure resource has a minimum traversal time per vehicle. Unlike our model, all their infrastructure resources have capacity 1. Using our notation, assuming all infrastructure resources have capacity 1, transport resource $v \in R^{tr}$ has route $Rt_v = (r_{v,1}, r_{v,2}, \ldots, r_{v,N_v})$, schedule $Sd_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,N_v})$ and likewise for transport resource $w \in R^{tr}$, Hatzack and Nebel model conflicts as the following constraints:

$$\Big(r_{v,i} \quad = \quad r_{w,j}\Big) \Rightarrow \Big((v = w) \vee [t_{v,i}, t_{v,i+1}) \cap [t_{w,j}, t_{w,j+1}) = \varnothing\Big), \tag{3.2}$$

$$\Big(r_{v,i} = r_{w,j+1} \wedge r_{v,i+1} = r_{w,j}\Big) \Rightarrow \Big(t_{v,i+1} \neq t_{w,j+1}\Big). \tag{3.3}$$

Equation 3.2 states that if two different transport resources $v \in R^{tr}$ and $w \in R^{tr}$ claim the same resource, their time-windows may not overlap. Equation 3.3 prevents two transport

resources to change position with each other instantly. The equations correspond to the type of conflicts illustrated in Figure 3.2.

In this thesis a similar notion of conflicts is used. However, the above definition is adapted to take capacities of the infrastructure resources into account. In Equations 3.4 and 3.5 conflicts for resources with non-unit capacity are defined, corresponding to the above equations. Equations 3.2 and 3.3 of Hatzack and Nebel (2001) can be generalized to cope with varying capacities using Equation 3.1. This is done by setting a constraint on the maximum number of vehicles that can swap simultaneously.

First, two auxiliary functions are defined to simplify specifying the constraints. Function $\mathit{exchanges}(r,r',t)$ defines the number of transport resources that exchange infrastructure resource $r \in R^{inf}$ for $r' \in R^{inf}$ exactly at time $t \in T$. Function $\mathit{stay}(r,t)$ is the number of vehicles that do not change their current infrastructure resource $r \in R^{inf}$ for another infrastructure resource at time $t \in T$.

$$\begin{aligned}
\forall (r,r') \in E_R, \forall t \in T: \quad & \mathit{exchanges}(r,r',t) = \lim_{\varepsilon \downarrow 0} \big| \{ v \in R^{tr} : \\
& (\mathit{claim}(v,t-\varepsilon) = r \wedge \mathit{claim}(v,t+\varepsilon) = r') \\
& \vee (\mathit{claim}(v,t+\varepsilon) = r \wedge \mathit{claim}(v,t-\varepsilon) = r') \} \big|, \\
\forall r \in R^{inf}, \forall t \in T: \quad & \mathit{stay}(r,t) = \lim_{\varepsilon \downarrow 0} |\{ v \in R^{tr} : \\
& \mathit{claim}(v,t-\varepsilon) = r \wedge \mathit{claim}(v,t+\varepsilon) = r \}|.
\end{aligned}$$

The number of exchanges $\mathit{exchanges}(r,r',t')$ between infrastructure resources $r \in R^{inf}$ and $r' \in R^{inf}$ at time t is defined by counting the number of transport resources $v \in R^{tr}$ that, at time t, just left resource r just and entered resource r'; or exactly the other way around. This holds for any small number ε approaching zero.

We can now specify the two requirements, one defined at the resource level, the other at the edge level, that ensure that the joint set of transportation plans is possible and safe to be executed by the transportation agents.

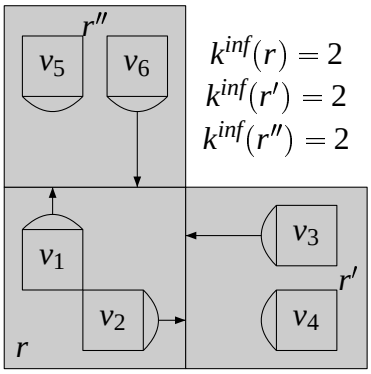
On the resource level, at all times the capacity of the resource must be satisfied:

$$\forall r \in R^{inf}, \forall t \in T : |\{ v \in R^{tr} : \mathit{claim}(v,t) = r \}| \leqslant k^{inf}(r). \tag{3.4}$$

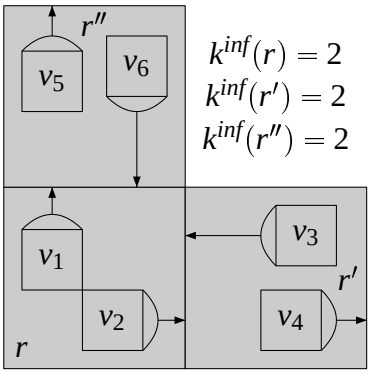
On the edge level, the following safety constraint is defined:

$$\begin{aligned}
\forall (r,r') \in E_R, \forall t \in T: \quad & \mathit{exchanges}(r,r',t) \leqslant \\
& \min(k^{inf}(r) - \mathit{stay}(r,t), k^{inf}(r') - \mathit{stay}(r',t)).
\end{aligned} \tag{3.5}$$

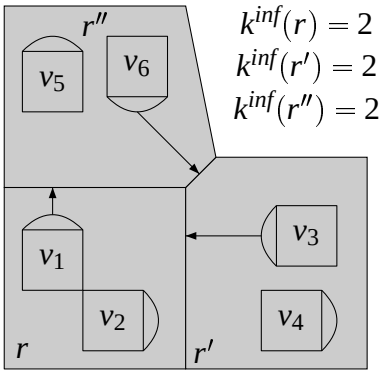
Equation 3.4 prevents the situation in Figure 3.2a, where more claims are done than allowed by the capacity $k^{inf}(r)$ of infrastructure resource $r \in R^{inf}$. Equation 3.5 avoids a situation like in Figure 3.2b. This equation ensures there is no spontaneous mutual exchange between infrastructure resources $r \in R^{inf}$ and $r' \in R^{inf}$ by more than $\min(k^{inf}(r) -$



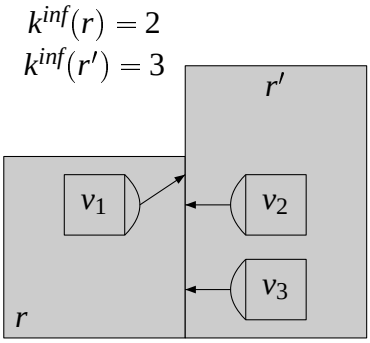
(a) Not allowed: $2 = exchanges(r, r', t) \nleq \min(2-0, 2-1) = 1$. Similar for (r, r'').



(b) Is allowed: $exchanges(r, r', t) = 2$ and $stay(r, t) = stay(r', t) = 0$. Similar for (r, r'').



(c) Is allowed, also in case both vehicles v_2 and v_3 would head to the right.



(d) Not allowed: $3 = exchanges(r, r', t) \nleq \min(2-0, 3-0) = 2$.

Figure 3.3: These examples illustrate which simultaneous exchanges are allowed by Equation 3.5. All arrows in these figures indicate the desire of the vehicle to move to the resource the arrow points to. All these movements are instantaneous, assume they are all at **exactly** the same time.

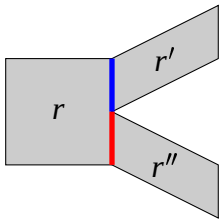


Figure 3.4: Border capacities $bordercap : E_R \rightarrow \mathbb{N}$, a possible extension to the model, $k^{inf}(r) = k^{inf}(r') = k^{inf}(r'') = 2$, $bordercap((r,r')) = 1$, $bordercap((r,r'')) = 1$.

$stay(r,t), k^{inf}(r') - stay(r',t))$ transport resources.

If one of the Equations 3.4 or 3.5 is violated, we say there is a conflict for a set of transport resources $R_c^{tr} \subseteq R^{tr}$ in resource $r \in R^{inf}$ (that has capacity $k^{inf}(r) \leqslant |R_c^{tr}|$) during time-window τ (τ is the intersection of the overlapping claims of all transport resources in R_c^{tr} for resource r).

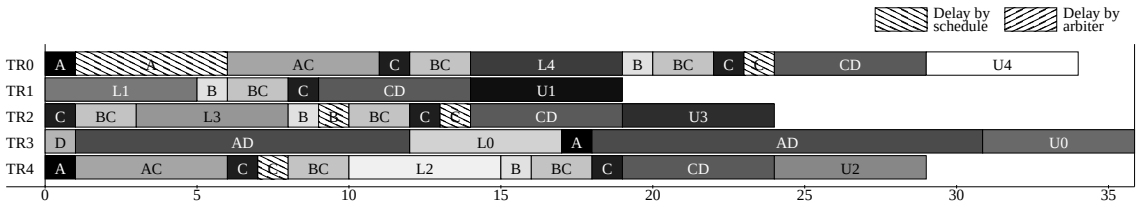
Remark 3.2 (Strenghtening the constraints) For some application domains, it could be desirable to strengthen (or weaken) the edge level constraint (Constraint 3.5). Strengthening the constraints is often required in cases where the transport resources are large relative to the infrastructure resources (e.g., airplane taxiing) and in cases where the infrastructure resources have many connections.

Equation 3.5 prevents more vehicles to swap resources simultaneously than the minimum of the capacities of these resources. If a resource has multiple outgoing arcs, see Figure 3.4, this might be too general – depending on the application domain. In such a case, the edges $e \in E_R$, whom in this model have no properties at all, can be given a border capacity $bordercap(e) \in \mathbb{N}$ representing the area size of the border between the infrastructure resources. Function $bordercap(e)$ then specifies the number of vehicles that are allowed to swap simultaneously. The edge level constraint, replacing Equation 3.5, would then be:

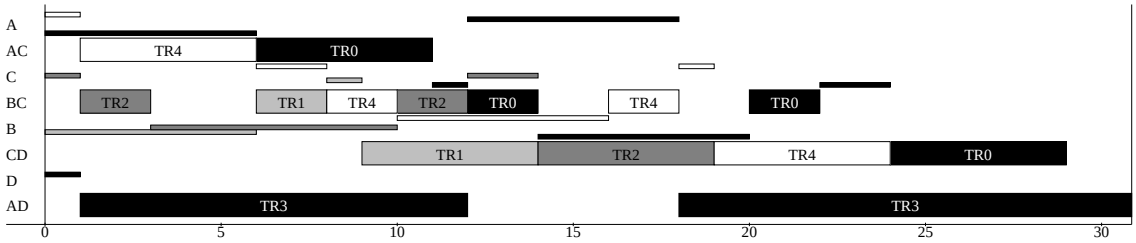
$$\forall (r,r') \in E_R, \forall t \in T : exchanges(r,r',t) \leqslant bordercap((r,r')). \tag{3.6}$$

Finally, we can state that Equations 3.4 and 3.5 form the requirements for a joint set of transportation plans, and there is one more requirement to the plan of a transport resource, which takes into account the plans of other transport resources:

- Given that reservations of other transport resources ($b_Q \in \mathbb{B}$) and incidents ($b_{\mathcal{I}} \in \mathbb{B}$) are or are not taken into account, not a single infrastructure resource is traversed faster than possible: $\forall i \in [1, N_v - 1] : t_{v,i+1} \geqslant t_{v,i} + \delta(v, r_{v,i}, t_{v,i}, b_Q, b_{\mathcal{I}})$.



(a) Infrastructure resource view.



(b) Transport resource view.

Figure 3.5: Example plans for five transport resources from two different perspectives.

The next section describes how the infrastructure agents and transport agents work together. The infrastructure agents compute reservations for infrastructure resource as asked by the transport agents. Furthermore, they actively monitor and inform the transport agents in case of abnormalities. If the transport agents create feasible plans that conform to the reservations computed by the infrastructure agents, the joint set of all transport resource plans is guaranteed to be possible and safe to execute.

3.3 Agents

Operational transport agents make agreements with infrastructure agents. For each infrastructure resource, a transport agent can claim the time-window for its transport resource when he wants to use the resource as determined in the schedule of the transport resource. This can, depending on the algorithms that are used, grant the agent certain rights, e.g., that no other transport resource is allowed to claim the infrastructure resource during this time-window. Or, they can be used by other agents to avoid heavily loaded infrastructure resources. In our model several functions are defined that specify the time required by transport resources to traverse resources – with or without taking into account previously committed plans – and the load of infrastructure resources over time.

3.3.1 Infrastructure agents

Access by transport resources to the infrastructure resources is controlled by one or more infrastructure agents, depending on the size of the infrastructure. Each infrastructure re-

source is controlled by one infrastructure agent. The infrastructure agent computes a reservation for a transport agent, given the time the transport resource enters the resource. Moreover, it is allowed to change these reservations at will. This is necessary, for example, in case an incident slows a transport resource down in such a way that another transport resource must also be delayed. The agent is informed that its prior reservation has to be delayed by the infrastructure agent. It is assumed that these infrastructure agents can be trusted by all other agents.

The tasks of an infrastructure agent are *(i)* to ensure a conflict-free situation and *(ii)* to maximize the performance of an infrastructure resource. It might be that infrastructure agents try to maximize the throughput of an infrastructure resource or they communicate with transport agents to determine which transport agent can go first. Intuitively, infrastructure agents can be compared to intelligent traffic lights. In Chapter 4 several methods are described that can be used by infrastructure agents.

The following section describes how the traversal speed of a transport resource for an infrastructure resource is determined.

3.3.1.1 Traversal speed

The speed at which transport resources traverse the infrastructure resources is determined by *(i)* their maximum speed, *(ii)* the maximum speed allowed at the infrastructure resource they are traversing, and *(iii)* incidents that affect either the transport resource or the infrastructure resource. Of course, their speed is zero if transport resources are not traversing infrastructure resources. For each transport resource $v \in R^{tr}$ and infrastructure resource $r \in R^{inf}$ we define the static speed $s_s(v,r)$ at which transport resource v traverses infrastructure resource r and the situation-aware speed $s_d(v,r,t)$ that takes into account incidents effective and known at time t. Note that it is possible that not all incidents at time t are known already at the time this function is computed. Using $E_i(r,t)$, which is the effective impact of incidents at time t for resource r (only taking into account incidents that are known at the time of computation), the static and dynamic speed are computed as follows:

$$s_s(v,r) = \min\{s^{tr}(v), s^{inf}(r)\}, \tag{3.7}$$

$$E_i(r,t) = \max\{i : (t_r, r, i, \tau) \in \mathcal{I} \wedge t \in \tau\}, \tag{3.8}$$

$$s_d(v,r,t) = \min\{s^{tr}(v) \cdot E_i(v,t), s^{inf}(r) \cdot E_i(r,t)\}. \tag{3.9}$$

The above speed functions do not include waiting time caused by other transport resources that claim the same infrastructure resource or by other transport resources driving in front of this transport resource in the case that overtaking is not allowed. This waiting

time is taken into account in the plans of transport resources.

Knowing the traversal speed it is possible for the infrastructure agents to compute a reservation for a transport resource and an infrastructure resource, given the time the transport resource will enter the infrastructure resource.

3.3.1.2 Computing reservations

Infrastructure agents that take into account both reservations of other transport resources as well as incidents search for the earliest time-window to claim an infrastructure resource that results in a conflict-free situation. In other words, the desired time-window $[t_1, t_2)$ in which transport resource $v \in R^{tr}$ traverses infrastructure resource $r \in R^{inf}$ at or later than time t is the solution to the following optimization problem (note that the last two equations are equal to Equation 3.4 and 3.5):

$$\begin{array}{ll} \text{minimize} & t_1 \\ \text{subject to} & t_1 \geqslant t \\ & t_2 = t_1 + s_d(v, r, t_1) \\ & \forall t \in [t_1, t_2) : |\{w \in R^{tr} : claim(w, t) = r\}| \leqslant k^{inf}(r) \\ & \forall (r, r') \in E_R, \forall t' \in \{t_1, t_2\} : exchanges(r, r', t') \leqslant \\ & \qquad \leqslant \min\left(k^{inf}(r) - stay(r, t'), k^{inf}(r') - stay(r', t')\right) \end{array} \tag{3.10}$$

If the infrastructure agent does not take into account incidents, we can replace $s_d(v, r, t_1)$ by $s_s(v, r)$. If reservations are not taken into account, the optimization problem becomes trivial: $t_1 = t$ and $t_2 = t + s_d(v, r, t)$. That greater values for $t_1 > t$ do not need to be considered is is proven by Proposition 3.3. Furthermore, this proposition supports the fact that we can minimize t_1 and do not have to worry about t_2 in the optimization objective.

We can now define function δ that computes the time required to traverse the infrastructure resource, given two Boolean parameters indicating whether or not to take into account reservations of other transport resources and/or incidents. Function $\delta(v, r, t, b_Q, b_{\mathcal{I}})$ represents the minimum time needed to traverse infrastructure resource $r \in R^{inf}$ by transport resource $v \in R^{tr}$ starting not before time $t \in T$ while taking into account reservations of other transport resources if and only if $b_Q \in \mathbb{B}$ is true and taking into account incidents

if and only if $b_{\mathcal{I}} \in \mathbb{B}$ is true.

$$
\begin{array}{lll}
\delta(v,r,t,b_Q,b_{\mathcal{I}}) = & t_2 - t \text{ in} & \\
\quad \text{minimize} & t_1 & \\
\quad \text{subject to} & t_1 \geqslant t & \\
& t_2 = t_1 + \begin{cases} s_d(v,r,t_1) & \text{if } b_{\mathcal{I}} \\ s_s(v,r) & \text{if } \neg b_{\mathcal{I}} \end{cases} & \\
& \forall t \in [t_1,t_2) : |\{w \in R^{tr} : claim(w,t) = r\}| \leqslant k^{inf}(r) & \text{if } b_Q \\
& \forall (r,r') \in E_R, \forall t' \in \{t_1,t_2\} : exchanges(r,r',t') \leqslant & \text{if } b_Q \\
& \qquad \leqslant \min\left(k^{inf}(r) - stay(r,t'), k^{inf}(r') - stay(r',t')\right) &
\end{array}
\tag{3.11}
$$

Proposition 3.3 shows that, while computing the reservation $[t_1,t_2)$ for an infrastructure $r \in R^{inf}$, it is sufficient to minimize the value of t_1, because that always leads to the minimum value for t_2 as well. Departing later (increasing t_1) always results in at least the same completion time t_2. Furthermore, the proposition generalizes this result over (partial) routes of more than one infrastructure resource.

Proposition 3.3 *Given any infrastructure $I = (R, E_R, k^{inf}, k^{tr}, d^{inf}, s^{inf}, s^{tr})$, a set of incidents $\mathcal{I}$, and a fixed route $Rt_v = (r_{v,1}, r_{v,2}, \ldots, r_{v,N_v})$, let $t, t' \in T$ to be two different departure times for transport resource $v \in R^{tr}$ such that $t' > t$. Let $\Delta(Rt_v, t)$ denote the minimal time needed to traverse Rt_v starting at time t. Then, at any point in time*[4]*, it holds that the completion time $t' + \Delta(Rt_v, t')$, when starting the traversal at time t', is at least the completion time when starting at time t:*

$$t' + \Delta(Rt_v, t') \geqslant t + \Delta(Rt_v, t).$$

□

PROOF: First it is proven the proposition holds for a single infrastructure resource. Then, it is shown how this result can be generalized over a complete route.

For any infrastructure resource $r \in R^{inf}$, and $\forall v \in R^{tr}, \forall b_Q, b_{\mathcal{I}} \in \mathbb{B}$, it holds that $t' + \delta(v,r,t',b_Q,b_{\mathcal{I}}) \geqslant t + \delta(v,r,t,b_Q,b_{\mathcal{I}})$. If the transport resource would leave at the early point in time t, it could in the worst case be blocked by an incident with impact one, meaning the transport resource would have speed zero during the incident. However, leaving at the later point in time t' – while traversing the same route – the transport resource will then arrive in the same situation. If incidents with impact one would not have been allowed, the inequality $\geqslant$ can even be replaced by the strict inequivalence relation $>$.

[4]Although the set of incidents $\mathcal{I}$ is fixed, it is still the case that at different points in time different incidents are known, because an incident is only known after its release time. A plan that was feasible at time t, is not necessarily still feasible at time $t + \varepsilon$ for some $\varepsilon > 0$.

Thus for a fixed route that has length one the proposition holds. Now notice that

$$\begin{aligned} t' + \Delta(Rt_v, t') &= t' + \delta(v, r_{v,1}, t', b_Q, b_{\mathcal{I}}) + \Delta\big((r_{v,2}, \ldots, r_{v,N_v}), t' + \delta(v, r_{v,1}, t', b_Q, b_{\mathcal{I}})\big) \\ &\geqslant t + \delta(v, r_{v,1}, t, b_Q, b_{\mathcal{I}}) + \Delta\big((r_{v,2}, \ldots, r_{v,N_v}), t' + \delta(v, r_{v,1}, t', b_Q, b_{\mathcal{I}})\big). \end{aligned}$$

That means it also holds for all fixed routes of length two then. By induction it can easily be shown that it holds for all fixed routes of arbitrary length. ■

This section described how the infrastructure agents compute reservations for transport resources that are not in conflict with any existing reservations. The next section describes the transport agents that can plan on a higher level, relying on the infrastructure agents to solve the lower level conflicts.

3.3.2 Transport agents

Transport agents are the planners for the transport resources. Their goal is to maximize performance (to be defined later in this section) by executing transportation requests. The set $O_v \subseteq O$ is the set of transportation requests in O that are assigned to transport resource $v \in R^{tr}$. Intuitively, they try to maximize the active performance indicator, for example, maximize the reward of individual transportation requests while minimizing the (traversal) costs. Each transport agent owns a transport resource, or possibly a set of transport resources. The task of transport agents is to compute plans for the transport resources they own.

At each point $t \in T$ in time, each transport resource $v \in R^{tr}$ claims exactly that infrastructure resource $claim(v,t) \in R^{inf}$, where transport resource $v \in R^{tr}$ resides in at time t. If r_1 and r_2 are infrastructure resources and $(r_1, r_2) \in E_R$ and transport resource $v \in R^{tr}$ holds a claim at infrastructure resource r_1, it can claim resource r_2 after having traversed resource r_1, while releasing its claim on resource r_1.

The behavior of a transport agent is mainly defined by how and which tasks it wishes to execute together with the planning method it uses to compute executable transportation plans.

The goal for each self-interested transport agent is to maximize its performance, e.g., to maximize the sum of reward values for the transportation requests it executes, and/or to minimize its costs for traversing infrastructure resources, waiting, etc. Although not necessary, for simplicity it is assumed that each agent is responsible for a single transport resource.

Equation 3.11 is used by the infrastructure agents to compute the reservations for the transport agents. The importance of this equation is, that, assuming the transport agents

only create feasible plans (see Definition 3.1) it can be proven they also only construct plans that meet the requirements for a set of transportation plans to be free of conflicts.

Proposition 3.4 *If the transport agents create feasible plans and the infrastructure agents compute the reservation time-windows for all infrastructure resources with limited capacities, then the joint set of all transportation plans for all transportation resources is free of conflicts.*

PROOF: We must prove that, if a transport agents created a feasible plan for its transport resource $v \in R^{tr}$ and an infrastructure agent determined the time-windows of the reservations, Equations 3.4 and 3.5 are met. Note that the transportation agent can choose the reservations for the *parking space* resources (the infrastructure resources with unlimited capacity) freely. Here no conflicts can occur and the vehicle can either stay idle here or load/unload freight.

Let route $Rt_v = (r_{v,1}, r_{v,2}, \ldots, r_{v,N_v})$ and schedule $Sd_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,N_v})$ be the feasible transportation plan for transport resource $v \in R^{tr}$. We will prove the proposition for an arbitrary index $i \in (1, N_v)$ in the plan (indexes 0 and N_v do not need to be considered, because these are parking space resources). At index i vehicle v resides in $r_{v,i}$ during the time-window $[t_{v,i}, t_{v,i+1})$.

From Equation 3.11 it follows that $\forall t \in [t_{v,i}, t_{v,i+1}) : |w \in R^{tr} : claim(w,t) = r_{v,i}| \leqslant k^{inf}(r_{v,i})$. This means that Equation 3.4 is satisfied.

Equation 3.11 also states that the infrastructure takes into account that $\forall t' \in \{t_{v,i}, t_{v,i+1}\} : exchanges(r_{v,i}, r_{v,i+1}, \leqslant) \min(k^{inf}(r_{v,i}) - stay(r_{v,i}, t'), k^{inf}(r_{v,i+1}) - stay(r_{v,i+1}, t'))$. And also that $\forall t' \in \{t_{v,i-1}, t_{v,i}\} : exchanges(r_{v,i-1}, r_{v,i}, \leqslant) \min(k^{inf}(r_{v,i-1}) - stay(r_{v,i-1}, t'), k^{inf}(r_{v,i}) - stay(r_{v,i}, t'))$. From these two observations it can be concluded that Equation 3.5 is also satisfied. ■

Proposition 3.4 does not mean that the planning of the transportation agents becomes trivial. First, it is their responsibility that the plans they create are feasible (by Definition 3.1). Second, it is also their responsibility to ask the infrastructure agents to create the reservations they need to construct feasible and efficient plans. Suppose that, at some time t, an infrastructure agents computes a reservation $[t_1, t_2)$ for the transportation agent. It is possible this reservation cannot be used by the transportation agent in its search for a feasible plan that executes all transportation requests. In such a case, the transport agent must ask for a new reservation with a different starting time $t > t_1$. How this is done will be described in the planning methods in Chapter 4.

The topic of the next section is to describe the quality of such a plan. Usually, there are many plans that achieve the same goals (i.e., delivery the freight), but those do not all have the same performance.

3.4 Performance criteria

Beamon (1998b) describes several important performance criteria for AGV systems such as vehicle travel time, vehicle utilization, queue length, and material handling cost. Le-Anh and de Koster (2004) states that generally, multi-criteria objective functions lead to a better quality. Morton and Pentico (1993) suggest to take into account revenue, tardiness, costs, and economic makespan as objectives. Fully developing an objective function according to them is not worthwhile and, since it is application specific, for the scope of this thesis only several general functions are specified to be able to work with. This gives a short-list with the following *objectives*:

- The costs for executing the transportation plan,
- Travel time, waiting time, idle time,
- Resource utilization, for transport network or transport resource,
- Queue length,
- Earliness and tardiness of the transportation requests,
- The number of accepted and correctly executed transportation requests,
- Economic makespan, trying to minimize resource utilization, penalizing intermediate (not final) waiting time.

The plan of a transport resource $v \in R^{tr}$ consists of a route Rt_v, a schedule Sd_v, and a load and unload function that specify at what time each transportation request $o \in O_v$ is loaded, $L_v(o)$, and at what time it is unloaded, i.e., $U_v(o)$.

The performance function of such a plan usually is a trade-off between cost and reward. In general, this cost-reward optimization is non-trivial due to the imposed pickup and delivery time-windows of the requests.

The following section describes how the cost of a transportation plan can be specified. The subsequent section describes other criteria, besides cost, which are often used in a performance indicator.

3.4.1 Cost model

To determine the cost for the total system, one usually distinguishes between fixed costs and variable costs. The fixed costs, for each transport resource, are independent of the usage of the resource, but are there because a company must own, rent, or lease the transport resource. The fixed costs of a transport resource $v \in R^{tr}$ can be expressed as a function of $C_f(v) \in T$. Note that we measure the costs of a plan in terms of time.

The variable costs depend on the amount of time the transport resource resides in any of its states. A transport resource can be *Waiting*, *Loading*, *Unloading* or *Driving.* In order to specify the costs of these states in terms of the schedule Sd_v of transport resource $v \in R^{tr}$, the loading costs Lc_v and unloading costs Uc_v are defined first:

Loading time costs $Lc_v = (l_{v,1}, l_{v,2}, \ldots, l_{v,N_v})$,where

$$l_{v,i} = \sum_{o_j \in O_v} \begin{cases} \delta_l(f_j) & \text{if } L_v(o_j) \in [Sd_{v,i}, Sd_{v,i+1}), \\ 0 & \text{otherwise.} \end{cases}$$

Analogously, the unloading time costs $Uc_v = (u_{v,1}, u_{v,2}, \ldots, u_{v,N_v})$,where

$$u_{v,i} = \sum_{o_j \in O_v} \begin{cases} \delta_u(f_j) & \text{if } U_v(o_j) \in [Sd_{v,i}, Sd_{v,i+1}), \\ 0 & \text{otherwise.} \end{cases}$$

Then the departure times from each resource are $Dt_v = (t'_{v,1}, t'_{v,2}, \ldots, t'_{v,N_v})$, where

$$t'_{v,i} = t_{v,i} + Lc_{v,i} + Uc_{v,i}.$$

The total loading time in transport resource v's plan is $Load(Sd_v) = \sum_{t \in Lc_v} t$ and the total unloading time is $Unload(Sd_v) = \sum_{t \in Uc_v} t$. The total drive costs are $Drive(Sd_v) = \sum_{i=1 \ldots N_v} s_d(v, r_{v,i}, t'_{v,i})$. The remaining time between the birth $t_{b,v}$ and death $t_{d,v}$ of transport resource $v \in R^{tr}$ is the waiting time $Wait(Sd_v) = (t_{d,v} - t_{b,v}) - Drive(Sd_v) - Load(Sd_v) - Unload(Sd_v)$.

The variable costs of a transport resource $v \in R^{tr}$ is a function $C_v(v, Drive(Sd_v), Drive(Sd_v), Load(Sd_v), Unload(Sd_v))$ of these different states.

The performance of an agent, however, is more than only the costs for executing its plan. One obvious aspect is the rewards agents can achieve by executing their transportation requests successfully.

3.4.2 Performance indicators

The performance of a multi-agent system is measured in terms of the performance of the individual transport resources. Several aspects are important in the transportation domain. Among others, these are *(i)* the cost a transport resource has to make for executing the tasks assigned to it, *(ii)* the reward the agent receives for successfully executing tasks, and *(iii)* the CPU cost a transport agent needs to compute its plan. The latter, CPU cost, gives important information about the scalability of a system. Or, in other words, if a problem instance increases in size (for example, more transportation requests, more agents, more incidents) can the multi-agent system still function within acceptable time?

The specified load and unload time-windows associated with transportation request $o_j = (f_j, s_j, \tau_j^s, d_j, \tau_j^d, \pi_j)$ are τ_j^s and τ_j^d respectively. Let $\breve{\tau}_j^s$ and $\breve{\tau}_j^d$ refer to the realized time-windows of request $o_j \in O_v$ and P_v^* to the optimal plan for transport resource $v \in R^{tr}$.

The average tardiness per request, infrastructure costs, transport resource reward, and finally the relative system welfare are defined as follows:

$$\begin{aligned}
\text{tardiness}(\tau_j^s, \tau_j^d, \breve{\tau}_j^s, \breve{\tau}_j^d) &= \max\big(0, ub(\breve{\tau}_j^s) - ub(\tau_j^s)\big) \\
&+ \max\big(0, ub(\breve{\tau}_j^d) - ub(\tau_j^d)\big), \\
\text{average request tardiness} &= \sum_{\{j:o_j \in O_\}} \text{tardiness}(\tau_j^s, \tau_j^d, \breve{\tau}_j^s, \breve{\tau}_j^d)/|O_|, \\
\text{relative infrastructure costs} &= \frac{\sum_{v \in R^{tr}} C(P_v)}{\sum_{v \in R^{tr}} C(P_v^*)}, \\
\text{relative transport resource reward} &= \sum_{\{j:o_j \in O_\}} \frac{\pi_j(\breve{\tau}_j^s, \breve{\tau}_j^d)}{\pi_j(\tau_j^s, \tau_j^d)}, \\
\text{system welfare} &= \frac{\sum_{\{j:o_j \in O_\}} \pi_j(\breve{\tau}_j^s, \breve{\tau}_j^d) - \sum_{v \in R^{tr}} C(P_v)}{\sum_{\{j:o_j \in O_\}} \pi_j(\tau_j^s, \tau_j^d) - \sum_{v \in R^{tr}} C(P_v^*)}.
\end{aligned}$$

In the experiments in Chapter 5, mostly the CPU cost (in time) and the relative transport resource reward will be used. Which performance indicator is most important depends on the problem domain. Also, at the strategic level one might be more interested in the transport resource costs (whether or not to allocate an additional transport resource) than for instance tardiness, while at the operational level this might be the other way around.

3.5 Summary

In this chapter a detailed description of a model for operational multi-agent transport planning is presented. The framework proposes a representation for a transport network, transport resources and their transportation plans, transportation requests and incidents.

Following Hatzack and Nebel (2001), the framework makes use of a non-classical graph representation. Instead of modeling a transport network as a classical graph with edges and vertices, where both usually have different properties, we make use of a set of infrastructure resources, having several properties, with connections that have no special properties. This eases, for instance, the modeling of crossroads.

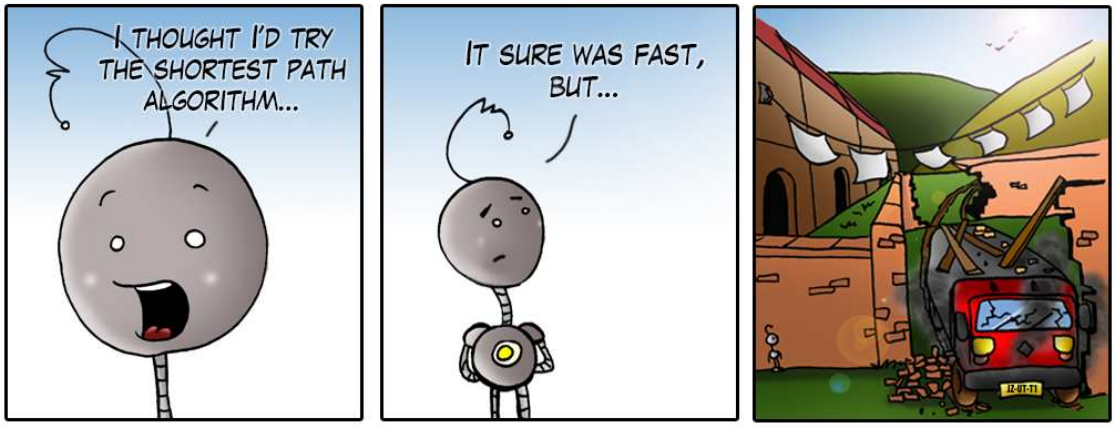
The key aspect of this framework is that it distinguishes between transport agents and infrastructure agents. The infrastructure agents make reservations for infrastructure resources, which aids the search for conflict-free plans for the transport resources.

Infrastructure agents guard the usage of infrastructure resources by only allowing transport resources access to the infrastructure resources if they have made a reservation. The exact duration of this reservation is previously determined by an infrastructure agent. This ensures, as proven in this chapter, that transport agents together create a set of joined plans that is free of conflicts.

In the next chapter a variety of planning methods are presented that make use of the framework presented here to construct transportation plans. Because, as proven in Appendix D, finding the best possible transport planning is a very hard problem, the focus is on searching approximation methods to big-sized transportation problems.

Chapter 4

Planning methods



Multi-agent route (transportation) planning refers to the problem of finding a conflict-free set of routes for a set of agents using a shared infrastructure. In the previous chapter a framework for operational transportation planning has been presented and Section 3.2 proves that a set of agent routes is guaranteed to be free of conflicts if the transport agents query the infrastructure agents for the start and end times at which they will claim the individual infrastructure resources. This chapter moves on to the question how to build complete transportation plans that that correctly execute the transportation requests and meet the customer's deadlines.

In this chapter several approaches for multi-agent route planning will be considered. The first of these is the classical solution to transportation planning. In this approach the transportation planning is split into a route-finding and a conflict resolution stage. This approach is characterized by the use of basic shortest path algorithms and simple resource usage rules that prioritize the agents on road crossings. Note that both stages might be done during planning time, or the conflict resolution stage might be delayed until the actual execution of the plans starts. The principal idea of the approach is to lower the complexity by first only considering which route to take, then to schedule the entry and

exit times of the infrastructure resources along the route.

The classical approach has several disadvantages. If conflicts are resolved during plan execution deadlocks might occur and this affects the predictability of travel times. But also if conflicts are resolved during planning time, performance might suffer from fixating the routes before considering potential conflicts with other transport resources.

Other researchers adopt an integrated approach, where conflict resolution is integrated with route planning (*context-aware* routing). The best known result is that of Kim and Tanchoco (1991), which has a high computational complexity. Furthermore, one might doubt whether it is useful to invest this much time in finding plans which can be destroyed by a few incidents (and cause a re-planning).

Context-aware routing intuitively seems a good approach in ideal circumstances, because it computes optimal plans assuming that the other agents do not make any plan changes and the absence of uncertainty in the environment. However, it stands to reason that context-aware routing only performs well in normal circumstances, and performance degrades in case of incidents. The reason is that the context-aware routing method – as opposed to the classical approach – delays agents based on information about the plans of other agents, but that information might no longer be valid in incident-rich environments.

Therefore, the final part of this chapter considers several approaches that are more robust in case of incidents and with respect to modeling uncertainties. Because of this uncertainty, it is practical that these methods operate both in the planning stage (before execution starts) as well as in the execution stage.

The next section describes the first approach to transportation planning, which is called the classical approach, because later on the other transportation planning methods are compared to this approach.

4.1 Classical approach

A straightforward solution for operational transportation planning is to leave the planning to the agents, but constrain the plan execution in a way similar to the everyday traffic regulation approach: use a set of operational conflict-resolution rules such as traffic rules (keep right), traffic lights and dynamic traffic guidance systems to ensure effective conflict resolution in the operational stage.

Hence, one could consider as a solution to multi-agent transportation planning to split the problem in two parts:

- *planning* stage: apply individual route-finding algorithms to find a best route for a single agent in a given infrastructure, and

- *conflict-resolutionconflict resolution* stage: conflict-resolution mechanisms that can

be used to resolve conflicts that arise due to the execution of routes found by route-finding algorithms.

The simplest way to determine the route for an agent, neglecting the presence of other agents on a given infrastructure, is to have each agent plan a shortest path from its current location to its destination. A basic *shortest-path algorithm*, such as Dijkstra (1959), can be used by the agents. Then comes the second stage of the standard approach. The operational conflict resolution is done by defining resource usage rules that prioritize the vehicles if they enter crossroads at the same time. These resource usage rules can be based on dynamic aspects, such as who arrives first at a crossroad, or static aspects related to the importance of the task. Examples of static resource usage rules are:

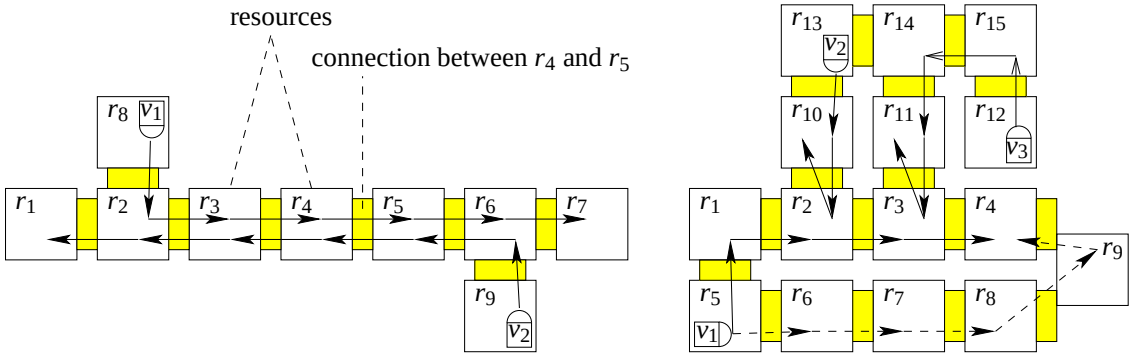
- highest-task-priority: the profits that can be earned for executing tasks is not constant. For example, an ambulance, police or fire brigade in action could precede regular traffic,
- highest-vehicle-priority: there could be several different types of transport resources, each with their own priority level.

Examples of dynamic resource usage rules are:

- first-come-first-served: for example, in the US at at 4-way-stop intersection, the vehicle that reaches the crossroad first gains the highest priority,
- longest-waiter-first: during plan execution, agents possibly have to wait at several occasions. This rule is similar to the previous one, but sums the waiting times of multiple crossroads,
- urgent-deadline-first: the agents are executing tasks with delivery deadlines. This rule is similar to the previous one, but considers the urgence with respect to time instead of task importance.

The problem with this classical approach is that travel times are becoming almost *unpredictable*: an agent must at least have some knowledge of what the other agents are doing to know how this affects its own plan. Even more important is the possibility of *deadlocks* to occur (see Figure 4.1a) which means the agents will not even be able to execute their plans. Finally, more knowledge about each others actions can improve the agent's decision making, which in turn improves the performance.

It is possible to move the conflict resolution mechanism into the planning stage as well. Such a generate-and-test approach first generates the routes and then a test approach is followed, which detects conflicts and finds a solutions to the problem. This makes the resource usage rules described above more effective, it removes the possibility for deadlocks to occur, and it makes the travel times more predictable. This approach, however,



(a) A deadlock occurs if both vehicles start driving without letting the other pass by first.

(b) Vehicle v_1 better takes the bottom detour, if it has to give way to vehicles v_2 and v_3, but it cannot know this without at least considering the plans of the other agents.

Figure 4.1: Two examples that show the necessity of agents to communicate their plans.

still starts with searching a route and then does not consider alternative routes anymore. We believe it is better to integrate the route searching with the conflict resolution. This is the essence of the *context-aware* approach we propose as an alternative to the classical approach.

4.2 Context-aware routing

Context-aware routing (Fujii et al., 1989; Kim and Tanchoco, 1991) refers to an approach where during the construction of an optimal route from source to destination also the consequences of the plans of other agents using the same infrastructure have been taken into account. This means that as a result of context-aware routing the set of routes of the agents are conflict-free and individually optimal given the routes of the other agents. Context-aware routing, however, requires that the results of route plans of agents can be stored and (locally) retrieved. Agents must be able *(i)* to make reservations of parts of the infrastructure and *(ii)* to be provided with detailed information about the availability of parts of the infrastructure.

Context-aware routing – assuming that no incidents occur – therefore

1. ensures that the set of routes determined is conflict-free,
2. ensures the predictability of the individual travel times,
3. renders a separate conflict-resolution stage obsolete.

The next section describes context-aware routing for a single agent. This algorithm solves the problem to find an optimal shortest path in time for an agent given a set of

transportation plans (which are assumed not to change) of other agents. Subsequently, Section 4.2.2 considers the context-aware routing for a set of agents.

4.2.1 Single-agent context-aware routing (SACA)

This section only considers planning for a single agent. It can be assumed that a subset of the agents have already created plans and made reservations – these reservations are assumed to be fixed and executed as planned. The rest of the agents do not have transportation plans at the moment, and can use the algorithm, which will be presented in this section, to find a transportation plan.

To make clear that agents have to consider the plans (reservations) of other agents let us consider an example. The example shows why it does not suffice to have all agents individually use a basic shortest path algorithm such as Dijkstra (1959).

Example 4.1 In Figure 4.1a a deadlock occurs if both vehicles plan to traverse their route as soon as possible. If vehicle v_1 enters resource r_2 before vehicle v_2, and vehicle v_2 enters resource r_6 before vehicle v_1, then a deadlock cannot be avoided unless at least one of the agents modifies its route. The only solution is to have one of both waiting until the other passes by.

In Figure 4.1b vehicle v_1 has to make a choice between the upper route $Rt_{v_1,1} = (r_5, r_1, r_2, r_3, r_4)$ and the lower route $Rt_{v_1,2} = (r_5, r_6, r_7, r_8, r_9, r_4)$. Note that there are no connections $(r_2, r_6) \notin E_R$, $(r_{10}, r_{11}) \notin E_R$, etc[1]. Obviously it chooses the upper route $Rt_{v_1,1}$, which it can finish in fewer steps. But let us consider the plans of all agents in more detail. The plan for vehicle v_2 is $P_{v_2,1} = (r_{13}, [0,1), r_{10}, [1,2), r_2, [2,3), r_{10}, [3,4))$. Vehicle v_3 wants to go to resource r_{11} via r_3 and has the plan $P_{v_3} = (r_{12}, [0,1), r_{15}, [1,2), r_{14}, [2,3), r_{11}, [3,4), r_3, [4,5), r_{11}, [5,6))$. The first part of the plan for taking the upper route $Rt_{v_1,1}$ is $P_{v_1,1} = (r_5, [0,1), r_1, [1,2), \ldots)$. But there is the first conflict. Vehicles v_1 and v_2 cannot both traverse resource r_2 during the time-window $[2,3)$. Let us assume that v_2 precedes v_1, then vehicle v_1 waits in resource r_1 and enters resource r_2 at time 3, i.e., $P_{v_1,1} = (r_5, [0,1), r_1, [1,3), r_2, [3,4), \ldots)$. The second conflict occurs in resource r_3 and, again, we assume that vehicle v_3 has higher priority than vehicle v_1, which results in the plan $P_{v_1,1} = (r_5, [0,1), r_1, [1,3), r_2, [3,5), r_3, [5,6), r_4, [6,7))$. This plan turns out inferior than the plan for taking the bottom route $P_{v_1,2} = (r_5, [0,1), r_6, [1,2), r_7, [2,3), r_8, [3,4), r_9, [4,5), r_4, [5,6))$. □

The above example showed that if an agent takes the plans of the other into account it is able to avoid conflicts that might occur otherwise. With the classical approach, in the first example the occurrence of a deadlock is likely, while in the second example the upper

[1] In the examples in this chapter it is assumed that all infrastructure resources – unless explicitly mentioned otherwise – have a capacity of 1 and can be traversed in 1 time unit.

route would incorrectly seem better than the bottom detour. This is the advantage of the context-aware routing algorithm over approaches that leave the conflict-solving to the execution stage (such as the classical approach).

4.2.1.1 Single-agent context-aware source-destination routing

In this section the core part of the context-aware routing algorithm is presented: how to find a conflict-free shortest path in time from a source to a destination location for a single agent. Conflict-free means that the plans of other agents are taken into account. The set of reservations of the other agents, which planned prior to the currently planning agent, is assumed to be fixed and to be executed as planned.

Before presenting the context-aware routing algorithm, a section is devoted to extend the framework presented in Chapter 3 with the notion of free time-windows and free time-window reachability. These are required for the context-aware routing algorithm.

Free time-window graph Instead of searching a shortest path in a graph, where the nodes represent locations (i.e., infrastructure resources), the context-aware routing algorithm searches through a graph of *free time-windows*. A free time-window is an interval during which the infrastructure resource has sufficient capacity available for an additional transport resource to occupy the infrastructure resources. The idea is to search through free time-intervals instead of reserved or forbidden time-intervals.

The framework presented in Chapter 3 includes, for each infrastructure resource $r \in R^{inf}$, a set $Q(r) \subseteq A \times W$ of agent time-window pairs representing the reservations of all agents. For the context-aware routing algorithm, a set of free time-windows specifying when the resource can accept additional presence of a transport resource is defined in terms of the set of reservations Q. Agents can safely create reservations in time intervals that are within the bounds of these free time-windows.

Definition 4.2 (Free time-window) Given resource r_i and the agent-reservations $Q(r_i) \subseteq A \times W$ on resource r_i, a free time-window on r_i is an interval $f_{i,v} = [\sigma_{i,v}, \phi_{i,v})$ such that:

1. $\forall t \in f_{i,v} : |\{(a_j, \tau_k) \in Q(r_i) : t \in \tau_k\}| < k^{inf}(i)$,
2. $\phi_{i,v} - \sigma_{i,v} \geqslant d^{inf}(r_i)$.

The first condition states that for an interval to be a free time-window, there should not only be sufficient capacity at any moment during that interval, and the second condition of Definition 4.2 ensures that it is also possible to traverse the resource before the end of the time-window. Note that the collection of free time-windows F_i on resource r_i is a list $(f_{i,1}, f_{i,2}, \ldots, f_{i,m})$ of disjoint intervals such that for all $j \in [1, \ldots, m-1]$, $f_{i,j}$ precedes $f_{i,j+1}$.

The context-aware routing algorithm is based on the idea of going from one free time-window on one resource to another free time-window on another resource. The reachability relation ρ defines when two free time-windows are reachable:

Definition 4.3 (Free time-window reachability) Given a resource r_i, a free time-window $f_{i,v}$ on this resource, and a time t, the free time-window $f_{j,w}$ on resource r_j is reachable from r_i at time t, denoted $f_{j,w} \in \rho(r_i,t)$, if:

1. $(r_i,r_j) \in E$,
2. $t \in (f_{i,v} \cap f_{j,w})$,
3. $t - \sigma_{i,v} \geqslant d^{inf}(r_i)$,
4. $\phi_{j,w} - t \geqslant d^{inf}(r_j)$.

The first condition in Definition 4.3 ensures that the resources r_i and r_j are connected. The second condition states that time t is contained both in the free time-window $f_{i,v}$ of resource r_i as well as in the free time-window $f_{j,w}$ of resource r_j, and these free time-windows overlap. The third condition ensures the transport resource does not exit resource r_i before it had time to traverse r_i and the fourth condition requires there will be enough time to traverse resource r_j if the transport resource enters r_j at time t.

Basic shortest-path algorithms, such as Dijkstra (1959), do not consider the plans of other agents while searching for a shortest path in time from the source to destination location. Context-aware routing algorithms do take this into account. In Dijkstra's shortest path algorithm, when a node is selected for expansion, it is sure that the current path to this node is the shortest, and the algorithm does not need to consider any other paths leading to this node. In context-aware routing, on the other hand, it does not suffice to consider infrastructure resources only once. The first time a resource r is considered, one has indeed found the fastest route from the source location r_s to resource r (say arriving at time t), but in an optimal plan from location r_s to destination r_d it might be necessary to enter resource r at some time t' later ($t' > t$) due to reservations of other agents.

Example 4.4 Figure 4.2 shows why it is not sufficient only to consider the earliest possible visit of a resource. From the first (and only) free time-window on start resource r_s, both free time-windows on resource r_1 (which is on a direct path to the destination resource r_d) can be reached. However, from the first free time-window on r_1, $f_{1,1} = [0,2)$, no free time-window on r_d can be reached, because on the destination resource there is a reservation until time 5. Hence, a transport resource must go from r_s to r_1 at time 4 (assuming travel times of 1 for all resources, by time 4 resource r_s can be traversed). The transport resource can leave r_1 at time 5, entering r_d at time 5, at the start of the free time-window on the destination resource. □

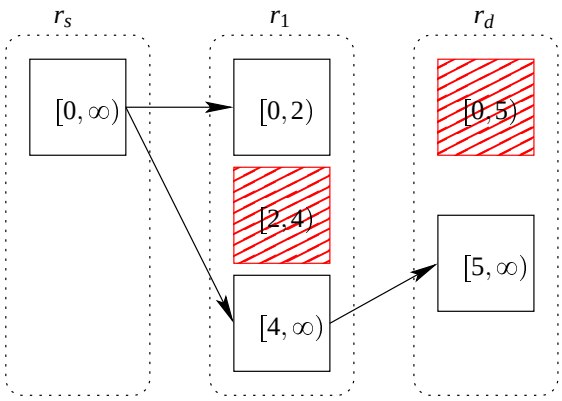


Figure 4.2: The first arrival at resource r_1, at time 1, will not lead to the shortest path to destination resource r_d. Instead, the path that visits r_1 during its second free time-window, starting at time 4, must also be considered.

Now the notion of free time-windows and free time-window reachability has been introduced, an algorithm can be described that takes reservations of other agents into account by considering these free time-windows.

Context-aware source-destination algorithm For a clear presentation our context-aware routing algorithm will be presented in steps. This section considers how an agent searches a shortest path in time, from a source to a destination location, taking into account the reservations of a set of other agents.

Because agents usually have multiple transportation requests assigned to them, in Section 4.2.1.2 the algorithm is modified to search a shortest-path in time along a sequence of pickup and delivery locations (this sequence is referred to as the *visiting sequence*). Finally, Section 4.2.2 presents the multi-agent context-aware routing algorithm that finds a route for all agents instead of just for one.

In this section the context-aware source-destination algorithm is described, which searches a shortest path in time for a single agent, given a source (usually the location where the agent resides) and destination location. The context-awareness comes from the fact the algorithm takes information about the plans of the other agents into account. This is realized by storing reservations by other agents to occupy a resource in the infrastructure. Using this information the context-aware routing algorithm can search a shortest path in time avoiding any conflicts with other agents.

The context-aware routing algorithm maintains a queue Q of candidate solutions, which is sorted by the costs made in the partial solution so far. The cheapest partial solution, the element at the front of the queue, is retrieved from the queue. This candidate solution is then expanded by appending reachable free time-windows to the partial route, provided that no constraints are violated. If such an expansion means that the

Algorithm 4.1 Context-aware shortest-path routing.

```
1:  function ContextAwarePath(r_s, r_d, t_s)
2:      Pre: start resource r_s, destination resource r_d, start time t_s.
3:      Post: entry time into r_d for the shortest path from r_s to r_d.
4:      if ∃v [f_{s,v} ∈ F_s | t_s ∈ f_{s,v}] then
5:          Q ← {(r_s, t_s)}
6:      end if
7:      while Q ≠ ∅ do
8:          (r_i, t_i) ← argmin_{(r,t)∈Q} t + d^inf(r))
9:          Q ← Q\{(r_i, t_i)}
10:         if r_i = r_d then
11:             return followBackpointers(r_i, t_i)
12:         end if
13:         for all f_{j,v} ∈ ρ(r_i, t_i) do
14:             t_entry = max(t_i + d^inf(r_i), σ_{j,v})
15:             if constraintsOk(r_i, r_j, t_entry) then
16:                 Q ← Q ∪ (r_j, t_entry)
17:                 F_j ← F_j\{f_{j,v}}
18:                 backpointer(r_j, t_entry) ← (r_i, t_i)
19:             end if
20:         end for
21:     end while
22:     return NoPossiblePath
23: end function
```

new resource can be reached earlier than without the current expansion, the resource and reached free time-window at that resource are added to queue Q and a backpointer is stored to the resource and free time-window pair from which it is reached. Finally, these backpointers are traversed to reconstruct the shortest possible path from the source to the destination infrastructure resource.

Algorithm 4.1 presents the pseudocode of the single-agent source-destination context-aware routing algorithm, which will now be considered in more detail. The algorithm expands a partial plan by looking at which free time-windows can be reached, rather than by expanding the plan by reachable resources. In Line 5, the open list Q of free time-windows is initialized to the start resource and the start time. In Line 8, the open list element (r_i, t_i) with the lowest cost $t_i + d^{inf}(i)$ is retrieved. Hence, the open list elements are sorted in order of increasing (minimum) *exit* times. Only the (r_i, t_i) pairs are stored on the open list Q, but the complete route and schedule can be reconstructed using backpointers that point to the previous resource-time pair. These backpointers, set in Line 18, point to the resource-time pair from which the newly added resource-time pair is reached. The FOLLOWBACKPOINTERS function then simply reconstructs the route back-

wards from destination to source resource by following the backpointers, to return the shortest path in Line 11.

To expand the current free time-window, in Line 13, all (resource, free time-window) pairs that are in $\rho(r_i, t_i)$ are considered. The entry time into a reachable free time-window $f_{j,v} = [\sigma_{j,v}, \phi_{j,v})$ is either the entry time into the previous resource r_i plus the time it takes to traverse r_i, or, in case $f_{j,v}$ starts after $t_i + d^{inf}(r_i)$, the start time $\sigma_{j,v}$ of $f_{j,v}$.

Line 15 checks additional constraints with regard to the current expansion candidate. Function CONSTRAINTSOK returns true if the constraints with respect to head-on and catching-up conflicts are met (see Section 3.2.2.2). Line 16 adds the new element to the open list Q, and, Line 17 removes the free time-window $f_{j,v}$ from the set of free time-windows F_j of resource r_j. This is an important step, as it guarantees that we do not consider any free time-window for expansion more than once.

The next section presents a proof of the *correctness* of the context-aware routing algorithm.

Correctness of the single request variant An algorithm is considered *correct* if, and only if, assuming that the starting point is a set of transportation plans that are free of conflicts, any new transportation plan computed by the algorithm is possible to be executed and not in conflict with an already existing plan of some agent.

First, note that the plans constructed by the context-aware routing algorithm can always be executed, assuming no incidents occur, because the next resource to be visited is always selected using the reachability relation ρ.

Second, assuming the plans of the other agents are conflict-free, the resulting set of transportation plans are also free of conflicts. This follows directly from the fact that the resulting plan only creates new reservations within time-intervals that were free time-windows.

Optimality of the single request variant The context-aware routing algorithm finds the optimal route from a source to a destination resource, starting at a given time, and assuming that the environment is fully known (i.e., no incidents, no more changes to the plans of other agents and no uncertainty in the environment).

Proposition 4.5 *Algorithm 4.1 returns an optimal solution, given the plans and reservations of the other agents are fixed.*

PROOF: First we prove, by induction, that during the n^{th} execution of the while-loop in Line 7, each pair $(r_i, t_i) \in Q$ on the open list Q represents the earliest time to reach the free time-window $f_{i,v}$, having started from (r_s, t_s).

Initially, the open list contains only (r_s, t_s), and the induction hypothesis holds for $n = 0$. Suppose now that after $n \geqslant 0$ iterations of the while-loop, the pair (r_i, t_i) is retrieved

from the open list in Line 8. Let $f_{j,v} \in \rho(r_i, t_i)$ be the free time-window to be expanded, and the earliest exit time $t_{exit} = t_i + d^{inf}(r_i)$. Now there are two cases to consider:

1. $t_{exit} \leqslant \sigma_{j,v}$. In Line 14, the entry time into $f_{j,v}$ is determined to be $\sigma_{j,v}$. Clearly, the free time-window $f_{j,v}$ cannot be entered earlier than its start time $\sigma_{j,v}$, so the induction hypothesis also holds for the pair $(r_j, \sigma_{j,v})$ that is added to the queue.

2. $t_{exit} > \sigma_{j,v}$. The entry time into $f_{j,v}$ will be t_{exit}. To see that no earlier entry time into $f_{j,v}$ is possible, note that $\forall k \neq i, (r_k, t_k) \in Q : t_{exit} \leqslant t_k + d^{inf}(r_k)$ (this follows directly from Line 8). Hence, for any pair (r_k, t_k) such that $f_{j,v} \in \rho(r_k, t_k + d^{inf}(r_k))$, the entry time into $f_{j,v}$would be at least t_{exit}.
   A second point to note is that there will be no iteration $m > n$ such that a pair (r_m, t_m) can be inserted into Q, such that $t_m + d^{inf}(r_m) < t_{exit}$. If a new element (r_m, t_m) is inserted into the open list Q as a result of expanding $(r_k, t_k) \in Q$, for some $k \neq i$, then $t_m + d^{inf}(r_m) > t_m \geqslant t_k + d^{inf}(r_k) \geqslant t_{exit}$.
   Hence, there is no earlier entry time possible into window $f_{j,v}$ than t_{exit} and the pair (r_j, t_{exit}) satisfies the induction hypothesis.

It is now clear that in each step of the algorithm, a pair (r_i, t_i) is expanded to all free time-windows reachable from the free time-window determined by (r_i, t_i). This means that the free time-window $f_{j,v}$ can safely be removed in Line 17, because it is impossible to find a path that reaches this free time-window earlier.

The proposition now follows since also for the destination resource r_d it holds that if the pair (r_d, t_j) is taken from the open list, the backpointers to this pair represent an optimal path from r_s to r_d starting at time t_s. Hence, Algorithm 4.1 guarantees to find the first possible entry time into the first reachable time-window on destination resource r_d.■

Besides the fact that the algorithm is correct, it is also faster than any competing existing algorithm . The best known result is that of Kim and Tanchoco (1991). In the next section, an analysis is provided of the time complexity of our context-aware routing algorithm.

Time complexity of the single request variant Proposition 4.6 gives the run-time complexity of the single-agent context-aware source-destination algorithm. The proposition is followed by a proof. Subsequently, the run-time complexity is also expressed in terms of the transport network size and number of agents, to make it easier to compare this method to other approaches.

Proposition 4.6 *Algorithm 4.1 has a run-time complexity of* $\mathcal{O}(|F|\log(|F|) + |\rho|)$.

PROOF: A free time-window $f \in F$ can be put onto the open list Q at most once, and in every iteration of the while-loop in Line 7, one free time-window is removed from

Q. Hence, this while-loop is executed at most $|F|$ times, and none of the Lines 8-11 contribute more than $\mathcal{O}(\log(|F|))$.

An important observation is that Lines 14–18 (inside the for-loop) also execute at most $|F|$ times. In these lines, a free time-window will be added to Q, and that can occur at most $|F|$ times. Inserting an element into Q requires $\mathcal{O}(\log(|F|))$ time. Removal of the examined free time-window $f_{j,v}$ can also be done within $\mathcal{O}(\log(|F_j|)) = \mathcal{O}(\log(|F|))$ time.

Furthermore, at Line 13 every element of ρ might have to be inspected during the run of the algorithm – but no more than once. Hence, Line 13 contributes $\mathcal{O}(|\rho|)$ to the complexity of the algorithm.

Hence, Algorithm 4.1 has a run-time complexity of $\mathcal{O}(|F|\log(|F|) + |\rho|)$. ■

Usually, the run-time complexity of algorithms is specified in terms of the number of infrastructure resources and connection of the transport network. The same can be done here, after making the assumption that transport resources have at most a constant number c of reservations per infrastructure resource (if $c = 1$ this effectively disallows cyclic routes).

Corollary 4.7 *If transport resources cannot have more than a constant number of reservations per resource (e.g., only acyclic plans are allowed) then Algorithm 4.1 has a run-time complexity of* $\mathcal{O}(|R^{tr}||R^{inf}|\log(|R^{tr}||R^{inf}|) + |E_R||R^{tr}|)$.

PROOF: If transport resources can have at most a constant number c of reservations in a resource, each infrastructure resource can have at most $c|R^{tr}|$ reservations. This means $|F|$ is bounded by $c|R^{tr}||R^{inf}|$. At the same time, $|\rho| \leqslant c|E_R||R^{tr}|$, because for each arc in the transport network at most $c|R^{tr}|$ free time-windows can be reached in the resource the arc connects to.

Hence, we have that $|F|\log(|F|) + |\rho| \leqslant c|R^{tr}||R^{inf}|\log(c|R^{tr}||R^{inf}|) + c|E_R||R^{tr}|$. ■

Because agents usually have multiple transportation requests assigned to them at the same time, the next section describes how the context-aware algorithm can deal with a (visiting) sequence of pickup and delivery locations.

4.2.1.2 Single-agent context-aware visiting sequence routing

The source-destination routing described in Section 4.2.1.1 computes a shortest path in time for an agent to traverse from its initial location to a destination location. But in transportation planning, agents have to look further. They can be assigned multiple orders, and hence have to create a plan to visit multiple location after each other. The sequence of loading and unloading locations that the agent has to travel to is referred to as the *visiting sequence* of the agent.

Algorithm 4.2 Function VISITINGSEQUENCE computes a shortest path in time along loading and unloading resources in the plan of a transport resource, while taking into account capacity constraints of the vehicle as well as loading and unloading times.

```
1:  function VISITINGSEQUENCE(t, r_1, r_2, ..., r_m)
2:      Post: The shortest path in time along the visiting sequence
3:      (Rt_v, Sd_v) ← (r_1, t)
4:      for i ∈ 1..m do
5:          t ← Sd_{v,n} + (un)loading times in resource Rt_{v,n}
6:          (Rt'_v, Sd'_v) ← CONTEXTAWAREPATH(r_i, r_{i+1}, t)
7:          if (Rt'_v, Sd'_v) ≠ NOPOSSIBLEPATH then
8:              (Rt_v, Sd_v) ← (Rt_v, Sd_v)⌢(Rt'_v, Sd'_v)
9:          else
10:             return NOPOSSIBLEPATH
11:         end if
12:     end for
13:     return (Rt_v, Sd_v)
14: end function
```

The VISITINGSEQUENCE algorithm (Algorithm 4.2) computes shortest paths along the ordered pickup and delivery resources, i.e., the visiting sequence of resources, of the agent. Re-ordering of the resources in this visiting sequence, as well as exchanging them with other agents, is considered a decision at the tactical level.

Correctness of the multiple request variant The single-agent context-aware visiting sequence algorithm is in fact a series of single-agent context-aware source-destination route plannings. Hence, we already know that these plans are correct and only have to consider the visiting sequence resources themselves, the locations where the plans are concatenated.

This is, however, also very easy due to one of the invariants in Section 3.2.1. All visiting sequence resources are either the start or end location of the transportation resource, or it must be a pick-up or delivery resource. In all of these cases, the visiting sequence resource has sufficient capacity to make a conflict with other agents impossible.

Optimality of the multiple request variant In Chapter 3 the assumption was mentioned that loading and unloading resources have sufficient capacity to hold all transportation vehicles. This simplifies the application of the source-destination variant of context-aware routing to visiting sequence routing. Namely, the optimal route along the visiting sequence is a simple concatenation of the optimal routes between each pair of adjacent location in the visiting sequence. This can easily be seen, because the agent can wait indefinitely in any of the resources in the visiting sequence. It can, therefore, never

be an advantage to arrive later in any of the resources in the visiting sequence. Without the sufficient-capacity assumption one would have to consider multi-stage routing (Ter Mors, 2007), which can still efficiently be solved (in polynomial time).

In the next section one can see what happens to the time complexity of the single-agent context-aware routing algorithm if a set of transportation requests per agent is considered.

Time complexity of the multiple request variant In Section 4.2.1.1 it was shown that the single-agent context-aware source-destination algorithm (Algorithm 4.1) has a run-time complexity of $\mathcal{O}(|F|\log(|F|)+|\rho|)$.

The visiting sequence routing algorithm applies this algorithm at most twice for each transportation request $o \in O_v$ assigned to transport resource $v \in R^{tr}$ (once for the pickup location, once for delivery). This means Algorithm 4.2 has a run-time complexity of $\mathcal{O}(|F|\log(|F|)|O_v|+|\rho||O_v|)$.

With the assumption that all transportation plans are acyclic Algorithm 4.1 has a run-time complexity of $\mathcal{O}(|R^{tr}||R^{inf}||O_v|\log(|R^{tr}||R^{inf}|)+|E_R||R^{tr}||O_v|)$.

The single-agent context-aware visiting sequence algorithm can plan multiple transportation requests for a single agent, taking into account the reservations of other agents. In the next section, transportation planning for multiple agents is considered.

4.2.2 Multi-agent context-aware routing (MACA)

Until now this chapter considered the context-aware routing approach for a single agent, given fixed plans of the other agents. In the transportation problem, however, there usually is a whole fleet of agents for which transportation plans have to be constructed. The context-aware routing method can then be applied iteratively by all agents in the fleet, in an arbitrary order.

Clearly, in such an iterative approach, the quality of the plans of the agents depends on the *order* in which the agents create their transportation plans. Hence, the final set of plans of the agents might not be a global optimal set of plans (the plan an agent creates is still the best it can do, given the plans of the agents who preceded in the planning process remain fixed). Furthermore, there might not even exist an ordering of agents that results in a global optimal set of plans, due to the fact that the agents create a complete transportation plan one by one.

For the multi-agent version of context-aware routing an event-based approach is used. Each time a new transportation request is assigned to an agent, this agent searches a modified plan which includes the newly arrived transportation request. Hence, the arbitrary order in which the agents plan in this case is controlled by the order in which the transportation requests are assigned to the agents.

This new method (see Algorithm 4.3), used each time a new transportation request is

Algorithm 4.3 PROCESSNEWREQUEST method using insertion heuristic.

1: **function** PROCESSNEWREQUEST($f \in F, s \in R^{inf}, \tau^s \in W, d \in R^{inf}, \tau^d \in W, \pi \in \mathbb{R}$)
2: **Pre:** New request is assigned to transport resource $v \in R^{tr}$ of agent $a \in A$.
3: **Post:** Plan $P_v = (Rt_v, Sd_v)$ includes the execution of the new request.
4: **for all** $(i, j) : 2 \leqslant i < j \leqslant n+1$ **do** ▷ pickup detour at index i, delivery at j.
5: $(Rt'_v, Sd'_v) \leftarrow$ VISITINGSEQUENCE$(Sd_{v,1}, Rt_{v,1}, \ldots, Rt_{v,i-1}, s, Rt_{v,i}, \ldots, \ldots, Rt_{v,j-1}, d, Rt_{v,j}, \ldots, Rt_{v,n})$
6: **if** $\mu(Rt'_v, Sd'_v) > \mu(Rt_v, Sd_v)$ **then**
7: $(Rt_v, Sd_v) \leftarrow (Rt'_v, Sd'_v)$
8: **end if**
9: **end for**
10: **end function**

assigned to an agent, is called PROCESSNEWREQUEST. This method makes use of the VISITINGSEQUENCE algorithm described in the previous section (Algorithm 4.2), which computes shortest paths along the sequence of pickup and delivery resources, i.e., the visiting sequence of resources, of the agent.

The agents execute the PROCESSNEWREQUEST method in the order in which the transportation requests arrived and without interleaving the planning with other agents. Due to both of these factors the resulting plans are sub-optimal. First, due to the arbitrary ordering: suppose that agent a_1 plans earlier than agent a_2 in this ordering of agents, and that agents a_1 and a_2 share some infrastructure resources in their routes. If an optimal plan requires that agent a_2 precedes agent a_1 in at least one of these share infrastructure resources, this plan might not be found (in the case that agent a_1 reaches the infrastructure resource earlier than agent a_2). Second, due to the absence of interleaved planning: if agent a_1 first has to precede agent a_2, but later has to take priority in any optimal plan, then such a plan also cannot be found, because the agents create a complete plan for all of their transportation requests at once when it is their turn.

Algorithm 4.3 describes how a new transportation request is inserted into the possibly already existing plan of an agent. Note that, for simplicity, several algorithmic optimizations have been ignored here. For example, if a detour index j is not allowed due to capacity violation, one does not need to check indexes greater than j; instead, index i can be increased. Furthermore, routes can be cached. During the computation many similar routes are searched, often with more or less the same starting time. Hence, some time can be gained.

4.2.2.1 Correctness of the multi-agent variant

In the multi-agent context-aware algorithm an agent modifies its current route by inserting a pick-up detour for an additional transportation request and a delivery detour as well. It

then compares its plan with the old version and if its performance improves, the agent accepts the additional transportation request and modifies its plan accordingly.

Because this is assumed to be an atomic operation, and we already know the single-agent visiting sequence routing algorithm is correct, no conflicts can arise from this process. Therefore, assuming a conflict-free starting point, the multi-agent context-aware routing also results in a set of transportation plans that is also free of conflicts.

4.2.2.2 Optimality of the multi-agent variant

Section 4.2.1.2 showed that the single-agent context-aware visiting sequence algorithm correctly computes a shortest path in time for a single agent given reservations of the other agents.

The multi-agent context-aware routing applies this algorithm in a sequence. At each iteration, an agent uses Algorithm 4.2 of which it is known this produces a correct plan given the reservations of other agents. Hence, the number of reservations grows while more agents constructed their transportation plans and each new transportation plan does not conflict with any of the already existing plans. Therefore it can be concluded that the set of transportation plans that exists after the final agent constructed its transportation plan is a correct set of plans that can be executed without any problems (assuming execution goes according to the plans).

Furthermore, assuming no incidents and no hard pickup or delivery deadlines, a transportation plan that executes all transportation requests correctly always exists and is always found by the multi-agent context-aware algorithm. It can easily be seen that a plan exists for all of the agents, by considering the following: let each agent wait in its initial location (which by definition has sufficient capacity). Then select one by one all of the agents and let the selected agent execute all of its assigned transportation requests. The next agent selected only starts after the previous agent finished execution of all of its requests. Hence, in this plan no conflicts are possible, because no two agents reside in a location with limited capacity, a necessary condition for a conflict. This proves that a multi-agent conflict-free transportation plan always exists. If a better plan does not exists, Algorithm 4.2 will always find this plan.

In the next section the time complexity of the multi-agent context-aware routing algorithm is described.

4.2.2.3 Time complexity of the multi-agent variant

In Section 4.2.1.2 it was shown that the single-agent context-aware visiting sequence algorithm has a run-time complexity of $\mathcal{O}(|F|\log(|F|)|O_v| + |\rho||O_v|)$ and with the assumption that all plans are acyclic a run-time complexity of $\mathcal{O}(|R^{tr}||R^{inf}||O_v|\log(|R^{tr}||R^{inf}|) + |E_R||R^{tr}||O_v|)$.

In the multi-agent context-aware routing algorithm all agents compute their transportation plans in a sequence. The total number of context-aware shortest-path computations is simply twice the number of transportation requests, i.e., once for each pickup and once for each delivery location. Hence, the time-complexity of the multi-agent context-aware routing algorithm is $\mathcal{O}(|F|\log(|F|)|O| + |\rho||O|)$ and with the assumption that all plans are acyclic a run-time complexity of $\mathcal{O}(|R^{tr}||R^{inf}||O|\log(|R^{tr}||R^{inf}|) + |E_R||R^{tr}||O|)$.

In this context-aware routing section we have described the single-agent context-aware routing algorithm and how to construct a multi-agent variant by applying SACA iteratively. We are after all interested in searching transportation plans for a set of vehicles. The next section presents some interesting properties of the MACA algorithm and compares MACA to other approaches.

4.2.3 Properties of the multi-agent context-aware routing

This section describes properties of the MACA approach. First, the quality of the solution is considered. Subsequently, a comparison is made between the classical approach presented in Section 4.1 and the multi-agent context-aware routing approach. Finally, the approach is compared to two related approaches; that of Kim and Tanchoco (1991) and of Hatzack and Nebel (2001).

4.2.3.1 Solution quality

A Nash equilibrium (Nash, 1950) is a situation in which no single agent is capable of improving its performance by making changes to its plan, *without* other agents changing their plans as well. The importance of reaching a Nash equilibrium is that the situation is stable; no single agent can make any progress modifying its plan, without forcing other agents to make other decisions.

The multi-agent context-aware routing approach always results in a Nash equilibrium – a local optimum. Each of the agents computes an optimal plan, given the reservations of other agents at that time. Then it immediately creates reservations for this plan. Hence, none of the agents can improve its plan, without forcing other agents to make other choices.

In general, however, the multi-agent context-aware routing approach results in a local optimum, not a globally optimal solution. Finding a global optimum, however, is a much more difficult problem, because, for each possible conflict, all possible orderings of agents must then be considered.

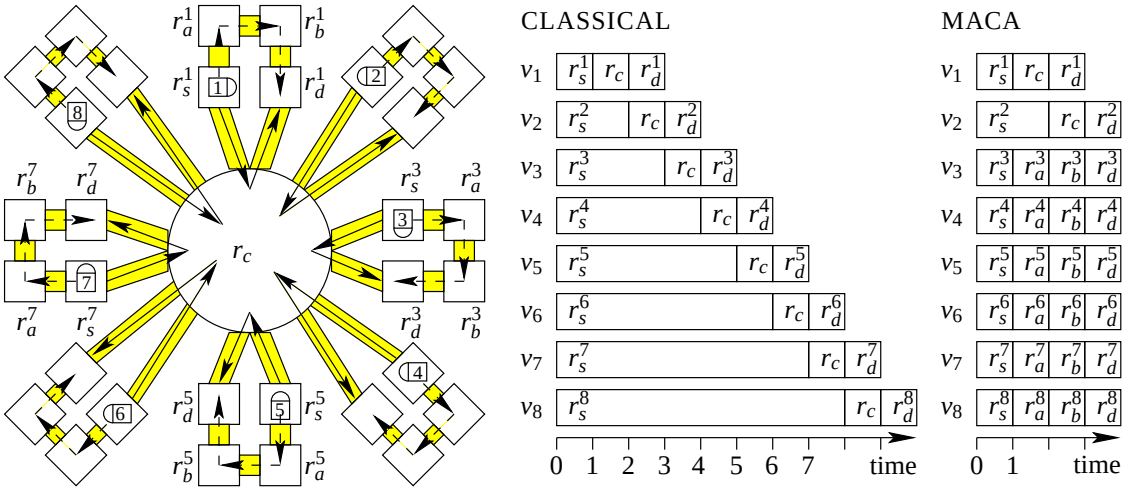


Figure 4.3: classical versus context-aware approach. Each agent can choose to travel to its destination via the center resource r_c following the solid arrows, or take the alternative route given by the dashed arrows.

4.2.3.2 Comparing MACA to CLASSICAL

In this section, the classical approach is compared to the context-aware method by giving examples. The first example shows how a central resource can become a bottleneck in the classical approach, while the context-aware approach easily finds alternative, non-congested routes. In this example the context-aware method significantly outperforms the classical approach. The second example illustrates that the freedom of the context-aware planner can sometimes lead to plans that make it harder for subsequent agents to find efficient plans. This shows that the classical approach can be superior too.

Based on these examples a general conclusion cannot be drawn. Therefore, in the next chapter experimental evidence is used to compare both approaches.

Example 4.8 Suppose that there are m agents in the transport network illustrated by Figure 4.3. Each agent a_i has the choice between two alternative routes: *(i)* from its initial location via the center resource r_c to its destination resource, (r_s^i, r_c, r_d^i), or *(ii)* take the outer detour, which is one step longer, following the dashed arrows, $(r_s^i, r_a^i, r_b^i, r_d^i)$. Using the classical approach, all of the m agents will choose the former route, traversing resource r_c. The first agent will then reach its destination at time 3, the second agent has to wait 1 time unit, the third agent one more, etc. This results in total costs $3+4+\ldots+(m+2) = (m+2)(m+3)/2-3$ and a makespan of $m+2$. If Dijkstra's shortest path algorithm with Fibonacci heap was used, the time complexity would be $\mathcal{O}(|R^{tr}| \cdot |R^{inf}| \log(|R^{inf}|) + |E_R|)$, where R^{tr} is the set of transport resources, R^{inf} the set of infrastructure resources and E_R the set of connections between infrastructure resources.

If the context-aware method is used, the first agent would compute the same plan

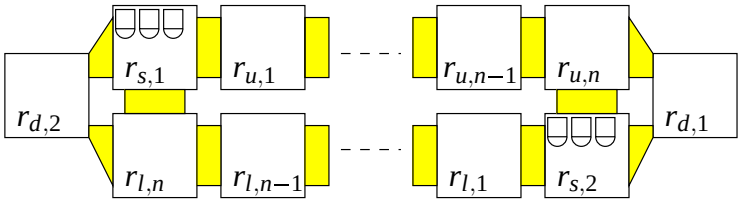


Figure 4.4: Another example comparing CLASSICAL to MACA. The vehicles in resource $r_{s,1}$ are named $v_{1,1}$, $v_{1,2}$ and $v_{1,3}$, and the vehicles in resource $r_{s,2}$ are named $v_{2,1}$, $v_{2,2}$ and $v_{2,3}$.

and go via resource r_c. The second agent would select either way, with equal probability. Let us say that it would also go via resource r_c. Then, all other agents will avoid resource r_c and take the alternative route. The total costs now are $3+4+4+\ldots+4 = 3+4(m-1)$ and the makespan is only 4 (or 3 if $m = 1$). Time complexity is $\mathcal{O}(|R^{tr}||R^{inf}||O|\log(|R^{tr}||R^{inf}|) + |E_R||R^{tr}||O|)$.

Both of the given performance indicators, summed total costs and makespan, have improved by a factor m and the context-aware method is better for any number of agents $m > 2$ in this example (and equal for $m \leqslant 2$). □

The context-aware method has some amount of overhead in computation time by solving conflicts in advance (although it takes no more than polynomial time). We expect the context-aware method to be somewhat slower than the classical approach, even though, if the classical approach is used, conflicts will still have to be solved during the execution phase.

Due to the arbitrary ordering in which agents plan, it is also possible, however, to construct a negative example in which the classical approach outperforms the multi-agent context-aware method.

Example 4.9 This time consider Figure 4.4. Its infrastructure has two long corridors of resources. Three vehicles in resource $r_{s,1}$ want to go to resource $r_{d,1}$, while the three vehicles in resource $r_{s,2}$ want to go to resource $r_{d,2}$. All locations, except for the pick-up locations $r_{s,1}$ and $r_{r,2}$, and the delivery locations $r_{d,1}$ and $r_{d,1}$, have capacity 1. Furthermore, all resources have the same constant traversal time, say 1 time unit.

Assuming the vehicles are identical (except for their initial location), there are $\binom{6}{3} = 20$ possible sequences in which the vehicles can plan. Half of these are listed in Table 4.5. The first column displays the order in which the vehicles planned. Notice that the missing ten rows can be found by swapping $v_{1,i}$ with $v_{2,i}$ for $i \in \{1,2,3\}$ in the first column. These are the same due to the symmetry. The second column shows the performance of the vehicles, which is here computed as the sum of the travel times of the six vehicles. The third column displays the makespan, which is the worst performance (the maximum individual vehicle performance).

Order						Performance	Makespan
$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$9n+17$	$2n+4$
$v_{1,1}$	$v_{1,2}$	$v_{2,1}$	$v_{1,3}$	$v_{2,2}$	$v_{2,3}$	$6(n+3)$ (50%), $9n+17$ (50%)	$n+4$, $2n+4$
$v_{1,1}$	$v_{1,2}$	$v_{2,1}$	$v_{2,2}$	$v_{1,3}$	$v_{2,3}$	$6(n+3)$ (50%), $9n+17$ (25%), $11n+15$ (25%)	$n+4$, $2n+4$, $3n+2$
$v_{1,1}$	$v_{1,2}$	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{1,3}$	$6(n+3)$ (50%), $11n+15$ (50%)	$n+4$, $3n+2$
$v_{1,1}$	$v_{2,1}$	$v_{1,2}$	$v_{1,3}$	$v_{2,2}$	$v_{2,3}$	$6(n+3)$ (100%)	$n+4$
$v_{1,1}$	$v_{2,1}$	$v_{1,2}$	$v_{2,2}$	$v_{1,3}$	$v_{2,3}$	$6(n+3)$ (100%)	$n+4$
$v_{1,1}$	$v_{2,1}$	$v_{1,2}$	$v_{2,2}$	$v_{2,3}$	$v_{1,3}$	$6(n+3)$ (100%)	$n+4$
$v_{1,1}$	$v_{2,1}$	$v_{2,2}$	$v_{1,2}$	$v_{1,3}$	$v_{2,3}$	$6(n+3)$ (100%)	$n+4$
$v_{1,1}$	$v_{2,1}$	$v_{2,2}$	$v_{1,2}$	$v_{2,3}$	$v_{1,3}$	$6(n+3)$ (100%)	$n+4$
$v_{1,1}$	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{1,2}$	$v_{1,3}$	$6(n+3)$ (100%)	$n+4$
					Minimum	$6(n+3)$	$n+4$
					Expected	$\frac{69}{10}n+\frac{88}{5}$	$\frac{53}{40}n+\frac{77}{20}$
					Maximum	$11n+15$	$3n+2$

Table 4.5: Possible performance outcomes for Example 4.9.

$v_{1,1}$	$r_{s,1}$	$r_{u,1}$	...	$r_{u,n}$	$r_{d,1}$			n+2
$v_{2,1}$	$r_{s,2}$	$r_{l,1}$	...	$r_{l,n}$	$r_{d,2}$			n+2
$v_{1,2}$	$r_{s,1}$		$r_{u,1}$	...	$r_{u,n}$	$r_{d,1}$		n+3
$v_{2,2}$	$r_{s,2}$		$r_{l,1}$	...	$r_{l,n}$	$r_{d,2}$		n+3
$v_{1,3}$	$r_{s,1}$			$r_{u,1}$	...	$r_{u,n}$	$r_{d,1}$	n+4
$v_{2,3}$	$r_{s,2}$			$r_{l,1}$	...	$r_{l,n}$	$r_{d,2}$	n+4
								6(n+3)

Table 4.6: An optimal plan for Example 4.9. This plan represents that all vehicles starting in resource $r_{s,1}$ take the upper route and all vehicles starting in resource $r_{s,2}$ take the bottom route.

The MACA algorithm often finds the optimal solution, which is the plan shown in Table 4.6. It can go wrong if the planning sequence starts with two vehicles and the second one chooses the alternative route. A sub-optimal plan will always be reached if the three vehicles in resource $r_{s,1}$ plan before the vehicles in resource $r_{s,2}$, or vice versa.

Table 4.7 shows the final plan if first all vehicles in resource $r_{s,1}$ reserve their plans, and then the vehicles in resource $r_{s,2}$. The optimal plan cannot be constructed in this case. Although the first vehicle will always choose the upper route, because this is the fastest, the the second vehicle might also (now the two alternatives are equidistant), the third vehicle will then surely choose the bottom route. Therefore, the optimal route will never be constructed in this case. This corresponds to the first row of Table 4.5.

$v_{1,1}$	$r_{s,1}$	$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$						$n+2$
$v_{1,2}$	$r_{s,1}$			$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$				$n+3$
$v_{1,3}$	$r_{s,1}$	$r_{l,n}$	$\ldots$	$r_{l,1}$	$r_{s,2}$	$r_{d,1}$					$n+3$
$v_{2,1}$	$r_{s,2}$				$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$			$2n+2$
$v_{2,2}$	$r_{s,2}$					$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$		$2n+3$
$v_{2,3}$	$r_{s,2}$						$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$	$2n+4$
											$9n+17$

Table 4.7: A sub-optimal plan for Example 4.9. If vehicles $v_{1,1}$ and $v_{1,2}$ both use the upper route, then vehicle $v_{1,3}$ always chooses the bottom route, because it then reaches resource $r_{d,1}$ faster. Hence, the optimal plan is never reached if the three vehicles in resource $r_{s,1}$ reserve their plans first (and vice versa).

$v_{1,1}$	$r_{s,1}$	$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$						$n+2$
$v_{1,2}$	$r_{s,1}$	$r_{l,n}$	$\ldots$	$r_{l,1}$	$r_{s,2}$	$r_{d,1}$					$n+3$
$v_{2,1}$	$r_{s,2}$				$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$			$2n+2$
$v_{1,3}$	$r_{s,1}$			$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$				$n+3$
$v_{2,2}$	$r_{s,2}$					$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$		$2n+3$
$v_{2,3}$	$r_{s,2}$						$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$	$2n+4$
											$9n+17$

Table 4.8: A sub-optimal plan for Example 4.9, which occurs if vehicles $v_{1,1}$ and $v_{1,2}$ do not choose the same route.

If first two vehicles in resource $r_{s,1}$ plan, then one vehicle from resource $r_{s,2}$, immediately followed by the remaining vehicle of resource $r_{s,1}$, then there are two possibilities that are even likely. Either vehicles $v_{1,1}$ and $v_{1,2}$ both choose the upper route and the result is the optimal plan as illustrated in Table 4.6, or vehicle $v_{1,2}$ selects the alternative route and the result is as given by Table 4.8 which has a performance of $9n+17$..

The third row of Table 4.5 shows three possibilities. The planning order of the vehicles here is $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}, v_{1,3}, v_{2,3})$. If the first two vehicles choose the upper route, the optimal plan is reached. Because vehicle $v_{1,1}$ always chooses the upper route, this happens in 50% of all cases. For the other 50% of the cases, the upper and bottom part of Table 4.9 are both even likely. The difference between those two is whether vehicle $v_{2,2}$ will choose the upper or bottom route, which have exactly the same distance. Which choice $v_{2,2}$ makes, however, has a great influence on the performance of vehicle $v_{1,3}$.

The final situation that can result in a sub-optimal plan for the vehicles is summarized on the fourth row of Table 4.5. Again, if vehicles $v_{1,1}$ and $v_{1,2}$ would both choose the upper route, then the optimal plan would have been found. The other 50% results in the plan given by Table 4.10.

Table 4.5 lists all possible performance outcomes of this example. The best performance of $6(n+3)$ is found in 75% of all possible orderings in which the vehicles can plan.

$v_{1,1}$	$r_{s,1}$	$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$							$n+2$
$v_{1,2}$	$r_{s,1}$	$r_{l,n}$	$\ldots$	$r_{l,1}$	$r_{s,2}$	$r_{d,1}$						$n+3$
$v_{2,1}$	$r_{s,2}$				$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$				$2n+2$
$v_{2,2}$	$r_{s,2}$					$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$			$2n+3$
$v_{1,3}$	$r_{s,1}$		$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$						$n+3$
$v_{2,3}$	$r_{s,2}$						$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$		$2n+4$
												$9n+17$

$v_{1,1}$	$r_{s,1}$	$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$							$n+2$
$v_{1,2}$	$r_{s,1}$	$r_{l,n}$	$\ldots$	$r_{l,1}$	$r_{s,2}$	$r_{d,1}$						$n+3$
$v_{2,1}$	$r_{s,2}$				$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$				$2n+2$
$v_{2,2}$	$r_{s,2}$				$r_{u,n}$	$\ldots$	$r_{u,1}$	$r_{s,1}$	$r_{d,2}$			$2n+3$
$v_{1,3}$	$r_{s,1}$							$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$	$3n+2$
$v_{2,3}$	$r_{s,2}$					$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$			$2n+3$
												$11n+15$

Table 4.9: Assuming that vehicles $v_{1,1}$ and $v_{1,2}$ make a different choice, there are two equidistant choices for vehicle $v_{2,2}$. Two sub-optimal plans for Example 4.9.

$v_{1,1}$	$r_{s,1}$	$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$							$n+2$
$v_{1,2}$	$r_{s,1}$	$r_{l,n}$	$\ldots$	$r_{l,1}$	$r_{s,2}$	$r_{d,1}$						$n+3$
$v_{2,1}$	$r_{s,2}$				$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$				$2n+2$
$v_{2,2}$	$r_{s,2}$					$r_{l,1}$	$\ldots$	$r_{l,n}$	$r_{d,2}$			$2n+3$
$v_{2,3}$	$r_{s,2}$				$r_{u,n}$	$\ldots$	$r_{u,1}$	$r_{s,1}$	$r_{d,2}$			$2n+3$
$v_{1,3}$	$r_{s,1}$							$r_{u,1}$	$\ldots$	$r_{u,n}$	$r_{d,1}$	$3n+2$
												$11n+15$

Table 4.10: The final sub-optimal plan for Example 4.9. Note that the plans of the vehicles $v_{2,2}$ and $v_{2,3}$ might be swapped, which results in the same performance.

In 7.5% of the possible orderings the worst plan with performance of $11n+15$ is found. In that case the performance is almost twice as bad as the classical approach, which will always find the optimal solution to the problem of this example. Furthermore, the expected value of the performance is $69n/10+88/5$, which is about a factor $7/6$ worse than the classical approach. With respect to the worst agent, which always has performance $n+4$ in the classical approach, we can see that MACA in the worst case generates a plan that is about a factor 3 worse. It can be concluded from this example that, in some situations, the classical approach outperforms the multi-agent context-aware approach. □

This section has shown that there are situations in which the context-aware routing significantly outperforms the classical approach. However, it also showed that the opposite can be true. This means it cannot be said which method is the better one at the moment. Extensive experiments are required that test and compare context-aware routing to the

classical approach. Empirical evidence gathered by the experiments will be presented in the next chapter.

In the next section context-aware routing is compared to related approaches. The most important differences are described with two approaches that were already described in Chapter 2, which are Hatzack and Nebel (2001) and Kim and Tanchoco (1991).

Comparison with related approaches In this section our multi-agent context-aware routing method is compared to two related reservation-based approaches, which also attempt to solve the routing problem for a fleet of vehicles.

The first related approach is Kim and Tanchoco (1991), see Section 2.2.4.1. Fujii et al. (1989) inspired us with the notion of free time-windows and Kim and Tanchoco (1991) present an algorithm, accompanied by a comprehensive analysis, for free time-window routing.

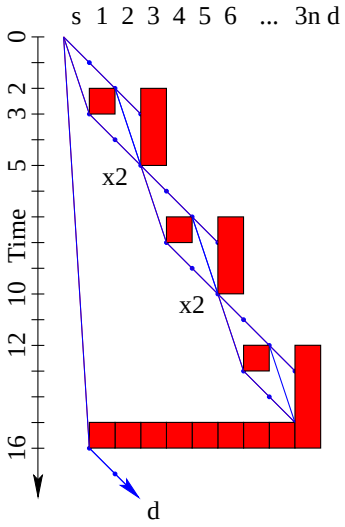
The second related approach is that of Hatzack and Nebel (2001), see Section 2.2.4.2. They present a fast algorithm to solve what they call the traffic control problem. Their algorithm is used to route a fleet of airplanes taxiing on the ground. Although the algorithm might in practice have turned out to be fast, we detected a flaw in the algorithm, which results in the fact that the computation time of their algorithm is not polynomially bounded.

Comparing MACA to Kim & Tanchoco In Section 2.2.4.1 the free time-window graph routing algorithm of Kim and Tanchoco (1991) was presented. The run-time complexity of their single-agent algorithm is $\mathcal{O}(|R^{tr}|^4|R^{inf}|^2)$, while our single-agent run-time complexity is $\mathcal{O}(|R^{tr}||R^{inf}|\log(|R^{tr}||R^{inf}|) + |E_R||R^{tr}|^2)$.

The reason for this difference is that their conflict detection procedure is inefficient. Kim and Tanchoco (1991) did not make use of the fact that it is enough to consider only the direct successor and predecessor to check for catching-up conflicts (instead, they iterated through all present transport resources).

Furthermore, they used a different framework in which they both had to check for conflicts in the locations, as well as on lanes. In our framework, presented in Chapter 3, lanes are also modeled as resources with the same properties as locations.

Comparing MACA to Hatzack & Nebel In this section a comparison is made between multi-agent context-aware routing and the approach by Hatzack and Nebel (2001) (see Section 2.2.4.1). The approach of Hatzack and Nebel (2001) consisted of two phases. In the first phase each agent computes a context-unaware shortest path from its current to its destination location. In the second phase, one by one the agents create a schedule that does not conflict with schedules other agents created previously. At the end, all agents have a plan and the joint plan is guaranteed to be free of conflicts.



n	Algorithm 4.1 [s]	H&N [s]	recursive calls
1	0	0	10
2	0	0	26
4	0	0	110
8	0	1.0	1,682
16	0	2.6	426,026
20	0	41.5	6,815,798
24	0	696.3	109,051,970
10,000	2.3	-	-
20,000	12.1	-	-

Figure 4.11: Difficult instances for Algorithm 2.2 of Hatzack and Nebel. The figure on the left illustrates how the instances are created, the table on the right gives the time required to find a plan for different problem sizes n, and the number of recursive calls used by the algorithm of Hatzack and Nebel.

Hatzack and Nebel (2001) do not take free time-windows into account in their algorithm, obviously not in the first phase, but also not in the second phase. The importance of free time-windows is the fact that algorithms can benefit greatly from the fact that only the earliest possible time a free time-window can be visited has to be considered by an algorithm. Any computations that visit the same free time-window at a later time are superfluous, because one could have used the earlier visit and just wait, resulting in the same plan costs.

The importance of considering which free time-windows are already visited can be made clear by looking at the difference in performance (with respect to CPU cost in time) between Algorithm 2.2 of Hatzack and Nebel and Algorithm 4.1 on a special set of problem instances. This example exploits the fact that the worst-case time complexity of Algorithm 2.2 of Hatzack and Nebel is not polynomially bounded.

Their algorithm makes use of backtracking, and since they do not make use of the idea that a free time-window needs to be expanded at most once, it is possible to construct examples in which the algorithm keeps backtracking through the same paths of time windows.

Figure 4.11 depicts an example where this worst-case behavior is realized. The figure illustrates the reservations on the sequence of resources (horizontally from source resource r_s via $r_1, r_2, r_3 \ldots$ to destination resource r_d); vertically, the progress of time is shown. Each resource is assumed to have a traversal time of 1. To create such an instance where their algorithm needs $2^n + 1$ updates, no more than the following $5n$ reservations

are needed in a route $r_s, r_1, r_2, \ldots, r_{3n}, r_d$ of $3n+2$ resources:

- resources r_{3i-2} are reserved during $[5i-3, 5i-2)$ for $1 \leqslant i \leqslant n$,
- resources r_{3i} are reserved during $[5i-3, 5i)$ for $1 \leqslant i \leqslant n$,
- resources r_i are reserved during $[5n, 5n+1)$ for $1 \leqslant i \leqslant 3n$.

The table shown in Figure 4.11 clearly shows that if such a structure of reservations occurs Algorithm 2.2 of Hatzack and Nebel will not be able to solve the instance within acceptable time, while Algorithm 4.1 has no problem with these instances at all.

Another difference to other approaches, such as the classical approach, is that our framework is flexibly due to the separation between infrastructure and transport agents. The next section describes several conflict-resolution rules that can be used by the infrastructure agents to prioritize the agents if a conflict occurs.

4.2.4 Refined conflict-resolution rules

Our framework presented in Chapter 3 separates between transport agents and infrastructure agents. The infrastructure agents compute reservations for the transport agents while ensuring a global situation that is free of conflicts. To do so, the infrastructure agents must prioritize agents with conflicting plans. The infrastructure agent uses conflict-resolution rules to determine which agents precedes the others, and the other agents have to wait until it is their turn to access the congested infrastructure resource.

We distinguish between *simple* and *plan-based* conflict-resolution rules. The difference is that in case of *simple* rules the priority values are not directly dependent on the plans of agents, while the *plan-based* rules are. Furthermore, one can also distinguish between *static* and *dynamic* rules (see Table 4.12), where dynamic rules use information that changes during plan execution, while static rules do not. All plan-based heuristics are assumed to be dynamic (because the plan of an agent changes over time).

The following are examples of simple rules:

- Randomly assign an entity precedence.
- First-In-First-Out (FIFO): an entity that arrived first takes precedence.
- Longest queue takes precedence: an entity residing in the longest waiting queue takes precedence.
- Longest queue with increment: the same as above, though here starvation is taken into account. It could happen that an agent waits alone for a crossroad where other agents arrive continuously taking precedence. This traffic rule virtually increments the queue length with 1 of each queue that is not chosen as the longest queue.

Plan-based	N/A	Longest plan Urgent deadline Decreasing reward Decreasing reward sum
Simple	Random	Longest queue (inc) First-In-First-Out
	Static	Dynamic

Table 4.12: Distribution of the resource usage rules over simple versus plan-based rules and static versus dynamic rules.

The following rules are plan-based, they take the plans of the agents into account. Within the scope of this thesis, it is assumed that all agents are honest and benevolent.

- An agent with the longest plan takes precedence.
- An agent with the most urgent deadline takes precedence.
- Each agent can compute the decrease in reward resulting from having to wait. An agent with this maximum decrease takes precedence.
- The previous decrease in reward due to waiting can also be summed for the queue. The queue having the maximum decrease takes precedence.

In the next chapter the performance of these conflict-resolution rules are compared to each other by presenting the results of experiments in which the conflict-resolution rule used by the infrastructure agents is varied. The next section describes several ways to refine the MACA algorithm to deal with the arbitrary ordering in which the agents plan, and to make it more robust to incidents.

4.3 Dealing with uncertainty

Typically, the assumptions made at planning time do not always hold at execution time, either due to calibration errors or disturbances. For instance, at unpredictable times, agents can be slower or faster in executing their actions. Uncertainty refers to both these modeling errors and incidents, such as described in Section 3.1.4.

In principle, incidents do not offer any problem to the classical approach described in Section 4.1, since it is assumed that operational conflict resolution methods should be sufficient to handle conflicts due to incidents as well.

We know that the multi-agent context-aware routing algorithm performs well under normal circumstances. An important question, however, for every route-planning system is, how robust it is, i.e., how it behaves in case incidents are happening. It might be

Algorithm 4.4 PROCESSINCIDENT method.

1: **function** PROCESSINCIDENT($t_r \in T, r \in R, 0 \leqslant i \leqslant 1, \tau \in W$)
2: **Pre:** An incident occurred that possibly influences the plan of transport resource $v \in R^{tr}$ of agent $a \in A$.
3: **Post:** Agent a updates the plan P_v for vehicle $v \in R^{tr}$, if needed.
4: **if** $r \in Rt_v \vee r = v$ **then**
5: $P_v \leftarrow$ VISITINGSEQUENCE(t_r, COMPUTEVISITINGSEQUENCE(Rt_v))
6: **end if**
7: **end function**

the case that in incident-rich conditions, any work done on solving conflicts between the agents a priori is rendered useless by the occurrence of unexpected failures, and, hence, the classical approach actually leads to the same agent performance.

Of course, one could add operational conflict resolution systems to context-aware routing systems just to complement them and to remove any conflicts due to incidents. But then it easily might turn out that such systems are not better than the classical route planning approach. Therefore, we propose to improve the context-aware routing method itself to deal with such conflicts, integrating execution-time conflict resolution with re-planning.

This section describes several approaches that increase the robustness of the system in environments where incidents do occur. First of all, the multi-agent context-aware routing described in Section 4.2.2 is adapted. Besides the PROCESSNEWREQUEST method, one can add the PROCESSINCIDENT method, which is used by the agents each time an incident occurs. Algorithm 4.4 presents the function that is called if an incident occurs. The affected resource $r \in R$ can either be a transport resource, or an infrastructure resource. If it is determined that the incident might affect the plan of the agent, the agent will recompute its plan.

Furthermore, in this section two types of refinements are described. With respect to the *execution stage*, simple and plan-based conflict resolution rules are considered. It might be the case that some rules work better in incident-rich situations, while other perform better under normal circumstances. These rules will be tested and compared in experiments in Chapter 5.

But there are also two revisions of the multi-agent context-aware routing algorithm. The first is to revise priorities. The plans of the agents were constructed with the assumption that no incident would occur. If an incident does occur, it makes sense to reconsider the priorities of the agents that are no longer to execute their current plan due to the incident(s).

The second revision to the multi-agent context-aware routing algorithm reconsiders, besides priorities due to incidents, also the routes chosen by the agents. With the addi-

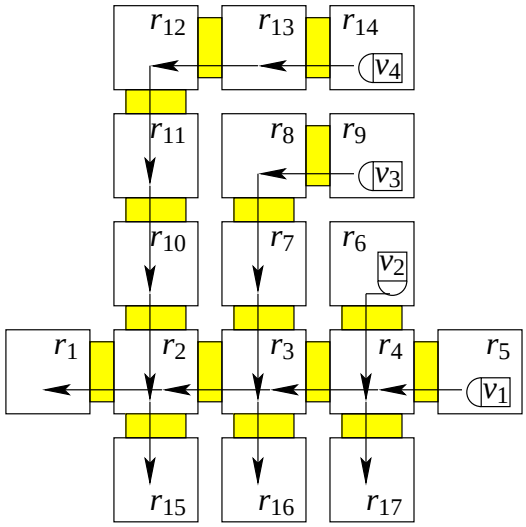


Figure 4.13: Example that shows the improvement of the MACA-RP method. Suppose that vehicle v_1 plans at a later time than the other vehicles, and its target resource is r_1. It turns out that has to wait in all resources, and its plan will be $P_1 = (r_5, [0,2), r_4, [2,4), r_3, [4,6), r_2, [6,7), r_1, [7,\infty))$. If any vehicle, preferably v_2, would give way to vehicle v_1, that would be of great help. This is exactly what the MACA-RP method can arrange.

tional information about the incident that occurred, an agent might prefer to take a detour. The MACA-RR method considers alternative routes while replanning the priorities. An important property of this method is that the spread of agents over the infrastructure after an incident occurs is improved with respect to the other methods.

In the next chapter, experiments are presented that will test and compare the rules listed in this section. Empirical evidence will show which rules perform best under normal circumstances, and which perform best in incident-rich conditions.

The next section describes the MACA-RP method, which reconsiders priorities each time new information, such as incidents, becomes available.

4.3.1 Revising priorities (MACA-RP)

In the previous approaches, if agents accepted new transportation requests and changed their reservations accordingly, these reservations were permanent. Other agents planned around these reservations. It was already noted that with this approach the welfare of the system depends on the order in which the agents create the reservations for their transportation plans. The performance can be improved by reconsidering the potential conflicts.

Due to the arbitrary order in which the agents created their initial transportation plans, unlucky choices might have been made. The *essence* of the MACA-RP method is to attempt to improve the performance of the agents by re-evaluating the resource usage rules now that more (or even less) transportation requests are assigned to the agents.

Regularly, the agents can request to revise priorities. This can be started, for instance, when an agent has accepted a new transportation request, a transportation request has been modified, or when an agent is bothered by an incident on its path. Algorithm 4.5 can be used to revise the priorities. Typically, the group of agents that revise priorities is

Algorithm 4.5 MACA-RP.

```
1:  function MACA-RP(v ∈ R^tr', a_s ∈ A, b ∈ ℕ)
2:      Pre: Vehicle v, part of the set of rescheduling vehicles R^tr' ⊆ R^tr, reschedules in
    blocks of size b
3:      Post: Vehicle v has an up-to-date schedule
4:      ADDTOGROUP(a_s, v)
5:      Sd_v ← null
6:      while |Sd_v| < |Rt_v| do
7:          Compute provisional schedule Sd'_v for index i to the end
8:          Compute selection value v_v = h(P_v)
9:          winner ← HASHIGHESTPRIORITY(a_s, v_v)
10:         if winner then
11:             i ← RESERVEBLOCK(v, Rt_v, Sd_v, Sd'_v, i, i + b)
12:         end if
13:     end while
14:     LEAVEGROUP(a_s, v)
15: end function
```

formed by the agent that requested the rescheduling together with all agents that share at least one infrastructure resource with the requesting agent. This set of agents can quickly be determined by looking at the reservations of agents for the infrastructure resources in the plan of the requesting agent.

To coordinate the group of rescheduling agents, the algorithm makes use of a scheduling agent $s_a \in A$. This can just be one of the agents within the group, or a special set of agents for the purpose of rescheduling that can be trusted by all the agents. There can be more than one of these scheduling agents and each scheduling agent could take care, for example, for a certain geographical area.

If the request to revise priorities is granted by a scheduler agent s_a, all of the participating agents throw away their schedule, but maintain their route. Iteratively, an agent is selected to recompute a part of its schedule. This selection is done using the agent selection heuristic. A voting round is needed to determine the winner. This agent now recomputes a part of its schedule, as determined by the resource block size parameter. Then, Algorithm 4.5 moves on to the next iteration, where the next agent is being selected to recompute part of its schedule. The algorithm terminates when each agent in the group has a new and complete schedule.

Line 4 adds transport resource $v \in R^{tr}$ to the rescheduling group administered by the scheduling agent $a_s \in A$. Subsequently, vehicle v throws away its schedule Sd_v and enters a loop, which only terminates when its agent has re-computed a complete schedule. In Line 7 an interim schedule Sd'_v is computed, which would be the optimal schedule if there would be no interference with other agents. Of course, this interim schedule has to take

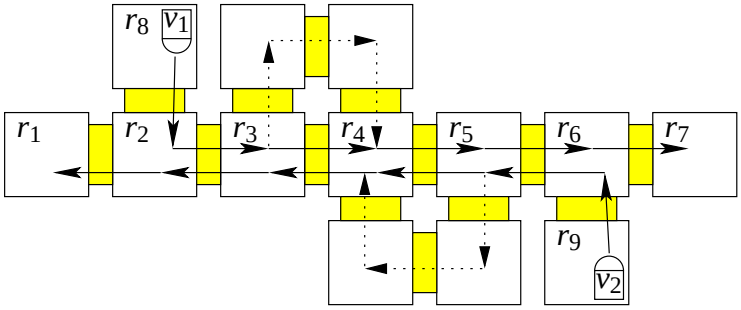


Figure 4.14: This example is similar to Figure 4.1a. Context-aware routing could solve the deadlock by having one vehicle wait for the other. MACA-RP could improve by determining which vehicle should wait. The MACA-RR method can further improve by letting the vehicle that was selected to wait by the maca-rp method take a detour.

into account loading and unloading actions as specified in the plan P_v of vehicle v. In Line 8 the agent selection heuristic $v_v = h(P_v)$ computes the priority value for vehicle v. The agents call the HASHIGHESTPRIORITY function, which only returns true for the agent with maximum value $v_v = \max_{w \in R^{tr}} v_w$. In Line 11, the RESERVEBLOCK method creates reservations for the vehicle with the highest priority, which it already computed in the provisional schedule Sd'_v. It creates reservations for the next b resources in its plan, but also for some more resources, because even temporary plans should always end in a resource with sufficient capacity. If this would not be the case, it cannot be ensured that the agent can find a feasible plan for the rest of its provisional schedule without changing prior reservations. Finally, the LEAVEGROUP function removes the agent from the rescheduling group.

The next section goes one step further. Besides reconsidering priorities of the agent, the MACA-RR method allows agents to traverse an alternative route as new information about incidents arrives.

4.3.2 Revising routes (MACA-RR)

In the previous methods the routes of an agent were only modified in case a new transportation request was assigned to the agent, or when an incident rendered the route of the agent infeasible.

A natural improvement to revising priorities, is to revise routes as well. The advantage of the MACA-RR method over MACA-RP is that it is expected to have a higher performance. Disadvantage is that, due to consideration of alternative routes, the computation costs (in CPU time) are also higher. The next chapter presents a comparison between these methods based on empirical data gathered with experiments.

Algorithm 4.5 can be extended to include consideration of alternative routes during

Algorithm 4.6 MACA-RR.

1: **function** MACA-RR($v \in R^{tr\prime}, a_s \in A, b \in \mathbb{N}$)
2: **Pre:** Vehicle v, part of the set of rerouting vehicles $v \in R^{tr\prime}$, reschedules in blocks of size b
3: **Post:** Vehicle v has an up-to-date schedule
4: ADDTOGROUP(a_s, v)
5: $v_s \leftarrow$ visiting sequence of resources in which load and/or unload actions take place
6: $(Rt_v, Sdv) \leftarrow null$
7: **while** $v_s \neq \varnothing$ **do**
8: Compute provisional plan (Rt'_v, Sd'_v) for visiting the resources in v_s
9: Compute selection value $v_v = h(P_v)$
10: *winner* $\leftarrow$ HASHIGHESTPRIORITY(a_s, v_v)
11: **if** *winner* **then**
12: $i \leftarrow$ RESERVEBLOCK$(v, Rt_v, Sd_v, Rt'_v, Sd'_v, i, i+b)$
13: **end if**
14: **end while**
15: LEAVEGROUP(a_s, v)
16: **end function**

the replanning. In Algorithm 4.6, instead of computing a provisional schedule Sd'_v, a provisional route Rt'_v as well as a schedule Sd'_v is computed. The route Rt'_v must visit all loading and or unloading resources as specified in the original plan P_v, in the same order as before. Hence, there are no changes to the order in which the transportation requests are executed.

Algorithm 4.6 shows the pseudo-code for the MACA-RR algorithm. In Line 5 the visiting sequence is constructed. This sequence contains all resources in plan P_v in which a loading and/or unload action takes place. The sequence contains these resources in the same order in which they occur in plan P_v. Subsequently, not only the schedule, but also the route contained in plan P_v is thrown away by the agent. Similar to the MACA-RP algorithm the plan for vehicle v is then re-computed. In Line 8, besides an interim schedule Sd'_v, also a route Rt'_v is computed, which might very well be different from the original route Rt_v.

4.3.3 Properties of the refined methods

Section 4.2.3 describes that the multi-agent context-aware routing algorithm always results in a Nash Equilibrium, which is considered a crucial property.

For both the MACA-RP as well as for the MACA-RR method, after a finite number of replannings due to transportation requests and incidents these methods also result in a Nash Equilibrium (as always assuming that execution will go according to plan for all

agents).

Furthermore, if this process turns out to be too slow, for example if there are many incidents in a short period of time, it is possible to skip a percentage of replannings and rely on the operational conflict resolution rules – or alternatively to only replan the first several locations to visit of the current transportation plan. The MACA-RP and MACA-RR methods can be used as *anytime* or *interruptible* methods, which continuously attempt to improve the performance even during the execution of the transportation plans.

4.4 Summary

In this chapter a new single-agent context-aware source-destination algorithm (SACA) is introduced. The best known result of Kim and Tanchoco (1991) has run-time complexity $\mathcal{O}(|R^{tr}|^4 \cdot |R^{inf}|^2)$. By a careful analysis of this approach, we succeeded in lowering the run-time complexity to $\mathcal{O}(|F|\log(|F|) + |\rho|)$ or $\mathcal{O}(|R^{tr}||R^{inf}|\log(|R^{tr}||R^{inf}|) + |E_R||R^{tr}|)$, making it much more scalable.

Subsequently we presented a multi-agent context-aware routing method (MACA) based on the SACA algorithm. MACA guarantees that, if execution goes exactly according to plan, deadlocks do not occur. Furthermore, if all agents use the context-aware approach the end result is a Nash equilibrium – no agents can improve by changing their plan (without other agents making changes).

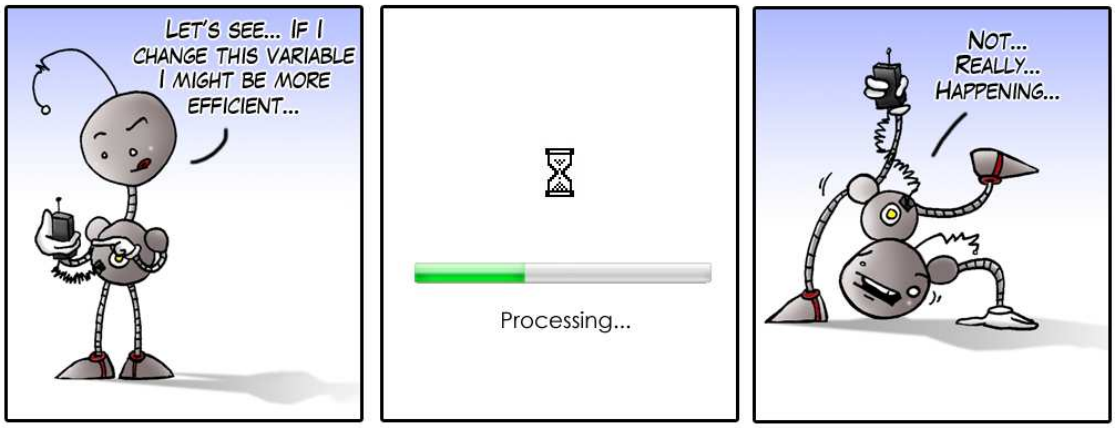
By solving conflicts already in the planning stage, MACA improves the predictability of travel times. For many realistic environments, however, it is also required to consider uncertainty. This is why the framework presented in Chapter 3 contains incidents, which model malfunctioning transport and infrastructure resources.

The second part of this chapter refines the MACA with respect to incidents. To improve the plan quality in incident-rich environments points in the planning process where agents are in conflict are reconsidered to improve on earlier solutions (with now more information available). This is what the MACA-RP method is for. Subsequently, this method is further improved by allowing the agent to change the routes to execute their requests – the MACA-RR method.

Section 4.2.3 compared the context-aware routing approach to the classical approach and showed that both positive examples, where the context-aware routing approach outperforms the classical approach, as well as negative examples exist. Besides that, there is another possible drawback; the improved algorithms do consume more CPU time. So, the question is whether the increased plan quality is worth the additional cost in CPU time. In the next chapter experiments and the obtained empirical results will be described to test and compare the transport planning methods presented in this chapter.

Chapter 5

Experiments



In the previous chapter several operational planning methods have been described for operational pickup and delivery transportation planning with time-windows. The introduced methods attempt to improve the performance of the agents as well as the robustness of the plans. On the one hand, methods were presented that take the plans of other agents into account in order to have better knowledge about the plan execution already at planning time. On the other hand, methods were introduced that have better capabilities to deal with incidents.

In Section 4.2.3 the multi-agent context-aware (MACA) routing was compared to classical routing by considering examples. It was shown that both situations where MACA outperformed classical routing as the other way around exists. In this chapter the two will be tested and compared more thoroughly with experiments. The experiments are also needed to discover the effect of the other methods that were described, such as the MACA-RP and MACA-RR methods.

This chapter reports the outcomes of our experiments. First, empirical results are presented about the gain in performance by the MACA approach as compared to the CLASSICAL approach. Second, the influence of the different conflict-resolution rules, which are

used by the infrastructure agent to prioritize agents, is determined empirically. Finally, experiments are described in which the incident level increases. This will show whether the MACA approach is also to be preferred in situations where incidents do occur.

A single experiment is defined by specifying the available transport resources, transportation requests, transport network topology and incident level. The experiments are divided into two sets: *(i)* general experiments using a collection of synthetic problem instances referred to as the *test set*, and *(ii)* a realistic setting with airplanes taxiing on the Schiphol airport network.

The general experiments do not focus on a particular real-life application. Instead, the transport network topology, the set of transportation requests, the set of transport resources, the behavior of the agents, and the incident level are varied in order to study the effect of this factor on the system performance. These synthetic instances are needed, because we need to control many parameters.

The behavior of the agents is defined by the various planning methods described in the previous chapter. The experiments attempt to establish relations between the performance of the planning methods under these varying circumstances. There have been other, similar, empirical validations, such as the benchmarks presented in Chapter 2 of Taillard, Golden, and Van Breedam (The VRP Web, 2007), as well as the large-scale vehicle routing problem benchmark by Li, Golden, and Wasil (2005). These benchmarks, however, are for variants of the Vehicle Routing Problem that abstract from the routing problem. The distance between each pair of customers is simply looked up in a distance matrix. No capacities of locations, or the congestion that results from these capacities, are considered. Furthermore, these benchmarks compare the performance of different planning methods, but do not follow a *systematic* approach to discover potential relations between the performance of the planning methods and influential environmental or infrastructural factors.

The other set of experiments in this chapter consider the taxiing of airplanes at Schiphol airport in the Netherlands. In current practice at Schiphol, airplanes follow a fixed route from runway to gate and vice versa. These experiments show the benefit of having the airplanes choose alternative routes at any point in time themselves. This fixed-path assumption is not particular to Schiphol airport, it is also common practice at, e.g., Frankfurt airport (Trüg et al., 2004), the fixed-routes assumption by Hatzack and Nebel (2001), or in bus line domains (Hickman and Blume, 2000).

5.1 Expectations

It is expected that the MACA approach results in statistically significantly better performance than the CLASSICAL approach, because all alternative routes that agents can take are considered, while using the CLASSICAL approach some infrastructure resources might

be overused. Because the *order* in which the agents plan is arbitrary, and the MACA-RP and MACA-RR methods revise the potential conflicts, we expect those to perform even better. Most is expected of the latter method, MACA-RR, which also considers alternative routes for the transport resources while reconsidering priorities in case of conflicts. The gain in performance comes at a price, we expect that the the higher the performance of the methods, the higher also the CPU time required to compute the final transportation plans.

With respect to the *conflict-resolution rules* described in Section 4.2.4, we expect the plan-based rules to outperform the simple rules. This makes sense, because the performance of the transport agents depends on their final plans. However, the first-in-first-out rule to prioritize vehicles accessing a resource has already proven itself in many domains.

We expect the transport network *topology* to have a significant influence as well. An important factor here is the average number of alternative (approximately equi-distant) routes between the source and destination locations. The more alternative routes, the better the context-aware methods can spread the traffic, and also the less sensitive to incidents because the agents can avoid infrastructure resources affected by incidents. The negative influence of the arbitrary order in which agents plan will also be smaller on networks with many alternative route choices.

Finally, we will see the effect of incidents on the different planning methods. We expect that the context-aware methods still outperform the CLASSICAL approach if the incident level increases. The strength of MACA-RR to select alternative routes will prove most useful in incident-rich circumstances. The next section describes the experimental set-up, after which the results of the experiments will be described.

5.2 Experimental set-up

For the general experiments a synthetic set of problem instances, referred to as the *test set*, has been used to obtain empirical data. As a starting point, an 8×8-grid network is used in which 32 transport resources are traveling around. The transportation request load is 192 requests[1]. Sequential Vickrey auctions (Vickrey, 1961; Sandholm, 1995) determine the task assignment and then the task execution by the transport resources can begin. The outcome of the auctioning process already depends on the planning methods that the agents use, because these are used to compute the bid values for each transport request by the agents. Using this as a starting point, several parts will be varied, referred to as the independent variables.

The agents are forced to do their best to execute all transportation requests assigned to them; that is, even without any reward they will make the cost for traversing to the

[1]The number 192 is twice the minimal request load, viz., two instances were merged together. For more details on the generation of the test set, see Appendix I.

pick-up and delivery location, unless they are unable to do so due to, e.g., deadlocks or incidents.

The role of information in our experiments is measured by the choice of planning method; some planning methods use different information than others. This is how the information that is available to the agents in the experiments is manipulated.

Independent variables The *independent variables* (or experimental variables) are the planning method, the request load, the incident level, the size, and topology of the transport network, and the number of transport resources. The planning methods, which can all be found in the previous chapter, are divided into 5 categories: *(i)* the classical approach, *(ii)* multi-agent context-aware (MACA) planning, *(iii)* revising priorities (MACA-RP), *and (iv)* revising routes (MACA-RR).

The *request load* is simply measured by the number of transportation requests (issued in parallel at the beginning). It can be more accurate to take into account the minimum distance from source to destination, as well as the specified time-windows. However, that is still an approximation as it, e.g., does not take into account the initial locations of the transport resources[2].

An alternative approach would be to define a *request rate* instead of a request load. If using a request rate (a set of transportation requests per time unit), one can postpone doing any measurements during the first few bursts of requests, to prevent including the cold start (all agents starting from their home locations), which in this thesis is included in the measurements. However, a small experiment showed it did not influence the results much, due to the fact that all different planning methods suffer from the same cold start and that the effect is small because it is averaged over a long time.

The *incident level* is not simply measured by counting the number of incidents. For each incident $(r, i, \tau) \in \mathcal{I}$, affecting infrastructure or transport resource $r \in R^{inf} \cup R^{tr}$, the impact $0 \leqslant i \leqslant 1$, and the duration $ub(\tau) - lb(\tau)$ are also considered. The following is used to measure the incident level:

$$\text{incident level} = \sum_{(r,i,\tau)\in\mathcal{I}} i \cdot \big(ub(\tau) - lb(\tau)\big).$$

Three different *network topologies* were used in the experiments, which are *(i)* random networks, *(ii)* small-world networks, and *(iii)* grid networks, all with 64 infrastructure resources and 128 arcs.

[2]For the scope of our experiments just counting the number of transportation requests per time unit sufficed.

Dependent variables The *dependent variables* (or performance indicators) computed for the experiments are the percentage of successfully executed transportation requests, the average tardiness of all transportation requests, the system welfare, and the CPU cost (in time) required to finish the particular simulation.

The network utilization is the usage of the infrastructure resources by the agents. The agent utilization is the amount of time that agents are in a non-idle state (i.e., they are loading, unloading, waiting or driving). To define the network utilization, ranging between 0 and 1, the infrastructure resources are divided into two disjoint sets: the resources with sufficient capacity to hold all agents $R^{ps} = \{r \in R^{inf} : k^{inf}(r) \geqslant |A|\}$ and the other resources $R^{nps} = R^{inf} \backslash R^{ps}$. The set of *parking space* resources R^{ps} is ignored while computing the network utilization. Recall that the route of an agent $Rt_a = (r_{a,1}, r_{a,2}, \ldots, r_{a,N_a})$ is a sequence of infrastructure resources, its schedule $Sd_a = (t_{a,1}, t_{a,2}, \ldots, t_{a,N_a})$ a sequence of times at which theses resources are entered and reservations $Q(r) \subseteq A \times W$ is a set of agent time-window pairs representing the reservations in resource r (Section 3.3.2).

$$
\begin{aligned}
\text{agent utilization } U_A &= \sum_{a \in A} \max_{t \in Sd_a} t, \\
\text{network utilization } U_N &= \sum_{r \in R^{nps}} \sum_{(a,[t_1,t_2)) \in Q(r)} \frac{t_2 - t_1}{|A| \cdot t_c}, \\
\text{plan quality (relative reward) } \mu &= \sum_{\{j : o_j \in O\}} \frac{\pi_j(\breve{\tau}_j^s, \breve{\tau}_j^d)}{\pi_j(\tau_j^s, \tau_j^d)}, \\
\text{CPU cost in time } \psi &= \text{computation time of a complete experiment.}
\end{aligned}
$$

The definition of performance μ is the relative system reward, i.e., the achieved reward for all transportation requests divided by reward that would have been obtained if all requests would have been processed within the specified pickup and delivery time-windows. The relative system performance is just one of the many possible system performance indicators, which is used as a default in this chapter.

The CPU cost in terms of time ψ is the measured amount of time that the computer program used to execute the experiment, from the begin to the end and including everything. The machine on which the experiments were performed guarantees that it dedicates all processor time to the experiments[3]. An alternative approach would be to measure only the algorithmic time, instead of the full time of the experiment from start to end, for CPU cost ϕ. However, as can be seen from the enormous speed (low CPU cost) of the CLASSICAL method, which is used on exactly the same problem instances, this did not influence

[3]Note that no other applications can interfere and increase the CPU cost ψ of an experiment. Besides that the operating system measures the percentage of CPU cycles given to the program, such that one could also correct in case other processes would be running simultaneously.

the experiments in this thesis.

For the lowest request load of 96 requests, it is possible to gain the maximum reward for each request, due to the way the test set is constructed[4]. For higher request loads, however, this is not possible. Transportation requests of multiple problem instances are merged together to form more challenging problem instances, but these requests are to be executed by the same set of transport resources. Hence, the maximum possible reward can usually not be reached because of the increased request load, same network, and same set of transport resources.

All results presented have been obtained by making use of the transport planning simulator TRAPLAS, see Appendix E. A free software environment for statistical computing and graphics called R (R Development Core Team, 2007) has been used to combine the output of TRAPLAS, to plot the graphs and for all further data analysis. All experiments were done on the Distributed ASCI Supercomputer (DAS-3, see Figure E.2).

5.3 General experiments

This section describes the results of the general experiments with synthetic problem instances. First, Section 5.3.1 tests and compares the CLASSICAL approach to the MACA and its variants under normal circumstances. Second, in Section 5.3.2 the results of experimenting with the different conflict-resolution rules for the infrastructure agents are presented. Finally, Section 5.3.3 presents empirical results on the performance of the different transportation planning methods in case there are incidents.

After the general experiments Section 5.4 describes experiments on a real-life transport network (the Schiphol airport network) which demonstrates that the context-aware methods are usable on real-life networks as opposed to syntesized problem instances.

5.3.1 Classical versus context-aware variants

Consider the situation where the request load is 192 transportation requests. There are no incidents and the network topology has a grid structure. Figure 5.1 shows a box-and-whisker plot (Tukey, 1977) for 4 different planning-method categories. The plots are based on 100 samples (10 simulations and 10 different sets of transportation requests). If the notches of two box-and-whisker plots do not overlap this is strong evidence that the two medians statistically differ (Chambers et al., 1983, page 62).

A relative reward of 1 means that the maximum possible reward is obtained for all of the transportation requests, as determined by the reward function $\pi_j(\check{\tau}_j^s, \check{\tau}_j^d) \in \mathbb{R}$ for each request $o_j \in O$, where $\check{\tau}_j^s \in W$ and $\check{\tau}_j^d \in W$ are the executed pick-up and delivery time-

[4]More about how the test set is constructed can be found in Appendix I.

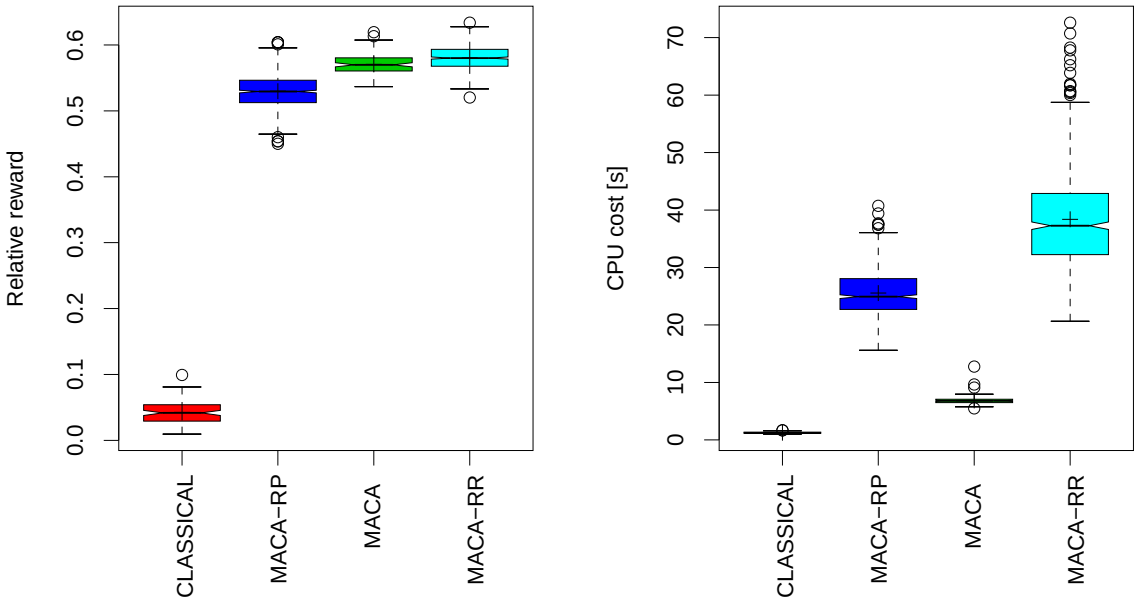


Figure 5.1: Role of information. With one exception, all methods execute all transportation requests. The CLASSICAL method, however, executes only 48.5% on average. The '+' symbol indicates the means and the '○' symbol is used for outliers.

Planning method	CLASSICAL	MACA-RP	MACA	MACA-RR
CLASSICAL	0.000	-0.487	-0.528	-0.538
MACA-RP	0.487	0.000	-0.041	-0.051
MACA	0.528	0.041	0.000	-0.010
MACA-RR	0.538	0.051	0.010	0.000

Table 5.2: Tukey Honest Significant Differences table for the planning-method categories. The value of each cell is the difference in performance means between the two planning methods. The cell is highlighted if this difference is not statistically significant (i.e., $p \geqslant 0.05$). The planning methods are ordered by increasing average performance (relative reward).

windows respectively. This maximum reward cannot always be achieved, especially not if the request load or the incident level increases, or if the number of transport resources decreases.

ANOVA shows that the null hypothesis that the means of all planning-method categories in Figure 5.1 are equal has to be rejected; there is at least one that significantly differs from the others. In such a case the post hoc TukeyHSD (honestly significant difference, See Miller (1991)) test provides more information. The corresponding TukeyHSD is presented in Table 5.2 and provides a pair-wise comparison between all planning-method

categories. In this table, the planning-method categories are sorted in increasing difference in mean, which is why the gray cells, indicating the difference in mean is not statistically significant, are along the diagonal.

It can be concluded that the MACA method has been underestimated in the expectations. Its performance is actually in between the MACA-RP and MACA-RR methods. The expectations about the CPU time costs were correct (see the right plot in Figure 5.1), it can be seen the CPU cost (in time) required by the planning methods increases if more information is considered by the planning methods.

As opposed to the MACA methods, the CLASSICAL method is not able to execute all transportation requests. This is due to deadlocks that occur without any coordination between the agents. On average, the CLASSICAL method executed 48.5% of the transportation requests.

That the performance, the relative system reward, depends on the chosen planning method and nothing else, can be shown by considering the correlation coefficient. The correlation coefficient r between performance and planning method in Figure 5.1, using the model $\mu = \beta_0 + \beta_1 M$, is 0.99. This means that 97% of the total variance in performance is under experimental control (is due to the choice of planning method).

The right plot in Figure 5.1 shows the sole strength of the CLASSICAL method, it is the cheapest in terms of CPU cost. Although this might be an advantage for very large transportation instances, the MACA method is usually fast enough for practical instances.

In Figure 5.3, the CLASSICAL approach is given a second chance at problem instances with higher request loads. It might turn out that it is the only method that can still operate on large problem instances. However, the figure clearly shows that the CLASSICAL planning does not work well at all. This bad performance is due to the occurrence of deadlocks. Of course, due to the fact that only few transportation requests are executed successfully, the relative reward of the agents is close to zero as well. It is clear that, for an agent, the plans of the other agents play an important role and must be taken into account to avoid deadlocks. Without incidents, and with the restriction that agents always drive to a resource with sufficient capacity at the end of their plan, it can easily be proven that deadlocks cannot occur with neither the MACA method, nor with MACA-RP, nor with MACA-RR.

It can be concluded that agents must consider the plans of other agents while creating their transportation plans. The MACA method performed better than was expected, but the best performing method is still the MACA-RR method. Furthermore, the MACA-RR method is also the most expensive in terms of CPU costs.

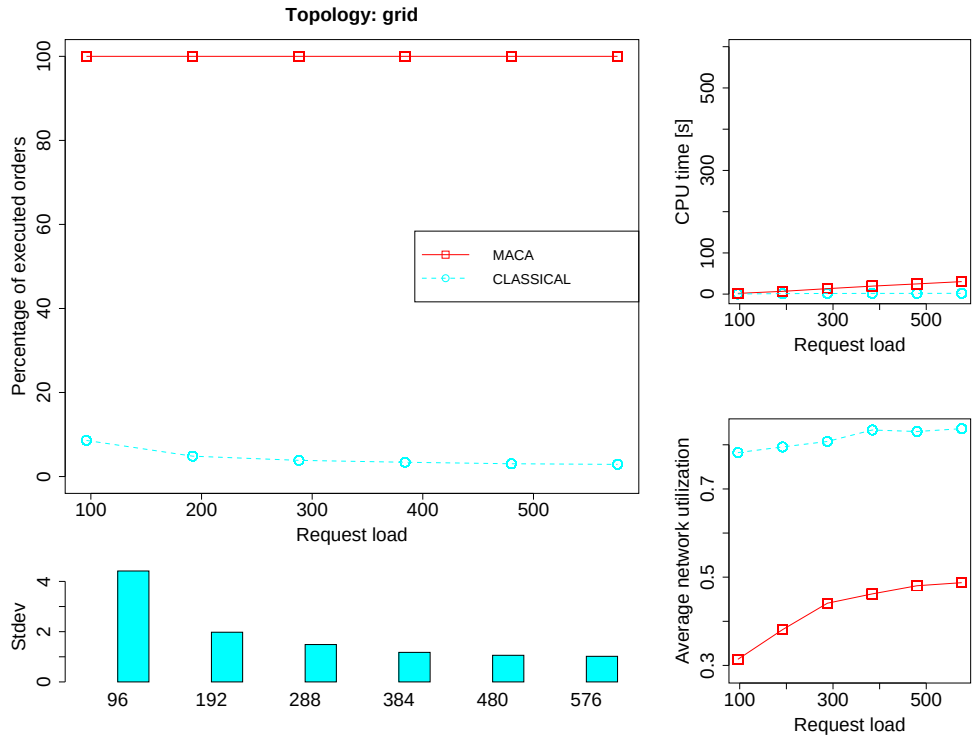


Figure 5.3: The percentage of executed transportation requests with CLASSICAL planning on grid networks.

rule	description
random	for baseline comparison, chooses a random agent to go first.
delays	agent with highest sum of expected delays goes first.
deadlines	agent with lowest value $\frac{t-\phi_o}{M_o}$ goes first, where $t-\phi_o$ is the amount of time until the deadline and M_o is the expected time required to execute the request, for all requests $o \in O_a$ assigned to the agent.
profits	agent with lowest expected profits goes first.
wait	agent that waits longest to enter its current location goes first.
task	agent that is assigned the task that has the highest reward goes first.
inv task	the reversed ordering of the task heuristic, used to see the effect of bad versus good heuristics.

Table 5.4: List of conflict-resolution rules used by the MACA-RP and MACA-RR methods.

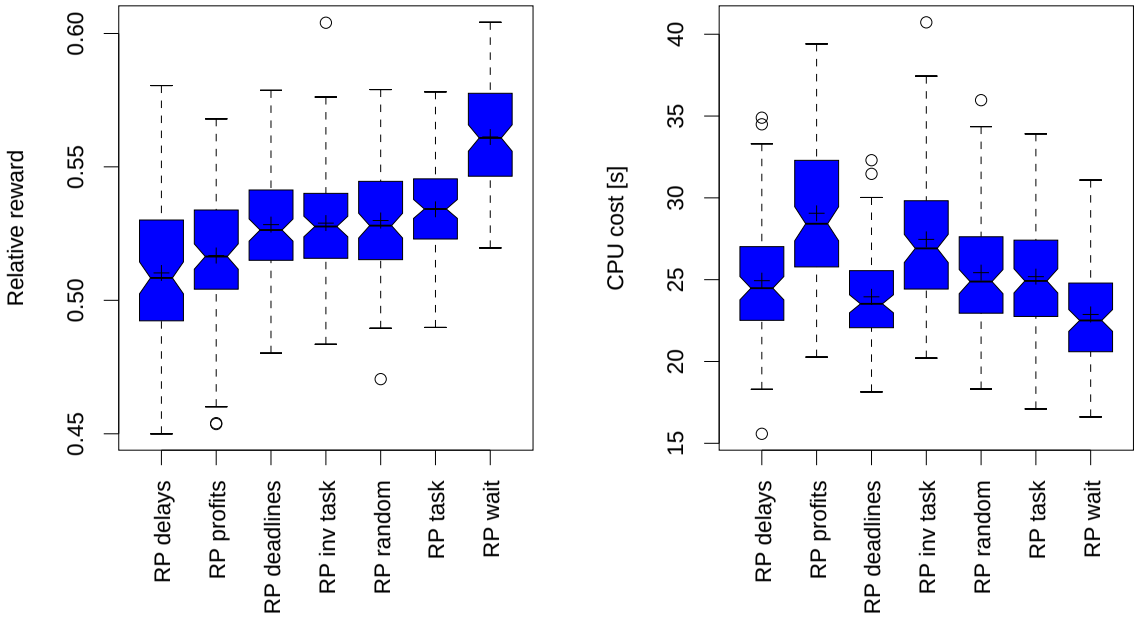


Figure 5.5: Role of information, zooming in on the MACA-RP methods. The '+' symbol indicates the means and the '○' symbol is used for outliers.

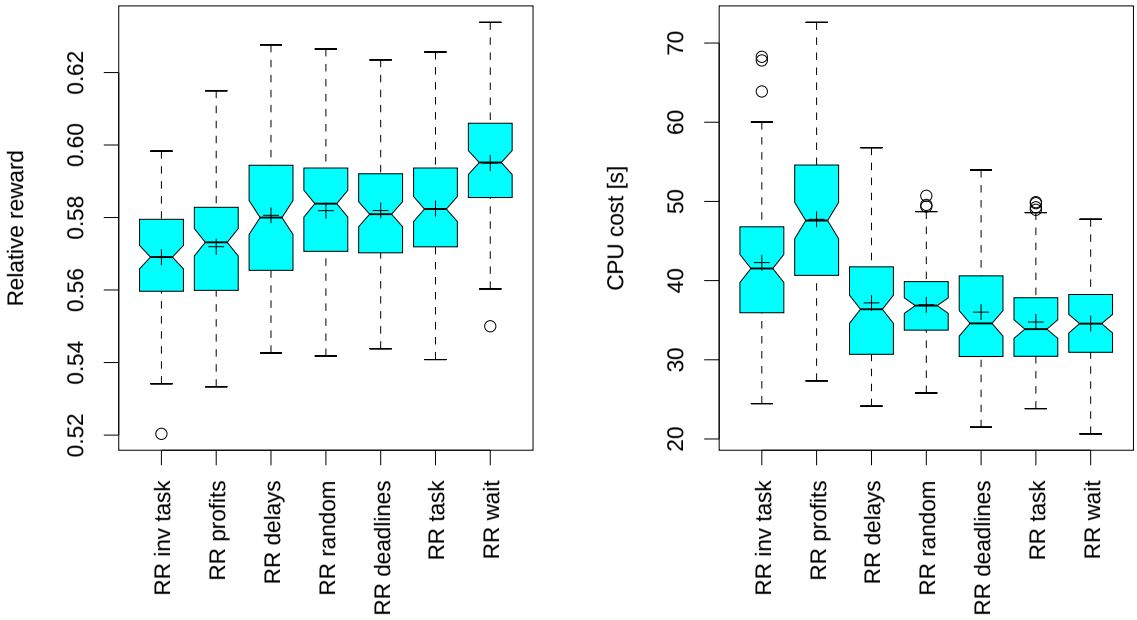


Figure 5.6: Role of information, zooming in on the MACA-RR methods. The '+' symbol indicates the means and the '○' symbol is used for outliers.

Planning method	RP DELAYS	RP PROFITS	RP DEAD-LINES	RP INV TASK	RP RANDOM	RP TASK	RP WAIT
RP DELAYS	0.000	-0.007	-0.018	-0.019	-0.020	-0.024	-0.051
RP PROFITS	0.007	0.000	-0.012	-0.013	-0.013	-0.017	-0.044
RP DEADLINES	0.018	0.012	0.000	-0.001	-0.002	-0.006	-0.033
RP INV TASK	0.019	0.013	0.001	0.000	-0.001	-0.005	-0.032
RP RANDOM	0.020	0.013	0.002	0.001	0.000	-0.004	-0.031
RP TASK	0.024	0.017	0.006	0.005	0.004	0.000	-0.027
RP WAIT	0.051	0.044	0.033	0.032	0.031	0.027	0.000

Table 5.7: Tukey Honest Significant Differences table for the MACA-RP methods. The value of each cell is the difference in performance means between the two planning methods.

Planning method	RR INV TASK	RR PROFITS	RR DELAYS	RR RANDOM	RR DEAD-LINES	RR TASK	RR WAIT
RR INV TASK	0.000	-0.003	-0.012	-0.013	-0.013	-0.013	-0.026
RR PROFITS	0.003	0.000	-0.009	-0.010	-0.010	-0.011	-0.023
RR DELAYS	0.012	0.009	0.000	-0.001	-0.001	-0.002	-0.014
RR RANDOM	0.013	0.010	0.001	0.000	0.000	-0.001	-0.013
RR DEADLINES	0.013	0.010	0.001	0.000	0.000	-0.001	-0.013
RR TASK	0.013	0.011	0.002	0.001	0.001	0.000	-0.013
RR WAIT	0.026	0.023	0.014	0.013	0.013	0.013	0.000

Table 5.8: Tukey Honest Significant Differences table for the MACA-RR methods. The value of each cell is the difference in performance means between the two planning methods.

5.3.2 Conflict-resolution rules

Figures 5.5 (for MACA-RP) and 5.6 (for MACA-RR) show box-and-whisker plots for the different conflict-resolution rules listed in Table 5.4. The *wait* rule, which considers the amount of time a vehicle has already been waiting to determine its priority, works best in both MACA variants. Interestingly, this conflict-resolution rule also is among the fastest with respect to the consumed CPU time.

Tables 5.7 and 5.8 display the TukeyHSDs for the MACA-RP and MACA-RR methods respectively. The MACA-RR method using the *wait* heuristic results in the highest performance on grid networks with relatively small request load and no incidents.

The good performance of the *wait* heuristic can be understood by appreciating the intuitive resemblance with the first-come-first-served heuristic (Kruse, 1984), which works so well in scheduling. The *wait* heuristic aims to minimize the waiting time of transport resources, by giving priority to the longest waiting transport resource. Hence, the *wait*

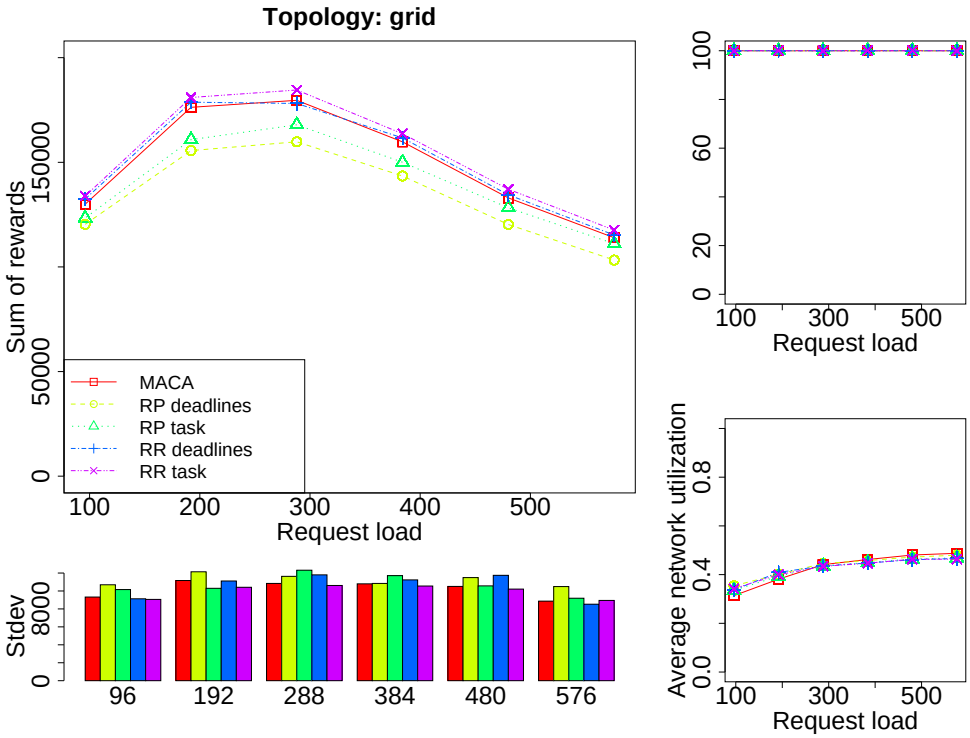


Figure 5.9: The main effect of varying the request load on grid networks.

heuristic attempts to increase the throughput of transportation requests, which increases the performance of the agents.

In Figure 5.9 the results are shown for increasing request load. The performance along the vertical axis is the sum of the rewards of all agents. It can be seen that the performance of all agents increases up to almost 300 transportation requests. The agents have time left, so they can increase the total reward by the arrival of additional transportation requests. There is a point at which the total reward stops growing, which happens to be at almost 300 transportation requests. This was expected because of network saturation. Figure 5.9 also shows that the network utilisation does still grow beyond 300 transportation requests, but not as much as below 300 requests. Beyond 300 requests, the drop in total reward is due to the fact that agents are enforced to execute (instead of simply drop) requests assigned to them, and this has a negative effect on their subsequent transportation requests.

Later, in Figure 5.10, it can be seen that the *relative reward* performance decreases right from the beginning when the request load is increasing. This is because the relative reward is computed relative to an upper bound on the reward. The upper bound assumes all transportation requests are executed within the specified time-windows, which cannot be reached by the agents and the difference increases with the number of transportation requests.

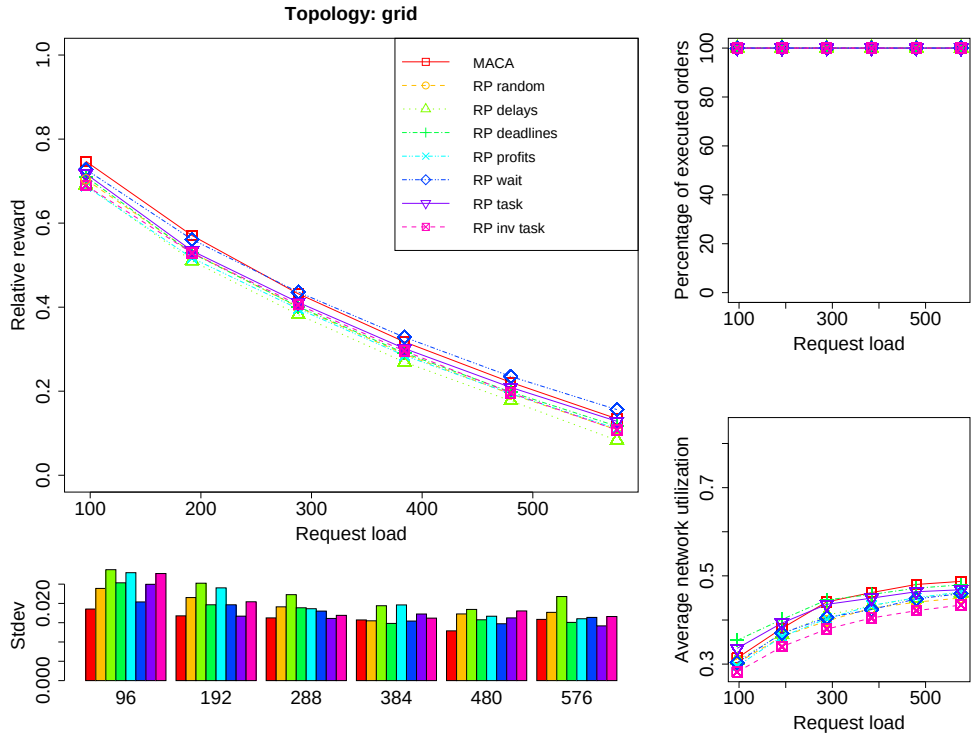


Figure 5.10: The relative reward for the MACA-RP method with different heuristics on grid networks.

5.3.2.1 MACA-RP

Figure 5.10 depicts the performance difference of the variants of the MACA-RP method. The MACA method is presented for comparison, and the other methods differ in the heuristic used to prioritize the agents during the rescheduling process.

The *wait* heuristic again outperforms the others (statistically significant) when looking at the average relative reward. In fact, it is the only used heuristic that significantly outperforms the MACA approach.

The network utilisation grows along with the request load until almost 50% utilization. It can be seen that the speed at which the network utilization grows decreases and at the same time the performance decreases slower.

The CPU cost (in time) depicted in Figure 5.11 shows that the MACA planning method can be used for realistically-sized applications. For the MACA-RP methods it will depend. If there is enough time, it is worth it (due to the improved plan quality). However, if speed is very crucial and there is few time available, the MACA method might be a better choice.

Figure 5.11 also reveals that the CPU cost required to do a simulation run also depends on the choice of heuristic. Generally, heuristics that lead to better performance in plan quality seem to be faster with regard to simulation speed as well. This can be understood

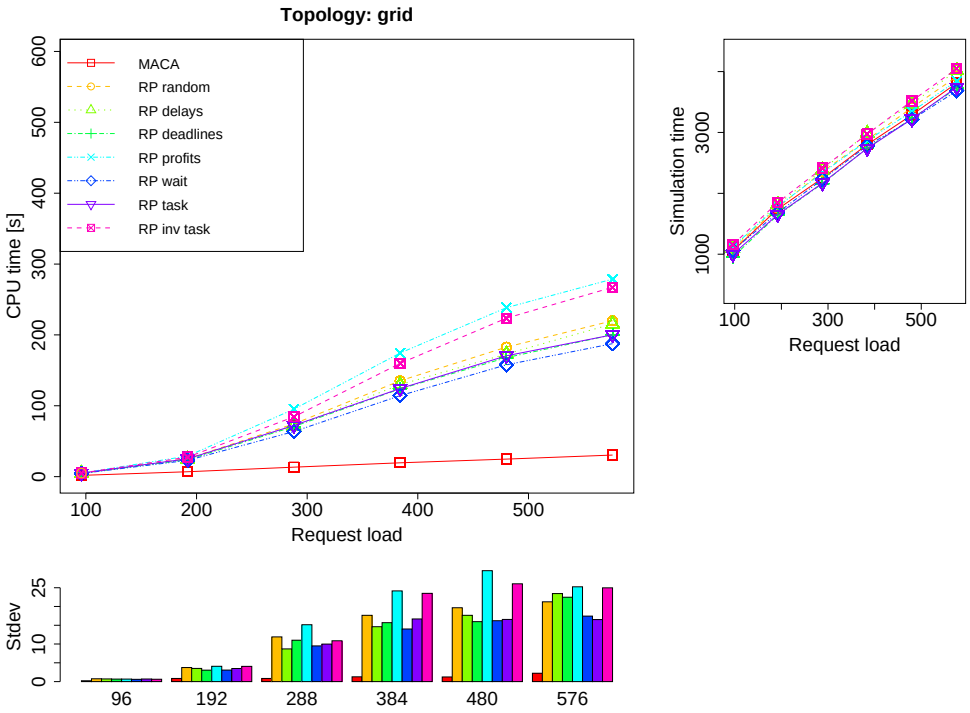


Figure 5.11: CPU cost (in time) required by MACA-RP for different heuristics on grid networks.

by recognizing that heuristics that make good choices about which agent should create reservations first result in fewer conflicts that have to be solved later on. Hence, the remaining process can be done in less time.

Except for the MACA method, which is just a lot faster, the MACA-RP method using the *wait* heuristic must be mentioned again as being the best method, this time with respect to the consumed CPU costs.

5.3.2.2 MACA-RR

Figure 5.12 shows that MACA-RR with the *wait* heuristic results in a slightly higher relative reward (yet statistically significant) than the MACA-RP and MACA methods. At the same time, as shown in Figure 5.13, CPU cost (in time) is higher for the MACA-RR variants. Again, with respect to the CPU costs, the choice of heuristic is important and the *wait* heuristic is cheaper than the other heuristics in terms of CPU costs.

Figure 5.13 shows that instances with less than, say, 200 requests can be solved quickly. Afterwards the required CPU time grows rapidly, especially with approximately 350 transportation requests. The reason why the CPU cost increment reduces beyond this point is that the transport network becomes congested and agents are just going to take much longer to execute the requests. As can be seen in the airplane taxiing experiments

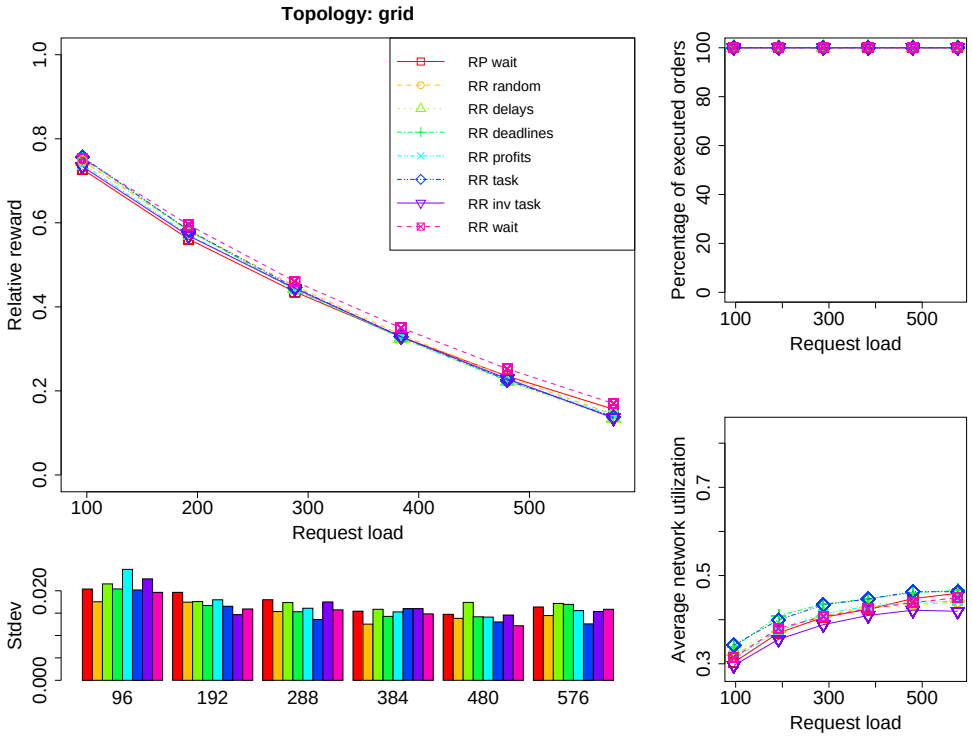


Figure 5.12: Average relative reward for the MACA-RR method for different heuristics on grid networks.

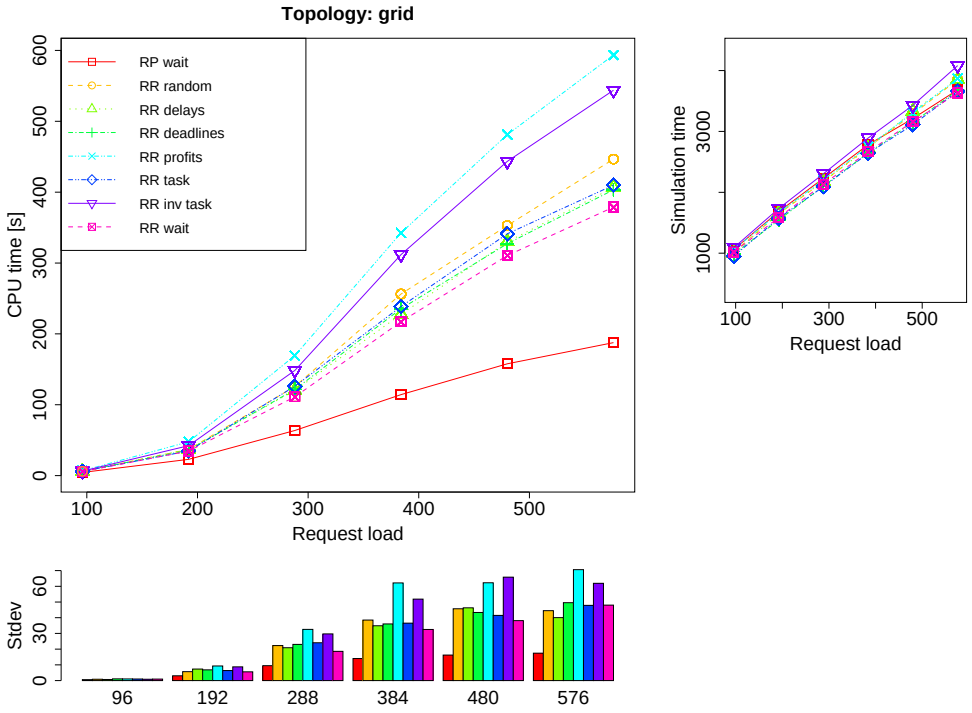


Figure 5.13: CPU costs (in time) required by MACA-RR for different heuristics on grid networks.

Model M3: $\pi = \beta_0 + \beta_1 W$.
Model M2: $\pi = \beta_0 + \beta_1 W + \beta_2 M$.
Model M1: $\pi = \beta_0 + \beta_1 W + \beta_2 M + \beta_3 WM$.

Model	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)	R^2
3	12094	109.0					0.81
2	12074	31.3	20	77.7	4158.5	< 2.2e-16	0.94
1	12054	11.3	20	20.0	1071.1	< 2.2e-16	0.98

Table 5.14: General linear test: comparison of the full model and two reduced models. Variable π refers to the relative reward over all transportation requests, W to the request load and M the planning method.

later in this thesis (Section 5.4), it is possible that, when adding a lot more transportation requests, the growth in CPU cost will increase again at some point, and oscillates like this. The question whether MACA-RR methods are to be preferred over MACA-RP methods again depends very much on the problem at hand, mostly with respect to the time available to the planning agents. A general conclusion here cannot easily be made.

5.3.2.3 Evaluation of the chosen planning method and request load

Looking at the plots of Figure 5.3, 5.10 and 5.12, which show the performance of the planning methods while increasing the request load, it seems the slopes and intercepts are not equal for all different planning methods. Analysis of covariance can show whether these differences are indeed statistically significant. The request load here is used as the covariate.

Three different models are computed. The first is the full model M1, which takes into account the planning method, the request load and their interaction. In the first reduced model M2 the interaction effect is ignored, and a second reduced model M3 also lacks the planning method. If model M2 differs statistically significant from model M3, this means the planning method is required to model the performance. If model M1 in turn differs statistically significant from model M2, this means also the interaction effect between the planning method and the request load adds to the accuracy of the model.

For the full model it is verified that the error has a normal distribution and that the variance is the same for all data (homoscedasticity). The results of the general linear test, which is used to compare the three models to each other, are presented in Table 5.14. Also included is the explained variance R^2.

The full model M1 is the best model according to Table 5.14, the slope and intercept are statistically significantly different. Three models were tested: model M1 stating that both the slopes and intercepts are different for at least one of the planning methods. Model M2 stating that the intercepts differ, but the slopes are all the same. And model M3 stating

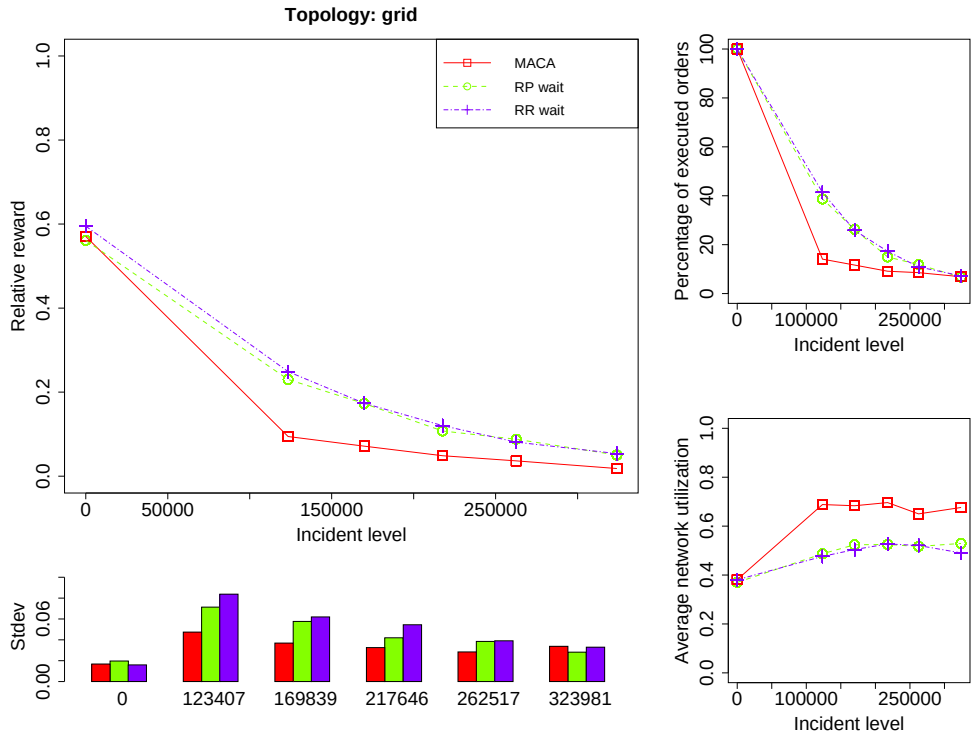


Figure 5.15: The relative reward (averaged over all requests) for a selection of planning methods on grid networks; the incident level is increasing and the request load is fixed to 192 requests.

that all slopes and intercepts are the same. Considering Pearson's R^2, which indicates the explained variance in the model, it must be noted the difference of 4% between model M1 and M2 should not be overestimated. The interaction effect between planning method and request load does not add much.

If the request load increases, the differences between the planning methods increases as well. The *wait* heuristic results in the best planning method. The MACA-RP method is preferred, as it performs equal to MACA-RR, but is also faster.

5.3.3 Incidents

In this section experiments are described that show what changes with respect to the role of information in operational transport planning if the incident level increases. First, the effect of increasing incident level on the performance of the planning methods is investigated. Later, the combination of various request loads as well as various incident levels is considered.

Figure 5.15 and 5.16 shows that the two methods that were superior so far, i.e., MACA-RP and MACA-RR with the *wait* heuristic, are also the best performers if the incident level

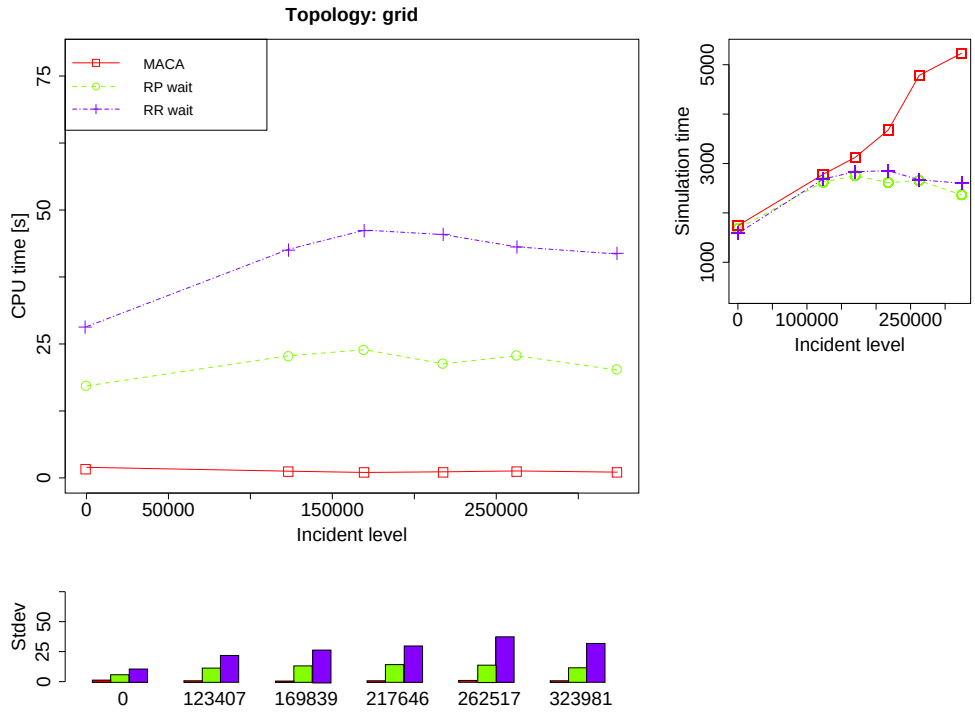


Figure 5.16: The CPU costs (in time) required by the selection of planning methods on grid networks; the incident level is increasing and the request load is fixed to 192 requests.

increases. The reason for this is that all methods degrade approximately equal if the incident level is increased. Apart from the small difference between RR-wait and RP-wait, the difference in performance of all runs with incidents is statistically significant.

Attempts to create methods that degrade less if the incident level increases, should, if possible, adopt a different approach. One such possibility is to insert slack into the plans of the agents, which is described in the following section.

5.3.3.1 Slack insertion

One idea to try to improve the robustness of the plans of the agents is to introduce slack into the plans of the agents. Slack is additional waiting time, that slows the agent down if no incidents occur, but might improve the situation in case of incidents. For a low incident level, agents might not even have to replan, because they can just subtract the effect of the incident from their slack and proceed as they would have without incidents. Inserting slack is realized by having agents multiply the distance of each resource by 1.1 or 1.2 (for 10% and 20% slack respectively) if no incidents are known. If an incident occurs, the slack is decreased likewise (sometimes removed completely).

Obviously, if every action of the agents goes exactly according to plan, then inserting slack only decreases the performance. However, the idea of slack inserting is especially

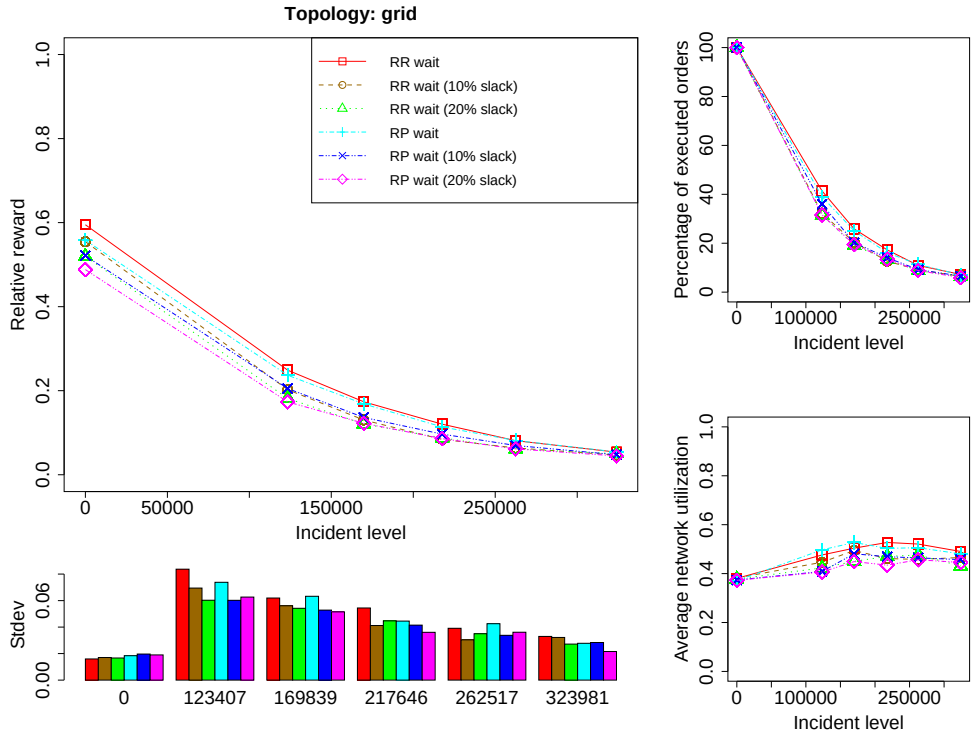


Figure 5.17: The relative reward (average per task) for a slack-inserting version of the *wait* heuristic on grid networks; the incident level is increasing and the request load is fixed to 192 requests.

meant for situations when incidents are *expected* to occur. The intention of slack insertion is that, even if agents are somewhat delayed, they can still travel within the time-window of the reservation they made and arrive at their next location in time. Inserting slack is an attempt to achieve graceful degradation of the performance of the agents in situations where the incident level increases.

In Figure 5.17, the MACA-RP and MACA-RR methods are used, both with the *wait* heuristic. To both of these methods first 10%, then 20%, slack is used for each reservation (i.e., each location they visit) the agents make.

The figure shows a negative result. The relative reward of the agents only gets worse if 10% or 20% slack is used compared to the situation where no slack is used. There is no turning point beyond which it is profitable to insert some slack the plans. The reason for this is that the MACA-RP and MACA-RR method already have the agents reschedule if an incident occurs. But since other experiments already showed that these are our best-performing methods, inserting slack cannot result in more reliable plans for the transportation planning problem that is central in this thesis.

Model M6: $\mu = \beta_0 + \beta_1 I$.
Model M5: $\mu = \beta_0 + \beta_1 I + \beta_2 M$.
Model M4: $\mu = \beta_0 + \beta_1 I + \beta_2 M + \beta_3 IM$.

Model	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)	R^2
6	11628	96.5					0.74
5	11609	77.6	19	19.0	180.5	< 2.2e-16	0.79
4	11590	64.1	19	13.4	127.9	< 2.2e-16	0.83

Table 5.18: General linear test: comparison of the full model and two reduced models. Variable μ refers to the relative reward over all transportation requests, I to the incident level, and M the planning method.

5.3.3.2 Evaluation of the factors information and incident level

The full model M4 is the best model according to Table 5.18. The differences between the planning methods become smaller if the incident level increases. Furthermore, the performance of the MACA method drops when the incident level is high. In Appendix J, Table J.2 shows the coefficients for model M4 in its first column.

5.3.4 Network topology

In this section the influence of network topology on the performance is considered. In Section 5.1 we expected that performance depends on network topology, via certain properties of the network topology, such as the network degree, diameter, or average number of alternative paths between source and destination. Figure 5.19 and 5.20 depict the differences in performance for the grid, random, small-world, and scale-free topologies, while increasing the request load and incident level respectively.

To make the comparison between different network topologies more fair, an equal number of resources (locations) *and* connections between the resources is used. In cases where the transport network has fewer connections (e.g., grid networks and scale-free networks have fewer connections than small-world networks), random connections are added to the network to obtain the desired number of connections.

In Figure 5.19 it can be seen that random and small-world networks perform better than grid networks, and a lot better than scale-free networks, if the request load increases. Of course, for scale-free networks, the hubs (locations with a lot of connections) create bottlenecks in the network that lead to a decrease in performance.

Figure 5.20 shows a similar situation for increasing incident level. Random networks, followed by small-world networks, seem less sensitive to incidents than the other network topologies.

Figure 5.21 show the average network utilization on different transport network

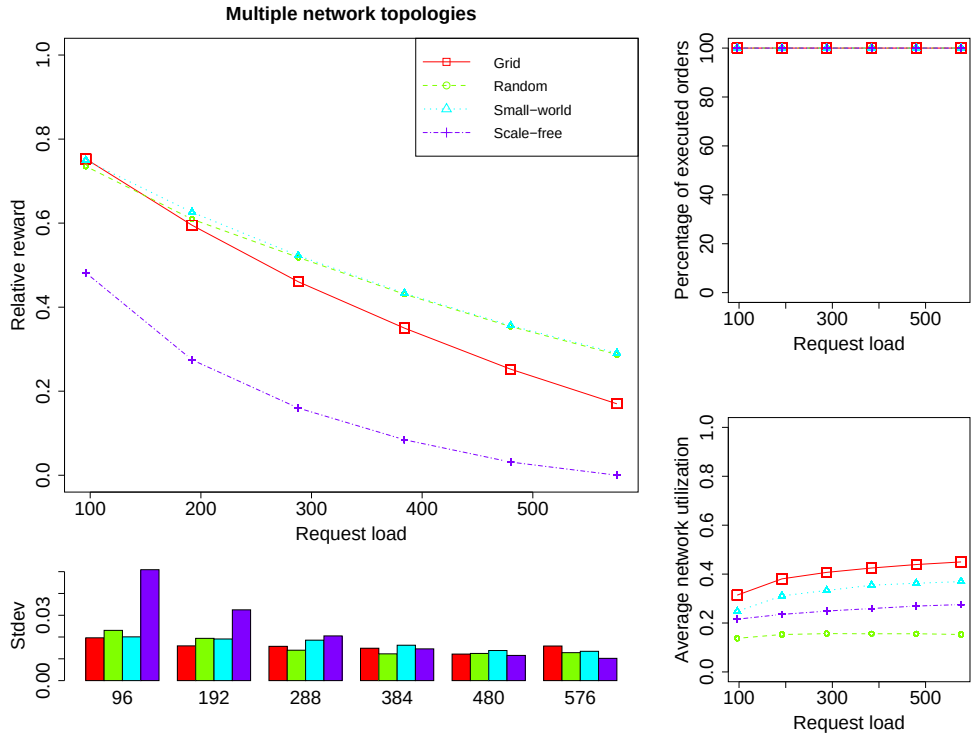


Figure 5.19: The relative reward (average per task) for the MACA-RR method with *wait* heuristic on four different network topologies.

topologies. The upper-left plot represents a base situation (low workload, no incidents). The plots on the right side have a high workload, the plots on the bottom include incidents.

Combining this figure to Figures 5.19 and 5.20 it can be seen that high network utilization results in decreased performance. Looking further at the properties of the networks, the diameter of the network explains why. The grid topology has the greatest diameter, the randomized network has the smallest diameter.

Small-world networks have the property that the path length grows logarithmic in the number of infrastructure resources. Wang and Chen (2003) present an overview of this and other properties of small-world and scale-free networks. This path-length property explains why it performs so well, because small-world networks have a smaller average distance from source to destination location.

5.4 Airplane taxiing experiments

Besides the experiments based on a synthetically generated test set, a real-life transport network is also considered. Experiments were conducted on the Schiphol airport network

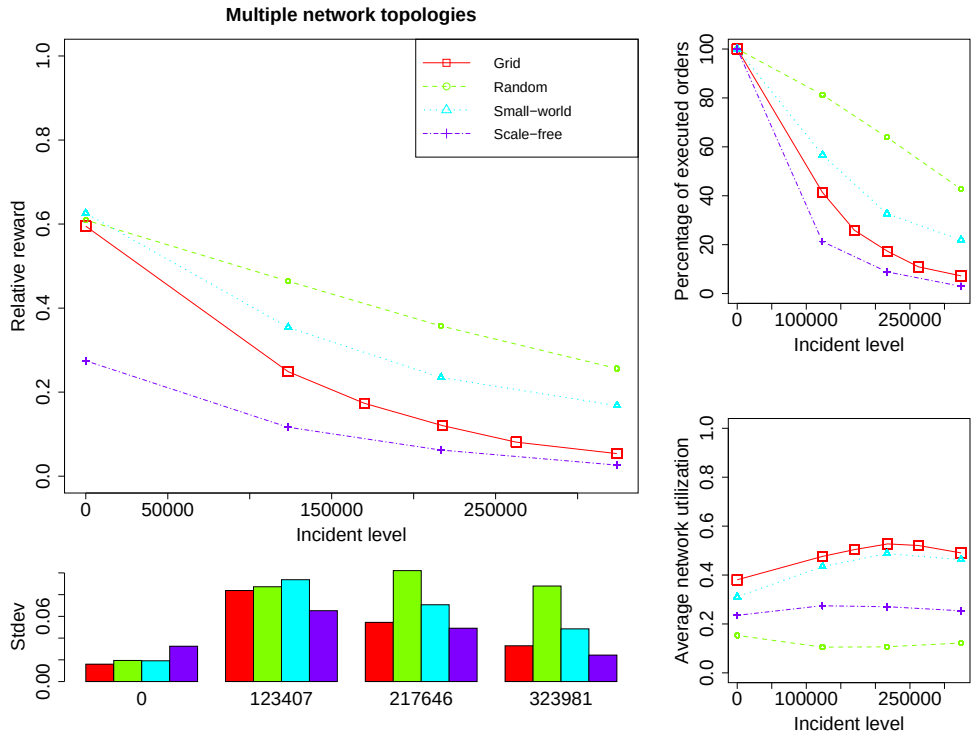


Figure 5.20: The relative reward (average per task) for the MACA-RR method with *wait* heuristic on four different network topologies; the incident level is varied and the request load is fixed to 192 requests.

in the Netherlands (see Figure 5.22)[5]. For this transportation network, the taxiing problem of aircrafts (on the ground) plays an important role that is crucial to the performance of the airport (Ter Mors et al., 2007). The usual sequence of an airplane after touch-down is to taxi to a gate, then wait for services, such as cleaning, boarding, safety checks. Finally, before taxiing to a runway for take-off, sometimes a de-icing station must be visited. Due to the approximately 300 airplanes (a number that is increasing) per day that go through this process, efficient and robust routing methods are required.

The goal of the airplane taxiing experiments is to compare current practice at Schiphol to a more sophisticated approach. Current practice is to use a *context-unaware* approach (such as the CLASSICAL method or Algorithm 2.2 of Hatzack and Nebel), which in these experiments is compared to a context-aware approach, such as the MACA method (Algorithm 4.1).

The role of information on agent performance is reconsidered, while zooming in on the context-aware shortest-path algorithm. For the routing component, the type of information might also be crucial to the performance of the system. A secondary goal is to

[5]The network model of Schiphol airport was kindly provided by the National Aerospace Laboratory (NLR).

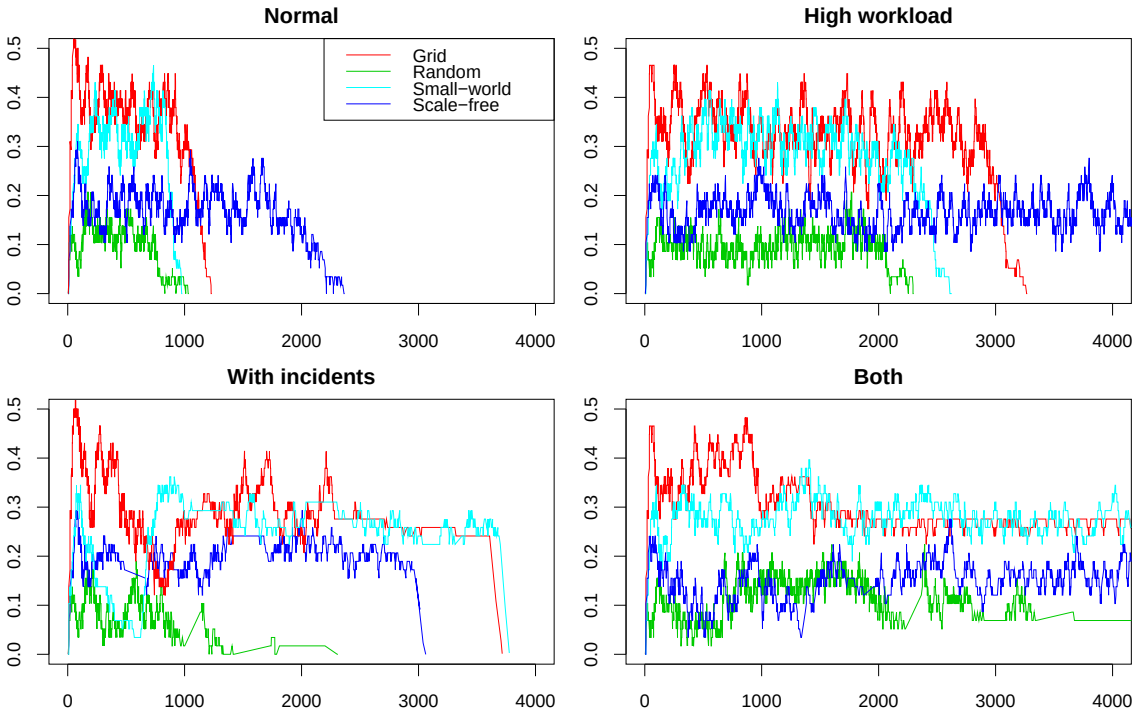


Figure 5.21: Detailed view on the network utilization for a single simulation on different topologies. The horizontal axis represents time [s], the vertical axis the average network utilization (from 0 to 1).

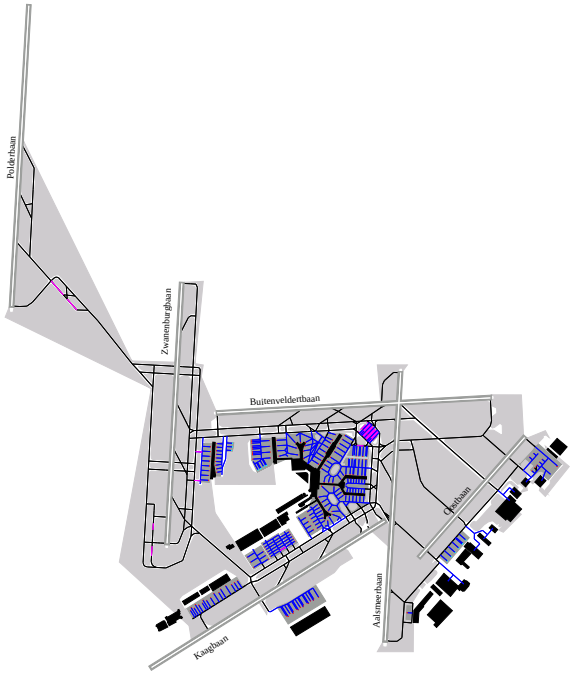


Figure 5.22: Schiphol airport network consisting of 1016 road resources.

show the usefulness of the context-aware routing approach on a real-life transport network.

In a multi-agent system, such a routing algorithm can be applied sequentially, once for each agent. An individual agent profits from being early in this sequence, as it is less bothered by reservations of other agents. The relation between the actual sequence and the performance of the total system is less clear and also part of the experiments. This is not yet a comparison between sequential and parallel routing. Parallel routing might further improve the performance. A parallel approach is typical in MIP formulations, which, unfortunately, cannot yet be applied to realistically sized problem instances.

To make a fair comparison between the two approaches, each algorithm was used to calculate a route for the same start-destination pair, given the same set of prior reservations on the infrastructure. For each set of reservations, the average time to find a conflict-free path for 20 randomly chosen start-destination pairs is computed. To get an impression of how plan quality and CPU costs (in time) depend on the number of reservations, the experiment starts with an empty set of reservations and then grows. For each set of reservations, the last conflict-free plan (out of the 20 in total) found is used to obtain new reservations. Those are then added to the existing set of reservations, and the new set is used to calculate again the time for route finding. This procedure is repeated for 3000 iterations. This means that at the end of the procedure reservations for 3000 source-destination paths are stored in the transportation network. The experiment is run twice: the first time, plans generated by the context-aware approach with distance heuristic were used to make reservations, the second time the plans obtained by the algorithm of Hatzack and Nebel (see Algorithm 2.2) were used. At all times, the start time of an experiment is $t_s = 0$.

In Section 4.2.1.1 the time complexity of the single-agent source-destination routing algorithm was analyzed. Recall that Corollary 4.7 expressed the time complexity in the number of infrastructure resources and transport resources in the case that each transport resource does not visit any infrastructure resource more than a constant number of times. Therefore, a simple variant of the context-aware routing algorithm is introduced in the experiments that visits each resource at most once (only acyclic routes). Furthermore, because of the resemblance to A*, at first sight, a context-aware A* variant is also included to be able to see the differences between the two algorithms in the experiment. Remark 5.1 defines these two variants.

Remark 5.1 In the upcoming experiments, two variants of the context-aware source destination algorithm (Algorithm 4.1) are also considered for comparison. These are:

- A context-aware variant of A* algorithm (Dechter and Pearl, 1985), the context-aware routing algorithm can use an admissible and consistent distance heuristic $h()$, which guides the search process towards the destination. A good candidate for

this function $h()$ is the distance between source and destination if the presence of other agents is ignored.
For the A* variant of the algorithm, several modifications are necessary: *(i)* $t + D(r)$ in Line 8 must be replaced by $t + D(r) + h(r, r_d)$ where r_d is the destination resource, *(ii)* Line 17, where the visited free time-window is removed from the set of free time-windows, has to be removed from the algorithm (it is not anymore sufficient to consider free time-windows only once) and *(iii)* besides an OPEN list a CLOSED list must also be considered. The OPEN list keeps track of those nodes that need to be examined, while the CLOSED list keeps track of nodes that have already been examined.

- A simple modification of the context-aware routing algorithm is not to allow cycles in an agents plan. For the *acyclic* variant of the algorithm, in Line 13, only successors are considered that are both reachable (in $\rho(r_i, t_i)$) and resource r_i does not occur in the previous part of the plan of the agent[6].

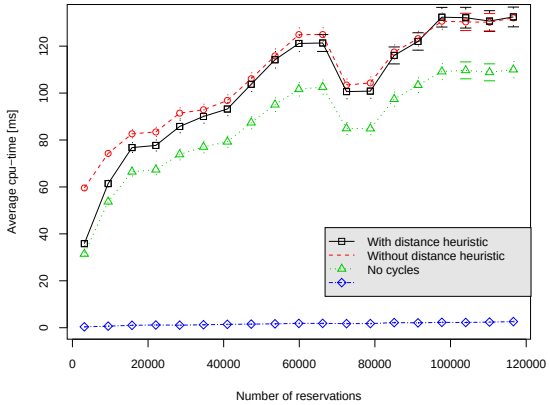
In Section 5.4.1 the results of the comparison between a context-aware and a classical context-unaware routing approach are presented. Subsequently, Section 5.4.2 describes the effect of the sequence in which agents plan. It is obvious an agent is better off when allowed to make reservations for their plan before other agents, but this section will show whether the performance of the total system is also influenced by the order in which agents create reservations.

5.4.1 Context-aware versus classical routing

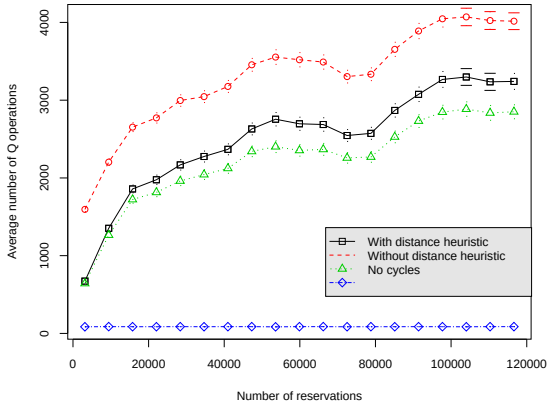
From Figure 5.23a it can be concluded that the context-unaware approach of the algorithm of Hatzack and Nebel is so fast, the context-aware algorithms look slow by comparison. A closer look reveals that the context-aware algorithms are still quite fast, as a solution is found on average within two tenths of a second. Also, the 95% confidence intervals are reasonably small, so this performance is reasonably stable. With regard to the different variants of the context-aware algorithms, it can be seen that the no-cycles variant is significantly faster than the other two, despite the fact that this version has to check for cycles in the routes of the agents. Note that the context-aware approach with distance heuristic requires about the same amount of CPU costs (in time) as the version that utilizes no heuristic. The cost of the additional open list operations – the version with distance heuristic must check for duplicates on the open list – more or less cancels out the benefits of having an context-aware search.

Furthermore, a clear notch is visible in Figures 5.23a and 5.23b, just before 80,000 reservations, which shows the CPU cost (in time) as well as the number of open list

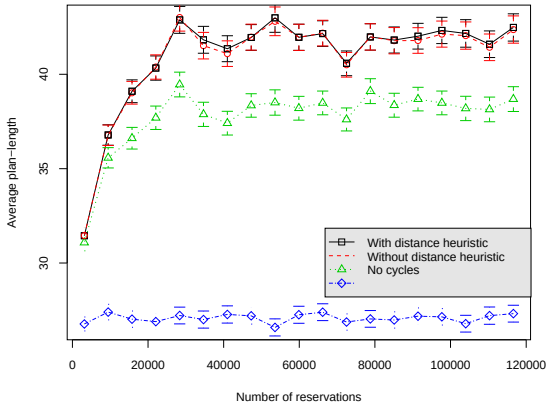
[6]Note that Corollary 4.7 applies here.



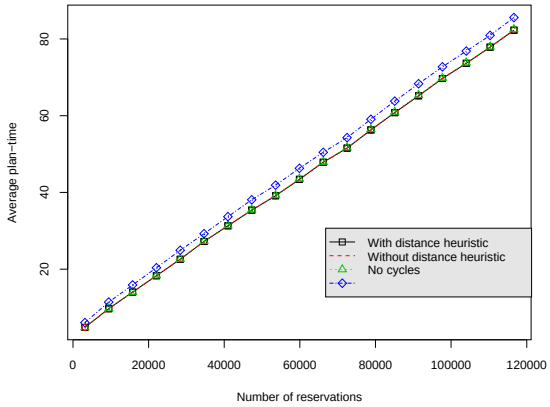
(a) Average CPU-time for increasing amount of resource reservations.



(b) The average number of open list insertions and removals performed to find the plan.



(c) The average number of resources traveled to reach the destination resource.



(d) Average end-times of plans for increasing amount of resource reservations (confidence intervals too small to display).

Figure 5.23: Results for the case where plans generated by the free-path method were used to make reservations.

operations is not growing monotonically for an increasing number of reservations in the transport network. Intuitively, the more reservations there are, the more difficult it is to search a path through the transport network. However, after adding certain reservations, the search might become easier all of a sudden, because a “difficult” part of the network does not have to be searched anymore (due to a reservation now prohibiting this). The exact position of such a notch depends on the transport network and the order in which agents create these reservations for traveling to their desired target locations.

Looking at the cost of the generated plans (Figure 5.23d), the plans generated by the no-cycles variant are equally expensive as those generated by Algorithm 4.1 with or without distance heuristic, both of which are optimal. The plans made by the context-unaware algorithm of Hatzack and Nebel are slightly longer (in time). The context-aware planner

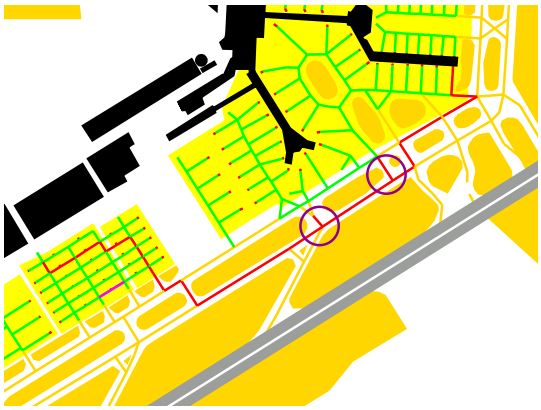


Figure 5.24: If cycles are allowed and there are many reservations of other agents present in the network, the context-aware algorithm often produces plans with a cycle. An example of a produced shortest path is highlighted in red, and the side-steps are indicated with the purple circles. Such plans are not considered by the acyclic version of the routing algorithm.

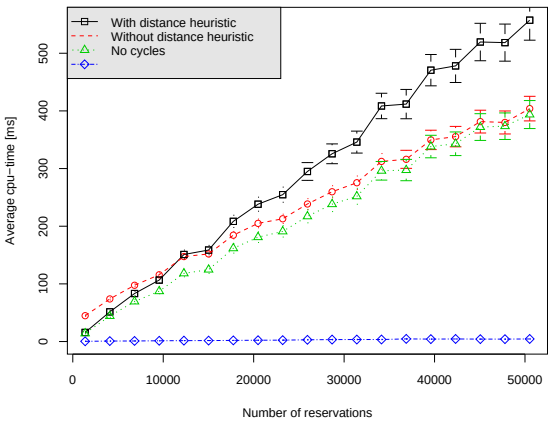
with or without distance heuristic often steps aside from a straight line for other agents to pass in their plans, see Figure 5.24. This is not allowed by the no-cycles planner (as it would induce a cycle). However, the no-cycles planner still generates equally expensive plans, because it usually can insert extra waiting time earlier in the plan. This also results in the shortest plan lengths. In general, this can lead to sub-optimal plans.

For the plots in Figure 5.25, the results are given using plans made by the context-unaware algorithm to make new reservations. Although the aforementioned is still very fast, the plans made by the context-unaware algorithm are significantly more expensive. The reason is that many shortest paths will make use of the same resources, so after a while a number of bottleneck resources will emerge, dramatically deteriorating the performance of the algorithm of Hatzack and Nebel.

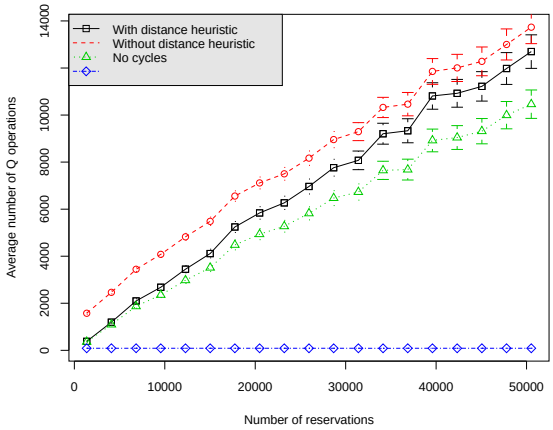
The context-aware planners still manage to plan around the bottleneck resources to a large extent, but the search process is slowed down considerably, with an average CPU costs (in time) of half a second per shortest path call, and frequent outliers of one or even two seconds for a single shortest-path call. The context-aware planner with distance heuristic suffers especially, presumably because this distance heuristic, which is based on the shortest path without reservations, directs the search right into the congested area of the infrastructure.

5.4.2 Planning in sequence

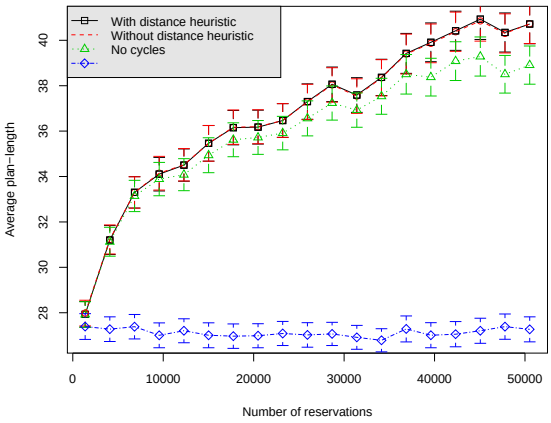
From Figure 5.23d it is already clear why agents would prefer to plan before others make any reservations: the cost of the average plan increases linearly with the number of reser-



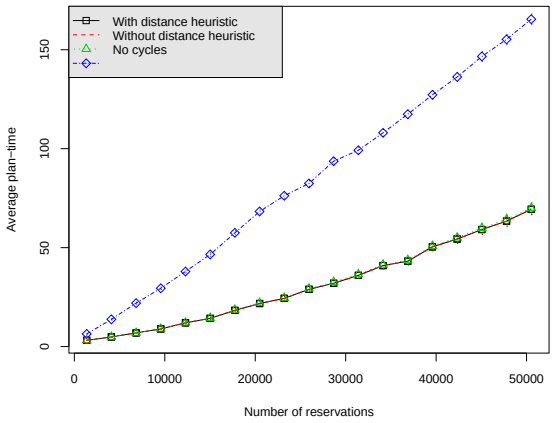
(a) Average CPU-time for increasing amount of resource reservations.



(b) The average number of open list insertions and removals performed to find the plan.



(c) The average number of resources traveled to reach the destination resource.



(d) Average end-times of plans for increasing amount of resource reservations (confidence intervals too small to display).

Figure 5.25: Results for the case where plans generated by context-unaware Algorithm 2.2 of Hatzack and Nebel were used to make reservations.

vations in the system. Figure 5.26 shows that if an agent is 400^{th} in line to make a plan, then its plan cost will approximately be twice the cost of the minimum-cost plan, which is the shortest path when reservations are not taken into account.

This does, however, not mean that the sequence in which airplanes plan makes a difference to the performance of the total *system*. We show the performance of the total system for a group of 500 airplanes routing from and to random resources in the Schiphol infrastructure. The same tasks are repeated 100 times with different random permutations in which the airplanes create their plans and make the reservations.

The relatively small 95% confidence intervals shown in Figure 5.26 indicate that for the Schiphol airport network the order in which airplanes reserve their plans is not signif-

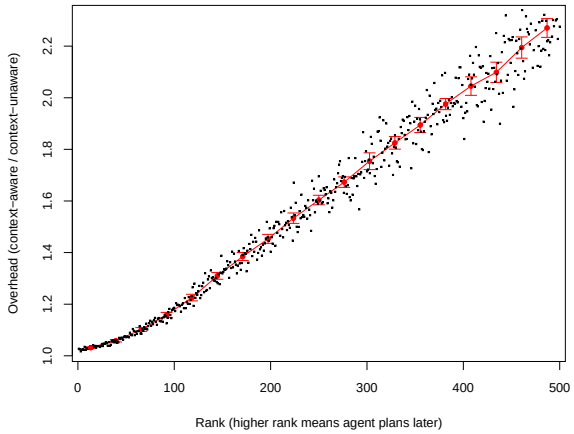


Figure 5.26: The average overhead (plan cost divided by minimum plan cost) increases when an agent plans later.

icant; the overhead for being n^{th} in line does not vary greatly.

5.5 Summary

In this chapter two sets of experiments were described: the general experiments on operational transportation planning and the airplane taxiing experiments.

General experiments The first part of the chapter is about the general experiments. The planning methods were ordered by increasing plan quality. The MACA method has been underestimated and should actually be in between the MACA-RP and MACA-RR methods according to the experiments. It outperforms the MACA-RP method not only in plan quality, but besides that it has much lower costs in terms of CPU time. The ordering of the planning methods with respect to CPU cost in time matched our expectations. Knowing the algorithms, this was much easier to expect in advance than the plan quality.

Interestingly, the general experiments showed that the costs in terms of CPU time depend on the choice of heuristic. Generally, it turned out that heuristic that resulted in higher performance also performed better with respect to CPU time. If agents that make better choices (i.e., use better heuristics) finish creating their plans more quickly.

With respect to incidents, the experiments in Section 5.3.3 showed that the performance of the agents decreases (following an S-shaped curve) if the incident level increases. Furthermore, the experiments showed that the difference between the performance of the various planning methods fades out if the incident level increases.

Section 5.3.4 described the experiments for different transport network topologies. The scale-free topology showed the worst performance, which can be understood, because of the nodes with many connections leading to bottlenecks in the infrastructure. The data

obtained with the experiments further showed that it was the path length that influenced the performance. No significant relation between the number of alternative paths between source and destination resource and the performance was found in the experiments.

Airplane taxiing experiments After the general experiments airplane taxiing on Schiphol airport is considered. Current practice at Schiphol is that airplanes follow fixed routes from runway to gate and back. The experiments in this chapter show, however, that using a context-aware routing method that dynamically inspects alternative routes at any time results in much better performance. Of course, ethical or legal aspects, which might prevent Schiphol to change their current practice, are beyond the scope of this thesis.

The context-aware shortest path algorithm is, on this realistic Schiphol airport transport network, compared to a context-unaware routing approach. Furthermore, two variants of the context-aware shortest path routing, are also considered in the experiments. These are a context-aware A* variant and a version that does not allow to visit resources more than once (only acyclic routes are allowed).

The first thing to notice is that the context-unaware approach is much faster than the other algorithms. The context-aware routing algorithms, however, are also fast. Out of the three context-aware variants, the acyclic version is quite a bit faster than the other two.

Furthermore, it is clear from the experiment that the context-aware algorithms perform much better than the context-unaware approach, especially in situations where detours might be profitable. There happens to be no significant difference in performance between the context-aware shortest path algorithm and its acyclic variant. Apparently, in these experiments it is the case that when cyclic routes are used by the former algorithm, the latter can usually introduce some additional waiting time to end up with the same performance.

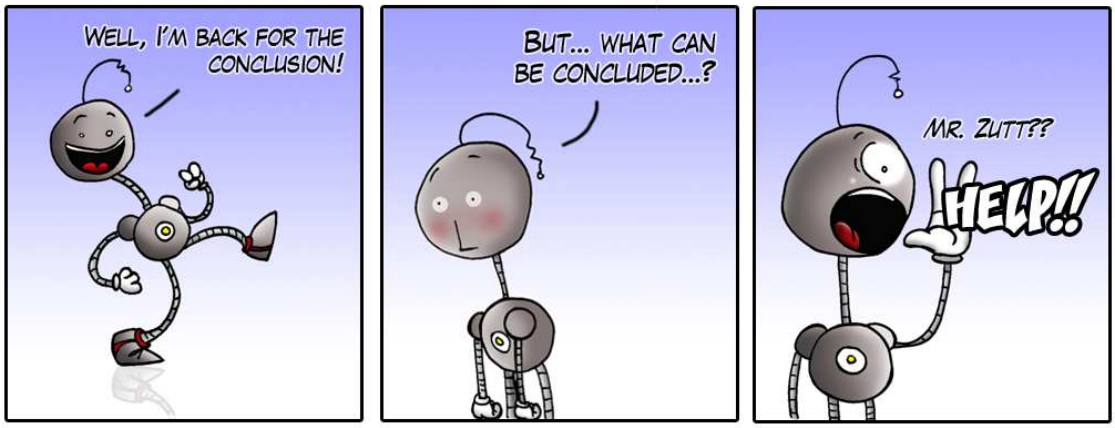
Although it is clear that agents can create better plans if they plan before other agents store their reservations, in the Schiphol airport experiments the order in which agents plan does not make a big difference to the total performance of all agents together. It appears that these performance differences that result of going first or second are leveled out between the agents. We certainly do not claim that this result holds for different network topologies.

Final note In general the default transportation planning algorithm should be a context-aware algorithm. If there is enough time, use the MACA-RR method with the *wait* heuristic. If time is critical then the MACA method might be the best choice. If there is some time for research available, it might be possible to improve on this with perhaps some domain specific heuristics[7].

[7] In such situations the transport planning simulator TRAPLAS, which is used for all experiments in this thesis, and its 3D-visualization project called TRAPLASVIZ are both available at SourceForge.

Chapter 6

Conclusions and extensions



In this thesis we have introduced a new framework for pickup and delivery transport planning, and we tested and compared several new approaches to compute a pick-up delivery transportation planning for a set of vehicles taking into account time constraints for loading and unloading.

The framework distinguishes transport agents and infrastructure agents. The transport agents create transportation plans and they communicate with the infrastructure agents to compute the reservations for accessing the infrastructure resources by the vehicles. This results in a flexible framework where the infrastructure agents can use different policies (such as first-come first-served). Furthermore, in case of incidents the infrastructure agent can inform the transport agents that will be affected by the incident, and modify their reservation(s) accordingly.

Chapter 5 describes the experiments that were performed, varying the transport network topology, the request workload, and the level of incidents. We have seen that the influence of incidents greatly depends on the transport network topology. Intuitively, networks with more alternative equidistant routes between the pickup and delivery locations are less sensitive to incidents than other types of networks.

The main results of our research are the following:

- First of all, by a careful analysis of the context-aware routing approach, we have succeeded in lowering the time complexity of the context-aware routing algorithm of Kim and Tanchoco in a significant way making it much more scalable.
- Secondly, we have investigated the effect of several policies of the infrastructure agents. Of these policies, the wait policy outperforms the other policies both in plan quality as well as in computation time.
- Thirdly, we have investigated the role of incidents in context-aware routing. We can conclude that MACA-RP and MACA-RR are indeed more robust than MACA. Random and Small-World networks are the least sensitive to incidents, while scale-free networks have the worst performance.
- Fourthly, with the airport taxiing experiments we have shown that MACA also works on a realistic network. Dynamic route planning can significantly improve the throughput of the airport.

In general the default transportation planning algorithm should be a context-aware algorithm. If there is enough time, use the MACA-RR algorithm with wait heuristic. If time is criticial, the MACA algorithm is the best choice.

There are many more experiments imaginable, as well as many extensions. The following list presents some extensions, which we found interesting.

Delta heuristic While reconsidering the example given in the introduction, Section 1.3, we came up with the idea of a heuristic that might outperform the ones described in this thesis. The example showed that it would do good to the total performance of the system if the priority of airplane A_4 would be increased. However, none of the mentioned heuristics does so. TODO

How to compute the delta value of agent A_4? Assume that airplanes A_1, A_2 and A_3 reserve their plans, while ignoring any conflicts between those. Now compute the optimal plan for A_4, given that the plans of A_1, A_2 and A_3 remain unchanged and A_4 is not allowed to introduce any new conflicts with those plans. The delta value of agent A_4 is the cost of this plan minus the cost of the plan in case A_4 could reserve its plan prior to A_1, A_2 and A_3. Note that this delta value is always zero or positive (plan costs never decrease after letting another airplane make its reservations first).

Using this delta heuristic airplane A_4 would in all cases reserve its plan (along resources $r_{16},\ldots,r_{12}$) prior to the other airplanes. This still is not the optimal plan, in which airplane A_4 must select the bottom route along resources $r_{21},\ldots,r_{11}$.

K-shortest path routing The approach of Hatzack and Nebel has been used in the experiments in this thesis as an example of an approach that first computes a route (or uses

static routing), and then solves conflicts. This approach can be extended by considering k shortest paths with $k > 1$. Then, for each of the k routes, the conflicts are solved and the best route will be chosen. This approach can significantly improve the results if the transport network allows k routes that do not have too much overlapping resources and at the same time are all approximately of the same length. There are many algorithms published to find these k shortest paths, such as Eppstein (1998).

Distributed version Although the MACA-RP and MACA-RR are not completely centralized algorithms (the supervisor can reschedule any subset of agents and there can be multiple supervisors), some steps can be made towards completely distributed algorithms. In the field of Dynamic Traffic Management (Zuurbier et al., 2006), for example, it is not acceptable to have a routing algorithm with a time complexity depending on the number of transport resources or the number of infrastructure resources in the system. Methods developed in this field determine crossroad priorities based on local information, which might be necessary for large-scale systems.

Multi-objective routing There are many different performance indicators. Hence, it is likely that the actors have different objectives. It is possible to optimize multiple objectives while searching plans for the vehicles. This can be achieved by integrating Traplas and Samcra (P. Van Mieghem and Kuipers, 2001; Kuipers and Mieghem, 2005). Samcra can search for an optimal plan given multiple objectives.

As an example, we consider the use of the following three metrics: *(i)* time, like before (but scaled between 0 and 1), *(ii)* network utilization (minimize resource load), and *(iii)* incident count (gathered historic data). The idea of the second objective is to avoid resources with many reservations, because the probability for an incident is greater for such resources). The idea of the third objective is to avoid resources, where there have been problems in the past. Hence, the two additional objectives attempt to minimize the probability of incidents along the route.

An experiment we performed is to give the first metric the highest priority. In case of ties we considered the second objective. If there was again a tie, the third objective decided on the plan. This did not significantly improve our results. However, Samcra actually uses a different mechanism to combine the different objectives, which might lead to better results. It considers all objectives and chooses the plan that minimizes the worst objective.

Reconsidering waiting time The context-aware routing algorithm always claims subsequent resources as soon as possible, never spending extra time in the current resource if not necessary. This fixed strategy might lead to congested resources. It could sometimes be advantageous to delay the traversal (if possible by waiting in a parking space resource),

thereby reaching the congested area later, while still reaching the destination at the same time. This can decrease network utilization and leave more room for the other vehicles that still have to create reservations.

Arriving too early For problem instances where agents arrive at the source or destination location of a transportation request too early, it is possible to compute whether the agent should slow down before arriving at this location, or arrive too early and take the corresponding penalty. The latter might be a better decision if the agent is in a hurry to reach other locations later in its plan. This problem can be solved by formulating this as a linear program, which can be solved in polynomial time. Of course, if this is done often, it can slow down the planning method considerably. For the experiments in this thesis this is not an issue, because the instances are intentionally made such that agents are always too late, or just-in-time in the optimal case.

Optimization stage Several approaches are described in this thesis that all produce feasible transportation plans. If there is remaining time available, one possible idea to make use of this time is to introduce an optimization stage. Many local search techniques exists that can be used in an attempt to further increase the performance.

Communication incidents The incidents that have been considered in the experiments in this thesis are malfunctioning resources. The speed of infrastructure or transport resources was reduced by a factor (the impact of the incident) between 0 and 1. Another common source of incidents is failing communication between transport resources and the planning system. For example, in container terminals the speed of transport resources is reduced if the transport resources have no communication with the system.

Two more topics that we have considered are collaboration by exchanging transportation requests, and the effect of mixing different transportation planning methods. Because we can also show some experimental results here, a section is devoted to each.

6.1 Collaboration

The transport planning methods described in Chapter 4 do not modify the assignment of transportation requests to the vehicles, and also do not modify the order in which the requests are executed. It is possible that an assignment, that might have been a good choice in the past, becomes a problem later – either due to an incident, or a better plan might have become available due to changes in the plans of other agents.

For this reason, a *collaborative* strategy is included, in which the transport agents

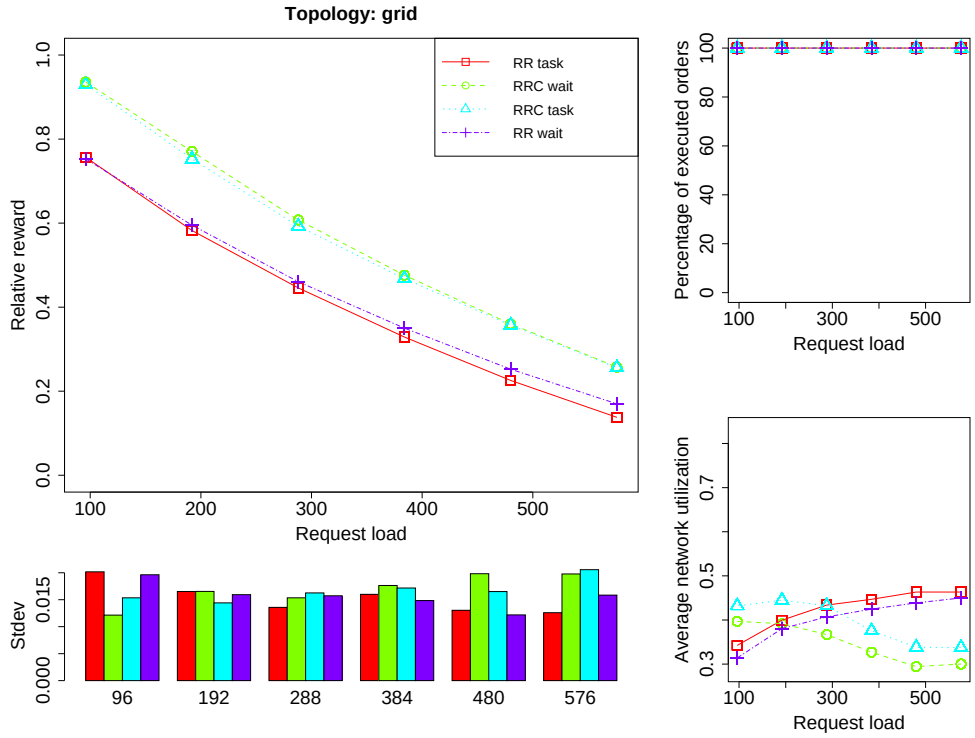


Figure 6.1: Average relative reward for MACA-RR with *wait* and *task* heuristic on grid networks. The two extreme values 0 (RR) and 1 (RRC) are used to set the collaboration level of the agents.

exchange transportation requests with each other. We claim that this will lead to a statistically significant increase in the performance of the agents.

To model collaboration between a group of agents, a property called *collaboration level* is added to each agent, which is a number between 0 and 1 indicating its collaboration level. If this value approaches 0, the agent only wants to exchange transportation requests to its own benefit. If it approaches 1, the agent is more aware of the system as a whole and exchanges transportation requests with other agents if it is better for the system. For the experiment in this section, we restricted to exchanging transportation requests one by one, i.e., each agent tries to give each of its requests to another agent. It is also possible that, in a certain situation, performance can be improved by giving away two requests and receiving one requests at the same time. Finding such request exchanges, however, will cost much more in terms of CPU time.

Figure 6.1 shows the results of the two extreme values 0 and 1 for the collaboration level. This clearly shows there is a big advantage letting the agents exchange transportation requests with each other. Using collaboration is always advisable. It can be used to improve the current plans until the time is reached at which an immediate decision is to be made (i.e., as an *anytime* algorithm).

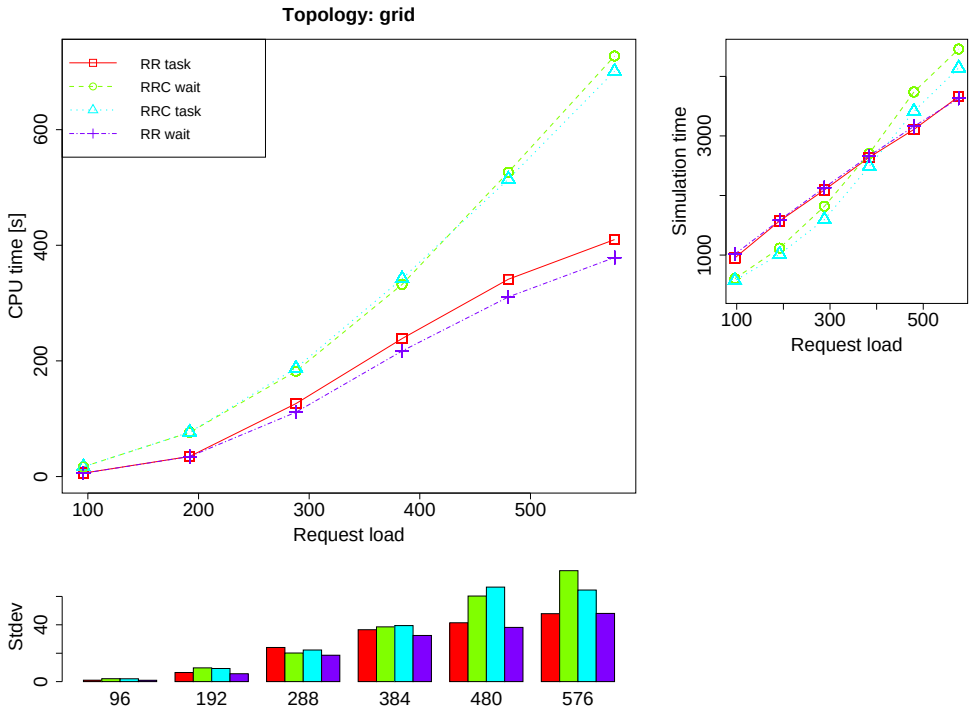


Figure 6.2: CPU costs (in time) required by MACA-RR with *wait* and *task* heuristic on grid networks. The two extreme values 0 (RR) and 1 (RRC) are used to set the collaboration level of the agents.

Figure 6.2 shows the additional CPU time that is needed by the collaborative version of the MACA-RR method. Up to a certain number of transportation requests it might be acceptable, but with many transportation requests, it will become too slow. Notice, however, that in such cases the method can easily be sped up by not trying all possible exchanges of requests.

6.1.1 Collaboration and incidents

For the experiment described in this section, the number of transportation requests is fixed to 192 requests. Then, we varied the incident level. It can be seen in Figure 6.3 that the performance decreases in a similar way. If the incident level reaches its maximum value, it does not matter much which method is used, the performance reaches its minimum value. Hence, we can see the methods come closer to each other as the incident level increases.

Furthermore, the MACA-RRC *wait* method, which performs best without incidents, also performs best when the incident level increases. Because the performance of all methods decreases in a similar way, there is no reason to use one method for a certain level of incidents, while choosing another method at another incident level, when performance

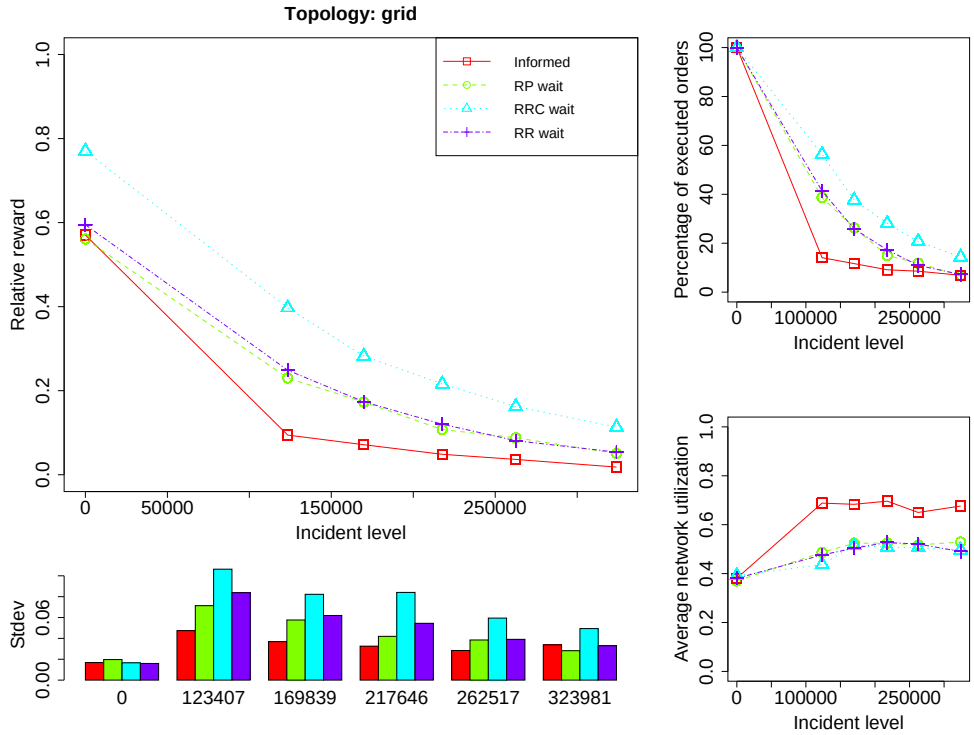


Figure 6.3: The relative reward (averaged over all requests) for a selection of planning methods on grid networks; the incident level is increasing and the request load is fixed to 192 requests.

is considered.

Figure 6.4 shows the CPU cost (in time) of the same experiment. Here it becomes clear that the CPU time required by the MACA-RR *wait* method grows rapidly if the incident level grows. It grows significantly faster compared to the results of the previous section, where the number of transportation requests increased along the horizontal axis.

The next section considers the effect of mixing different transport planning methods in the same environment.

6.2 Mixed strategy

Until now it was assumed that all agents in the system applied the same planning method. Combining different methods in the same setting might also influence the performance of the agents. For example, suppose that a system with altruistic agents reaches the highest performance encountered in experiments. But now add an equal amount of selfish agents. Which group will have the best total performance? Combinations of planning methods occur in many real-life systems, where one usually does not have full control over all agents in the system. Rather, there are several parties (e.g., companies) involved that each

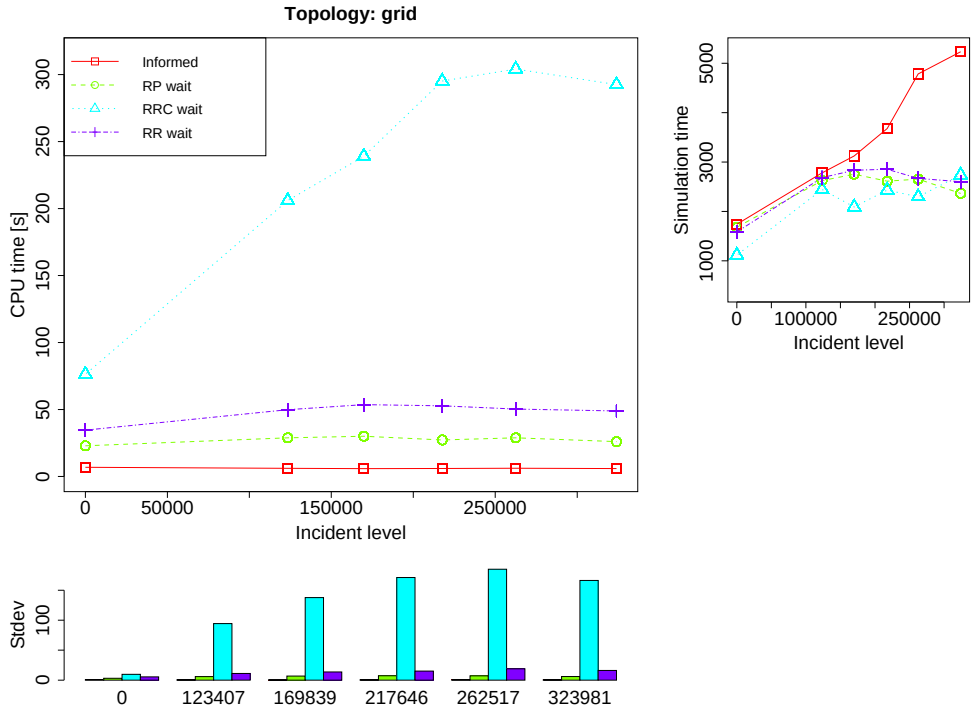


Figure 6.4: The CPU costs (in time) required by the selection of planning methods on grid networks; the incident level is increasing and the request load is fixed to 192 requests.

control a subset of agents in the system. The possible influence of having mixed planning methods will be investigated.

In many application domains, the system designer does not have full control over the system. This might be the case when multiple companies are involved who do not want to share all of their private data, or when the agents of the system designer should be able to mingle with existing traffic. In these situations the question arises whether planning methods that have been proven to be the most efficient in the previous sections will still be the best performing methods. Perhaps other methods outperform these methods, because they can handle these external traffic flows better. In this section the planning methods are tested in this setting.

The agents are divided into groups. Agents within the same group have the same behavior, they use the same planning method. Other groups contain agents that use other planning methods. The relative rewards presented in Figure 6.5, are not summed over all agents in the system, but instead, it is the contribution to the system performance of the agents per group. This means the total relative reward is actually the sum of the contributions of all groups.

As can be seen in Figure 6.5, the MACA-RR method with *wait* heuristic and collaborative agents gains the highest performance. The ordering of planning methods did not change compared to the situation where all agents in the system used the same planning

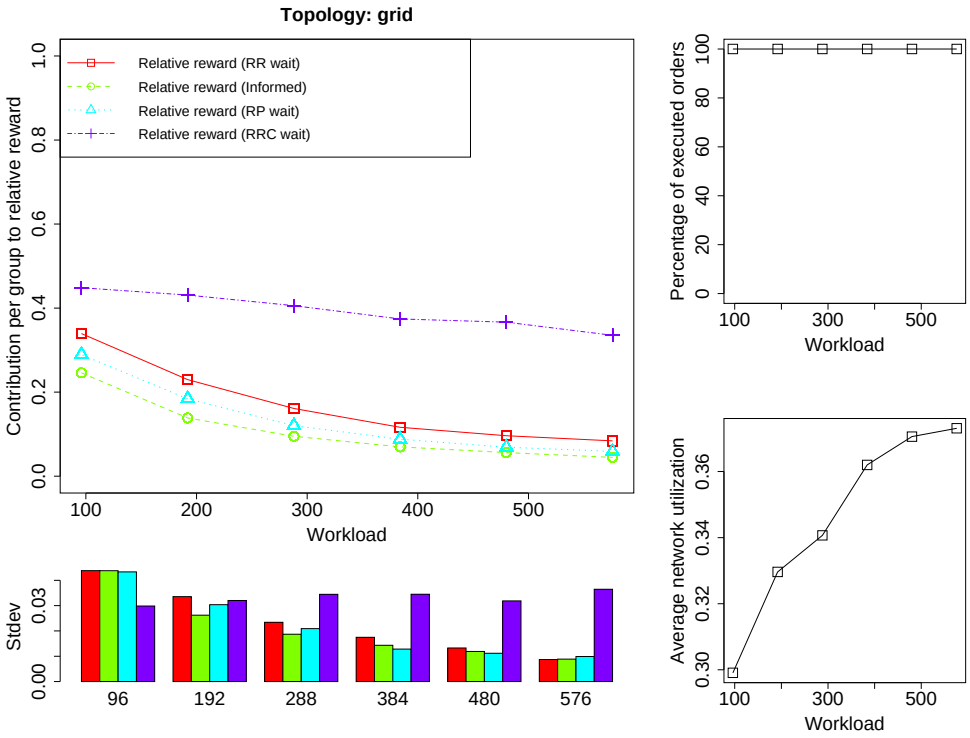


Figure 6.5: The 32 agents are divided into four groups of eight agents. The agents within the four groups all use the same planning method. Sum the contributions to the relative reward of the individual groups to obtain the relative reward of the complete system. The subplots on the right are for the whole system.

method.

It is apparent that one cannot just apply one of the simple planning methods and hope for good performance in general. Even with some noise of other agents using other planning methods, it turns out the better planning methods (MACA-RRC *wait*) still outperform the other planning methods.

Appendix A

Notation

As a guide for the notation used throughout this dissertation, the following table lists several important used symbols. Most of the time, a standard is adopted that uses small letters for simple variables, capital letters for sets, small Greek letters for composed variables and a calligraphic font for structures. The section column refers to the section where the symbol is defined.

symbol	meaning	section
I	infrastructure $I = (R, E_R, k^{inf}, k^{tr}, d^{inf}, s^{inf}, s^{tr})$ is a tuple of resources R, arcs E_R, infrastructure resource capacity function k^{inf}, transport resource capacity function k^{tr}, infrastructure resource distance function d^{inf}, infrastructure resource speed function s^{inf} and transport resource speed function s^{tr}.	3.1.1
R^{inf}	set of infrastructure resources.	3.1.1
R^{tr}	set of transport resources.	3.1.1
R	set of all resources $R = R^{inf} \cup R^{tr}$.	3.1.1
E_R	infrastructure resource connectivity relation $E_R \subseteq R^{inf} \times R^{inf}$.	3.1.1
k^{inf}	$k^{inf}(r) \in \mathbb{N}$ is the capacity of resource $r \in R^{inf}$.	3.1.1
k^{tr}	$k^{tr}(v) \in \mathbb{N}$ is the capacity of transport resource $v \in R^{tr}$.	
L_R	a family of functions, where $L_r(t) \in \mathbb{N}$ is the current load at infrastructure resource $r \in R^{inf}$ at time $t \in T$. For transport resource $v \in R^{tr}$, $L_v(t) \in \mathbb{N}$ is the sum of the volumes of the loaded packages at time t.	3.3.2
d^{inf}	$d^{inf}(r) \in \mathbb{R}$ is the distance of resource $r \in R^{inf}$.	3.1.1
...	(continued on next page)	

symbol	meaning	section
s^{inf}	$s^{inf}(r) \in \mathbb{R}$ is the maximum possible speed at resource $r \in R^{inf}$.	
s^{tr}	$s^{tr}(v) \in \mathbb{R}$ is the maximum possible speed at resource $v \in R^{tr}$.	
E_f	$E_f(r,t) \in [0,1]$ is the effective impact of incidents with resource $r \in R$ at time t taking into account incidents known at the time of computation.	
s_s	$s_s(v_t, r_i) \in \mathbb{R}$ is the static speed of traversing infrastructure resource $r_i \in R^{inf}$ by transport resource $v_t \in R^{tr}$ not taking into account incidents.	
s_d	$s_d(v_t, r_i, t) \in \mathbb{R}$ is the speed of traversing infrastructure resource $r_i \in R^{inf}$ by transport resource $v_t \in R^{tr}$ at time $t \in T$ taking into account incidents known at the time of computation.	
T	$T = \mathbb{R} \cup \{-\infty, \infty\}$ is the set of all possible time points.	3.1
W	set of all possible time-windows (intervals of time) $W = T \times T$.	3.1
t	a single point in time $t \in T$.	3.1
τ	a single time-window $\tau \in W$.	3.1
$\check{\tau}$	an actual time-window $\check{\tau} \in W$ that was used for some event, e.g., loading freight, during simulation.	
$lb(\tau)$, $ub(\tau)$	lowerbound and upperbound of window $\tau = (lb(\tau), ub(\tau))$.	
O	total set of transportation requests.	3.1.2
O_v	set of transportation requests assigned to transport resource $v \in R^{tr}$.	3.1.2
o_j	$o_j = (f_j, s_j, \tau_j^s, d_j, \tau_j^d, \pi_j) \in O$ is a transportation request: freight should be picked up in source location $s_j \in R^{inf}$ within time-window $\tau_j^s \in W$, it should be delivered in destination location $d_j \in R^{inf}$ within time-window $\tau_j^d \in W$, and the reward for doing so is $\pi_j(\check{\tau}_j^s, \check{\tau}_j^d) \in \mathbb{R}$ where $\check{\tau}_j^s \in W$ and $\check{\tau}_j^d \in W$ are the realized pick-up and delivery time-windows respectively.	3.1.2
...	(continued on next page)	

symbol	meaning	section
μ	performance, if not explicitly mentioned otherwise the *relative system reward*, i.e., the sum of realized pickup and delivery rewards of all requests divided by the pickup and delivery rewards in case the requests would have been loaded and unloaded within the specified time-windows.	
ψ	CPU cost in terms of time	
$\mathcal{I}$	set of incidents. An incident $(t_r, r, i, \tau) \in \mathcal{I}$ is a resource incident, which is announced to the agents at release time $t_r \in T$, affects infrastructure or transport resource $r \in R$, has impact $0 \leqslant i \leqslant 1$ and is effective during time-window τ.	3.1.4
A	set of agents.	3.3.2
Rt_v	route Rt_v of transport resource $v \in R^{tr}$ is $Rt_{v,1}, \ldots, Rt_{v,n}$.	3.3.2
Sd_v	schedule Sd_v of transport resource $v \in R^{tr}$ is $Sd_{v,1}, \ldots, Sd_{v,n}$.	3.3.2
Q	$Q(r) \subseteq A \times W$ is the set of agent and time-window pairs stored at infrastructure resource $r \in R^{inf}$.	
L_v	$L_v(o) \in T$ is the time at which transport resource $v \in R^{tr}$ picks up the freight of transportation request $o \in O_v$.	3.3.2
U_v	$U_v(o) \in T$ is the time at which transport resource $v \in R^{tr}$ delivers the freight of transportation request $o \in O_v$.	3.3.2
$\operatorname{argmin}_{s \in S} f(s)$	the argmin denotes an arbitrary value $m \in S$ for which $f(m) = \min_{s \in S} f(s)$ holds.	
$\operatorname{argmax}_{s \in S} f(s)$	the argmax denotes an arbitrary value $m \in S$ for which $f(m) = \max_{s \in S} f(s)$ holds.	

Table A.1: List of symbols.

Appendix B

Glossary

Agent

AGV Abbreviation for Automatic Guided Vehicle or Autonomous Guided Vehicle. The Material Handling Institute defines an Automatic Guided Vehicle as “a vehicle equipped with automatic guided equipment, either electromagnetic or optical. Such a vehicle is capable of following prescribed guide paths and may be equipped for vehicle programming and stop selection, blocking, and any other special functions required by the system”.

Algorithm An algorithm is any well-defined computational procedure that takes some value, or a set of values, as input and produces some value, or a set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

Deadlock [Gridlock is a deadlock due to spill-back?]

Dependent variables The variables that are measured, as opposed to variables that are varied, during an experiment. Also referred to as performance indicators or score variables.

Gridlock Gridlock is a term describing an inability to move on a transport network. The term originates from a situation possible in a grid network where intersections are blocked, prohibiting vehicles from moving through the intersection or backing up to an upstream intersection.

The term gridlock is also widely used to describe high traffic congestion with minimal flow (a traffic jam), whether or not a blocked grid system is involved. By extension, the term has been applied to situations in other

fields where flow is stalled by excess demand, or in which competing interests prevent progress. [copied from Wikipedia, for a visual hint, look here: http://en.wikipedia.org/wiki/Image:New_York_City_Gridlock.jpg]

Independent variables Variables that are under control of the experimenter. Also referred to as experimental variables.

MACA Multi-Agent Context-Aware routing, see Section 4.2.2.

Makespan The makespan of a schedule refers to the total execution time. This is the time at which the last agent finishes plan execution. Often, one attempts to minimize the makespan, which comes down to minimizing the maximum completion time of all agents.

Mobile entity

Resource

Transport resource

Appendix C

Introduction to agents

In 1950 Alan Turing proposed his famous *Turing test*, designed to provide a satisfactory operational definition of intelligence. A human operator interrogates the computer via a teletype. If the human cannot distinguish whether there is another human being or a computer at the other end of the line, the computer has passed the test and should be called intelligent. Shortly after the Turing test, the term *artificial intelligence* was first mentioned.

The term *agents* was introduced by Putnam (1960). It is likely the idea originates from psychology. Skinner (1953) tried to define the psychology of organisms by solely using input/output or stimulus/response mappings. Nowadays, there exist many definitions for the concept agent, though none of them is generally accepted. Starting of with a weak definition of an agent by Russell and Norvig (1995), several properties are listed here that are generally required to be present for an entity to be called an agent. Then, some properties are listed that for some people are necessary, for others superfluous, conditions to call something an agent.

Definition C.1 (Agent) An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors (Russell and Norvig, 1995).

In general, it is accepted that agents must at least have the following properties:

- Autonomy – agents have the ability to act without being told what to do and when by others.
- Persistence – agents are not products that are produced or consumed, but live a relatively long life.
- Computational abilities – presence of non-trivial computations; this, for example, excludes thermostats from being agents.

Comment: - Insert image here: overview of an agent: e.g., beliefs, control, goals, strategy, input, actions.
- The term "agent" describes a software abstraction, an idea, or a concept, similar to OOP terms such as methods, functions, and objects. The concept of an agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its user. But unlike objects, which are defined in terms of methods and attributes, an agent is defined in terms of its behavior. [Introduction to MultiAgent Systems, Michael Woolridge].

In addition, some people require even more before they call something an agent. Some of these properties (Graham, 2001) are:

- Mobility – the opportunity for an agent to move around.
- Veracity – agents will not knowingly communicate false information.[1]
- Benevolence – agents occupy the unbiased kindness to do good.
- Rationality – an agent will act in order to achieve its goals and not to prevent its goals from being achieved.

Agents can either be humans, software or robotic agents. There are many realistic systems that consist of a mixture of human and software agents (e.g., intelligent user interfaces, e-commerce, search engines).

Agent-based theory has been well studied. This has resulted in the many agent platforms that are available, for example, the Java Agent Development Environment (Jade), Java Agent-based Simulations (JAS), System for Parallel Agent Discrete Event Simulation (SPADES) and many others. And also agent communication languages like Knowledge Query Manipulation Language (KQML) or the one used at Foundation for Intelligent Physical Agents (FIPA).

A group of these agents can form a multi-agent system. According to Graham (2001) the main multi-agent system specific design issues are *communication*, *interaction*, *coherence* and *coordination*. Multi-agent systems offer a high level of encapsulation and abstraction. Agents can be created by different developers, as long as they can agree on how they communicate with each other. Multi-agent systems can very well be used in the transportation domain.

Definition C.2 (Multi-agent system) A multi-agent system is a loosely coupled system network of problem solvers that work together to solve problems that are beyond their individual capabilities (Durfee, 1999).

[1] Note that this does not include honest mistakes.

Definition C.3 (Coordination) Coordination is the regulation of diverse elements into an integrated and harmonious operation. Coordination means integrating or linking together different parts of an organization to accomplish a collective set of tasks (Malone and Crowston, 1994).

Appendix D

Complexity of transport planning

When evaluating algorithms for transportation planning, a point of reference is desired. Without this, only empirical results can be gained for the performance of a candidate algorithm. There might exists other ones that are much better. One way to give such a reference point is to look at complexity theory (see Moret (1998) for an extensive survey on complexity theory). The objective of complexity theory is to establish bounds on the behavior of the best possible algorithm for solving a given problem — whether or not such algorithms are known.

Within a complexity class, a problem is called *complete*, if every other problem in the same class can be reduced to this problem. When a problem is complete, it belongs to the most difficult problems of that complexity class.

A well-known and often used complexity class is NP (Non-deterministic Polynomial Time). To this class belong, for example, famous problems like Satisfiability, the Traveling Salesman Problem, and the Vehicle Routing Problem. It has not been shown that any of the problems in the class NP truly requires exponential time; however, completeness in this class may safely be taken as strong evidence of intractability. If any complete problem would also be solvable efficiently, then all problems in the class would be solvable efficiently.

We are now going to prove, that the decision variant of the transportation planning problem belongs to NPC, the class of NP-complete problems. Optimization problems can easily be changed into decision problems by setting a threshold value to the optimization criterion. The question "what are the minimal costs to execute all transportation requests?" is transformed, after adding a variable $K \in \mathbb{N}$, into "is there a solution to execute all transportation requests with costs less than or equal to K?". It is obvious that the search/optimization variant of a decision problem never is easier than the corresponding decision variant.

Definition D.1 Decision variant of transport planning (DTP)

Given the model of transport planning in Section 3.1 and the performance criterion that measures the total distance traversed by the transport resources, does there exist a plan for the agents that execute all transportation requests with total distance traversed by the transport resources less than or equal to K?

To prove the NP-completeness, the decision variant of a well-known NP-complete problem Shortest Hamilton Path is used. This problem is among a huge list of problems known to be NP-Complete in the compendium of NP-complete problems by Love (1999).

Comment: This reference might be about the non-decision variant SHP.

Definition D.2 Decision variant of Shortest Hamilton Path (DSHP)

Given an undirected and complete[1] graph $G = (V,E)$ and a distance function $d : E \rightarrow \mathbb{R}^{\infty}$, does there exist a path that visits all nodes exactly once of which the sum of the distances of all edges in the path is less than or equal to K?

Theorem D.3 *The decision variant of the transport planning problem (DTP) belongs to NPC, the class of NP-complete problems.* □

Comment: The following proof requires major revision.

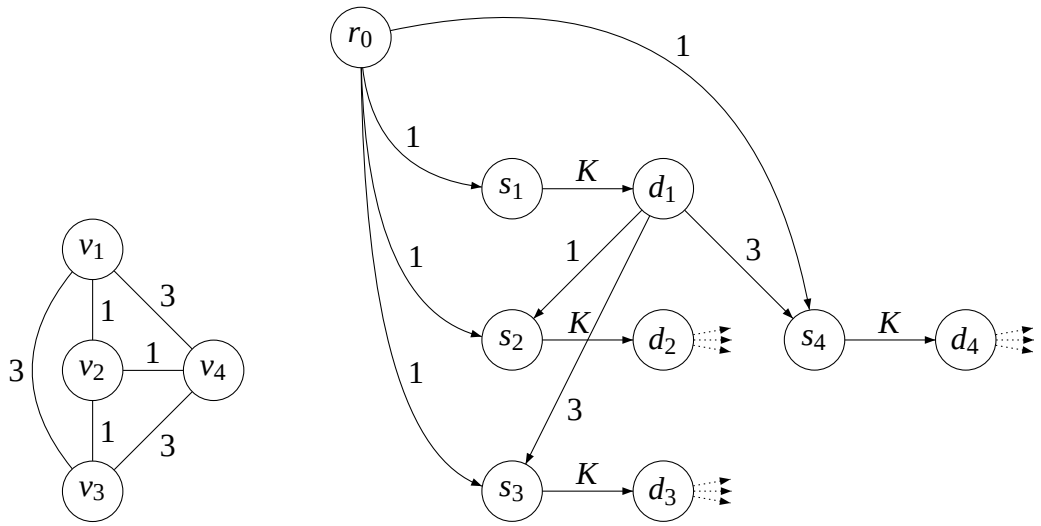
PROOF: We have to show that DTP belongs to NP and that any other problem in NP can be reduced to DTP.

The fact that DTP belongs to NP can easily be seen. We must be able to verify in polynomial time that a 'yes'-instance Y_{DTP} of DTP has indeed costs less than or equal to K given the plan for all the trucks. This means we only have to calculate the cost function and confirm it is less than or equal to K.

Next we prove, that the NP-complete problem Shortest Hamilton Path (DSHP) can be reduced to DTP. Since DSHP is NP-complete (Love, 1999), any problem in NP can be reduced to DSHP. And if DSHP can be reduced to DTP, then any other problem can be reduced to DTP too and DTP is NP-complete.

Figure D.1 illustrates the transformation from an arbitrary DSHP instance to a DTP instance. Every node $v_i \in V$ in DSHP is duplicated and represented by two infrastructure resources s_i and d_i in DTP. A connection is added from infrastructure resource s_i to infrastructure resource d_i with distance $K+1$. A transportation request must be planned from pick-up resource s_i to delivery resource d_i. The volume of all transportation requests is equal to the capacity of the truck. The pick-up and delivery time-windows of all transportation requests are infinite $[-\infty,\infty]$, so these are never violated. There is one transport

[1] Complete graphs have a connection between each pair of nodes. Its distance might be infinite however.



(a) Arbitrary instance of SHP.

(b) Result when Figure D.1a is transformed (arcs leaving d_2, d_3, and d_4 are omitted for clarity).

Figure D.1: Transformation from an arbitrary Shortest Hamilton Path (DSHP) instance to the decision variant of transport planning (DTP).

resource starting in an extra created infrastructure resource r_0, that has connections to every pick-up resource. These connections have length 1. Every undirected edge (v_i, v_j) in DSHP is transformed into two directed connections (s_i, p_j) and (d_j, s_i) of the same length in DTP.

More formally, the transformation from a DSHP instance (G, d, K) to a DTP instance $(L, A, d', T, cap, loc, O, K')$ is the following:

- For every node in the DSHP graph, we create two locations s_i and d_i in DTP. And for the transportation resource, we create one extra infrastructure resource r_0, so $L = S \cup D \cup \{r_0\}$, where $S = \{s_i : v_i \in V\}$ and $D = \{d_i : v_i \in V\}$.

- There is one transport resource, initially placed in location r_0 and with a capacity of one: $R_t = r_t$, $loc(r_t) = r_0$ and $cap(r_t) = 1$.

- Each node in DSHP results in one transportation request after the transformation. The volume of the package in this transportation request is equal to the capacity of the truck and all time-windows are infinite (they can never be violated): $O = \{(s_i, d_i, cap(r_t), [-\infty, \infty]) : s_i \in S \wedge d_i \in D\}$.

- The infrastructure resource connectivity relation in the DTP instance is $E = E_1 \cup E_2 \cup E_3$ where E_1, E_2, and E_3 are defined as follows:

1. There are connections from every pick-up location to its corresponding delivery location. The travel time of these arcs is K:
   $E_1 = \{(s_i, d_i) : s_i \in S \wedge d_i \in D\}$
   $d'(a,t) = K + 1 \; \forall a \in E_1, \forall t \in T$
2. There are connections from infrastructure resource r to every pick-up location. The travel time of these arcs is 1:
   $E_2 = \{(r,s) : s \in S\}$
   $d'(a,t) = 1 \; \forall a \in E_2, \forall t \in T$
3. There are connections from every delivery location to every pick-up location of an other transportation request. The travel time of these arcs is the same as the distance between the corresponding locations in the DSHP instance:
   $E_3 = \{(d_j, s_i) : s_i \in S \wedge d_j \in D \wedge i \neq j\}$
   $d'((d_j, s_i), t) = d(v_j, v_i) \; \forall (d_j, s_i) \in E_3, \forall t \in T$

- $K' = (|V| + 1)(K + 1)$

The correctness of the transformation is proven by showing *(a)* that a yes-instance of DSHP is always transformed into a yes-instance of DTP and *(b)* a no-instance of DSHP is always transformed into a no-instance of DTP. Also, *(c)* the transformation must be computable in polynomial time.

a) Suppose we have a yes-instance Y_{DSHP} of DSHP. This means there must exist a path $(v_{x_1}, v_{x_2}, v_{x_3}, \ldots, v_{x_n})$ that visits all n nodes and has costs less than or equal to K. After the transformation, the transport resource can follow the same path $(r_0, s_{x_1}, d_{x_1}, s_{x_2}, d_{x_2}, \ldots, s_{x_n}, d_{x_n})$. This path will execute all transportation requests, because it contains all pick-up and delivery locations (and the capacity of the transport resource is no problem, since never more than one transportation request is loaded). The truck will not violate time-windows, because we only have infinite time-windows. The costs are 1 for going to the first pick-up location, $K + 1$ for traveling from each pick-up location to its corresponding delivery location and R for the rest of the plan. The latter costs R are exactly the costs of instance Y_{DSHP} and, since Y_{DSHP} is a yes-instance of SHP, we have that $R \leqslant K$. The total costs for the agent are

$$\begin{aligned} 1 + |O|(K+1) + R &\leqslant 1 + |O|(K+1) + K \\ &= |V|(K+1) + K + 1 = (|V|+1)(K+1) \\ &= K' \end{aligned}$$

And therefore, the instance of DTP is also a yes-instance.

b) Suppose the transformed instance $R(I)$ is a yes-instance of DTP. This means that there is a plan with costs less than or equal to K'. In such a plan, all arcs of length $K+1$ are traversed, because all transportation requests must be executed. Thus we have

$$C(Plan(t)) \leqslant K' = (|V|+1)(K+1)$$
$$C(Plan(t)) \geqslant |V|(K+1)$$

Combining these equations shows that the truck has $K+1$ left to visit all pick-up locations. The first action of the truck is to drive from its initial position to the first pickup, this arc has costs 1. Note that for visiting all other pick-up locations, an arc is used that has the same costs as an edge in the DSHP instance. Since we stated that $R(I)$ is a yes-instance of DTP, there is a path that visits all pick-up locations and has costs less than or equal to K. This path cannot visit any location more than once, because then, for some i, an arc (s_i, d_i) must be traveled more than once with costs $K+1$. The path corresponds to a Hamilton path in DSHP with costs less than or equal to K. And therefore, if instance $R(I)$ is a yes-instance of DTP, then instance I is a yes-instance of DSHP.

c) It is easy to see that this transformation can be done in polynomial time. One transport resource is created and $2|V|+1$ infrastructure resources. There are $|V|$ connections from the initial position of the transport resource to the pick-up locations, $|V|$ connections from the pick-up to the corresponding delivery locations and $2|E|$ additional (directed) connections (i.e., the connections in the DSHP-instance). Then, $|V|$ transportation requests are created and the value K' is computed (in $\mathcal{O}(1)$ time). The complete transformation can be done in $\mathcal{O}(|V|+|E|)$ time. ■

The decision variant of transport planning (DTP) and thus also the transport planning problem are NP-complete.

Appendix E

Transport planning simulator - TRAPLAS

Intro ... Before that several important properties of the simulation tool are described. We start with the Pamela run-time library that can be viewed as the simulation kernel. Pamela supports light-weight processes and semaphores and here we describe how that can be used for communication between the agents and enforcing capacity constraints on the transport network.

Comment: - Some TraplasViz screenshots.

E.1 Pamela run-time library

TRAPLAS is based on the Pamela run-time library [cite: technical report AvG].

The Pamela run-time library provides a concurrent, general-purpose performance simulation interface, based on the procedure-oriented ("P/V-style") paradigm [cite: Andrews and Schneider, "Concepts and notations for concurrent programming"].

The library contains two important data types, *(i)* processes and *(ii)* semaphores.

Processes are light-weight threads with their own local time stamp, in which the global simulation time is stored either at which it has been suspended (in the past) or at which it is resumed (in the future). At any time, only one process can be *running*. All other processes are either *runnable* – i.e., scheduled to run at their local time stamp – or they are *blocked* – i.e., waiting on a semaphore until another process lifts this block by executing a `pam_V` operation on this semaphore. Pamela processes are scheduled non-preemptively. This means that execution of a process continues until it voluntarily releases control.

Several important function in this library with respect to the processes and semaphores are listed here together with their description.

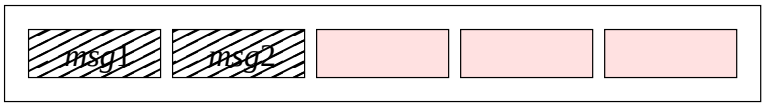


Figure E.1: Message queue with buffer size 5.

`pam_time()` Get the current global simulation time.

`pam_delay(`*s*`)`** Calling process delays *s* seconds. Control is (in general) given to another process.

`pam_alloc()` ...

`pam_P(`*sema*`)`** If semaphore *sema* does not have sufficient credit, the calling process is blocked until another process lifts the semaphore's credit. If the semaphore does have enough credit, it is decreased and the calling process remains the running process.

`pam_V(`*sema*`)`** The credit of semaphore *sema* is increased. The calling process always remains running.

`pam_T(`*sema*`)`** This function returns (tests) the current credit of semaphore *sema*. If it is negative, it indicates the number of processes that are blocked on this semaphore.

`pam_fork()` ...

`pam_exit()` ...

`pam_quit()` ...

E.2 Communication

Comment: Type of simulation (event-based, about time, etc., discrete event simulation mechanism)
- H&N scheduling, event graph (using messages) for starting replanning rounds. Maybe also planning by agents in general (e.g., message flow diagrams)
- deadlock (cycle) detection: Tortoise and Hare (optimal complexity).

Communication functions.

```
 1: procedure SEND(Message m)
 2:     pam_P(room)
 3:     push(m)
 4:     pam_V(data)
 5: end procedure

 6: procedure RECEIVE
 7:     pam_P(data)
 8:     pop(m)
 9:     pam_V(room)
10:     return m
11: end procedure

12: procedure RECEIVE_IMMEDIATELY
13:     if pam_T(data) > 0 then
14:         return Receive()
15:     else
16:         return NO_MSG_AVAILABLE
17:     end if
18: end procedure
```

E.2.0.1 Execution

The same execution method regardless of what planning method is chosen (good safety check).

Capacity of infrastructure resources safeguarded by semaphores.

A cycle detector watches for deadlocks. Two-cycles are not allowed. Simultaneous exchanges are.

Comment: What about too many swaps?

E.2.0.2 Statistics

MEASURE(variable, value, save_to_log) [lunchlezing]

Number of samples, Minimum, Maximum, Sum, Average, Variance, Skewness (symmetry around mean), Kurtosis (peakedness/flatness)

Last four (four moments) can be used to determine the distribution using Generalized

Traveling from infrastructure resource r_1 to resource r_2.

```
1: procedure DRIVE(r1, r2)
2:     pam_P(cap(r2))
3:     arbitrated_pam_V(cap(r1))
4:     pam_delay(drive_cost)
5: end procedure

6: procedure ARBITRATED_PAM_V(sema)
7:     pam_V(sema)
8:     Reschedule processes blocked for sema
9: end procedure
```

Lambda Distribution (GLD).

$$M_r = \frac{1}{N} \sum_{i=1}^{N} (X_i)^r \tag{E.1}$$

E.3 Distributed ASCI Supercomputer (DAS-3)

Figure E.2: The DAS-3 cluster at the Delft University of Technology consists of 68 dual-CPU 2.4 GHz AMD Opteron DP 250 compute nodes, each having 4 GB of memory and 250 GB of local HD space. The cluster head node consists of a dual-CPU/dual-core 2.4 GHz AMD Opteron DP 280 with 4 GB of main memory and an additional RAID6 storage system of 5 TB. The cluster is equipped with 1 and 10 Gigabit/s Ethernet.

Appendix F

Transport network topologies

This section describes different transport network topologies, which occur frequently in practice. For all of the topologies exist algorithms that can create random instances, which are used for the experiments in Chapter 5 to generate the required transport networks. But first, several transport network properties are listed, by which the topologies differ. The performance of these different topologies with corresponding different network properties will be tested and compared in Chapter 5; it is likely not only the performance in normal circumstances, but also in case there are incidents will be very different. For example, a network with multiple alternative routes from a source to a destination location is likely to be less influenced by incidents, because agents can take a detour if needed.

The network properties characterizing the different topologies are:

- diameter: the longest path in the network,
- options: the average number of equidistant alternative routes between any pair of locations,
- average path length,
- clustering coefficient: how many of a location's connections are also connection to each other?
- betweenness: degree the shortest path between two other locations is through this location.

Regarding the clustering coefficient property, if a location has n connections, there can be at most $n(n-1)/2$ connections between the connected locations. If there are in fact m connections between those connected locations, the clustering coefficient C is defined as $C = 2m/n(n-1)$. Hence, the clustering coefficient is always $0 \leqslant C \leqslant 1$. It is 1 for fully-connected networks; it is approximately $1/|R^{inf}|$ for random networks, but it has a high

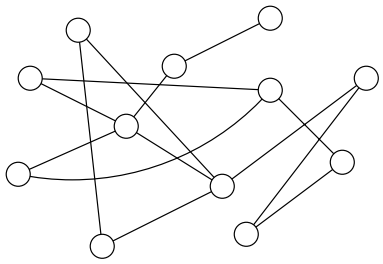


Figure F.1: Example of the random topology.

variability. The betweennees is measured by computing all shortest paths between all pairs of locations and incrementing counters for intermediate locations on these shortest paths.

The different topologies are the random, tree, grid, small-world, and scale-free topologies, which will be described in turn in this section.

F.0.1 Random topology

Although it was stated before in this thesis that it is not sufficient to look at random transport networks – one would not have enough time to process a representative set of transport networks in experiments, it is still an important topology to consider.

Three different ways can be adopted to create a random transport network, these are:

1. create a spanning tree, then randomly add more edges,

2. create a fully-connected network, then randomly remove edges,

3. create an edge for each of the $\binom{n}{2}$ pairs of locations with a given probability.

An advantage of the first method, which will be used in Chapter 5, is that the resulting transport network is always connected, i.e., it is possible to traverse from any given location to any other location. If a transport network would not be connected, one would first have to divide the problem instance into smaller instances, taking into account that some transportation requests simply cannot be executed.

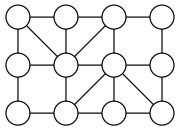


Figure F.2: Example of the grid topology with some diagonals.

Comment: Gregory Provan (from Cork), who among other things worked on automated model generators, claimed that random networks can well capture generic structure of a complex system, but are not very good at detailed structure. He also mentions alternative approaches to changing to random Small-World structures.
Also, he claims that random networks *are* very good estimations in the domain of model based diagnosed, and probably also for many other domains.
Gregory claims that all complex systems have *(i)* short distance between any two nodes and *(ii)* high clustering (these are power law properties).
Diagnosis is especially difficult when subcomponents are highly clustered; graph coloring problems are difficult when neighboring lands have many connections (up to the point the instance is uncolorable).

F.0.2 Tree topology

A tree network is actualy constructed by the step of the first method to generate a network with random topolgoy of the previous section. In this step, a random spanning tree is constructed.

The interesting thing about a tree network is that there is exactly one shortest path between any source and destination location. This makes (re-)routing computations trivial. The downside of this, is that tree networks are very sensitive to incidents, as the agents are not able to take detours.

With respect to the network properties, the options property is 1 – its smallest possible value.

F.0.3 Grid topology

Figure F.2 shows an example of a grid network. A distinction is made between a regular grid network, having only horizontal and vertical connections, and grid networks where some (at most two per square) diagonals are also allowed. The non-regular diagonal connections are introduced to be able to make a more fair comparison between different network topologies. For example, in Chapter 5, it is attempted to compare networks of different topologies that have an equal number of locations and connections.

Regular $n \times n$-grid networks have a diameter of $n + n$. The number of equidistance

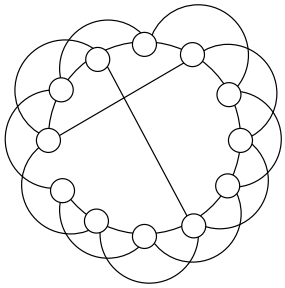


Figure F.3: Example of the small-world topology.

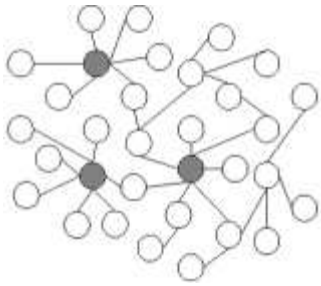


Figure F.4: Example of the scale-free topology (taken from Wikipedia).

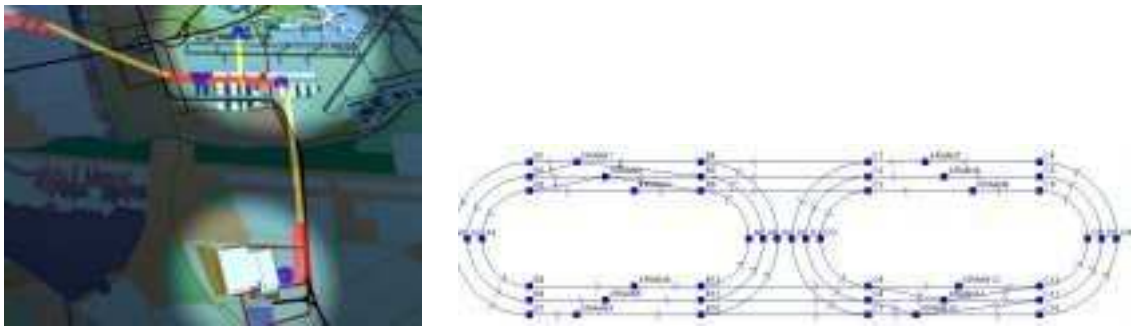
shortest paths grows while source and destination are further apart, and is quite big (options is large).

F.0.4 Small-world topology

The characteristic property of small-world networks is that the average path length between locations grows logarithmic with the number of locations in the network. The type of network is also popular in social networks. Although the earth population grows rapidly, it is conjectured that within seven handshakes each pair of persons know each other.

In a small-world network, each location is connected to nearby locations. And, only sometimes, a node is connected to some node far away. Such a network is usually created by starting with a circle of nodes. Then, each node is connected to the $k/2$ nearest neighbors to the left, and its $k/2$ nearest neighbors to the right (in Figure F.3 $k = 4$ is used). Subsequently, with a small probability $0 < p \ll 1$ each connection is rerouted to a different node. If p approaches 1 the resulting network is of the random topology.

Considering the network properties given at the start of this section, the clustering coefficient remains high, while the average path length grows slowly (logaritmically).



(a) Courtesy of IE&ICT University of Twente.

(b) Screenshot of the Sealand terminal in AGVSim.

Figure F.5: The underground logistic system connecting Aalsmeer, Schiphol and Hoofddorp and the Sealand terminal of Europe Container Terminals (ECT). The latter has a stocking yard at the top and quay cranes (un)loading ships at the bottom, the AGVs drive in unidirectional circles.

F.0.5 Scale-free topology

As an example of a scale-free topology, compare a roadmap to an airline routing map. On the roadmap, one can see each city connected to a highway with nearby cities connected. On the airline routing map, however, one usually sees that big airports are connected to many other airports.

F.0.6 Fully-connected topology

If the path between source and destination is shorter, the probability of malfunctioning resources and conflicts with other agents is smaller. In a fully-connected network, there are so many connections the agents can always take the direct connection while traveling and hence have maximum opportunity to avoid malfunctioning resources and conflicts with other agents (whereas cities were connected to only one or a few highways).

F.0.7 Realistic topologies

Another example of a realistic topology that is used in the experiments of Chapter 5 is the Schiphol airport network that is presented in Figure 5.22.

Not only for the transport networks it was important to look at different topologies, but this also holds for incidents. In the next section several incident models are described, which are commonly used to model the occurrence of incidents in real-life situations.

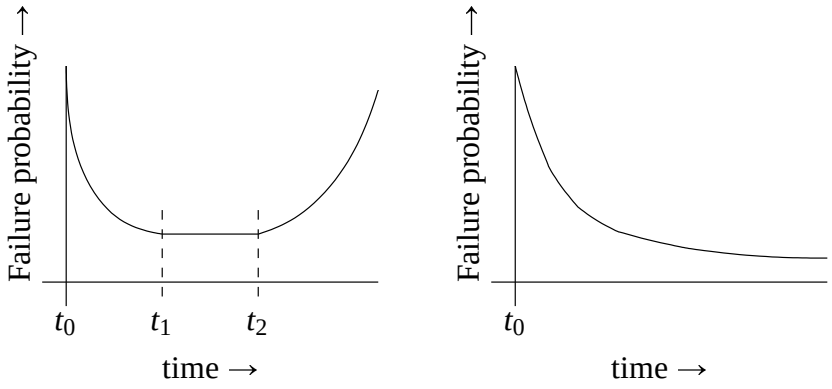
Appendix G

Incident models

In reliability engineering, it is common to model the so-called hazard rate function by one of the functions illustrated in Figure G.1. On the left, a hazard rate curve is shown known as the *bathtub*, due to its shape. The time interval $[t_0, t_1]$ is referred to as *infant mortality*, usually due to production failures. Then comes the normal life during $[t_1, t_2]$ with an almost constant failure rate. And, finally, after time t_2 the failure rate increases again as the product exceeds its design lifetime (*wearout* failures). The bathtub curve is often used to model the failure rate of electronic components; the failure rate for mechanical components might very well follow a different curve.

On the right, a negative exponential curve is shown. This failure rate function is ever decreasing and is an easy-to-use failure rate function when only the number of failures per time unit is known from practice and the failure rate is independent of age.

The mean-time-between-failure (MTBF) that one often encounters in literature, is the average time between failures. When the failure rate is constant, the MTBF is simply the reciprocal of the failure rate.



(a) Bathtub curve.

(b) Negative exponential curve.

Figure G.1: Hazard rate curves.

G.0.8 Exponential distribution

The exponential distribution is the continuous analog of the discrete geometric distribution. It is used to model Poisson processes (Poisson, 1837). The exponential distribution can be used as a good approximation model to estimate the time at which a next incident occurs. In queuing theory, the inter-arrival times (i.e., the times between customers entering a system) are often modeled with random variables that are exponentially distributed. The exponential distribution can be a good choice for modeling the arrival time of incidents. However, note that the right choice for a distribution eventually depends on the problem at hand and can best be determined by examining real data.

The probability density function $f(t)$ of an exponentially distributed random variable X, where constant λ denotes the average number of failures per time unit, and the cumulative distribution $F(t)$ are defined as:

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & 0 \leqslant t \\ 0 & t < 0, \text{ and} \end{cases}$$

$$F(t) = \begin{cases} 1 - e^{-\lambda t} & 0 \leqslant t \\ 0 & t < 0. \end{cases}$$

The exponential distribution is used in the model to generate time-windows for incidents. The start of this time-window is called the *failure time* and the length of this time-window is referred to as the *repair time* of the resource. The failure times of all incidents are assumed to be independent of each other. This means it is assumed that a resource performs like a new one after it has been repaired. For some applications that assumption is not valid, in which more MTBF theory can be applied, for example, by increasing the λ value dynamically in situations where it is more and more likely for a resource to malfunction the more times this resource has been malfunctioning in its recent history.

The expected failure time of a resource and its variance for random variable X that has an exponential distribution are given by

$$E[X] = \frac{1}{\lambda}, \text{ and}$$
$$Var[X] = \frac{1}{\lambda^2}.$$

The exponential distribution is said to be *memoryless*, i.e., the probability that no incident will occur within the next ten minutes, given no incident occurred in the past half hour, is equal to the probability, for example, that the first incident occurred after ten

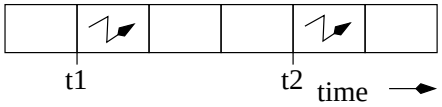


Figure G.2: Generating arrival times of incidents by selecting adjacent time intervals with probability $1/\lambda$. The width of the boxes is the repair time. This process is repeated for each resource.

minutes. Formally, $P(X > 40 \mid X > 30) = P(X > 10)$.

The importance of this method is that it avoids discretization that could lead to undesired artifacts. These undesired artifacts can occur, for instance, when one uses running intervals (with length equal to the repair time) and then, for all resources, with probability $1/\lambda$ generate an incident during this time-window. The problem with this approach, sketched in Figure G.2, is that, at different infrastructure resources, incidents often occur at exactly the same times, which is simply not so realistic and an unnecessary simplification.

In real world scenarios, the constraint that there are λ failures per time unit is rarely satisfied. That, however, does not render this approach useless. One approach is to focus on a more specific time interval, such as peak hours versus quiet hours, where the constraint is more or less (locally) satisfied. Another approach is to combine several incident models with different λ values together.

Appendix H

Planning Domain Definition Language (PDDL)

PDDL – the Planning Domain Definition Language, which can be seen as successor to STRIPS[1], has set a standard language for planning domains used by many planning tools. The language is particularly interesting to the setting of this thesis, because in some competitions benchmarks were generated that contained problem instances for logistic domains.

The original version of the language (PDDL 1.7) was developed by Drew McDermott, with the help of the 1998 Planning Competition committee. Fahiem Bacchus selected a subset of the original language as the language for the 2000 competition. To result in the 2002 version (PDDL 2.1) Maria Fox and Derek Long extended the language with time and objective functions. In 2004 (PDDL 2.2) derived predicates and timed initial literals were added. The most recent competition known at the time of this writing is PDDL 3.0, used in the 2006 competition (Gerevini and Long, 2006). In this version first suggestions for constraints and preferences, expressed in a restricted temporal logic, were added.

Figure H.1 gives an example of the planning domain definition language, taken from the fifth International Planning Competition (IPC 2006) hosted at the International Conference on Automated Planning and Scheduling in 2006. How to interpret the different building blocks of this example is described in Yannis Dimopolus and Saetti (2006). After specifying the name of the problem instance and the domain it starts with listing the objects in `(:objects)`, several trucks, packages, locations, and truck areas. Truck areas represent the loading space of the trucks. Subsequently, in `(:init)`, the initial state is given. `(free a1 truck1)` denotes that truck area `a1` of truck `truck1` is empty and `(closer a1 a2)` means that truck area `a1` is closer to the door of the truck than

[1] STRIPS (Stanford Research Institute Problem Solver) is an automated planner invented by Richard Fikes and Nils Nilsson in 1971. The same name was later used to refer to the formal language of the inputs to this planner.

truck area `a2` and, hence, truck area `a1` must be empty if something is loaded or unloaded into truck area `a2`.

The goal state `(:goal)` specifies that the three packages must be delivered to the correct destination locations and then in `(:constraints)` the trajectory constraints are specified that should be valid all the way (while the goal state should be met at the end). Inside the `(:constraints)` tag there are several preferences. Those are soft constraints that are desired to be satisfied, but not necessarily. The objective function, specified in `(:metric)` uses these, multiplied to a number indicating the priority of the particular preference.

In the specification of the domain `Trucks-ComplexPreferences`, which is omitted here, it is specified that there are four actions: `load`, `unload`, `drive` and `deliver`. Here it is also specified that a load or unload action can only be performed in a truck area if all truck areas closer to the door of the truck are empty.

On the homepage of McDermott it is stated that Opt – Ontology with Polymorphic Types – is a successor to PDDL. Opt includes durative actions, autonomous processes, a completely revised hierarchical planning notation and a more robust type system. McDermott further promised that soon Opt will include all features that PDDL 3.0 has.

```
(define (problem truck-1)
(:domain Trucks-ComplexPreferences)
(:objects
  truck1 - truck

  package1 - package
  package2 - package
  package3 - package
  l1 - location
  l2 - location
  l3 - location

  a1 - truckarea
  a2 - truckarea)

(:init
  (at truck1 l2)
  (free a1 truck1)
  (free a2 truck1)
  (closer a1 a2)
  (at package1 l3)
  (at package2 l3)
  (at package3 l1)
  (connected l1 l2)
  (connected l1 l3)
  (connected l2 l1)
  (connected l2 l3)
  (connected l3 l1)
  (connected l3 l2)
  (= (drive-time l1 l2) 406.3)
  (= (drive-time l1 l3) 73.1)
  (= (drive-time l2 l1) 406.3)
  (= (drive-time l2 l3) 356.8)
  (= (drive-time l3 l1) 73.1)
  (= (drive-time l3 l2) 356.8))

(:goal (and
  (delivered package1 l1)
  (delivered package2 l2)
  (delivered package3 l2)))
```

```
(:constraints (and
  (forall (?p - package ?t -
truck)
    (preference p1A (always
    (forall (?a - truckarea)
    (imply (in ?p ?t ?a)
    (closer ?a a2) )))))

  (preference p1B
(sometime-before
    (delivered package2 l2)
    (delivered package1 l1)))

  (preference p4A (within 919.7
    (delivered package1 l1)))

  (preference p4B (within 919.7
    (delivered package2 l2)))

  (preference p4C (within 1813.7
    (delivered package3 l2)))

  (forall (?p - package)
    (preference p2A (at-most-once
    (exists
    (?t - truck ?a - truckarea)
    (in ?p ?t ?a)))))))

(:metric minimize (+
  (* 1 (is-violated p1A))
  (* 1 (is-violated p1B))
  (* 2 (is-violated p2A))
  (* 4 (is-violated p4A))
  (* 4 (is-violated p4B))
  (* 4 (is-violated p4C))))

)
```

Figure H.1: PDDL (version 3.0) example `truck-1` from Truck domain (IPC 2006) with complex preferences. Preferences are specified within the `(:constraints)` tag and the objective function within the `(:metric)` tag.

Appendix I

The test set

The test set is a set of problem instances designed for the experiments. To compare, for example, two different planning methods, those planning methods can be run for each problem instance within the test set. A problem instance consists of a transport network, a set of operational agents, a set of transportation requests, and a model (or a set of) incidents. How these individual components are chosen is described in this section. At the end, attention is paid to the relevance of the test set.

I.1 Experimental setting

In this section information is provided that is indispensable to anyone who desires to reproduce the experiments presented in the sequel of this chapter. Reproducibility

Each experiment consists of a set of simulation runs with TRAPLAS. These individual runs differ in several ways. The selected problem instance consists of a set of transportation agents, a set of tasks, the transport network, and the incident model. The chosen operational planning method is part of the definition of an agent in the problem instance. A problem instance is a single element from the test set, which is described in the subsequent section.

Furthermore, simulation runs are affected by environmental settings. TRAPLAS, which of course is based on the model described in Chapter 3, is a highly parametrized simulation tool.

Take over If take over is not allowed then all drive actions of transport resources complete in order. Hence, without take over, faster transport resources sometimes have to wait for slower transport resources. In these experiments, take over is allowed.

Arbiter policy If agents use faulty planning methods, or plans are rendered infeasible due to incidents, there is always a fallback to simple resource usage rules. In

these experiments first-in-first-out is chosen as arbiter policy, meaning transport resources enter a resource in the same order as they announced their desire to enter the resource.

Shortest path planning There are many shortest-path algorithms. Mostly, they are guaranteed to produce the optimal shortest path and in that case they only differ in computation time. In case reservations are considered, however, the situation changes. The complexity increases, which can intuitively be understood by recognizing that now it matters at what time a resource is entered and optimal paths can now contain cycles.

Accepted penalty The maximum negative change in individual performance for an agent to still accept the task. This value might also be negative. In these experiments a value of zero has been used, tasks are not accepted by the agents during task (re)allocation if the difference in performance (given the selected performance indicator) between the new plan and the old plan is positive.

Task limit The maximum number of transportation requests that can simultaneously be assigned to an agent. In these experiments, the task limit is set to five. This is done to avoid that a single agent can disturb the system by accepting all tasks, and to speed up some methods (e.g., the insertion method for task allocation is quadratic in the plan length of an agent).

Seed value If two simulation runs use the same seed value for the random generator, then each time a random number is asked for the same number is generated. In these experiments, a random seed value is chosen each time, among other things based on the system clock. This option allows for exact reproducible simulation runs.

I.2 Network topologies

In Section **??** several network topologies have been described. Those included random (but connected) networks, which can in fact represent any of the other network types. However, because only a small finite number of simulation runs can be done within acceptable time, it is not a good idea to use random networks only. Namely, a large number of networks would be required to have a representative set of transport networks in the test set. Therefore, networks are generated of each network topology – including the random topology (in fact, even three different methods were used to generate random topologies).

To make the comparison as accurate as possible, where possible the above generated networks all have the same number of infrastructure resources and arcs. Of course, a

regular $n \times n$ grid network has exactly $2n(n-1)$ arcs. That is why diagonal arcs have been introduced. For each grid cell (four resources that are in a cycle) one diagonal (out of two possibilities) might be added, adding a maximum of $(n-1)^2$ additional arcs. This (limited) flexibility to select the desired number of arcs for a grid network makes it possible to create a network that has the same number of arcs as a Small-World network. A Small-World network with n^2 infrastructure resources (where each resource is connected to its neighbors and the neighbors of its neighbors, i.e., $k = 4$) has $2n^2$ arcs and $3n^2 - 4n + 1 > 2n^2$ for $n > 3$. On the other hand, for tree networks, the number of edges is always $n-1$, so comparison must still be done carefully. Besides these synthetic transport networks, the problem instances also contain some networks inspired by real-life, for instance, the Sealand terminal, the underground logistic system (OLS), and a network resembling the taxiways at Schiphol Airport have been included.

When the set of infrastructure resources and the set of connecting arcs is known, the properties of each infrastructure resource must be considered. Among these are the capacities of the infrastructure resources. In Section 3.2.1 it was assumed that all transport resources initially start in a resource with sufficient capacity. This is done for simplification. If it would have been allowed, transport resources might have to move to a resource with sufficient capacity in order to give way to other transport resources, or else, in the worst case, the system is prone to deadlocks. The more resources there are with sufficient capacity, the more easy it is for transport resources to pass each other. Again, for an accurate comparison it is desired to have about the same number of resources with sufficient capacity for the different transport networks. Either these resources were chosen at random or the endpoints of the diameter were chosen (then, the endpoints of the next longest path, etc). These resources are also used as pick-up and delivery resources, such that, if a transport resource finishes a task by unloading at the destination, there is no need for transport resources to route to a resource with sufficient capacity.

Finally, the maximum allowed driving speed and the distance are to be specified. These are of course related. In the experiments the maximum allowed driving speed was set to 1, and the distance was drawn from a normal distribution with mean $\mu = 10$ and standard deviation $\sigma = 1$.

I.3 Agent behavior

Planning is considered part of the behavior of an agent throughout this thesis. This means the methods used for task allocation and the planning methods used to compute a route and a schedule for the plans of the agents are considered part of its behavior.

Two different approaches are used to assign tasks to agents. The first approach is *vehicle-oriented*, the second *task-oriented*. In the vehicle oriented approach, new tasks are announced by the customers by putting them on a blackboard (or some other information

system, the idea is just that the agents are informed about this new task). Any agent, who happens to be looking for a new task, can check this blackboard and retrieve tasks from it. At the moment an agent pulls a task off the blackboard, it is assigned to this agent and others cannot see it anymore. This operation is atomic to avoid the situation where multiple agents think they have been assigned the same task. Incoming tasks are assigned one by one to the agents sequentially, while the agents consider adding new tasks in parallel.

For the *task-oriented* task allocation, a well-known auctioning protocol, the Vickrey auction, is used. An auction is held for each incoming transportation request. Again, this is done one by one sequentially. Each agent computes a bid for this transportation request and it will be assigned to the agent with the highest bid. This agent has the pay the amount of the second highest bid. It can be proven for this auctioning protocol that the best strategy for an agent is to bid his true value. It could very well be that agents are more interested in some tasks given that they will also get some other tasks. In other words, what an agents wants to bid for a certain task can depend on whether it wins the auctioning for some other task. This dependency between auctions is taken into account in the field of combinatorial auctions (Sandholm, 2002), which is beyond the scope of this thesis.

Furthermore, three different types of *acceptance* of new tasks are used. There are agents that only accept positive reward for the transportation tasks they execute. They will not accept tasks to have a negative reward when they are assigned to the agent nor will they ever accept a situation where it is expected the task will lead to a negative reward. If such a situation occurs, e.g., due to incidents, the transportation task will be dropped immediately. The second type of agents are a little less self-interested. They also do not accept transportation tasks with a negative reward, though they will continue to execute tasks that lead to a negative reward later, possibly due to incidents. Then, there is a third category consisting of agents that accept everything. These are most useful in situations where each task must be executed. Transportation tasks can be assigned to those agents even if it is known in advance that the agent will receive a negative reward even if no incidents at all would occur.

Unless explicitly mentioned the experiments are performed with 32 agents. Each of these agents uses the same planning method, which is varied over different simulation runs. TRAPLAS is able to keep the rest of the environment completely identical, such that the results can be compared to each other. Additionally, mixed strategy experiments are performed. In these experiments, the 32 agents are divided into four groups of eight agents. Here, the members of the same group use the same planning method. This is done to investigate whether there are dependencies between the different planning methods and to see whether it is true that the best performing planning methods are still performing best in case there are other agents in the system that use other planning methods.

I.4 Transportation requests

For each network topology, ten different instances are created for 32 transport resources with five tasks per agent (request load of 160). Then, instances are merged together to create a higher request load, as is described later in this section.

Algorithm **??** describes a way to generate problem instances, such that *(i)* optimal system welfare gained by executing a set of optimal plans is known and *(ii)* the instances look reasonably realistic (e.g., there must not be a huge amount of idle time in the optimal plans or other trivialities).

To compare the planning methods to the best possible performance, best possible performance must be known. That is not possible for sufficiently large arbitrary problem instances due to the complexity of the problem. The following technique is used to overcome this problem: instead of computing the optimal performance for a problem instance (which can only be estimated, and not even precisely), a random plan for each agent is generated. Subsequently, this is used to compute a problem instance. The interesting part is that it is ensured that the generated plan is an optimal plan with respect to the generated problem instance. This is always the case if (but not only if) the following conditions are true:

1. the route driven by the agent from pick-up to delivery location has minimal costs. There does not exist a faster path and there is no waiting time in the plan,

2. the realized loading and unloading time-windows maximize the rewards of the transportation requests,

3. an agent being idle has zero cost.

Because the transport resources move along shortest paths from source to destination of their transportation requests and they gain maximum reward for the execution of the request, these plans are likely to be optimal. The only exception is when the agents have a loading capacity greater than one and by combining transportation requests they can decrease their driving costs more than the rewards decrease due to violation of the time-windows of the transportation request. The latter is not the case in these experiments, because the loading capacities of the agents are set to one.

The task-generating agents select a new destination resource for each task at random in Algorithm **??**. An unfortunate choice here could lead to more idle time in the plans of the agents, hence fewer transportation requests per time unit. There is room for some improvement by considering several alternative destination resources. In Figure I.1 the idle times of the test set are shown, and those instances are considered well enough already.

Setting the task limit or time limit to a greater value increases the number of generated transportation requests, but it does not increase the request load computed as the number

Agents generating transportation requests.

1: **procedure** GENERATEREQUESTS
2: Simulation time $t \leftarrow t_0$.
3: Generate a new request o_j with as source the current infrastructure resource.
4: **while** both time-limit and task-limit not yet reached **do**
5: Set loading time-window to $\tau_j^s \leftarrow [t, t + loadingtime(o_j)]$
6: Compute the set D of potential delivery resources, agent must be able to drive there without any delay
7: **if** $D \neq \varnothing$ **then**
8: Pick a random destination d from D and place the agent there.
9: $t \leftarrow t + loadingtime(o_j) + drivetime(o_j)$
10: Set unloading time-window to $\tau_j^d \leftarrow [t, t + unloadingtime(o_j)]$
11: $t \leftarrow t + unloadingtime(o_j)$
12: Add request o_j to the set of transportation requests.
13: Generate a new request o_j with as source the current infrastructure resource.
14: **else**
15: $t \leftarrow t + t_c$, where t_c is the minimal non-zero duration of all conflicts encountered in Line 6 (agent is idle t_c time units).
16: **end if**
17: **end while**
18: **end procedure**

of transportation requests per time unit. Therefore, an additional technique is used to create problem instances of a higher request load. This is done by merging transportation request files. An upper-bound on the optimal performance of the merged instance is the sum of (upper-bounds of) the optimal performances of the individual instances.

Reward functions The reward function, which can be different for each separate transportation request, specifies the reward the agent gets for successfully executing the transportation request. The reward for a transportation request is the sum of the reward for loading and the reward for unloading the freight.

The reward typically is maximal when the loading or unloading event takes place within the specified time-window. When the loading or unloading event is too late, the reward either is zero immediately, or decreases according to some function. Furthermore, if the loading or unloading event is too early, there might also be a drop in reward. For example, when trucks arrive too early to deliver their goods at a supermarket, they consume parking space close to the supermarket that temporarily cannot be used by other trucks.

In case a function is used that has a global maximum (e.g., a horizontally asymptotic function), there exists an easy upper-bound on the total reward that is the sum

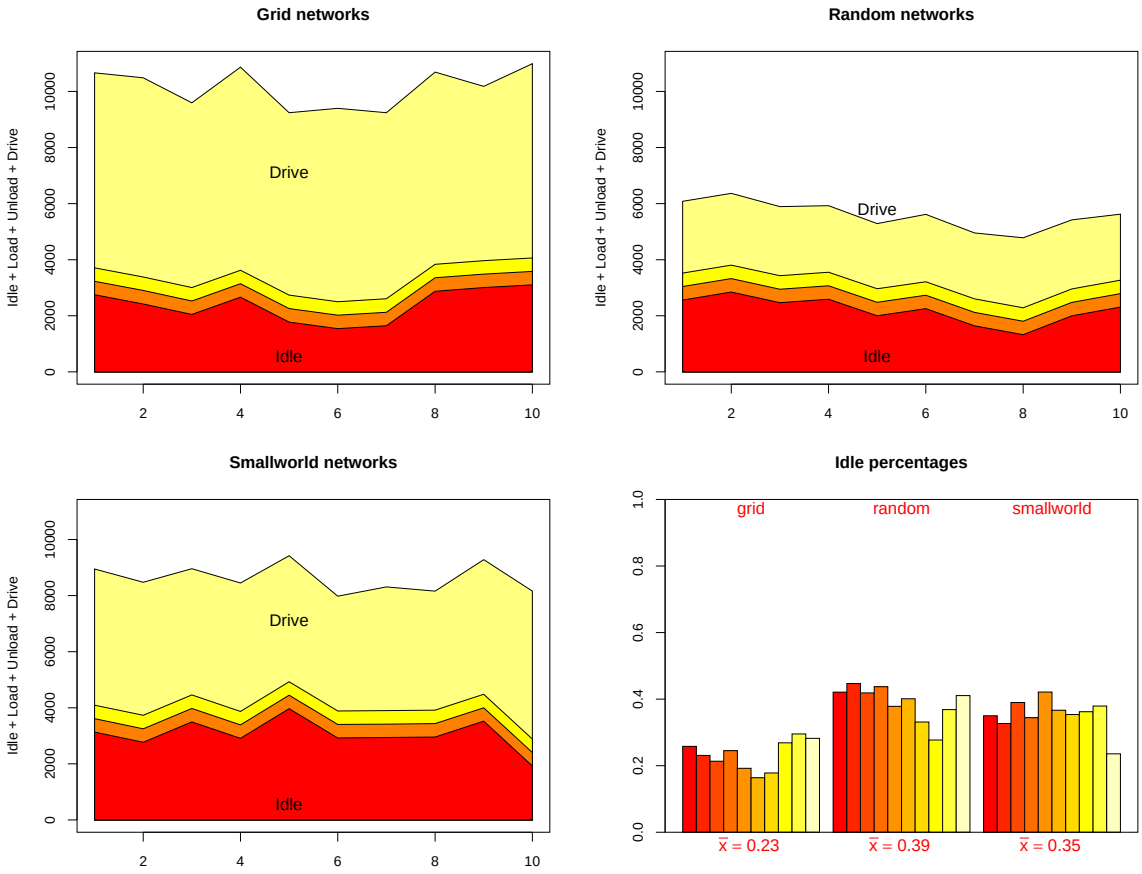


Figure I.1: Idle times in the optimal plans for ten arbitrary transport networks for each topology.

of these global maxima of each transportation request. Examples of such reward functions are a constant reward (e.g., zero outside the desired time-windows and a constant within the time-windows), a summed arctan (too late results in decreasing reward, but never more than the horizontal asymptote), linear distance outside the time-windows, or the gamma distribution. The latter, illustrated in Figure I.2, is used in the experiments. In case loading or unloading starts too early at time $t < lowerbound(\tau)$ and too early is considered of equal importance as too late, this time can be transformed to $upperbound(\tau) + (lowerbound(\tau) - t)$ to compute the reward.

I.5 Incidents

An incident (r, τ, i) is defined by specifying a *resource* r, either a transport resource $r \in R^{tr}$ or an infrastructure resource $r \in R^{inf}$, an *impact* factor $0 \leqslant i \leqslant 1$, for which $(1-i)$ is multiplied with the allowed speed at or maximum speed of the resource, a *time-window* $\tau \in W$ during which the incident is in effect and its *release time*, the time at which the

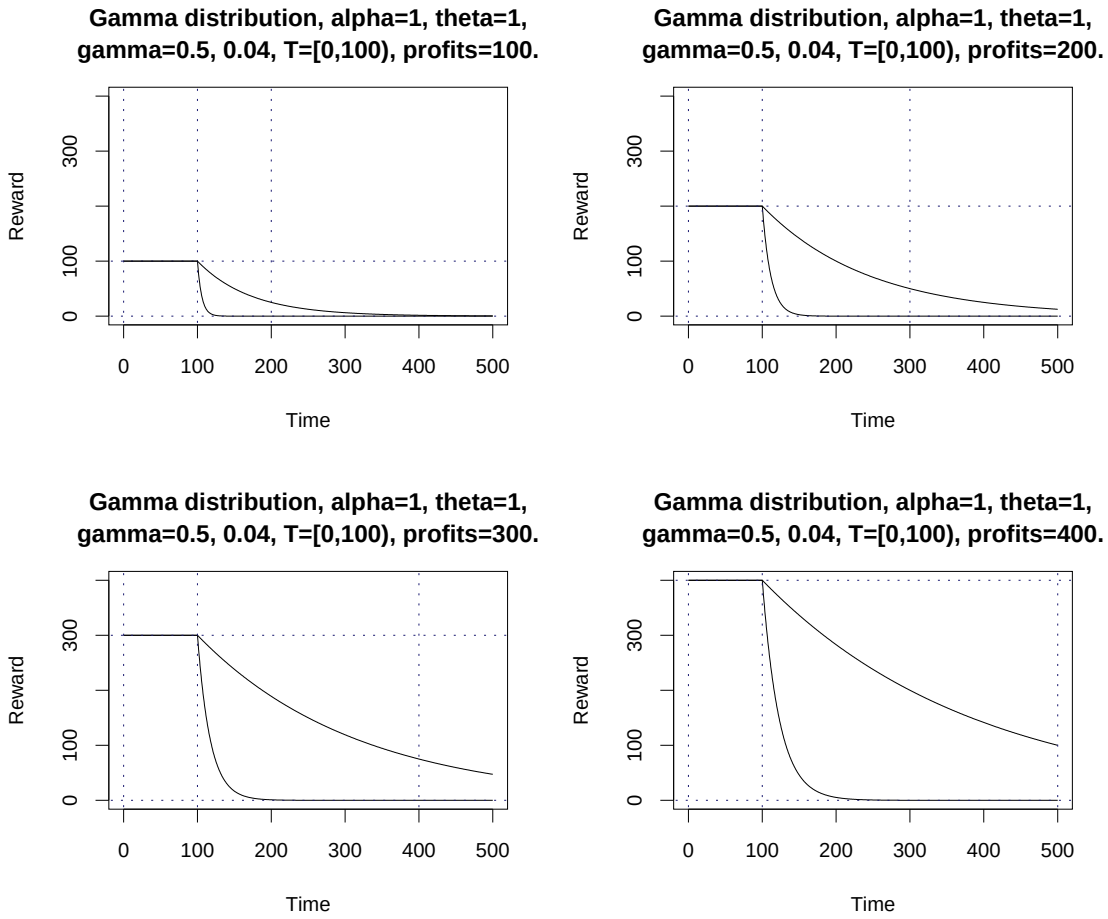


Figure I.2: Example of the Gamma distribution reward function. The figure illustrates $\alpha = \theta = 1$, simplifying to $Reward(t) = profits \cdot \exp(\frac{-lateness \cdot log(2)}{\gamma \cdot profits})$ for $lateness > 0$.

incident is known to the system. If the release time is smaller than the start of the time-window, this means the system knows in advance the incident will take place; the release time is never greater than the start of the time-window.

As already stated in Section 3.1.4, the effective time-windows of incidents are often modeled using the Mean-time-between-failure (MBTF) approach. As an example, let us model a resource that malfunctions 10% of the time, i.e., $p = 0.1$, with an average repair time of 3. Recall the probability density function $f(t)$ and cumulative distribution $F(t)$ that belong to the exponential distribution, see Section 3.1.4.

This means there is on average one failure in thirty time units, i.e., the average number of failures per time unit λ equals $\frac{1}{30}$.

The total area below the probability density function is, of course, equal to 1. What we want is to compute the next start time for an incident according to the probability density function. Therefore, we start by taking a list of random numbers in $[0,1)$, say $N = (0.254, 0.928, 0.498, 0.529, 0.139)$. Now let each random number correspond to the

fraction of the total area below the probability density function before the start time of the incident. To compute the starting time x for all random numbers $n \in N$ the following equation is used.

$$\begin{aligned} \int_0^x f(t)dt = n \quad &\Leftrightarrow \\ F(x) - F(0) = n \quad &\Leftrightarrow \\ 1 - e^{-\lambda x} = n \quad &\Leftrightarrow \\ x = \frac{-ln(1-n)}{\lambda} \quad &\lambda \neq 0 \wedge n < 1 \end{aligned}$$

This yields $L = (8.791, 78.933, 20.675, 22.587, 4.490)$. List L now denotes subsequent times between intervals. Finally, from this list we can easily compute the time-windows of the incidents: these are $[8.791, 11.791)$, $[90.724, 93.724)$, $[114.398, 117.398)$, $[139.985, 142.985)$, and $[147.475, 150.475)$ respectively. Five incidents of length 3 in approximately 150 time units indeed corresponds to a resource malfunctioning 10% of time. Note that the repair time does not have to be constant over all incidents.

To measure the incident level the following function, that sums the product of impact and duration of each incidents, is used:

$$\text{incident level} = \sum_{(r,\tau,i)\in\mathcal{I}} i \cdot \big(ub(\tau) - lb(\tau)\big).$$

For the experiments the failure probability was varied from 0 to 0.2 (20%) and the repair time was drawn from normal distribution with $\mu = 400$ and $\sigma = 50$. The reason behind this was previous research of Maza and Castagna (2005), and some tests done to repeat this. Samia and Castagna report that with frequent incidents that have a short repair time there are no significant differences between their different planning methods. This also holds for the methods in this thesis. They claim that when the failures are frequent and short, they are mutually compensated. The methods in this thesis that consider rerouting the vehicle will not find much of an improvement as the vehicle will also be bothered by incidents along all alternative routes.

I.6 Relevance

The set of problem instances that are used for the experiments throughout this thesis is synthetic. Although attempts were made to have the problem set resemble reality as much as possible concerning many aspects, this possibility, or simply the data, was not always available. Nevertheless, it is conjectured that the set of problem instances is realistic

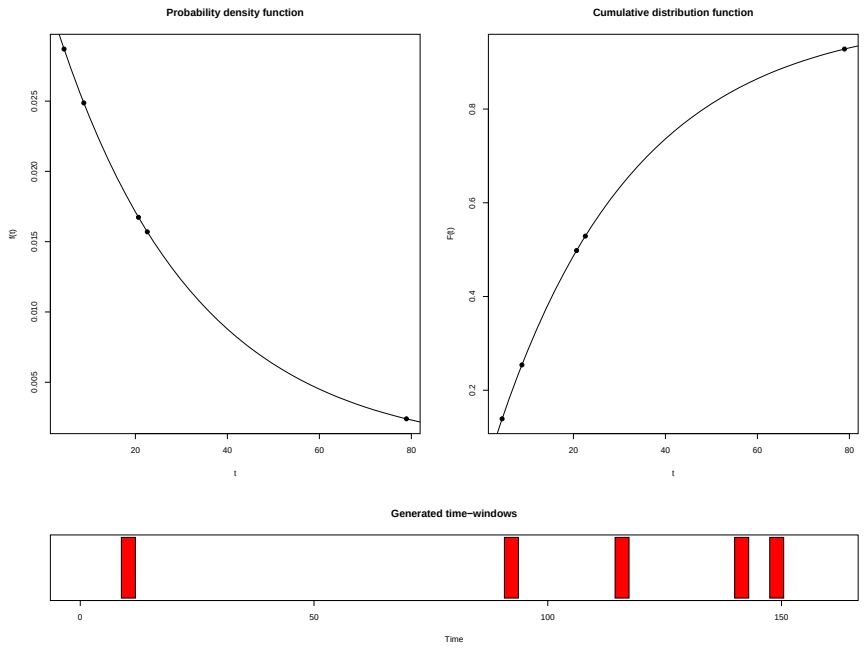


Figure I.3: Probability density function and cumulative distribution function with $\lambda = \frac{1}{30}$. The separation times L between the generated incidents are plotted. The bottom figure displays the resulting time-windows.

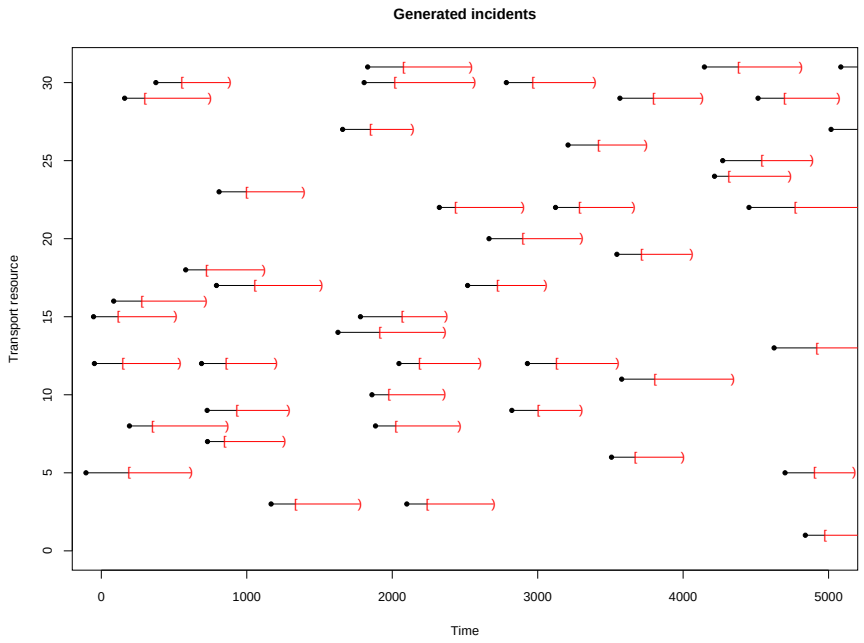


Figure I.4: Example of generating incidents. The dots indicate the release times of the incidents (i.e., the time it is known to the agents). This data is generated like described in this section, with failure probability 0.1 for all 32 transport resources, but with a repair time (and time known in advance) drawn from a normal distribution.

enough to be of practical importance.

To start with, the transport networks include some realistic instances. There is the network resembling the Sealand terminal, the underground logistic system (OLS), and the Schiphol airport network. For the rest of the transport networks, an attempt was made to generate many networks of several well-known structures (grid networks, small-world networks, etc.), but also to generate many random networks to see the effect of the network structure on the performance of the system.

Both the task request load and the density of incidents are varied from light to extreme circumstances. Although it would be impossible to look at all possible situations, simply because there are way too many, the experiments are focused on the most interesting things that were encountered.

Finally, the model is quite general. It includes many details that can be tuned to a real-life application domain. For example, overtaking, vehicle capacities, per task reward functions, etc.

Appendix J

Model details

Table J.1 and J.2 list the coefficients for the models M1 and M4 respectively that are described in Section **??** and 5.3.3. The first column lists the β-constants to be filled in in the formula provided in the table caption. The variable Used(m) is a dummy encoding: 1 if method *m* is used, 0 otherwise. If Used$(m) = 0$ for all methods *m*, the method used is the one not listed in the table; this method is the MACA method for both tables.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	8.27e-01	2.84e-03	2.91e+02	0e+00
Used(Classical)	-7.57e-01	4.02e-03	-1.88e+02	0e+00
Used(RR random)	8.58e-03	4.07e-03	2.11e+00	3.5e-02
Used(RP random)	-4.51e-02	4.03e-03	-1.12e+01	6.46e-29
Used(RR delays)	9.42e-03	4.05e-03	2.32e+00	2.01e-02
Used(RP delays)	-5.86e-02	4.03e-03	-1.45e+01	1.76e-47
Used(RR deadlines)	1.20e-02	4.02e-03	2.99e+00	2.81e-03
Used(RP deadlines)	-4.46e-02	4.03e-03	-1.11e+01	2.30e-28
Used(RR profits)	-1.32e-03	4.08e-03	-3.23e-01	7.47e-01
Used(RP profits)	-6.4e-02	4.04e-03	-1.59e+01	4.5e-56
Used(RR wait)	1.04e-02	4.05e-03	2.57e+00	1.03e-02
Used(RR wait 10% slack)	-2.29e-02	4.05e-03	-5.66e+00	1.59e-08
Used(RR wait 20% slack)	-5.15e-02	4.04e-03	-1.27e+01	6.43e-37
Used(RP wait)	-2.76e-02	4.03e-03	-6.86e+00	7.44e-12
Used(RP wait 10% slack)	-5.62e-02	4.04e-03	-1.39e+01	9.08e-44
Used(RP wait 20% slack)	-8.76e-02	4.03e-03	-2.18e+01	5.07e-103
Used(RR task)	1.53e-02	4.04e-03	3.78e+00	1.56e-04
Used(RP task)	-3.93e-02	4.03e-03	-9.74e+00	2.38e-22
Used(RR inv task)	-3.76e-03	4.08e-03	-9.21e-01	3.57e-01
Used(RP inv task)	-5.36e-02	4.03e-03	-1.33e+01	5.13e-40
request load	-1.26e-03	7.6e-06	-1.66e+02	0e+00
Used(Classical)×request load	1.16e-03	1.08e-05	1.08e+02	0e+00
Used(RR random)×request load	8.03e-06	1.1e-05	7.3e-01	4.65e-01
Used(RP random)×request load	3.51e-05	1.08e-05	3.24e+00	1.20e-03
Used(RR delays)×request load	-1.30e-05	1.09e-05	-1.20e+00	2.32e-01
Used(RP delays)×request load	2.04e-05	1.08e-05	1.88e+00	6.02e-02
Used(RR deadlines)×request load	-1.50e-05	1.08e-05	-1.39e+00	1.63e-01
Used(RP deadlines)×request load	4.41e-05	1.08e-05	4.07e+00	4.64e-05
Used(RR profits)×request load	1.83e-05	1.11e-05	1.65e+00	9.81e-02
Used(RP profits)×request load	7.57e-05	1.08e-05	6.98e+00	3.00e-12
Used(RR wait)×request load	4.85e-05	1.09e-05	4.44e+00	9.01e-06
Used(RR wait 10% slack)×request load	2.13e-05	1.09e-05	1.96e+00	5.05e-02
Used(RR wait 20% slack)×request load	-6.38e-06	1.08e-05	-5.88e-01	5.57e-01
Used(RP wait)×request load	8.98e-05	1.08e-05	8.31e+00	1.05e-16
Used(RP wait 10% slack)×request load	5.65e-05	1.08e-05	5.23e+00	1.76e-07
Used(RP wait 20% slack)×request load	3.26e-05	1.08e-05	3.02e+00	2.56e-03
Used(RR task)×request load	-1.73e-05	1.09e-05	-1.59e+00	1.12e-01
Used(RP task)×request load	5.63e-05	1.08e-05	5.2e+00	2.04e-07
Used(RR inv task)×request load	1.93e-05	1.11e-05	1.73e+00	8.3e-02
Used(RP inv task)×request load	6.03e-05	1.08e-05	5.57e+00	2.54e-08

Table J.1: The coefficients for Model M1: $\mu = \beta_0 + \beta_1 W + \beta_2 M + \beta_3 WM$. Variable Used(method) is 1 if the specified planning method has been used, and 0 otherwise.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	4.28e-01	6.09e-03	7.02e+01	0e+00
Used(Classical)	-3.83e-01	8.62e-03	-4.45e+01	0e+00
Used(RR random)	7.48e-02	8.7e-03	8.6e+00	9.11e-18
Used(RP random)	3.08e-02	8.61e-03	3.58e+00	3.43e-04
Used(RR delays)	6.3e-02	8.7e-03	7.25e+00	4.52e-13
Used(RP delays)	1.59e-02	8.63e-03	1.84e+00	6.61e-02
Used(RR deadlines)	6.95e-02	8.62e-03	8.06e+00	8.27e-16
Used(RP deadlines)	3.42e-02	8.64e-03	3.96e+00	7.69e-05
Used(RR profits)	5.91e-02	8.77e-03	6.74e+00	1.63e-11
Used(RP profits)	1.96e-02	8.65e-03	2.27e+00	2.34e-02
Used(RR wait)	8.3e-02	8.66e-03	9.59e+00	1.07e-21
Used(RR wait 10% slack)	6.55e-02	9.03e-03	7.26e+00	4.13e-13
Used(RR wait 20% slack)	2.94e-02	9.02e-03	3.25e+00	1.14e-03
Used(RP wait)	5.78e-02	7.49e-03	7.73e+00	1.19e-14
Used(RP wait 10% slack)	3.93e-02	9.05e-03	4.35e+00	1.39e-05
Used(RP wait 20% slack)	6.82e-03	9e-03	7.59e-01	4.48e-01
Used(RR task)	7.05e-02	8.64e-03	8.16e+00	3.66e-16
Used(RP task)	3.59e-02	8.64e-03	4.15e+00	3.31e-05
Used(RR inv task)	5.5e-02	8.72e-03	6.31e+00	2.92e-10
Used(RP inv task)	2.98e-02	8.64e-03	3.45e+00	5.53e-04
incidents	-7.48e-04	1.37e-05	-5.45e+01	0e+00
Used(Classical)×incidents	7e-04	1.95e-05	3.59e+01	5.81e-268
Used(RR random)×incidents	-1.36e-05	1.97e-05	-6.92e-01	4.89e-01
Used(RP random)×incidents	6.81e-05	1.95e-05	3.50e+00	4.75e-04
Used(RR delays)×incidents	-2.12e-06	1.96e-05	-1.08e-01	9.14e-01
Used(RP delays)×incidents	8.45e-05	1.95e-05	4.33e+00	1.51e-05
Used(RR deadlines)×incidents	-1.38e-05	1.95e-05	-7.1e-01	4.78e-01
Used(RP deadlines)×incidents	4.77e-05	1.96e-05	2.44e+00	1.47e-02
Used(RR profits)×incidents	5.57e-06	1.98e-05	2.82e-01	7.78e-01
Used(RP profits)×incidents	7.66e-05	1.95e-05	3.92e+00	9e-05
Used(RR wait)×incidents	-3.64e-05	1.96e-05	-1.86e+00	6.31e-02
Used(RR wait 10% slack)×incidents	-4.33e-05	2.09e-05	-2.08e+00	3.79e-02
Used(RR wait 20% slack)×incidents	2.31e-05	2.08e-05	1.11e+00	2.66e-01
Used(RP wait)×incidents	6.14e-06	1.69e-05	3.62e-01	7.17e-01
Used(RP wait 10% slack)×incidents	1.51e-05	2.08e-05	7.27e-01	4.67e-01
Used(RP wait 20% slack)×incidents	6.3e-05	2.09e-05	3.01e+00	2.58e-03
Used(RR task)×incidents	-1.31e-05	1.95e-05	-6.72e-01	5.02e-01
Used(RP task)×incidents	5.44e-05	1.95e-05	2.79e+00	5.26e-03
Used(RR inv task)×incidents	2.01e-05	1.97e-05	1.02e+00	3.07e-01
Used(RP inv task)×incidents	6.41e-05	1.95e-05	3.28e+00	1.03e-03

Table J.2: The coefficients for Model M4: $\mu = \beta_0 + \beta_1 i + \beta_m + \beta_2 i \beta_m$. Variable Used(method) is 1 if the specified planning method has been used, and 0 otherwise.

Appendix K

List of hypotheses

Throughout this thesis several hypotheses were presented. Those were supported or falsified using empirical evidence gained from the experiments described in Chapter 5. For convenience, these hypotheses are listed here.

Bibliography

The numbers in brackets behind each entry list the page numbers with references to this entry.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843. ISSN 0001-0782. pages 55

ANP (2006). Groei van Rotterdamse haven stagneert (in Dutch). http://www.nu.nl/news/794723/32/rss/Groei_Rotterdamse_haven_stagneert.html. pages 15

Aronson, L. D., van der Krogt, R. P. J., and Zutt, J. (2002a). Automated transport planning using agents. In *Proceedings of the International Congress on Freight Transport Automation and Multimodality: Organisational and Technological Innoventions (FTAM'02)*. Delft. pages 12

Aronson, L. D., van der Krogt, R. P. J., and Zutt, J. (2002b). Incident management in transport planning. In *Proceedings of the 7th TRAIL Congress (TRAIL'02)*. Rotterdam. pages 12

Beamon, B. M. (1998a). Performability-based fleet sizing in a material handling system. *International Journal of Advanced Manufacturing Technology*, 14(6):441–449. pages 46

Beamon, B. M. (1998b). Performance, reliability, and performability of material handling systems. *International Journal of Production Research*, 36(2):377–393. pages 46, 70

Beamon, B. M. (1998c). Reliability in design of fixed-path material handling systems. *Engineering Design and Automation*, 4(4):285–292. pages 45, 46

Benneker, B. (2006). Haven Rotterdam blijft achter bij concurrenten (in Dutch). http://www.elsevier.nl/nieuws/economie/artikel/asp/artnr/108880. pages 15

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., and Laporte, G. (2006). Static pickup and delivery problems: A classification scheme and survey. *TOP: Official Journal of the Spanish Society of Statistics and Operations Research*. pages 16

Bjarne E. Helvik, O. W. (2001). Using the cross-entropy method to guide/govern mobile agent's path finding in networks. In *Proceedings of the 3rd International Workshop on Mobile Agents for Telecommunication Applications - MATA'01*. Montréal, Canada. pages 27

Bos, A., van Gemund, A. J. C., and Zutt, J. (2002). System health tracking and safe testing. In Willett, P. K. and Kirubarajan, T., editors, *Proceedings of the International Society for Optical Engineering (SPIE'02)*, volume 4733 of *Component and Systems Diagnostics, Prognostics, and Health Management II*, pages 25–36. Orlando, Florida. pages 12

Bräysy, O. (2001). Genetic algorithms for the vehicle routing problem with time windows. *Arpakannus*, 1. Special issue on Bioinformatics and Genetic Algorithms. pages 27

Broadbent, A. J., Besant, C. B., Premi, S. K., and Walker, S. P. (1985). Free ranging AGV systems: promises, problems and pathways. In Hollie, R. H., editor, *Proceedings of the 2nd International Conference on Automated Guided Vehicle System*, pages 221–237. pages 30, 39

Chambers, J. M., Cleveland, W. S., Kleiner, B., and Tukey, P. A., editors (1983). *Graphical methods for data analysis*. The Wadsworth statistics/probability series. Wadsworth, Pacific Grove, CA, USA. ISBN 0-534-98052-X. pages 112

Cheon, S. (2001). An overview of automated highway systems (ahs) and the social and institutional challenges they face. pages 2

Cherkassky, B. V., Goldberg, A. V., and Radzik, T. (1994). Shortest paths algorithms: Theory and experimental evaluation. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*. pages 36

Chiang, W.-C. and Russell, R. A. (1996). Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63(1):3–27. pages 27

Cordeau, J.-F. and Laporte, G. (2003). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research (4OR: A Quarterly Journal of Operations Research)*. pages 16

Cordeau, J.-F., Laporte, G., Potvin, J.-Y., and Savelsbergh, M. W. P. (2004). Transportation on demand. In Barnhart, C. and Laporte, G., editors, *Handbooks in Operations Research and Management Science*. Elsevier, Amsterdam. pages 16

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W. P., and Vigo, D. (2005). Vehicle routing. In Barnhart, C. and Laporte, G., editors, *Handbooks in Operations Research and Management Science*. Elsevier, Amsterdam. pages 16

Davenport, A. J. and Beck, J. C. (2000). A survey of techniques for scheduling with uncertainty. Unpublished manuscript. pages 47

Davenport, A. J., Gefflot, C., and Beck, J. C. (2001). Slack-based techniques for robust schedules. In *Proceedings of the Sixth European Conference on Planning (ECP-2001)*. pages 47

de Feijter, R. (2006). *Controlling high speed automated transport network operations*. Ph.D. thesis, Delft University of Technology. pages 33

de Weerdt, M. M., Bos, A., Tonino, J. F. M., and Witteveen, C. (2003a). A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence, special issue on Computational Logic in Multi-Agent Systems*, 37(1–2):93–130. pages 48

de Weerdt, M. M., van der Krogt, R. P. J., and Zutt, J. (2003b). Plan merging: Experimental results. In *Proceedings of the Fifteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'03)*, pages 315–322. Nijmegen. pages 12

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality af A*. *J. ACM*, 32(3):505–536. ISSN 0004-5411. pages 130

Desrosiers, J., Dumas, Y., and Soumis, F. (1986). A dynamic programming solution to the large-scale single-vehicle dial-a-ride problem with time window. *American Journal of Mathematics and Management Scienc*, 6:301–325. pages 23

Desrosiers, J., Dumas, Y., and Soumis, F. (1991). The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22. pages 23

Di Caro, G. and Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365. pages 27

Di Caro, G., Ducatelle, F., and Gambardella, L. M. (2004). Anthocnet: an ant-based hybrid routing algorithm for mobile ad hoc networks. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*. Birmingham, UK. pages 27

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271. pages 35, 77, 79, 81

Dorigo, M., Maniezzo, V., and Colorni, A. (1991). Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy. pages 27

Durfee, E. H. (1999). Distributed problem solving and planning. In Weiß, G., editor, *A Modern Approach to Distributed Artificial Intelligence*, chapter 3. The MIT Press, San Francisco, CA. pages 154

Eppstein, D. (1998). Finding the *k* shortest paths. *SIAM J. Computing*, 28(2):652–673. pages 139

Fisher, M. L. (1994). Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, pages 626–642. pages 23

Fisher, M. L. and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124. pages 26

Fujii, S., Sandoh, H., and Hozaki, R. (1989). A routing control method of automated guided vehicles by the shortest path with time-windows. In *Production Research: Approaching the 21st Century*, pages 489–495. pages 43, 78, 97

Gao, H. (1995). *Building Robust Schedules using Temportal Protection - An Empirical Study of Constraint Based Scheduling Under Machine Failure Uncertainty*. Master's thesis, University of Toronto, Toronto, Ontario, Canada. pages 47

Garrido, A., Salido, M. A., Barber, F., and López, M. A. (2000). Heuristic methods for solving job-shop scheduling problems. In *European Conference on Artificial Intelligence (ECAI 2000)*. pages 38

Gendreau, M., Laporte, G., Musaraganyi, C., and Taillard, É. D. (1999). A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & OR*, 26(12):1153–1173. pages 26

Gendreau, M., Laporte, G., and Semet, F. (1998). A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, 32(4):263–273. pages 26

Gerevini, A. and Long, D. (2006). Plan constraints and preferences in pddl3, the language of the deterministic part of the fifth international planning competition. In *International Conference on Automated Planning and Scheduling*. pages 179

Graham, J. R. (2001). *Real-Time Scheduling in Distributed Multi-Agent Systems*. Ph.D. thesis, University of Delaware. pages 154

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC*, 4(2):100–107. pages 35

Hatzack, W. and Nebel, B. (2001). Solving the operational traffic control problem. In Cesta, A., editor, *Proceedings of the 6th European Conference on Planning (ECP'01)*. pages 38, 39, 41, 53, 60, 61, 72, 91, 97, 108

Hickman, M. and Blume, K. (2000). A method for scheduling integrated transit service. In *the 8th International Conference on Computer-Aided Scheduling of Public Transport (CASPT)*. pages 108

Huang, J., Palekar, U., and Kapoor, S. (1993). A labelling algorithm for the navigation of automated guides vehicles. *Journal of Engineering for Industry*. pages 43

Jaw, J., Odoni, A., and Psaraftis, H. N. (1986). A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with windows. *Transportation Research*, 8(20):243–257. pages 26

Kaspi, M. and Tanchoco, J. M. A. (1990). Optimal flow path design of unidirectional AGV systems. *International Journal of Production Research*, 28(6):1023–1030. pages 46

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 359–363. John Wiley & Sons. pages 24

Kautz, H. and Selman, B. (1999). Unifying SAT-based and graph-based planning. In Minker, J., editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. Computer Science Department, University of Maryland, College Park, Maryland. pages 24

Kim, C. W. and Tanchoco, J. M. A. (1991). Conflict-free shortest-time bidirectional AGV routeing. *International Journal on Production Research*, 29(12):2377–2391. pages 7, 31, 38, 43, 51, 76, 78, 85, 91, 97, 106

Koenig, S. and Likhachev, M. (2002). Improved fast replanning for robot navigation in unknown terrain. Technical report, Georgia Institute of Technology and Georgia Institute of Technology. pages 48

Konings, R., Pielage, B.-J., Visser, J., and Wiegmans, B. (2009). New hinterland transport concept for port of rotterdam: Extended gateways with innovative connecting transport systems. In *Transportation Research Board 88th Annual Meeting*. pages 2

Kruse, R. L. (1984). *Data structures & program design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. ISBN 0-13-196253-1. pages 117

Kuipers, F. and Mieghem, P. V. (2005). Conditions that impact the complexity of qos routing. *IEEE/ACM Transaction on Networking*, 13(4):717–730. pages 139

Laumonier, J., Desjardins, C., and Chaib-draa, B. (2006). Cooperative adaptive cruise control: a reinforcement learning approach. In *In The Fourth Workshop on Agents in Traffic and Transportation*. pages 3

Le-Anh, T. and de Koster, M. B. M. R. (2004). A review of design and control of automated guided vehicle systems. Technical report, Erasmus Research Institute of Management. ERIM Report Series Reference No. 2004-030-LIS. pages 6, 70

Li, F., Golden, B., and Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Comput. Oper. Res.*, 32(5):1165–1179. ISSN 0305-0548. pages 28, 108

Lindeijer, D. G. (2003). *Controlling Automated Traffic Agents*. Ph.D. thesis, Delft University of Technology, Delft, The Netherlands. pages 33

Lindeijer, D. G. and Evers, J. J. M. (1999). Traces, the agile traffic-control and engineering system. In *Proceedings of the Fifth Annual Congress on Transport, Infrastructure and Logistics (TRAIL-99)*. TRAIL research school, Delft, the Netherlands. pages 33, 34

Love, T. (1999). A compendium of NP optimization problems. Internet site. Http://www.nada.kth.se/∼viggo/problemlist/compendium.html. pages 158

Malone, T. W. and Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119. ISSN 0360-0300. pages 155

Maza, S. and Castagna, P. (2005). A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry*, 56:719–733. pages 31, 191

Miller, R. G. (1991). *Simultaneous Statistical Inference*. Springer Series in Statistics. Springer Verlag. pages 113

Mitrović-Minić, S. (1998). Pickup and delivery problem with time windows: a survey. Technical report, Simon Fraser University. pages 22

Möhring, R. H., Köler, E., Gawrilow, E., and Stenzel, B. (2004). Conflict-free real-time AGV routing. Pre-print from the Combinatiorial Optimization & Graph Algorithms group from Universität Berlin. pages 31

Moret, B. M. (1998). *The Theory Of Computation*. Addison-Wesley. pages 157

Morton, T. E. and Pentico, D. W. (1993). *Heuristic Scheduling Systems with Applications to Production Systems and Project Management*. Wiley series in engineering and technology management. pages 37, 70

Nash, J. F. (1950). Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*. pages 91

P. Van Mieghem, H. D. N. and Kuipers, F. (2001). Hop-by-hop quality of service routing. *Computer Networks*, 37(3-4):407–423. pages 139

Pereira, F. B., Tavares, J., Machado, P., and Costa, E. (2002). Gvr: a new genetic representation for the vehicle routing problem. In *Proceedings of the 13th Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2002)*, pages 95–102. Limerick, Ireland. pages 27

Poisson, S. D. (1837). *Recherches sur la probabilité des jugements en matière criminelle et en matière civile*. Adamant Media Corporation. pages 176

Post, H. N. (2004). Transport, routing- en schedulingproblemen. Course manual in Dutch. pages 36

Psaraftis, H. N. (1980). A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14:130–154. pages 23

Psaraftis, H. N. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time-windows. *Transportation Science*, 17(3). pages 23

Putnam, H. (1960). Minds and machines. In Hook, S., editor, *Dimensions of Mind*, pages 138–164. Macmillan, London. pages 153

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. pages 112

Rodney K. Lay, Gene M. McHale, W. B. S. (1996). The u.s. dot status report on the automated highway systems program. Technical report, Mtretrek Systems, Center for Telecommunications and Advanced Technology, McLean, Virginia. pages 2

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:127–190. pages 27

Russell, S. and Norvig, P., editors (1995). *Artificial Intelligence, A Modern Approach*. Prentice-Hall International, Inc. pages 35, 153

Sammarra, M., Cordeau, J.-F., Laporte, G., and Monaco, M. F. (2006). A tabu search heuristic for the quay crane scheduling problem. *Journal of Scheduling*. pages 16

Sandholm, T. W. (1995). Limitations of the Vickrey auction in computational multiagent systems. In Lesser, V. R., editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*. AAAI Press, distributed by The MIT Press, San Francisco, CA. pages 109

Sandholm, T. W. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1–2):1–54. pages 186

Savelsbergh, M. W. P. (1990). An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47:75–85. pages 26

Savelsbergh, M. W. P. and Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, 29(1):17–29. pages 16, 17, 19, 20, 21

Schoonderwoerd, R., Bruten, J. L., Holland, O. E., and Rothkrantz, L. J. M. (1997). Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207. ISSN 1059-7123. pages 27

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1–2):231–252. pages 32

Skinner, B. F. (1953). *Science and Human Behavior*. Macmillan, London. pages 153

Stentz, A. T. (1994). Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 4, pages 3310 – 3317. pages 48

Taghaboni, F. and Tanchoco, J. M. A. (1988). A lisp-based controller for free-ranging automated guided vehicle systems. *International Journal on Production Research*, 26(2):173–188. pages 31

Taghaboni-Dutta, F. and Tanchoco, J. (1995). Comparison of dynamic routing techniques for automated guided vehicle system. *International Journal of Production Research*, 33(10):2653–2669. pages 43

Tavares, J., Pereira, F. B., Machado, P., and Costa, E. (2003). On the influence of gvr in vehicle routing. In *Proceedings of the 2003 ACM Symposium On Applied Computing (SAC 2003)*, pages 753–758. Melbourne, Florida USA. pages 27

Ter Mors, A. W. (2007). Robustness in context-aware routing planning. Doctoral consortium. pages 88

Ter Mors, A. W., Zutt, J., and Witteveen, C. (2007). Context-aware logistic routing and scheduling. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*. Providence, Rhode Island, U.S.A. pages 128

The VRP Web (2007). http://neo.lcc.uma.es/radi-aeb/WebVRP/. pages 108

Trüg, S., Hoffmann, J., and Nebel, B. (2004). *KI 2004: Advances in Artificial Intelligence*, volume 3238 of *Lecture Notes in Computer Science*, chapter Applying Automatic Planning Systems to Airport Ground-Traffic Control – A Feasibility Study. Springer Berlin / Heidelberg. pages 108

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison Wesley. pages 112

Valk, J. M., Witteveen, C., and Zutt, J. (2001a). Approximation results for multi-agent planning systems. In *Proceedings of the fourth Pacific Rim International Workshop on Multi-Agents (PRIMA'01)*. Taipei, Taiwan. pages 12

Valk, J. M., Witteveen, C., and Zutt, J. (2001b). A chain manager strategy for a multimodal transportation system. In Kröse, B., de Rijke, M., Schreiber, G., and van Someren, M., editors, *Proceedings of the Thirteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'01)*, pages 449–456. Amsterdam. pages 12

Valk, J. M., Witteveen, C., and Zutt, J. (2001c). Task allocation policies for logistic problems. In *Proceedings of the First Seminar "Joint TNO TRAIL Transport Research T3" (TRAIL'01)*. TRAIL Research School, Delft. pages 12

van Arem, B., van Driel, C. J., and Visser, R. (2006). The impact of cooperative adaptive cruise control on traffic-flow characteristics. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):429–436. pages 3

van der Krogt, R. P. J. (2005). *Plan Repair in Single-Agent and Multi-Agent Systems*. Ph.D. thesis, Delft University of Technology. pages 48

van der Krogt, R. P. J., Aronson, L. D., Roos, N., and Zutt, J. (2002). Tactical planning using heuristics. In Blockeel, H. and Denecker, M., editors, *Proceedings of the Fourteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'02)*, pages 187–194. Leuven. pages 12

van Gemund, A. J. C. (1994). The pamela run-time library. Technical Report 1-68340-44(1994)03, Delft university of technology. Version 1.0. pages 13

Verbraeck, A. and Versteegt, C. (2001). Logistic control for fully automated large-scale freight transport systems. In *IEEE conference on Intelligent Transportation Systems*. Oakland, California, USA. pages 2

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37. pages 109

Wang, X. F. and Chen, G. (2003). Complex networks: Small-world, scale-free and beyond. *IEEE Circuits and Systems Magazine*, 1:6–20. pages 127

Withers, R. M. J. and Rijnsdorp, J. E. (1978). Work of the social effects of automation committee from bad boll to enschede. In *Automatica*, volume 14, pages 189–191. Pergamon Press. pages 1

Yannis Dimopolus, Alfonso Gerevini, P. H. and Saetti, A. (2006). The benchmark domains of the detrministic part of ipc-5. In *International Conference on Automated Planning and Scheduling*. pages 179

Zhan, F. B. (1997). Three fastest shortest path algorithms on real road networks: Data structures and procedures. *Journal of Geographic Information and Decision Analysis*, 1(1):69–82. pages 36

Zhan, F. B. and Noon, C. E. (1998). Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73. pages 36

Zhan, F. B. and Noon, C. E. (2000). A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. *Journal of Geographic Information and Decision Analysis*, 4(2):1–11. pages 36

Zutt, J. (2001). *Cooperative Transport Planning*. Master's thesis, Delft University of Technology, Delft. pages 12

Zutt, J., Aronson, L. D., van der Krogt, R. P. J., Roos, N., and Witteveen, C. (2002). Multi-agent transport planning. In Blockeel, H. and Denecker, M., editors, *Proceedings of the Fourteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'02)*, pages 387–394. Leuven. pages 12

Zutt, J. and de Weerdt, M. M. (2000). Cooperative transport planning (abstract for a demo). In van den Bosch, A. and Weigand, H., editors, *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'00)*, pages 371–374. Tilburg. pages 12

Zutt, J., van Gemund, A., de Weerdt, M., and Witteveen, C. (2009). Dealing with uncertainty in operational transport planning. In Negenborn, R., Lukszo, Z., and Hellendoorn, J., editors, *Intelligent Infrastructures*, pages Ch. 14, 355–382. Springer, Dordrecht, The Netherlands. pages 12

Zutt, J. and Witteveen, C. (2004). Multi-agent transport planning. In *Proceedings of the Sixteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'04)*, pages 139–146. Groningen. pages 12

Zuurbier, F., van Lint, H., and van Zuylen, H. J. (2006). Comparison study of decentralized feedback strategies for route guidance purposes. In van Zuylen, H. J., editor, *Proceedings of the 9th TRAIL Congress (TRAIL'06)*. pages 139

Summary

Operational Transport Planning in a Multi-Agent Setting

Red thread

Classical approaches to operational transport planning treat planning separately from conflict resolution, thereby seriously affecting travel predictability aspects, even under incident-free conditions. Many researchers attempted to improve with MIP formulations, but those turned out not sufficiently scalable for realistic problems at the operational level. Furthermore, there have been developments in shortest-path algorithms, where planning and conflict resolution is integrated. These *context-aware* routing algorithms seem abandoned because of their high cost in computation time.

In this thesis, we aim to design and evaluate a distributed approach, based on improved context-aware routing. By means of this method predictability of travel plans can be greatly enhanced under incident-free conditions. But modeling uncertainty and incidents are present in any real-life situation. Therefore, in order to improve the robustness and performance under incident conditions, special variants of the method are developed, which enable us to deal with incidents and modeling uncertainty.

We developed a new framework that supports both the classical approach as our context-aware approach and its extensions such that they can be compared. The framework includes infrastructure agents that (locally) guard the safety constraints of infrastructure resources. The infrastructure agents make a two-level approach possible. At the first level, infrastructure agents compute reservations that are guaranteed to be safe (no conflicts with other planned actions). At the second level, transport agents search plans and they communicate with these infrastructure agents.

The behavior of the transport agents is largely determined by the planning method they use. We developed a new multi-agent context-aware (MACA) approach, which is faster than its competitors. Subsequently, because of the arbitrary order in which transport agents create plans and the presence of incidents, the extensions MACA-RP and MACA-RR are presented. We show positive examples illustrating the methods perform better in some cases, though we do not provide theoretical bounds on the improvement.

Because we have shown that the MACA approach not always outperforms the classical

approach, we present an extensive set of experiments to show empirical results. Due to the many different parameters we want to control, we first make use of a synthesized set of problem instances. Next, we also experiment with airport taxiing on the Schiphol airport network to illustrate that our methods also work on a realistic example. The experiments show that MACA indeed outperforms the classical approach. Furthermore, the extended MACA-RP and MACA-RR methods also significantly improve the performance, even in the absence of incidents.

Samenvatting

Nederlandse vertaling van Operational Transport Planning in a Multi-Agent Setting

The problem setting is that there are a set of transportation orders with time windows both for pickup and delivery. To travel over infrastructure resources, an agent needs to claim these resources. In case an agent wishes to reserve a resource (rather than simply planning to go there and relying on social rules), it can see the reservations of other agents on all infrastructure resources. A transportation agent also needs to load packages into its hold, which has a limited capacity; at the destination location, a package must be unloaded again. The time of loading and unloading is associated with a reward function, and each task also has a loading or unloading duration associated with it (in the TRAPLAS simulator, at least). Currently, loading and unloading occurs at locations that have (practically) no capacity constraints.

Jonne extends the resource-based infrastructure model of Hatzack and Nebel (HZN) with the following: connectivity between resources a capacity of a resource: in HZN, all resources have unit capacity. Jonne also allows capacities greater than one. In his TRAPLAS simulator, though currently not in his PhD, Jonne also distinguishes between infrastructure resources where vehicles can overtake, or not. Resources have a distance, and a maximum speed, and vehicles also have a maximum speed. Together, these elements determine how long an agent will occupy a resource

Transportation resources (i.e., vehicles/agents) make plans consisting of: a route, which is a sequence of resources a schedule, which is a sequence time points; these time points may correspond to reservations the agent has made at the corresponding resources (it can also plan without reserving) a load function and an unload function, which specifies the times at which packages are loaded and unloaded, respectively.

Like HZN, Jonne's algorithms focus on operational (i.e., short-term) planning and

scheduling. Unlike HZN, he does not assume that agents have a set of unalterable routes in advance; rather, planning which route to take is a key part of his work (there is quite a bit of background research on shortest path planning in his thesis). In addition, planning and scheduling can be triggered as a result of new transportation orders, or incidents in the system. He proposes the following incident model: incidents arrive in the system according to an exponential distribution. An incident is either a communication incident or a resource incident. A communication incident affects a single transportation agent, and prevents it from communicating with other agents for a specified time interval. Inability to communicate means that an agent has to fall back to simpler coordination methods, as it can no longer see the reservations on resources of other agents [right?]. A resource incident affects the maximum speed of a resource. In case the resource is a vehicle, this is the maximum speed at which it can travel; in case of an infrastructure resource, it is the maximum allowed speed that is reduced.

Chapter 4 distinguishes different types of methods for task allocation, planning, and scheduling. Task allocation consists in assigning transportation orders to agents. When auctions are used, agents will bid on transportation orders, and, if the right kind of auction is chosen, an order should be awarded to the agent that has the highest private value for this order (however, if orders are auctioned off one by one, this does not imply that the optimal distribution of orders over agents will reached). Alternative to using auctions is that unassigned tasks are put on a blackboard, and as soon as some agent decides it wants that order, it can take it off the blackboard. If reallocation of orders at a later stage is possible, then the disadvantages of assigning orders to the 'wrong agents' is mitigated.

The planning scheduling of the agents consists of two separate parts: first, given a partial plan to deliver a set of transportation orders, a new order must be inserted somewhere in the 'visiting sequence'. Second, a route must be planned to and from the pickup and delivery locations of the new orders. So, if the pickup of a new package at location b is planned between visiting locations a and c, then a new route must be planned from a to b and from b to c, to replace the route from a to c. In TRAPLAS, a heuristic is used to determine where to insert a new order, but the insertion of new orders is currently not described in the thesis.

When an agent determines a route to deliver a (new) package, it must reserve the resources on this route. To determine the best route between two locations, the agents make use of shortest path planning algorithms. This chapter not only describes some of the research into the shortest path planning problem, it also presents an algorithm for shortest path planning that takes into account reservations that other agents have made.

When an incident occurs, agents have different options of reacting to it (this is also operational planning). The easiest is not to do any re-planning or scheduling, and rely on social rules to resolve any conflicts between agents. Social rules in this thesis are traffic rules of a special kind, that determine which agents can go first when more than one agent

contends for the use of a resource at the same time. The traffic rules are divided into static rules, such as FIFO or longest queue first, or they may make use of dynamic information that determines the priority of an agent. Assigning priority to agents can be done using commonly employed scheduling heuristics such as earliest deadline first, or longest plan first.

A more advanced method of dealing with an incident is to keep the agents' routes, but to reschedule their use of the resources. This can be initiated by an agent that is unsatisfied with the level of resource availability, for instance when it is trying to insert a new order into its plan. Typically, all agents sharing some (bottleneck) resources with the requesting agent will be invited to participate in this re-scheduling process, by throwing away their current reservations. The possibility of also re-planning routes is mentioned but not explored.

A final idea, that may be used for infrastructure analysis, is to look at the level of redundancy in a infrastructure network, that can be measured by counting the number of alternative paths between two locations that differ at most, say, 5% from the shortest path.

Curriculum Vitae

Write a short story about Jonne Zutt.

TRAIL Thesis Series

A series of The Netherlands TRAIL Research School for theses on transport, infrastructure, and logistics.

Nat, C.G.J.M., van der, *A Knowledge-based Concept Exploration Model for Submarine Design*, T99/1, March 1999, TRAIL Thesis Series, Delft University Press, The Netherlands

Westrenen, F.C., van, *The Maritime Pilot at Work*: E*valuation and Use of a Time-to-boundary Model of Mental Workload in Human-machine Systems*, T99/2, May 1999, TRAIL Thesis Series, Eburon, The Netherlands

Veenstra, A.W., *Quantitative Analysis of Shipping Markets*, T99/3, April 1999, TRAIL Thesis Series, Delft University Press, The Netherlands

Minderhoud, M.M., *Supported Driving: Impacts on Motorway Traffic Flow*, T99/4, July 1999, TRAIL Thesis Series, Delft University Press, The Netherlands

Hoogendoorn, S.P., *Multiclass Continuum Modelling of Multilane Traffic Flow*, T99/5, September 1999, TRAIL Thesis Series, Delft University Press, The Netherlands

Hoedemaeker, M., *Driving with Intelligent Vehicles: Driving Behaviour with Adaptive Cruise Control and the Acceptance by Individual Drivers*, T99/6, November 1999, TRAIL Thesis Series, Delft University Press, The Netherlands

Marchau, V.A.W.J., *Technology Assessment of Automated Vehicle Guidance - Prospects for Automated Driving Implementation*, T2000/1, January 2000, TRAIL Thesis Series, Delft University Press, The Netherlands

Subiono, *On Classes of Min-max-plus Systems and their Applications*, T2000/2, June 2000, TRAIL Thesis Series, Delft University Press, The Netherlands

Meer, J.R., van, *Operational Control of Internal Transport*, T2000/5, September 2000, TRAIL Thesis Series, Delft University Press, The Netherlands

Bliemer, M.C.J., *Analytical Dynamic Traffic Assignment with Interacting User-Classes: Theoretical Advances and Applications using a Variational Inequality Approach*, T2001/1, January 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Muilerman, G.J., *Time-based logistics: An analysis of the relevance, causes and impacts*, T2001/2, April 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Roodbergen, K.J., *Layout and Routing Methods for Warehouses*, T2001/3, May 2001, TRAIL Thesis Series, The Netherlands

Willems, J.K.C.A.S., *Bundeling van infrastructuur, theoretische en praktische waarde van een ruimtelijk inrichtingsconcept*, T2001/4, June 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Binsbergen, A.J., van, J.G.S.N. Visser, *Innovation Steps towards Efficient Goods Distribution Systems for Urban Areas*, T2001/5, May 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Rosmuller, N., *Safety analysis of Transport Corridors*, T2001/6, June 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Schaafsma, A., *Dynamisch Railverkeersmanagement, besturingsconcept voor railverkeer op basis van het Lagenmodel Verkeer en Vervoer*, T2001/7, October 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Bockstael-Blok, W., *Chains and Networks in Multimodal Passenger Transport. Exploring a design approach*, T2001/8, December 2001, TRAIL Thesis Series, Delft University Press, The Netherlands

Wolters, M.J.J., *The Business of Modularity and the Modularity of Business*, T2002/1, February 2002, TRAIL Thesis Series, The Netherlands

Vis, F.A., *Planning and Control Concepts for Material Handling Systems*, T2002/2, May 2002, TRAIL Thesis Series, The Netherlands

Koppius, O.R., *Information Architecture and Electronic Market Performance*, T2002/3, May 2002, TRAIL Thesis Series, The Netherlands

Veeneman, W.W., *Mind the Gap; Bridging Theories and Practice for the Organisation of Metropolitan Public Transport*, T2002/4, June 2002, TRAIL Thesis Series, Delft University Press, The Netherlands

Van Nes, R., *Design of multimodal transport networks, a hierarchical approach*, T2002/5, September 2002, TRAIL Thesis Series, Delft University Press, The Netherlands

Pol, P.M.J., *A Renaissance of Stations, Railways and Cities, Economic Effects, Development Strategies and Organisational Issues of European High-Speed-Train Stations*, T2002/6, October 2002, TRAIL Thesis Series, Delft University Press, The Netherlands

Runhaar, H., *Freight transport: at any price? Effects of transport costs on book and newspaper supply chains in the Netherlands*, T2002/7, December 2002, TRAIL Thesis Series, Delft University Press, The Netherlands

Spek, S.C., van der, *Connectors. The Way beyond Transferring*, T2003/1, February 2003, TRAIL Thesis Series, Delft University Press, The Netherlands

Lindeijer, D.G., *Controlling Automated Traffic Agents*, T2003/2, February 2003, TRAIL Thesis Series, Eburon, The Netherlands

Riet, O.A.W.T., van de, *Policy Analysis in Multi-Actor Policy Settings. Navigating Between Negotiated Nonsense and Useless Knowledge*, T2003/3, March 2003, TRAIL Thesis Series, Eburon, The Netherlands

Reeven, P.A., van, *Competition in Scheduled Transport*, T2003/4, April 2003, TRAIL Thesis Series, Eburon, The Netherlands

Peeters, L.W.P., *Cyclic Railway Timetable Optimization*, T2003/5, June 2003, TRAIL Thesis Series, The Netherlands

Soto Y Koelemeijer, G., *On the behaviour of classes of min-max-plus systems*, T2003/6, September 2003, TRAIL Thesis Series, The Netherlands

Lindveld, Ch..D.R., *Dynamic O-D matrix estimation: a behavioural approach*, T2003/7, September 2003, TRAIL Thesis Series, Eburon, The Netherlands

Weerdt, M.M., de, *Plan Merging in Multi-Agent Systems*, T2003/8, December 2003, TRAIL Thesis Series, The Netherlands

Comment: Add missing entries here.