The world according to MARP

Multi-Agent Route Planning

Adriaan W. ter Mors

The world according to MARP

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 maandag 15 maart 2010 om 15:00 uur
door Adriaan Willem TER MORS
informatica ingenieur
geboren te Zoetermeer

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr. C. Witteveen

Copromotor: Dr.ir. F.A. Kuipers

Samenstelling van de promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr. C. Witteveen	Technische Universiteit Delft, promotor
Dr.ir. F.A. Kuipers	Technische Universiteit Delft, copromotor
Prof.dr.-ing. I.J. Timm	Goethe-Universität
Prof.dr. M. Fox	University of Strathclyde
Prof.dr.ir. S.P. Hoogendoorn	Technische Universiteit Delft
Prof.dr.ir. J.A. La Poutré	Technische Universiteit Eindhoven
Dr. A.H. Salden	Almende BV

SIKS Dissertation Series No. 2010-11. The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

This research has been funded by the Dutch Ministry of Economic Affairs, project number CSI4006.

ISBN: 978-90-8559-937-1

Contents

Chapter 1

Introduction

The day is November 13, 1994. The streets of Adelaide, Australia are empty except for twenty-six autonomous agents that drive around a 3.78km track for a total of 81 laps. On lap 35, agent 5 and agent 0 try to occupy a piece of track that has room for only one. The resulting collision immediately puts agent 5 out of the race, while agent 0 manages to limp back to the pit lane, where he retires with a broken suspension. Having scored more points in previous races, agent 5 is World Champion; agent 0 misses the title by a single point.

The subject of this thesis is multi-agent route planning (MARP), in which each agent (a computational entity with a degree of autonomy to choose its actions — see section 1.2) has to plan a route from a current location to a destination location, while avoiding conflicts with the plans of the other agents. The scope of MARP is not quite as broad as to encompass Formula 1 racing, at least not until the FIA[1] allows non-human drivers to enter the championship. Until that time, there are many interesting situations in which multi-agent route planning *is* a relevant problem, including planning taxi routes for airplanes at airports [33], and coordinating the movements of automated guided vehicles in flexible manufacturing systems [7]. In addition, research into the complexity of multi-agent route planning has revealed similarities with other planning problems, including moving pianos (out of a room of movable objects) [82], dodging asteroids, and escaping from prison (by evading detection from search beam lights) [25].

The main applications we will consider in this thesis are taxiway route planning and route planning for automated guided vehicles. In airport taxi routing, aircraft (agents) have to taxi from a runway to a gate, and then, after all ground handling services have been performed, they have to taxi from the gate to the runway for take-off. These aircraft drive around a shared infrastructure of taxiways, runways, gates, aprons, parking places, etc., and they may never come into contact with (or even close to) another aircraft. Also, agents are self-interested in the sense that they care little whether the planes of rival airlines arrive on time, as long as they are on time themselves. Although at many airports taxi routes have been specified in advance, route planning algorithms can improve performance, because the pre-specified routes may not be optimal (for example with

[1] The Fédération Internationale de l'Automobile is the governing body for world motor sport.

regard to the minimization of delay). In addition, new routes have to be found in case the standard taxiways are covered with snow or are otherwise obstructed. Also, wintry conditions sometimes require snow and ice to be removed from airplanes prior to take-off. This *de-icing* process usually occurs at a de-icing station, which means that an agent cannot take its regular route to the runway, but it must first taxi from the gate to the de-icing station. Moreover, an aircraft must take off within a certain time limit of de-icing (called the *holdover time*, which is typically 15 minutes), to prevent ice from re-forming. Hence, we need algorithms to plan along a sequence of locations, possibly with timing constraints between different locations.

Other application domains of multi-agent route planning are those in which Automated Guided Vehicles (AGVs) are deployed. On factory floors and in warehouses AGVs are used to transport materials between locations of the facility. The AGV routing problem is typically but one of the optimization problems involving AGVs. It must also be decided which transportation task to allocate to which AGV [38], what to do with an AGV once it has completed a transportation task (e.g. the idle-vehicle positioning problem [8]), what the optimal AGV fleet size is [79], when an AGV should recharge its battery [62], etc. All of these problems interact with the problem of determining the best routes for the AGVs. Another prominent area where AGVs are used is at container terminals (e.g. in Singapore, Rotterdam, or Hamburg), where AGVs carry containers to and from ships.

1.1 Problem description

In multi-agent route planning, there are a number of autonomous *agents* (in the above examples: (auto-)pilots in taxiing aircraft, and automated guided vehicles), each with its own transportation task, which implies that each agent has a start location and one or more destination locations[2]. In the example domains, an aircraft has to taxi to the gate after landing, while an AGV may have to pick up a pallet at the warehouse and transport it to a production station. We assume that agents traverse a *road map*. That is, rather than moving around in free space, an agent must keep to pre-specified roads. A taxiing aircraft, for instance, can no longer travel as the crow flies.

With more than one agent active on the infrastructure, the agents need to coordinate their movements to avoid collisions, both during route planning and during the execution of the route plans. In multi-agent route planning, the coordination between the agents must ensure that none of the *resources* of the infrastructure (e.g. roads and intersections) are ever occupied by more agents than the capacities of the resources allow. Planning-time coordination should ensure that coordination during the plan execution phase (e.g., maintaining sufficient distance between the agents) is no longer a difficult problem. The challenge is to coordinate the agents while enabling them to *efficiently* perform their transportation tasks, by which we mean that they want to finish their transportation tasks as quickly as possible. In the airport domain, where delays can be very costly, an aircraft should spend as little time as possible traversing the taxiways. For manufacturing systems that use AGVs for materials handling, efficient transportation is important to

[2]We will only consider the case where destination locations have to be visited in a particular order that is known in advance. Otherwise, a Traveling Salesperson Problem (TSP) (see e.g. [26]) has to be solved.

ensure that no machine is idle. In addition, an efficient transportation system may require fewer AGVs than an inefficient one.

Some existing approaches to coordination sacrifice efficiency to ensure conflict-free behaviour (cf. [20, 48, 97]). For example, the coordination problem for AGV systems can be solved by arranging the infrastructure as one large, uni-directional loop along which all pickup and delivery locations lie [85]. AGVs simply drive around this loop and pick up a load as soon as they encounter one, unless they already carry a load. Other approaches, such as always traversing the same taxiways from the gate to the runway, do not take into account the congestion on the taxiways when deciding on a route. To tackle the problem of finding efficient, conflict-free routes, we pursue a *planning* approach in this thesis, in which an agent takes into account the intended movements of other agents. That is, rather than postponing conflict resolution until execution time, or preventing any possibility of conflicts prior to planning, we integrate the processes of route planning and conflict resolution.

When agents try to carry out their plans, they may find that the environment has changed in ways they did not expect, and this may require them to revise their original plans. If these revised plans are much less efficient than the original ones, then the objective of efficient coordinated behaviour is still not met, for in practice circumstances frequently do change. Hence, a special area of focus in the thesis will be the evaluation of the *robustness* of agent plans, i.e., the property of plans to remain efficient even after moderate revisions.

1.2 Research question

This thesis aims to answer the question,

> Can *agents*, operating on a shared *infrastructure*, find *conflict-free* route plans that are both *efficient* and *robust* under changing circumstances, given limited *computation time*?

There are many definitions of an *agent* (cf. [23]), and one that fits our research well is from Maes [55]: "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed." In this thesis, we consider agents that perform transportation tasks, and they have to take into account the actions of other agents both in the planning and the execution of their tasks. We assume that each agent plans and executes the actions of a single vehicle, which can be anything from a small AGV to an aircraft. Depending on the application domain agents can be *self-interested*, as in the airport domain, or *cooperative*. An example of cooperative agents are AGVs operating on a factory floor, where the total performance of the AGV system is more important than the performances of the individual AGVs.

An *infrastructure* is a graph of *resources*, each of which can hold a limited number of agents at the same time. Given our focus on logistic domains, the infrastructure is usually a *road map*, consisting of roads connected by intersections. A set of *conflict-free* plans must ensure that there are never more agents in a resource than the capacity allows. In

addition, the execution of the agents' plans should not lead to a *deadlock* situation, as illustrated in figure 1.1, where four agents approach an intersection that can hold at most one.

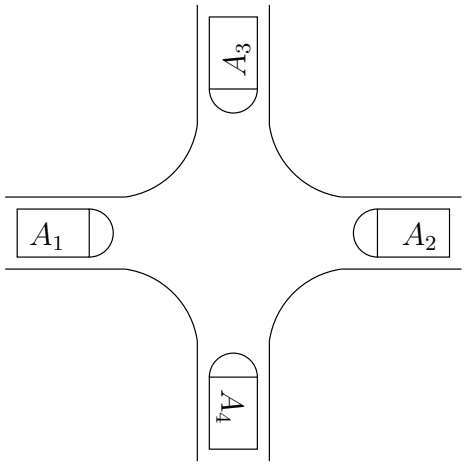


Figure 1.1: With four agents approaching an intersection, a deadlock will occur: only one agent can enter the intersection, and it cannot leave the intersection because all roads are blocked.

We assume that an *efficient* route plan is one that minimizes the completion time. Completion time is an important factor in logistic tasks, especially if other processes (or agents) in the environment depend on the completion of the transportation tasks. For example, workstations in manufacturing systems depend on the timely delivery of materials to start processing a job. An Automated Guided Vehicle System should therefore ensure a constant flow of materials to and from workstations.

When unexpected changes in the environment invalidate the plans of the agents (for example, aircraft can arrive late, AGVs may find their way blocked by humans that step into their path, etc.), then they must revise their plans in order to still achieve their (transportation) goals. The revised plans, however, are often less efficient than the original set of plans. We define the *robustness* of a set of plans in terms of the loss of efficiency (which, using the definition of efficiency given above, is measured in terms of delay), *given* a set of available plan repair techniques.

Finally, finding a plan should not require too much *computation time*. If we assume that an agent cannot start the execution of its route plan until the route has been fully computed, then the time spent finding the route should be negligible compared to the time traversing the route. For the same reason, the plan repair techniques should also require little computation time.

1.3 Contributions

Our research question states that we are looking for a way to compute efficient agent route plans. Unfortunately — although perhaps unsurprisingly — the problem of finding a globally optimal set of conflict-free route plans is intractable (NP-hard in fact, as we shall see in chapter 3). To be able to find efficient route plans in reasonable time, we take a heuristic approach where agents plan one by one, each agent ensuring that it does not create a conflict with any existing agent route plans. This thesis makes the following contributions to the field of multi-agent route planning.

A model for conflict-free route planning: What constitutes a set of conflict-free plans differs slightly for different application domains of conflict-free routing. The model we present in this thesis allows us to model several key application domains, including any special constraints that these domains might place on conflict-free travel (such as a constraint that forbids overtaking).

A fast and locally optimal context-aware route planning algorithm: We present an algorithm that finds the shortest-time route plan for a single agent, such that no conflict is introduced with its *context*, by which we mean any of the existing route plans from other agents. Similar algorithms from the literature are either slower (such as Kim and Tanchoco's algorithm [41]), or sub-optimal (such as Hatzack and Nebel's algorithm [33]).

A locally optimal multi-stage route planning algorithm: In conflict-free routing, the problem of finding the shortest-time route plan that visits a *fixed sequence* of locations is not trivially reducible to the problem of finding a route from a start location to a destination location. We first show that the naive approach of concatenating optimal route plans between successive locations is both sub-optimal and incomplete. Then, we present an optimal algorithm for the multi-stage routing problem that has a time complexity that is only slightly higher than that of the algorithm for the single-destination case.

A priority-based plan repair algorithm: Unexpected incidents in the environment can cause a delay for one or more agents. If we stick to the original set of plans in these situations, then non-delayed agents may have to wait for their delayed counterparts. We present an algorithm that allows non-delayed agents to take precedence over delayed ones, while still guaranteeing conflict-free plan execution. Our algorithm improves on an algorithm from Maza and Castagna [60] that also allows *priority changes*, but only under very strict conditions.

Empirical evaluation of the usability of route planning: It has been suggested in the literature that computing an optimal route is both too time-consuming (e.g. [88]), and a waste of time, because changes in the environment invalidate any plan (e.g. [47]). The latter criticism assumes either that no plan modifications are allowed, or that no repair techniques are available. As mentioned above, however, we *do* have repair techniques at our disposal, and our empirical evaluation of these repair techniques shows that route plans can be robust.

Additional experiments have been conducted to evaluate the cost of *multi*-agent plans. Although we can find the optimal route plan for a single agent, we have no guarantee that a set of individually optimal plans (sequentially obtained) is also optimal. In our experiments we try to establish whether or not the cost of a multi-agent plan is on average far removed from the cost of an optimal multi-agent plan, and how this cost gap depends on the chosen infrastructure topology. Our results indicate that if the agents manage to organize themselves into flows through the infrastructure, then the order in which they plan has no great impact on the cost of the multi-agent plan.

1.4 Overview

In chapter 2 we discuss related work. Research into multi-agent route planning has mostly been conducted in two separate fields: robot motion planning and route planning for automated guided vehicles. Research into robot motion is often concerned with low-level motor controls (such as acceleration, deceleration, turning angle, etc.), whereas AGV research is more concerned with route choices. Nevertheless, the methods to deal with multi-agent motion/route planning are similar in both domains. For example, an approach common to both is to plan the motions of the agents in sequence (which is also the approach we take in this thesis), subject to a particular priority of the agents.

In chapter 3 we present a framework for modelling conflict-free routing problems. The basic concept underlying the framework is that infrastructural elements such as roads and intersections are modeled as *resources* that can hold a finite number of agents at the same time. In addition to the resource capacity constraint, many application domains require agent plans to satisfy additional constraints, for example that agents are not allowed to overtake each other. These and other constraints can be expressed in our framework.

In chapter 4, we present our route-planning algorithms, both for the case where there is a single start location and a single destination location, and for the case where a fixed sequence of locations must be visited. Both algorithms are based on the idea that an agent must plan around the plans of previous agents; it can do so by making use of the *free time windows* on the resources. If we assume that agents A_1 to A_{n-1} have each made a plan, such that the set of $n-1$ route plans is conflict-free, then agent A_n can determine for each resource the time intervals during which its entry into the resource will not cause a conflict. At the least, this means that during a free time window there will be fewer agents on the resource than the capacity, but there may also be other constraints that the agent A_n has to take into account, depending on the application domain.

In chapter 5 we discuss *plan repair*[3], which is required when unexpected incidents disrupt the execution of plans. Our plan repair algorithm is based on the concept of priority: from a conflict-free set of agent plans, we can infer for any resource the order in which agents will visit the resource. The main idea of priority-based plan repair is that if this ordering is maintained during the execution of plans — even if some agents are delayed — then no deadlocks will occur, and all agents will eventually reach their destinations

[3]More specifically, we consider *schedule* repair, as our repair algorithms only change the timing of existing plans.

safely. To improve the performance of the agents, we can increase the priority of non-delayed agents over delayed agents as long as this does not introduce a deadlock. To judge whether a particular priority change will lead to a deadlock or not, we can use the plans of the agents to construct a graph that will contain a cycle if and only if the execution of these plans will lead to a deadlock.

In chapter 6, we present a set of experiments to evaluate the usability of our sequential approach to multi-agent route planning. First of all, we evaluate how agent delay is related to the number and severity of unexpected incidents, and we investigate whether increasing the priority of agents over delayed agents results in an overall reduction of delay. Second, we investigate the relation between the cost of the multi-agent plan and the order in which agents plan. In particular, we try to ascertain how much performance is lost by letting agents plan in the worst possible order, and how this loss of performance is related to characteristics of the infrastructure and distribution of start and finish locations of the agents.

In chapter 7, we revisit the research question posed in this chapter. We discuss the extent to which the contributions of this thesis provide a satisfactory answer to the research question, and we identify in which areas further work is still required.

Chapter 2

From Motion Planning to Route planning

In this chapter we will review the research on multi-agent route planning. The research question from the previous chapter provides us with a perspective on the literature.

> Can agents, operating on a shared infrastructure, find conflict-free route plans that are both efficient and robust under changing circumstances, given limited computational resources?

From the research question, we can derive the following three criteria on which to judge existing route planning approaches: (*i*) the *efficiency* in terms of the time required by the agents to finish all of their transportation tasks, (*ii*) the *robustness* of the plans in dynamic environments, in which unexpected changes can render plans infeasible, and (*iii*) the *computation time* required to find a solution, i.e., the speed of the planners. As we will see, many approaches to MARP focus on one of these criteria at the expense of one of the others. For example, there are approaches based on mixed integer programming that are capable of finding optimal solutions (e.g. [16]), but these typically require too much computation time for all but the smallest instances; other approaches focus on preventing deadlocks by taking routing decisions at real time, but these forego the opportunity of finding efficient plans in advance (e.g. [20]).

Before we discuss the literature on MARP, we will first discuss the related problems of robot motion planning (ROMP). The main difference between ROMP and MARP is that the latter assumes the existence of an infrastructure of roads and intersections, whereas robots typically enjoy full freedom of movement to avoid any obstacles they may find on their way. Consequently, the difference between a route plan and a motion plan is that the former specifies occupation times for the elements of the infrastructure, whereas the latter may specify turning angles and accelerations. Indeed, the research on robot motion planning typically considers more 'low-level' details, such as constraints on the acceleration capabilities of the agents, whereas in AGV research acceleration and deceleration are often not taken into account during route planning.

The ROMP problem can be said to be a more general version of the MARP problem, as we could solve MARP instances with ROMP solvers; MARP, in turn, can be viewed as an approximation for ROMP, in case it is possible to create a map of roads and intersections around all obstacles. The relevance of ROMP research to our current discussion is that many of the ideas and techniques used to solve motion planning problems also occur in MARP. Also, since many ROMP papers predate their MARP counterparts, these ideas can be said to have originated in robot motion planning.

This chapter is organized as follows. In section 2.1, we first discuss the field of robot motion planning. By way of introduction, we will not only consider multi-robot motion planning, but also discuss motion planning problems in which there is only a single robot, either avoiding a set of stationary obstacles, or avoiding a set of moving obstacles with known trajectories. Then, in section 2.2, we introduce research into automated guided vehicle systems, as much MARP research originates from this domain, and we briefly discuss some other planning problems in AGV systems such as task assignment and infrastructure design. In section 2.3, we will discuss multi-agent route planning approaches, most of which are in the context of AGV systems.

2.1 Robot motion planning

The initial interest in motion planning problems came from two research fields: computational complexity theory and robotics. Within the literature of these two fields, we can identify the following four classes of robot motion planning problems.

Kinodynamic motion planning: A single agent has to go from one place to another, avoiding collisions with a set of stationary obstacles. The word *kinodynamic* refers to the combination of *kinematic* and *dynamic* constraints [18]. Kinematic constraints limit the position of the agent, e.g. in the sense that it may not overlap with obstacles or that the different joints of the agent may not be in an impossible configuration. Dynamic constraints specify how the position of the robot may change over time, such as bounds on the maximum velocity and acceleration of the agent. Strictly kinodynamic constraints govern both the agent's position and the way it changes over time; an example of such a constraint is a speed-dependent obstacle avoidance margin.

Motion planning with moving obstacles: There is a single agent that has to go from one place to another, while avoiding collisions with a set of obstacles that move along known trajectories. The movement of the agents is subject to a set of kinodynamic constraints. Particular sub-classes of the problem can be distinguished based on the type of motion of the obstacles. In *asteroid avoidance* [74], the obstacles all move in a straight line. In *prison break* (cf. [25]), search beam lights sweep the grounds of the prison compound, and an escaping prisoner must make his way to the wall without ever being caught in the light.

Multi-robot motion planning: There is a set of agents, each with a destination location of its own. From the perspective of a single agent, the other agents are

moving obstacles with unknown trajectories (in addition to the agents, there may be stationary and moving obstacles with known trajectories).

Multi-agent route planning: There is a set of agents, each with a destination location in the infrastructure or *road map*. On the roads of the infrastructure, there are no stationary obstacles, and usually no moving obstacles with known trajectories either. Hence, an agent only needs to take into account the movements of the other agents.

In this section we will mainly discuss work in the field of (robot) motion planning in the first three problem classes. The fourth class of problems assumes that there exists an infrastructure on which the agents travel. The existence of an infrastructure is not a typical assumption in robot motion planning, which is why the fourth problem has not been studied much in robot motion planning. However, there is some work on constructing a road map, given the locations of a set of stationary obstacles, which we also discuss here. The research on multi-agent route planning in the domain of AGV systems is discussed in section 2.3.

2.1.1 Kinodynamic motion planning

Many robot motion planning approaches make use of the notion of a *configuration space* (or *Cspace*), first introduced by Lozano-Pérez [54]. This approach is based on characterizing the position and orientation of the object as a single point in the configuration space, in which each coordinate represents a degree of freedom in the position or orientation of the object. Only configurations of the object that do not intersect with any obstacles are allowed. The author shows how to compute which regions of the configuration space are forbidden due to the presence of other obstacles. Once these regions are known, finding a path from an initial configuration to a goal configuration can be done using a search through the configuration space.

Donald et al. [9, 18] consider kinodynamic motion planning for a point mass that must reach a goal configuration, while avoiding a set of polyhedral obstacles in two or three dimensions. They show that finding an optimal solution is NP-hard for three or more dimensions, and they develop a Fully Polynomial-Time Approximation Scheme (FPTAS). That is, if there exists an optimal-time solution requiring time t, then they can find a solution that requires at most $(1+\varepsilon)t$ time, for any $\varepsilon > 0$. Furthermore, the running time of their algorithm is polynomial in the input and in $\frac{1}{\varepsilon}$. Their algorithm performs a search through the *phase space*, which is the configuration space extended with a velocity dimension (which is required because of speed-dependent obstacle-avoidance margins). The main idea of their approach is to discretize the phase space into a "grid", where neighbouring grid points are reachable from each other using piecewise maximal accelerations. The "finer" the grid, the more likely it is to find a near-optimal trajectory. The overall complexity of their algorithm is $O(n^2V + V \log V)$, where n is the number of obstacles, and V is the number of grid points (the number of grid points depends on the parameter ε).

The approach by Donald et al. can be viewed as a graph search through the configuration space. An alternative approach is to let the motions of an agent be guided by an *artificial potential field*. The basic idea is that obstacles (and other agents) exert a

repelling force on the agent, whereas the goal configuration is an attractive force. The combination of these forces results in a potential field that the agent needs to follow in the direction of strongest attraction to arrive at its destination. A challenging problem is to design the potential field in such a way that there are no local minima, i.e., locations in the workspace other than the destination location from which the agent cannot escape (using the potential field). Rimon [77] shows how a potential field can be constructed from the geometric descriptions of the obstacles. Some approaches that utilize potential fields to control the motions of agents are from Loizou and Kyriakopoulos [52, 53], from Ogren and Leonard [70], and from Ge and Cui [28].

2.1.2 Motion planning with moving obstacles

If obstacles move around the environment, the configuration of the agent alone does not fully specify the state of the problem. Reif and Sharir [74] define the *free configuration space* by adding the dimension of time to the configuration space. Reif and Sharir analyze the complexity of the following problems: in case there are obstacles that may translate and rotate (along known trajectories), then the movement planning of a single disc with a bounded velocity is PSPACE-complete. In case there is no bound on the velocity of the disc, the problem is NP-hard. The authors also provide decision algorithms for motion planning in *asteroid avoidance*, in which obstacles move with a fixed translational velocity. A polynomial-time algorithm is presented for 2D asteroid avoidance for a constant number of obstacles, while an $O(2^{n^{O(1)}})$ algorithm is presented (where n is the number of obstacles) for the 3D case.

Fujimura [25] presents a polynomial-time algorithm for the problem of avoiding obstacles that move along known translational trajectories (though not necessarily with a fixed velocity). Fujimura mentions an interesting interpretation of his work: in *search beam avoidance*, the agent (e.g. a prison escapee) must avoid being caught by any one of the search light beams that sweep the prison grounds. Canny and Reif [10] showed that this problem is NP-hard in case some of the obstacles can move faster than the agent. Fujimura, however, assumes that the agent can move faster than any of the obstacles, and then the problem is solvable in polynomial time.

A popular approach to reduce the complexity is to decompose motion planning into spatial planning and velocity planning. The spatial planning phase ensures that the robot finds a path from its initial location to its goal location that avoids collisions with all stationary obstacles. The velocity planning should ensure that no two agents ever occupy the same location at the same time. The first decomposition approach is due to Kant and Zucker [39]. Kant and Zucker propose to use existing techniques to solve the path planning problem, for instance a graph search through the configuration space described by Lozano-Pérez [54]. The velocity planning problem can be solved by a graph search in two-dimensional *path-time* space, where time is explicitly represented as an additional dimension (with the additional restriction that no edges can be chosen that go back in time).

Wilfong [100] considers the case where obstacles do not move unless moved by the agent. In case the final positions of the objects are irrelevant, then finding a motion plan for the agent is NP-hard. The authors present an algorithm for the special case where

there is a single object in a two-dimensional space. If both the obstacle and the agent are convex polygons with $O(1)$ corners, and they are in a workspace with n corners, then an $O(n^3)$ algorithm can be formulated.

2.1.3 Multi-robot motion planning

The problem of transferring a set of objects from an initial configuration to a goal configuration has many interesting applications and interpretations. Hopcroft et al. [36] introduce the notion of the *warehouseman's problem*, whereas Schwartz and Sharir [82] coin the term *piano movers' problem*. Hopcroft et al. [36] show that the 'warehouseman's problem' is PSPACE-complete even in the case of moving polygons in a rectangular area. Hearn and Demaine [34] later repeated the result of Hopcroft et al. using a nondeterministic constraint logic model of computation. Using this model, they also demonstrated the PSPACE-completeness of various related puzzle problems, such as Sokoban and Rush Hour. Schwartz and Sharir [82] show that a polynomial-time solution is possible if the number of degrees of freedom is bounded by a constant.

Siméon et al. [84] propose a decoupled approach to the multi-robot motion planning problem. Each agent is assumed to have a fixed path that avoids all stationary obstacles, and the velocity planning is posed as a coordination problem. In [69], O'Donnell and Lozano-Pérez introduce the notion of a *coordination diagram* for two robots. A coordination diagram specifies the segments on the paths of the agents at which a collision might occur. Siméon et al. extend the coordination diagram to n agents, and search the n-dimensional space using an A* algorithm. Despite the worst-case exponential complexity of their approach, the authors were able to solve problems involving 150 robots within a few minutes.

A common approach in multi-robot motion planning is to plan the motions of the agents one by one. This approach consists of assigning priorities to the agents, then planning the motions one agent at a time, in the order specified by the prioritization. This means that for agent with priority i, a motion plan must be found that avoids the stationary obstacles, as well as the 'moving obstacles' that are agents $1, \ldots, i-1$. The approach of Erdmann and Lozano-Pérez [19] is to assume that the priority is given, and to plan the motions of the agents by searching the time-extended configuration space. To plan the motions of a single agent, the authors distinguish the following three subproblems: (i) how to construct the space-time configuration space, (ii) how much of the space-time configuration space to represent, and (iii) how to search the space-time configuration space. They describe their graph-search approach in two domains: planning the motions of polygons in the plane (where translations are allowed but rotations are not), and to plan the motions of two-link planar (robot) arms with rotary joints. Erdmann and Lozano-Pérez identified an interesting problem in prioritized motion planning: the planning process for agent i does not end when this agent has reached its goal configuration. It must also be checked that in this goal configuration, it will not hinder the movements of agents $1, \ldots i-1$ in case these agents are still moving when agent i is done. Another issue is that prioritized planning is not complete (this also holds for decoupling motion planning into path and velocity planning, of course): it is possible that the trajectory planning for agent i makes it impossible for agent $k > i$ to find a safe trajectory, even if

there do exist non-conflicting trajectories for agents $1, \ldots, k$. The authors remark that the prioritization itself may also be a planning process, e.g. the decomposition of a task into sub-tasks that can be planned in sequence by the agents.

Warren [98] combines prioritization with artificial potential fields. The potential field function is always the same for stationary obstacles, but for moving obstacles (i.e., agents that have already planned a trajectory) the potential field function may be different for each time slice of the space-time configuration space. The determination of the actual path is posed as an optimization problem: given an initial path consisting of a series of points, with the initial point the start configuration and the final point the goal configuration, change the location of the points (i.e., in space-time configuration space) such that the path passes through minimum potential, thereby avoiding all obstacles.

Both the decoupled approach and the prioritized approach trade optimality and completeness for computational complexity. As Erdmann and Lozano-Pérez [19] remarked, the multi-robot motion planning problem can be solved in the same way as the single-robot motion planning problem. If there are n robots with each k degrees of freedom, then we can view the set of robots as a single robot with kn degrees of freedom. A multi-robot motion plan can be found by searching the kn-dimensional configuration space. By comparison, the prioritized approach solves n problems of dimension $k + 1$ (a time dimension is needed to reason about the movements of agents with a higher priority). The need for approximation is thus quite clear, but none of the above authors discuss the impact of the approximation on the quality of the *multi-agent* plan. Savchenko and Frazzoli [80] do consider this issue, as they determine lower bounds and upper bounds on the *total transfer time* (i.e., the difference between the start time of the earliest agent and the finish time of the latest agent) for n vehicles in the plane. They assume the following setting: the workspace is a square of unit area, and there are no obstacles in addition to the agents. Agents appear in the workspace when they become active, and they disappear again when they have reached their destination. Hence, a solution always exists. The distance that agents must maintain between each other depends on their speed. In case this distance is non-zero when the agents stand still, then it is easy to show that the minimum transfer time is $\Omega(n)$, i.e., we cannot guarantee to do better than strictly sequential operation. For the case that agents may come arbitrarily close to each other if their speed is low enough, the authors derive the following bounds: for n arbitrarily chosen origin-destination pairs, the minimum transfer time is $\Omega(\sqrt{n}L)$, where L is the average distance between the start and end points. In case the origin-destination pairs are randomly drawn from a uniform distribution, the minimum transfer time is bounded by $O(\sqrt{n \log(n)})$. Sharma et al. [83] analyze the case where agents cannot communicate, and where they can only sense vehicles that are sufficiently close.

2.1.4 Multi-agent route planning in robotics

In multi-agent route planning, agent motion is restricted to an infrastructure of roads and intersections. While traversing the infrastructure, the agents 'only' need to take into account the movements of other agents, and there are no stationary obstacles placed on the roads or intersections. In robotics research, the existence of an infrastructure

is not a common assumption; most of the papers discussed above considered a set of (usually polyhedral) stationary obstacles, the avoidance of which can be a computationally difficult problem. In route planning for automated guided vehicles, the existence of an infrastructure *is* a common assumption, which is why we postpone the discussion of the main body of multi-agent route planning research (in section 2.3), until after the discussion of the automated guided vehicle systems domain (in section 2.2). However, there is a 'movement' in robot motion planning research that concerns itself with constructing an approximate (and conceptual) infrastructure (called road map in the robotics domain) given the stationary obstacles in the configuration space of the agents.

The work by Kavraki et al. [40] was motivated by the complexity of finding robot motion plans through configuration-space search, especially for robots with many degrees of freedom (recall that each degree of freedom corresponds to a dimension in the configuration space). The approach of Kavraki et al. is to learn a road map incrementally. The nodes of the road map are points in the configuration space where the agent does not overlap with any of the obstacles, and if those nodes in the road map are connected by an edge, then it is possible to find a motion plan from one node to the other. Initially, the road map is empty, and a free configuration point is chosen randomly. Next, a set of potential neighbour configurations are chosen that are not too far from the initial configuration. To determine whether two configurations are reachable from each other, a (motion) planning algorithm is used. If a very simple motion planning algorithm is used (e.g. simply try a straight line), then the planner may not find a trajectory even if one exists. Using a more advanced planner may reveal more connections in the road map, but will require more computation time. The road map can be used to find a motion plan for an agent as soon as its start and goal configurations are in the road map. However, by continuing to learn the road map (i.e., by adding more nodes and edges to the road map) it might be possible to find a more efficient motion plan later.

Van den Berg and Overmars [94] assume that a road map has been constructed that allows the robot to avoid static obstacles. To avoid collisions with other agents, they take a prioritized planning approach. Hence, when agent i develops its motion plan, it knows the trajectories of agents $1, \ldots, i-1$. Specifically, it knows the *free time interval* for each node in the road map, i.e., the interval during which it can occupy the node without fear of colliding with any other agent. To reach its destination configuration, the agent must find a motion plan that uses only the free time windows on the nodes. To see whether a free time window on one node is reachable from another free time window on an adjacent node, the authors send a *probe* from the originating node. This probe is a small planning process in itself that tries to find the earliest entry time into the neighbouring free time window. The probe works by traversing the edge in very small time steps. At each time step, it tries three different things: to go forward, to go backward, and to stand still. Any of these options is viable if they do not result in a collision with another agent. The agent tries to find a plan from its start configuration to its goal configuration by sending out probes to unexplored regions of the road map in an A* fashion.

In prioritized approaches, the quality of the multi-agent plan depends on the order in which agents make their plans. Van den Berg and Overmars [93] considered the case where the quality of the multi-agent plan is measured in terms of *makespan*, i.e., the difference between the start time and the time that every agent has reached its destination

configuration. The heuristic they present is to give the highest priority to the agent that has to cover the greatest distance. The idea is that if an agent has to travel only a short distance, then it has time to avoid the movement of higher-priority agents.

In [1], Akella and Hutchinson do not assume the existence of a road map, but they assume that the trajectory of the robots is fully specified, both in terms of the spatial path followed, and in terms of the velocity. What remains it to vary the starting times of the agents. The authors show that the problem of finding minimum time solutions is NP-hard, and they present a mixed integer linear programming formulation that allows the authors to solve problems with up to 20 robots. In Peng and Akella [71], the authors consider the problem of finding velocity profiles for the agents in case their spatial paths are fixed. This problem can be formulated as a mixed integer *non*linear programming problem. As an approximation, the authors solve two related mixed integer linear programming problems, which allow them to solve problems of up to 12 agents.

2.1.5 Lessons learned from motion planning

Many of the ideas and techniques used in robot motion planning are also used in multi-agent route planning. First of all, there is the decomposition of spatial and velocity planning [39]: a robot first makes a spatial plan that avoids contact with stationary obstacles, then it makes a velocity plan (equivalently, a schedule along its planned path) to avoid collisions with moving obstacles and other agents. Hatzack and Nebel [33] showed how this idea can be used in multi-agent route planning: in an airport taxi routing scenario, they assumed that each aircraft has a fixed path from e.g. runway to gate, and aircraft are scheduled along their paths in such a way that at most one aircraft occupies a taxiway segment at one time.

A second idea that occurs in both robot motion planning and multi-agent route planning is prioritized planning. Instead of trying to find an optimal plan for all agents at the same time, the agents or robots plan one after the other, such that a conflict with an existing plan is never introduced. The prioritized approach sacrifices the guarantee of an optimal global plan, and sometimes the guarantee of completeness, in favor of reduced computational complexity (cf. [19]).

The most significant difference between the fields of robot motion planning and multi-agent route planning is that the existence of an infrastructure or road map is a common assumption in the latter field, but not in the former. It is not uncommon, however, for robot motion planning approaches to approximate an infrastructure. The approximation by Donald et al. [18] discretizes the phase space (which is the configuration space extended with a velocity dimension) into a grid where neighbouring grid points are reachable from each other using piecewise maximal accelerations. Kavraki et al. [40] try to construct a road map by connecting points in the configuration space (thereby not taking time into account and only avoiding stationary obstacles). Van den Berg en Overmars [94] assume that such a road map has been learned, and they develop a route planning algorithm that makes use of the free time windows on nodes of the road map. In fact, the third idea that occurs in both fields is the idea of a free time window, that specifies when an agent can be at a location without coming into conflict with a moving obstacle or any of the other agents.

2.2 The automated guided vehicle domain

Before we discuss multi-agent route planning approaches in the next section, we first discuss the domain of automated guided vehicle research. This domain is relevant to our discussion for two reasons:

1. In most applications of AGVs, it is natural to model the workspace as an infrastructure consisting of roads and intersections (or other interesting locations such as drop-off points or workstations). Hence, route planning can be a computationally efficient approach to the agent transportation problem, especially in comparison with general robot motion planning, where the avoidance of stationary obstacles can already be a hard problem.

2. Though more efficient than general motion planning, multi-agent route planning can still require many computational resources (cf. [88]). In addition, it is not always clear whether a plan can be followed through in case the environment changes unexpectedly (i.e., the robustness of the plans are in question). This has led (AGV) researchers to look for other approaches, in which agents constantly monitor their execution for possible collisions with other agents.

In a survey paper on the design and control of AGV systems [96], Vis mentions three application domains of AGVs: manufacturing, transshipment, and distribution. The traditional AGV domain is manufacturing, where the material handling system is implemented using automated guided vehicles. These vehicles transport goods to and from a warehouse, and between production stations. A more modern application is transshipment at e.g. container terminals such as Rotterdam and Singapore, where the AGVs carry containers between the different modes of transportation, e.g. ships and trucks. An example application in distribution is the study by Van der Heijden et al. [95] on an underground system of tubes that connect a flower auction, an airport, and a railway station.

In addition to route planning, Vis mentions the following research topics in AGV research:

Infrastructure design: The infrastructure of an AGV system (sometimes called a flow path) can be designed to simplify the agent transportation problem. In a single-loop configuration (cf. [85]), all production stations are arranged around a single, unidirectional road. Since all agents drive in the same direction, only *catching-up conflicts* are possible. In a tandem configuration, the infrastructure is divided into multiple zones, and one zone is served by exactly one AGV [51]. Special transfer stations are used to transport a load from one zone to another. Since no collisions between the agents are possible, the only coordination between the agents occurs at the transfer stations.

Vehicle requirements: Since automated vehicles can be expensive, an important problem is determining the minimum number of vehicles that are required to satisfy all transportation demands [57, 56, 73].

Vehicle dispatching: In many AGV systems, the vehicles are homogeneous, which means that a transportation task can be performed by any of the vehicles. Typically, some heuristic function is used to select a vehicle that is currently idle. The heuristic functions are often based on a combination of the distance of the vehicle to the pickup location and the state of the buffers at the workstation (i.e., the pickup location) [42, 38, 2], or on the distances that vehicles drive empty [65].

Idle vehicle management: Once an AGV has finished a transportation task, it must be decided where it should go. Simply standing still can mean that the vehicle is blocking the way for other vehicles that are still carrying out transportation orders (compare this to the prioritized robot motion planning method of Erdmann and Lozano-Pérez [19], where a robot must check whether higher-priority robots will not conflict with the robot's goal configuration). In addition, if new transportation orders keep coming in, then it may increase system efficiency if the agent moves to a location where it is more likely to be close to a new transportation order. Bruno et al. [8] heuristically determine a new home location (a place where a vehicle parks) every time a new transportation order comes in, or when a transportation task has been completed.

Deadlock handling: A common and often implicit assumption in AGV research is that AGV systems are advanced enough to prevent actual collisions (using *forward sensing* [47]). If an AGV approaches another vehicle that is standing still, then it will halt before it collides with the stationary vehicle. If AGVs wait for each other, then the possibility of a *deadlock* may arise: a circular wait in which no vehicle can make progress, because they are all waiting for each other to move. In section 2.3, we will discuss approaches that ensure deadlock-free routes by planning trajectories in space and time. However, there also exist methods that combine (spatial) path planning, with an online controller that verifies that no deadlock is created for every step that an agent takes.

Many authors use Petri nets to model the AGV system [48, 101, 20]. The idea is that each time an agent is about to enter an infrastructure resource, an online controller checks whether this *transition* (i.e., in Petri net terminology) will lead to a deadlock. Viswanadham et al. [97] use the reachability graph of a Petri net to develop resource allocation policies for deadlock *prevention*. However, they report that this method is too time-consuming for problems of realistic size, which is why online deadlock *avoidance* mechanisms should be used in real-world AGV systems. Fanti [20], for instance, runs a small simulation in the Petri net to see if an intended transition will lead to a deadlock in the near future. Reveliotis [76] presents an approach in which the agent's path is not fixed, but instead it chooses where to go after each step. Using a variation of the Banker's algorithm [30], all of the agent's allowed successor resources are determined. From the set of allowable successor resources, some (unspecified) performance control policy should choose the most efficient one.

2.3 Multi-agent route planning

In the multi-agent route planning approaches we will discuss in this section, there exists a road map or infrastructure on which the agents travel. Hence, in contrast to the robot motion planning approaches discussed in section 2.1, an agent only needs to decide which roads to travel on, and at which speeds. In addition, the problem of avoiding collisions with other agents is solved in terms of the capacity of the *resources* of the infrastructure: an intersection (resource) can typically contain a single agent, while a lane (resource) can often contain multiple agents, as long as they do not overtake each other. The dimensions of the vehicles, or the dynamic constraints on vehicle movements, are often not taken into account in the multi-agent routing problem, with a few notable exceptions [63, 27], which are discussed in the context of prioritized planning approaches.

Finding optimal route plans for multiple agents is a hard problem. In chapter 3, we prove that a basic version of the multi-agent routing problem is NP-complete, while an additionally constrained version is suspected to be PSPACE-complete. Several methods exist in the literature to tackle the complexity of the problem, that mirror the techniques found in robot motion planning. First, there is the decoupling of path planning from velocity planning (section 2.3.1); second, there is the prioritized approach, in which agents plan in sequence, and agent $n+1$ should find a plan that is conflict free with regard to the plans of agents $1, \ldots, n$ (section 2.3.2). There also exist optimal (centralized) solvers (section 2.3.3), but because of their computational complexity they are rarely applicable to systems with more than a handful of agents. Also, as in all centralized approaches, there is a single point of failure. In section 2.3.4, we will discuss research in multi-agent routing that is concerned with dynamic environments and unexpected incidents.

2.3.1 Decoupled path and velocity planning

In Guo and Parker [29], every agent has a map of the environment that specifies the location of stationary obstacles. The first step in their approach is that each agent plans a path from its start to its destination using the D* algorithm [87]. The D* algorithm is similar to A*, except that it is dynamic in the sense that cost parameters can change. This implies that the D* algorithm can be used in partially known or changing environments, because the search algorithm can cope with e.g. changes in the locations of static obstacles. The second step in the approach of Guo and Parker is to identify the potential collision regions for the paths of all agents. These collision regions are considered static obstacles in an n-dimensional coordination diagram (where n is the number of agents) that is searched using the D* algorithm. The result of this search is a set of velocity profiles, one for each agent, in such a way that the total completion time is minimized. The second step of the algorithm runs in $O(2^n)$ time, where n is the number of agents.

Hatzack and Nebel [33] combine the decoupled approach with a prioritized approach: first, each agent plans a path through an infrastructure of resources (they model each intersection and road as a resource), and they assume that each agent simply chooses the shortest path. Then, for a given priority of the agents, each agent plans its velocity[1]

[1] As is common in the AGV research field, agents are assumed to have a maximum speed, but there are no bounds on acceleration and deceleration.

along its path in such a way that it does not create a conflict with any of the agents that have a higher priority. An agent plans its velocity by determining its entry time into each of the resources in its path, where the entry time into resource $i+1$ must equal the exit time out of resource i. To determine the possible entry times into a resource, Hatzack and Nebel make use of the concept of a *free time window*, which is a time interval in which the resource is empty of other agents (they assume that a resource can hold at most one agent at a time). Given that an agent is on a resource r at time t, within some free time window f, it will try to enter the next resource r' at the start of each free time window f' in case f and f' overlap. Their algorithm is a kind of depth-first search through the graph of free time windows. In the search process, a single free time window f needs to be considered only once[2], which means that the running time is bounded by the size of the *free time window graph*. If we suppose that each agent visits a resource only once (which is true if each agent plans the shortest path), then one resource can have at most n reservations, and $n+1$ free time windows, where n is the number of agents. If there are m resources in the infrastructure, then the algorithm of Hatzack and Nebel finds the optimal velocity profile for a given path in $O(mn)$ time.

A more general approach is to consider not one path for each agent, but multiple paths. Lee et al. [49] find the k shortest paths between every pair of locations in an offline planning stage. When an AGV must be dispatched to a new task, an online-controller finds a collision-free velocity profile for each of the k paths, and chooses the best one. Another approach that considers multiple fixed paths is from Ferrari et al. [22]. Each robot has k alternative paths that are conflict free with regard to stationary obstacles. For each robot, a family of k sub-robots is created, one for each path, and the synchronization problem between the robots is solved for all the families of sub-robots together (ignoring intra-family conflicts). Then, a best path is chosen from each family on the basis of a variety of different plan quality measures, that should ensure not only timely, but also robust plan execution. Some of the plan quality measures are: *running time*, which is the duration of the plan; *motion error*, which measures how much distance an agent keeps from stationary obstacles; *velocity error*, which is a measure of how much a robot can change its speed without worsening the global plan; *collision area factor*, which is the ratio of the length of a path inside a potential collision area, and the total length of the path. The approach by Ferrari et al. is an anytime algorithm[3] in the sense that k, the number of candidate paths per robot, can grow iteratively, as initially each robot starts with a single path.

2.3.2 Prioritized route planning

In the prioritized approach, an agent tries to find the shortest-time route[4] that avoids collisions with all agents that planned before it. An idea put forward by Broadbent et

[2]Hatzack and Nebel [33] do not make this observation in their paper; in case free time windows are considered multiple times, their algorithm has a worst-case exponential running time. See our earlier work [90] for an example.

[3]An *anytime algorithm* does not need to run until an optimal solution has been found, but it can always return a correct answer, the quality of which depends on the amount of computation time.

[4]We will use the term 'route' to refer to the combination of a path in the infrastructure (graph), and the occupation times of the path segments (i.e., the velocity profile).

al. [7] is to record the plans of higher-priority agents using node occupation times (which means that the occupation times on the links need not necessarily be recorded). From the node occupation times, we can derive the *free time windows* for the nodes. As explained in section 2.3.1, a free time window represents a time interval during which the agent can occupy the node without interfering with higher-priority agents. To find a conflict-free route plan, an agent must find a route that hops from free time window to free time window on adjacent nodes. This can be accomplished by constructing a graph of free time windows, and adapting standard graph search algorithms like Dijkstra's algorithm [17] or A* [32]. Three similar works that take this approach are from Fujii et al. [24], Huang et al. [37], and Kim and Tanchoco [41]. The work from Kim and Tanchoco is the best-documented research, and we will discuss it here. Because their work is also close to our research, we will discuss their approach in some more detail.

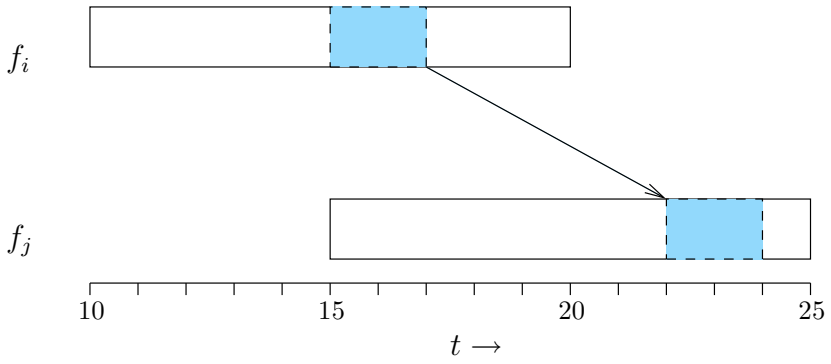


Figure 2.1: If an agent enters node i at time 15, then its earliest exit time is 17. The arrow represents the five time units required to traverse lane (i, j), so the agent enters j at time 22, leaving it at time 24, just before the end of free time window f_j at time 25.

Kim and Tanchoco [41] present an algorithm that is an adaptation of Dijkstra's algorithm for a search through the free time window graph. To determine whether a free time window on one node is reachable from a free time window on another node, three conditions must hold: (*i*) the nodes must be connected by a link, (*ii*) the second window should be reachable in time, and (*iii*) there should be no conflict with vehicles on the link between the nodes. The following short example (see figure 2.1) illustrates how we can determine whether the second condition of time reachability holds. Let $f_i = [10, 20)$ be a free time window on node i, and let $f_j = [15, 25)$ be a free time window on node j. The minimum traversal times of nodes i and j are 2 time units, and in between i and j is a link with minimum travel time 5. In case the earliest time that f_i can be reached is 15, then f_j is reachable from f_i: first, node i must be traversed, and it can be exited at time 17; then the link between i and j must be traversed, so node j can be reached at time 22. Traversal of node j may take another two time units, so an agent can exit j at time 24, which is within free time window f_j. Following the same reasoning, free time window f_j can *not* be reached in case f_i can be entered no earlier than time 17.

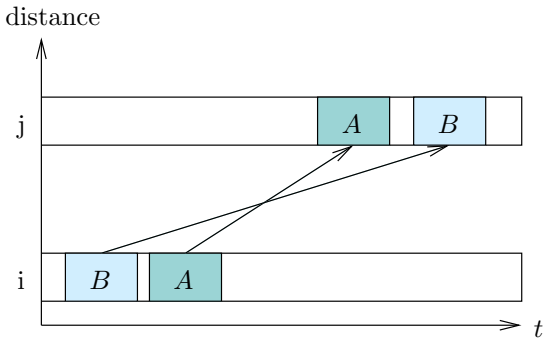


Figure 2.2: Travelling from node i to node j, agent A will overtake agent B.

In accordance with Broadbent et al. [7], Kim and Tanchoco distinguish two types of vehicle conflicts on links: a *head-on* conflict can occur when two vehicles travel in opposite directions on the same link; a *catching-up* conflict can occur when two vehicles travel in the same direction on the same link but one travels faster than the other. Kim and Tanchoco define the following procedure to determine whether there will be a catching-up conflict (the procedure for detecting head-on conflicts is similar) between agent A, which has not made a plan yet, with agent B that has made a plan (see figure 2.2). Suppose agent A wants to go from free time window f_i on node i to free time window f_j on node j. For all *reservations* on j that start after f_j, we first determine which agent they belong to. In figure 2.2, there is a reservation from agent B that starts after A's intended traversal time. For this agent B, we then check whether it has a reservation on node i that starts before the start of free time window f_i. In figure 2.2, we see that agent B does have a reservation that starts before f_i. This means that if agent A goes from f_i to f_j, then it will overtake agent B. This implies that f_j is not reachable from f_i. The procedure to determine free time window reachability is quadratic in the number of free time windows on a node. Since the authors assume that each agent makes at most one reservation on a node, this brings the complexity of this procedure to $O(n^2)$, where n is the number of agents.

The main algorithm works as follows (see algorithm 1 for the pseudo-code). In addition to the set F of free time windows, the algorithm maintains the sets T of free time windows to which the shortest route has been found, and $U \subset (F \setminus T)$ of windows that can be directly reached from a window in T. Initially, T contains the start window and U is empty. The first step of the algorithm, the labelling step, is to determine for each window in $F \setminus T$ whether it can be reached from some window in T. If a window is reachable, it is placed in U, and it is labelled with the earliest time that it can be entered. In the next step, the label setting step, the window in U with the smallest label is removed from U and placed in T. If this window has reached the destination node, the algorithm returns the solution, otherwise it continues with the next iteration of the labelling step.

Some notes on notation for algorithm 1: the k^{th} free time window on node i is denoted by f_i^k, and $\tau(f_j^q | f_i^p)$ stands for the earliest entry time into free time window f_j^q, coming

from free time window f_i^p, and taking into account the time at which f_i^p was entered (which is equal to its label $L(f_i^p)$). For the sake of brevity, we have omitted from algorithm 1 the code that maintains backpointers, which can be used to construct the actual plan once a free time window to the destination node has been reached.

Algorithm 1 Conflict-free shortest-time bidirectional AGV routing

Require: start node s, destination node d, set $F = \{F_1, \ldots, F_n\}$ of free time windows, where F_i is the set of free time windows on node i.
Ensure: shortest-time, conflict-free route plan from s to d.

1: $T \leftarrow F_s$
2: $U \leftarrow \varnothing$
3: For all $f_i^p \in (F \setminus T)$, set $L(f_i^p) \leftarrow \infty$ ▷ Initial labels
4: **for all** $f_j^q \in (F \setminus T)$ **do**
5: $\forall f_i^p \in T$ check reachability from f_i^p to f_j^q and calculate earliest entry time $\tau(f_j^q | f_i^p)$
6: $f_r^k \leftarrow \operatorname{argmin}_{f_i^p \in T} \tau(f_j^q | f_i^p)$
7: **if** $L(f_j^q) > \tau(f_j^q | f_r^k)$ **then** ▷ If current label is larger than entry time
8: $L(f_j^q) \leftarrow \tau(f_j^q | f_r^k)$
9: $U \leftarrow U \cup \{f_j^q\}$
10: $f_i^p \leftarrow \operatorname{argmin}_{f_j^q \in U} L(f_j^q)$
11: $U \leftarrow U \setminus \{f_i^p\}$
12: $T \leftarrow T \cup \{f_i^p\}$
13: **if** $i = d$ **then** ▷ Check if destination has been reached
14: **return** ▷ Construct plan using backpointers
15: **goto** line 4

The algorithm runs for at most $O(|F|)$ iterations, and in each iteration the labelling step may have to inspect $O(|F|)$ free time windows for reachability. Under the assumption that each agent makes at most one reservation per node, there are at most nm free time windows in the system, where n is the number of agents, and m is the number of resources. Kim and Tanchoco therefore conclude that the worst-case running time of their algorithm is $O(n^4 m^2)$.

The computational complexity of Kim and Tanchoco's algorithm, although polynomial, was considered by many (see Vis's review paper [96]) to be too high for practical use. For the same reason, Taghaboni-Dutta and Tanchoco [88] developed an incremental route planning approach that they compared with Kim and Tanchoco's algorithm. Their idea is to decide at every node where to go next. The choice of successor node is made heuristically, on the basis of a combination of factors such as the proximity of the next node to the destination, the traffic density at the adjacent nodes, and the estimated delay that will be incurred on the route following the next node. Although a speed-up was achieved by the incremental algorithm, the optimal algorithm of Kim and Tanchoco produced better results, especially when there are bidirectional links in the infrastructure.

Möhring et al. [63, 27] present a prioritized approach that takes into account the dimension of the vehicles, as well as their turning capabilities. Their approach is presented in the application domain of the container terminal Altenwerder at Hamburg harbour,

Germany. The infrastructure is a graph-like grid that consists of around 30000 arcs. When the center of an AGV is on a particular arc in the infrastructure, it may occupy parts of adjacent arcs. To avoid collisions with other agents, the authors draw a polygon around each arc that represents the area that an AGV can use while traversing the arc. If the polygons of two arcs overlap, then they may not be simultaneously occupied by different AGVs. Note that although this approach makes practical sense, it is not as efficient as the collision checks in many robot motion planning approaches, in which motions are only forbidden if the vehicles would actually collide, or if their vehicle-dependent safety margins overlap. Möhring et al. also consider the turning capabilities of the AGV, which means that not all arcs that are connected in the graph can be successively visited by the AGVs. For each arc, it is therefore recorded which outgoing arcs can be reached[5].

Möhring et al. consider a dynamic environment, where new transportation orders come in while the system is executing (although the task assignment is assumed to be given). For a new transportation order, a route is planned much in the same way as presented by Kim and Tanchoco [41]: Möhring et al. define a set of free time windows for each arc, and perform a Dijkstra-like search through the graph of free time windows. Taking vehicle dimensions into account is accomplished through the definition of the free time windows: if an arc e is occupied during the interval $[t_1, t_2)$, then any arcs of which the polygons overlap with e cannot have a free time window in that interval, either. The authors do not report on the complexity of their algorithm, other than to comment that it runs in polynomial time in the number of free time windows.

In all of the prioritized (and possibly decoupled) approaches discussed above, the priority of the agents is assumed to be given, and it has not been the subject of much research, as far as we know. Sometimes, the particular choice of priorities can have a big impact on the quality of the global plan, and for some priorities there may not even exist any global plan. Bennewitz and Burgard [5] study the problem of determining the right priority mainly for the second reason: to ensure that a global plan is found if one exists. They show that trying a number of priority schemes, which differ in randomly swapped agents, can greatly improve the chance of finding a global plan.

2.3.3 Optimal multi-agent route planning

A number of authors have tried using techniques with worst-case exponential running times like mathematical programming or constraint programming to tackle the multi-agent routing problem. These approaches can only be applied to problem instances with a small number of agents. For material handling systems with a limited number of AGVs, the task assignment problem (i.e., deciding which AGV performs a transportation task) is often more important to the system performance than the conflict-free routing problem, because (*i*) there is a small chance that an AGV is near the pickup location of a package, and (*ii*) there is less congestion in a system with only a few AGVs. It is therefore not surprising that many authors have combined optimal task assignment with optimal conflict-free routing.

[5] In [49], Lee et al. also consider agent turning capabilities by measuring the angle between two arcs, and only allowing traversal between two arcs if the angle is not too small.

In Desaulniers et al. [16], the objective of the material handling system is to finish all transportation orders with the minimal delay penalty. It is assumed that there is a set of transportation orders, each with a location and a time window for pickup, and a location and time window for delivery. Picking up or delivering a package earlier than the start of the respective time window is not possible, while delivery outside the window will result in a penalty. Desaulniers et al. present a mixed integer programming model to solve the problem of task assignment and conflict-free routing. In the model, each agent has a set of possible routes, and from each route (binary) parameters can be derived like: does agent A_k occupy node n at time step t, or does route r of agent A_k include pickup or delivery for task τ. The scheduling horizon is discretized into time intervals of equal length, and the authors chose a value of 15 seconds for each interval, with all processing and traversal times being an integer multiple of the minimum time interval. The solution to the model is to choose one route for each agent, such that the production delay penalties are minimized, no collisions occur between the agents, each transportation order is picked up and delivered by exactly one agent, and an agent never carries more than one load.

The proposed model is not suited to be solved directly by a mixed integer programming solver, because the set of possible routes for an agent is too large. Instead, the authors propose a column generation technique in which possible agent routes are explored incrementally. First, an initial set of routes is found by sequentially assigning a transportation task to the best agent, and planning a route for this agent that avoids collisions with other agents. The set of initial routes provides an upper bound on the time horizon for the subsequent steps. The column generation technique consists in dividing the problem into a master problem and a set of sub-problems. The master problem is a linear relaxation of the main problem, and operates on only a small set of possible routes for each agent. For each AGV, there is a sub-problem that should identify routes that can decrease the value (i.e., to minimize cost) of the objective function. Solving the master problem and the sub-problems alternates until no better routes can be found for the sub-problems. Finally, a branch-and-cut procedure is used to find an integer solution, in case the solution to the master problem is fractional. The optimality of the method by Desaulniers is subject to two restrictions: first, the length of the minimal-time interval influences the quality of the solution. Obviously, having a smaller minimum time step might increase the quality of the solution, but it results in longer computation times. A second restriction is that agents can only wait in nodes (i.e., intersections or pickup and delivery stations). When travelling on a lane between two nodes, the agent's speed is assumed to be a system-wide constant. The authors report solving instances for four AGVs in three minutes.

Corréa et al. [14] try to improve on Desaulniers et al. [16] by using constraint programming rather than mathematical programming for the master problem. A problem with the approach by Desaulniers et al. is that, if the initial heuristic fails to find a solution (note that many prioritized approaches are not complete), then there is no upper bound on the time horizon, and, according to Corréa et al., then the method will not find an optimal solution. The constraint programming approach of Corréa et al. can be described as follows: the master problem determines a task allocation and fixes end times for each of the AGVs. The sub-problems try to find conflict-free routes for each of the agents,

using mixed-integer programming, that satisfy the end times. If no set of conflict-free routes is possible, *no-good* constraints are added to the master problem. Instances with up to six AGVs can be solved within ten minutes of computation time.

Beaulieu and Gamache [4] present an approach that is based on dynamic programming, where legal states of the system are expanded to other legal states. An interesting feature of their approach is that they take into account the *displacement mode* of the vehicles, i.e., whether a vehicle moves forward or in reverse. The application domain is a haulage network in an underground mine, where a mining cart may only approach a service point when it is moving forward. The state of the system is therefore described by the positions (arcs) of the agents, how long each agent has to travel to reach the next intersection, and whether each agent is driving forward or in reverse. To generate a new state of the system, each agent has the option of (*i*) staying in its current arc, (*ii*) backing up into an adjacent arc, to make room for another vehicle at the intersection, or (*iii*) moving forward on an adjacent arc. Simply put, their approach iteratively selects the cheapest state from a set of reachable states, and expands it to all (non-dominated) reachable states, until the final state has been reached. The authors show that instances consisting of 4 agents and a network of 20 nodes can be solved in at most a few minutes.

A distributed approach to multi-agent route planning is presented by Nishi et al. [67, 66, 68]. The task assignment is assumed to be known a priori, so each agent has to find a route from its start location to its destination location, using a mixed integer programming solver. The agents plan a route concurrently, and then exchange plans with each other, to see whether their respective routes conflict with those of others. In the next iteration, the plans of the other agents are a given, and an agent finds a route plan that takes into account the plans of others. To avoid collisions with other agents, each collision results in a penalty. The constraint of not colliding with other agents is thus moved to the objective function (this technique is called Lagrangian relaxation), with the penalty for collisions increased in each iteration to ensure convergence. In each iteration, one or more agents are randomly selected *not* to change their plans. Otherwise, agents could end up in a cycle where they keep producing (the same) plans that conflict with each other. Although the approach of Nishi et al. is not guaranteed to find an optimal solution, the distance to the optimal solution is bounded by the *duality gap*. The authors report that instances with 15 AGVs were solved to within 5% of optimality in a few seconds of computation time.

2.3.4 Robust route planning

A route plan specifies both the path that an agent will take, and the schedule of occupation times for the path segments. In realistic application domains, unexpected incidents or changes in the environment can result in the planned path being unavailable, or the schedule becoming unattainable. According to many researchers that develop online controllers for deadlock handling, the question what to do with your route plan once it becomes invalidated is best answered by forsaking the planning approach altogether; in their opinion, it is better to determine at each step of the way where an agent should go, and to choose only directions that are safe (see the discussion on deadlock handling in section 2.2). There are also, however, researchers that aim to combine planning with

robust plan execution.

The starting point of the research of Maza and Castagna [58, 59, 60] is that a set of conflict-free agent route plans has been found using the algorithm from Kim and Tanchoco [41]. They assume that during the execution of these plans, agents can suffer an incident that temporarily immobilizes them. Agents that approach the immobilized agent are assumed to be capable of stopping before they collide with the stationary agent. Actual collisions will therefore not occur, but it is not hard to see how a deadlock situation can arise in case the other agents carry on with their plans as if nothing has happened. In figure 2.3, agent A_1 will go from A to D, and agent A_2 will go from F to C. Given this combination of origin-destination locations, one of the agents has to wait while the other traverses the link between B and E. Suppose that the agents have come up with the following plans, to the effect that A_1 will traverse the link (B,E) first (the nodes all have a minimum traversal time of 2, and the links a minimum traversal time of 4.).

$$\pi_1 = \langle A, [0, 2)\rangle, \langle B, [6, 8)\rangle, \langle E, [12, 14)\rangle, \langle D, [18, 20)\rangle \tag{2.1}$$

$$\pi_2 = \langle F, [0, 2)\rangle, \langle E, [14, 16)\rangle, \langle B, [20, 22)\rangle, \langle C, [26, 28)\rangle \tag{2.2}$$

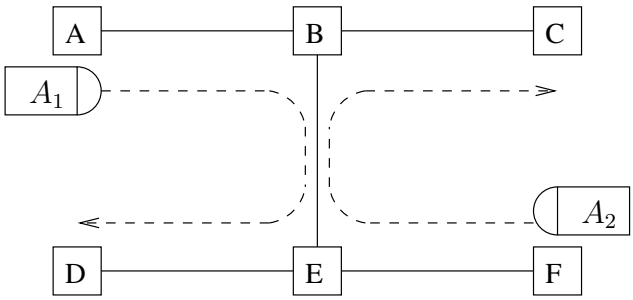


Figure 2.3: Agents A_1 and A_2 traverse the link between nodes B and E in opposite directions. If they meet each other on the link, a deadlock will occur.

If agent A_1 is delayed for e.g. 10 time units, and agent A_2 is unaware of A_1's problems, then the agents will meet on the link between B and E, and neither can make any progress in their plan.

Maza and Castagna observed that after all agents have made a plan, we can infer for each node in which order it will be visited by the agents[6]. For example, in the example of figure 2.3, the plans π_1 and π_2 specify that the nodes B and E should be visited first by agent A_1, and then by agent A_2. When agent A_1 was delayed in its start location, it was agent A_2 that entered node E first, while A_1 still entered node B first. Maza and Castagna showed that deadlock-free plan execution can be guaranteed if this (node) priority is maintained during the execution of the plans. In the example, agent A_2 should therefore have waited in link (F,E) until A_1 had exited node E.

[6]And from the specific visiting times we can infer a kind of inter-agent slack: how late can an agent be before it will interfere with the next agent to use the resource.

Maza and Castagna also identified opportunities for a non-delayed agent to increase its priority over a delayed agent. In the example, if the priority of A_2 is increased over A_1 for the entire path (E, (E,B), B) then agent A_1 must wait in link (A,B) for A_2 to pass, and both agents can reach their respective destinations without any deadlocks occurring. In general, a deadlock-free priority change is possible if the delayed agent has not yet reached the path that two agents share. For the case that there are more than two agents, Maza and Castagna proved that an agent can receive the highest priority on a certain path if and only if this path is currently empty of any other agents. Because this requirement is quite strict, Maza and Castagna also presented an alternative priority-changing algorithm [61], in which a delayed agent gets a lower priority, rather than increasing the priority of the non-delayed agent. Decreasing the priority of the delayed agent is safe if it does not yet occupy the path that it shares with the non-delayed agent. In this thesis, we also use Maza and Castagna's priority maintenance idea, and on its basis we will present an alternative priority-changing algorithm.

While the approach of Maza and Castagna aims to preserve existing route plans (i.e., it is a re-scheduling approach), a few re-planning methods have also been presented. In Narasimhan et al. [64], an agent selects a different route from a set of pre-computed alternatives in case its current route is no longer feasible, e.g. because of a path obstruction. It then checks whether the alternative route is free of conflicts with regard to the plans of the other agents. If it is not, the agent selects another of the precomputed alternatives. In case all of the agent's pre-planned routes create a conflict with other agents, then one of the other agents must select an alternative route. In Zutt and Witteveen [104], a prioritized planning approach is presented in which an agent plans only part of its route before other agents are allowed to plan a chunk of theirs. Once all agents have a complete plan, an agent is allowed to re-plan once, to see if it can find a more efficient plan. The authors also assume that incidents can disrupt the execution of plans, in which case agents also re-plan in order to find a feasible or more efficient route.

2.4 Concluding remarks

It is possible to distinguish between planning methods and non-planning methods (such as the Petri net approaches discussed in section 2.2) to the problem of multi-agent transportation in a shared environment (i.e., either an infrastructure or a workspace with a set of stationary obstacles). Our focus on planning methods — both in this chapter and for the rest of this thesis — is given in the sense that our research question states that we are *looking* for a good planning method. However, our choice is not motivated purely out of academic interest in planning methods, as we can point out various drawbacks of the non-planning methods briefly described in section 2.2.

The first of the non-planning methods is *infrastructure design*, possibly combined with traffic rules, that should ensure that no collisions or deadlocks are possible between agents. For example, in a tandem configuration of an infrastructure, each agent is assigned a non-overlapping area of the infrastructure, which means that two agents can never come into contact with each other. Although this method certainly simplifies an agent's routing decisions, it may not result in efficient use of the (AGV) system's resources. In addition,

even though agents can never collide, the system is not entirely robust, either, as the failure of a single agent can mean that an entire area of the infrastructure can no longer be serviced.

The use of online controllers, that verify the safety of every agent move, also has its drawbacks. Again, system resources may not be efficiently utilized, since agents rarely take congestion into account in their routing decisions. Second, verifying the safety of each agent move can require many computational resources, for example if the reachability graph of a Petri net has to be analyzed. For this reason, many online controllers cannot guarantee that no deadlocks will occur, in case they perform only a limited look-ahead to judge whether a move is safe. The main advantage of this approach is that no (plan) repair techniques are required in dynamic environments. Because there is no plan that must be maintained, the online controller can treat every situation the same, regardless of whether an incident has occurred or not. Whether or not this leads to a very robust system is unknown, because no route planning approach has ever been compared with an online controller approach, as far as we know, although Zutt's forthcoming thesis [103] will combine ideas from online control with route planning.

In this thesis we assume that there exists an infrastructure of lanes and intersections that the agents travel on. In this setting, collisions are avoided by ensuring that there are never more agents on an infrastructure resource than the capacity allows. In this sense, the robot motion planning approaches discussed in section 2.1 are less relevant because robot motions are planned in far greater detail, in the sense that trajectories are calculated in which the areas of two agents do not overlap. However, many solution methods found in (multi-)robot motion planning are also relevant in multi-agent routing: decoupling path and velocity planning, prioritizing agents and planning in sequence, and optimal centralized solving.

Among these three solution concepts, there is a clear trade-off between solution quality and computation time. At one end of the spectrum, optimal centralized approaches are, clearly, optimal in terms of solution quality, but these methods have so far been used to solve problems of no more than a few agents. At the other end of the spectrum, decoupled (and sometimes prioritized) approaches allow fast solving (e.g. Hatzack and Nebel's algorithm runs in linear time), but because the paths of the agents (i.e., the 'space trajectories') do not take into account the movement of others, it is possible that many agents make use of the same infrastructural resources, leading to congestion in the system. Prioritized approaches often allow a single agent to find an optimal route plan for itself, but as agent priorities are often determined heuristically if not arbitrarily, the distance between the quality of the *multi*-agent route plan and the optimum is often unknown. With regards to the computational complexity of the prioritized approach, an optimal plan for a single agent can be found in polynomial time, but the $O(n^4m^2)$ (n the number of agents, m the number of resources) worst-case complexity that Kim and Tanchoco [41] report of their algorithm is considered to be too expensive for practical use by many (see e.g. [96]).

With regard to the robustness, there is not known to be much difference between the three approaches (centralized, prioritized, or decoupled planning). In each approach, the result of the planning process is a route plan that specifies exactly where each agent

should be at each point in time. In case an unexpected incident disrupts the execution of the plans, there is the question what to do with the plans: find a new set of plans, or try to repair the existing plans. The method proposed by Maza and Castagna [60], which only affects the timing of the plans, can be applied to any route plan, regardless of the method used to obtain it (although the authors explicitly consider the planning method from Kim and Tanchoco [41]).

To recapitulate, we are looking for route planning methods that are efficient, robust, and fast in terms of computation time. In general, we can conclude that none of the described methods score well on all three counts. Methods that focus on plan efficiency often do so at the expense of planning speed (and vice versa), while a focus on robustness often disregards efficiency. In particular, our research is motivated by the following gaps in the literature.

1. Centralized solvers are optimal but too slow: optimal route plans have been found for up to four AGVs, but even then completion time frequently exceeds half an hour [16].

2. Prioritized planning is too time-consuming. Kim and Tanchoco's algorithm can find an optimal plan for a single agent, but their algorithm has a worst-case complexity of $O(n^4m^2)$ (n the number of agents, m the number of resources).

3. Prioritized and decoupled planning are not globally optimal. More importantly, no (empirical or analytical) approximation results have been found that bound the distance between the cost of an optimal plan and a plan obtained using a decoupled or prioritized approach.

4. Robustness of route plans needs more evaluation. In the literature, robustness is often considered with regard to an empty set of plan repair techniques (e.g. [47]), in which case a small disturbance can lead to a deadlock situation. Other than Maza and Castagna [60], who conducted some preliminary experiments to validate their approach, no-one has evaluated the robustness of route plans, as far as we know.

5. Additional research is required into plan repair techniques. Two schedule repair algorithms have been presented by Maza and Castagna [60, 61], but there is not yet a full understanding when a priority change is safe, and when it will lead to a deadlock. In addition, the two existing algorithms from Maza and Castagna still leave room for alternative priority-changing algorithms.

In our attempt to fill these gaps, we present a prioritized route planning algorithm that is faster than current approaches (chapter 4), we analyze priority-based schedule repair and present an additional schedule repair algorithm (chapter 5), and we evaluate empirically both the robustness of route plans and the multi-agent plan cost (chapter 6). First, in chapter 3, we will present the multi-agent routing problem and analyze its complexity.

Chapter 3

A Model for Multi-Agent Route Planning

Motion planning can be a very difficult problem in its most general form, in which a set of agents in a multi-(usually two- or three-)dimensional space must move from one location to another (cf. [75, 36]). An agent should decide on a value for continuous variables like speed and heading for each point in continuous time, all the while avoiding collisions with other agents that are solving the same problem. An agent can hardly be expected to have any time left to ponder the really important questions of existence like: "does P equal NP"?

The (single-agent) problem becomes tractable if agents move along an *infrastructure*: a collection of *intersections* and *lanes* connecting the intersections. An agent's motion planning problem can now be solved by deciding which lanes and intersections of the infrastructure to travel on (and during which time intervals), rather than plotting a complicated trajectory in space and time. The problem of avoiding collisions can be solved by defining for every lane and intersection in the infrastructure a *capacity*, which is the maximum number of agents that is allowed to occupy the lane or intersection at the same time. A natural approach to the multi-agent motion planning problem is *route planning*: each agent plans to visit a sequence of infrastructure *resources* during specified time intervals, in such a way that the capacity of all resources is never exceeded at any point in time.

Multi-agent route planning is applicable in many domains, from underground mining carts [4] to automated guided vehicles (AGVs) in manufacturing systems (e.g. [72, 21, 13]), to aircraft on taxiways (e.g. [33, 35]). Each of these domains may place constraints on route plans in addition to the resource capacity constraint. For example, we can imagine that AGVs can overtake each other on a lane, but underground mining carts cannot. On taxiways, aircraft should even maintain additional separation, because of their strong exhausts.

In this chapter, we present a resource-based model that allows us to define the problem of multi-agent route planning. In section 3.1, we define the basics of the model, which include a graph of resources, the definition of an agent plan, and the *resource load* function,

which specifies the number of agents on a resource at a particular time, given the set of agent plans.

In section 3.2, we identify a set of additional constraints that may be required to model logistic application domains. Most of the constraints of this section have appeared in different works in the literature; our aim in this section is to provide an overview of the constraints that are relevant for multi-agent route planning in logistics.

In section 3.3, we define the multi-agent route planning problem as finding a minimum-cost set of agent plans that are conflict free with regard to the resource load function (i.e., no resource may ever contain more agents than its capacity), and possibly with regard to some of the additional constraints identified in section 3.2. We will prove that the basic multi-agent route planning problem, in which no additional constraints are taken into account, is NP-hard. For some sets of additional constraints, the problem even appears to be PSPACE-complete[1].

3.1 Basic model

An *infrastructure* is a graph $G = (V, E)$, where V is a set of vertices representing *intersections* and *locations*, $E \subseteq V \times V$ is a set of edges. These edges can be both directed or undirected, and we will use the term *lane* to refer to both types of edge. The infrastructure graph contains no loops, where a loop is an edge with the same start and end point. From the infrastructure graph G, we derive a *resource graph* $G_R = (R, E_R)$. The set of vertices of the resource graph is the set of (infrastructure) *resources*: $R = V \cup E$. We derive the set of arcs E_R in the following manner: for each undirected edge $e = \{v, w\} \in E$, the set E_R contains the pairs (v, e), (e, w), (w, e), and (e, v); for each arc (directed edge) $(v, w) \in E$, E_R contains the pairs (v, e) and (e, w). See figure 3.1 for a translation of a simple infrastructure to a resource graph. The set of arcs E_R can be interpreted as a successor relation: if $(r, r') \in E_R$, then an agent can go directly from resource r to resource r'.

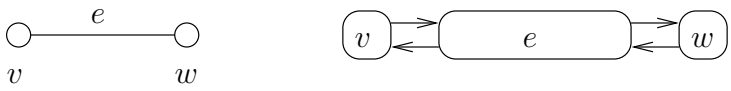


Figure 3.1: An infrastructure G, and the corresponding resource graph G_R.

We associate two attributes with every resource $r \in R$: a capacity and a minimum travel time. The function $tt : R \to T^+$ gives the minimum travel time of a resource, and it can typically be determined by dividing the length of a resource by the maximum speed of the agents, or by the maximum allowed speed. The set T is the set of possible time points, and it consists of the set of non-negative real numbers; the set T^+ consists of only positive real numbers. In case agents are heterogeneous for instance with regard to

[1]The PSPACE-completeness result is subject to the conjectured PSPACE-completeness of a problem from the literature, see section 3.3.1.

their maximum speeds[2], we use the function $at : \mathcal{A} \times R \to T^+$, such that $at(A_i, r)$ is the minimum travel time of agent A_i on resource r. The capacity is a function $cap : R \to \mathbb{N}^+$ that specifies the maximum number of agents that can simultaneously occupy a resource.

We will assume that all intersections and locations (i.e., resources in V) have a capacity of one (while resources in E can have a capacity of more than one). In case a crossing of roads is very large compared to the dimensions of a single agent, a clever division of the available space into resources is required to prevent a loss of infrastructure capacity. In figure 3.2(a), for example, we have a large central area where four roads meet. If only one agent were allowed to enter this area at a time, then the throughput of this infrastructure would be very low. In figure 3.2(b), we divide the central space into multiple resources that together form a roundabout.

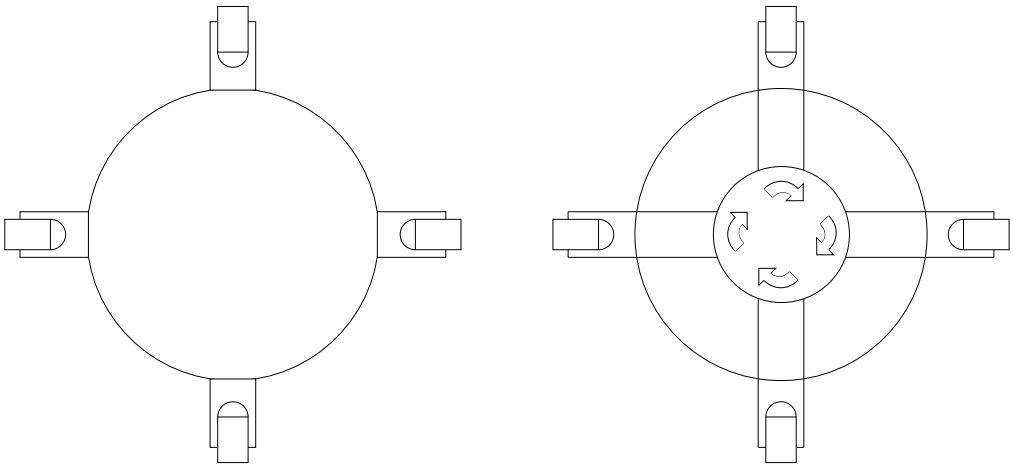
To understand why we do not model the intersection of figure 3.2(a) as one large resource with capacity four or more, one must understand the idea (philosophy would be too big a word) behind resource-based route planning. This idea is that potential conflicts between agents are prevented by ensuring that no resource ever contains more agents than its capacity. In case a resource can hold more than one agent at a time, avoiding collisions within a resource should be a simple problem. For example, in case a resource is one long lane, and two agents are driving behind each other, then a collision can be avoided if the following agent simply matches its speed to that of the leading agent. If, in figure 3.2(a), all four agents would enter the intersection at the same time, then it is not obvious how each should drive towards its destination exit without interfering with the other agents. In fact, in chapter 2, we saw that determining optimal conflict-free motions for a set of agents in the plane is PSPACE-complete (see [36]). Hence, to avoid conflicts at run-time, the agents would have to solve a complicated planning problem, even if the capacity of a resource is never exceeded.

At this point we should explain why we define both an infrastructure graph G and a resource graph G_R. The *resource* graph allows a uniform treatment of all infrastructure elements: agent collisions should be avoided both on lanes and on intersections, and not only lanes have non-zero travel time. The *infrastructure* graph ensures that the resource graph models a realistic infrastructure. For example, the infrastructure contains lanes with precisely two endpoints, both of which are intersections (or locations). This implies that a lane *resource* (i.e., in the resource graph) is only connected to other resources at its endpoints. Otherwise, we could construct resource graphs where resources are connected to each other 'in the middle', which would complicate the definition of a resource's travel time, for instance.

3.1.1 Agent plans

We define a set $\mathcal{A}$ of agents that can traverse the infrastructure. Each agent $A \in \mathcal{A}$ has one start location $r \in V$, and one destination location $r' \in V$ (that is, start and destination locations are always intersections). We approach the multi-agent routing problem as a planning problem: agents determine exactly for each time point which resource they will occupy.

[2]Agents can also be heterogeneous in other ways such as weight, which can influence minimum travel times on resources that slope upwards or downwards.



(a) A large area modeled as a single intersection resource.

(b) The same area divided into resources to form a roundabout.

Figure 3.2: A large central area can be used by more than one agent if we divide it into multiple resources.

Definition 3.1.1 (Agent plan)**.** *Given an agent $A \in \mathcal{A}$, a start location $r \in V$, a destination location $r' \in V$, and a start time $t \in T$, a plan for agent A is a sequence $\pi = (\langle r_1, \tau_1 = [t_1, t'_1)\rangle, \ldots, \langle r_n, \tau_n = [t_n, t'_n)\rangle)$ of n* plan steps *such that $r_1 = r$, $r_n = r'$, $t_1 \geq t$, and $\forall j \in \{1, \ldots, n\}$:*

1. *interval τ_j meets interval τ_{j+1} $(j < n)$,*
2. *$|\tau_j| \geq at(A_i, r_j)$,*
3. *$(r_j, r_{j+1}) \in E_R$ $(j < n)$.*

The first constraint in the above definition makes use of Allen's interval algebra [3][3], and states that the exit time out of the j^{th} resource in the plan must be equal to the entry time into resource $j+1$. The second constraint requires that the agent's occupation time of a resource is at least sufficient to traverse the resource in the minimum travel time. The third constraint states that if two resources follow each other in the agent's plan, then they must be adjacent in the resource graph.

The main objective in routing that we will consider is to minimize completion time. Hence, we define the cost of a plan for a single agent as the end time of the plan minus the start time. The cost of a set of agent plans is simply the sum of the costs of the agent plans, which is a suitable measure in case agents are self-interested, because then the cost of each individual agent should be reflected in the total plan cost.

[3]We make use of the *meets* predicate, which means that the end of one interval is equal to the start of the second, and the *precedes* predicate, which means that the end of one interval is earlier than the start of the second.

Definition 3.1.2 (Plan cost)**.** *Given a plan* $\pi = (\langle r_1, \tau_1 = [t_1, t'_1)\rangle, \ldots, \langle r_n, \tau_n = [t_n, t'_n\rangle)$ *and a start time* t*, the cost of* π *is defined as* $c(\pi) = t'_n - t$*. The cost of a set* Π *of agent plans is defined as* $c(\Pi) = \sum_{A_i \in \mathcal{A}} c(\pi_i)$*.*

As an alternative cost measure for the multi-agent plan, we will sometimes use the *makespan* of the plan, i.e., the time at which all agents have completed their plan. Given a set of agent plans Π, the makespan-cost is defined as $c_{\max}(\Pi) = \max_{\pi_i \in \Pi}(c(\pi_i))$, assuming the agents share a common start time. If each agent has a different start time, then the makespan is the difference between the latest finish time and the earliest start time. The makespan-cost can be a useful measure when agents are cooperative, and only the performance of the total system is relevant.

From a set of agent plans, we can derive for each point in time the number of agents that occupy a particular resource. This number should never exceed the capacity of a resource.

Definition 3.1.3 (Resource load)**.** *Given a set* Π *of agent plans, the* resource load λ *is a function* $\lambda : R \times T \to \mathbb{N}$ *that returns the number of agents occupying a resource* r *at time point* t*:*

$$\lambda(r, t) = |\{\langle r, \tau\rangle \in \pi \,|\, \pi \in \Pi \wedge t \in \tau\}|$$

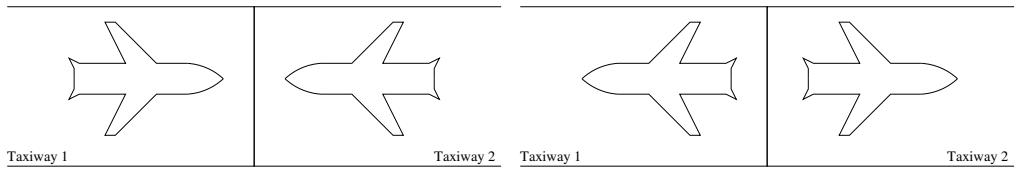
The constraint that the load of a resource may never exceed its capacity is at the heart of the multi-agent routing problem on infrastructures. It aims to solve the basic requirement that agents should never collide. In many realistic application domains, there are additional constraints that a set of agent route plans must satisfy. In the next section, we will examine a number of relevant constraints.

3.2 Additional constraints

In the previous section we introduced the basic constraint that there may never be more agents in a resource than the capacity. For some application domains, unsafe or impossible-to-execute plans will result if we do not impose additional constraints. For example, a long and narrow lane resource may be able to hold many agents at the same time, but there may not be traffic in opposing directions at the same time. In this section, we will give an overview of additional constraints, divided into three classes of related constraints, that may be desirable or required in (logistic) application domains.

The first class of additional constraints concerns how agents go from one resource to another. When we defined agent plans in definition 3.1.1, we specified that the time intervals of the plan steps should meet, rather than overlap. This implies that an agent always occupies exactly one resource, and that it never has one wheel in one resource and another wheel in a different resource. This, in turn, might mean that the situation of figure 3.3 occurs, which would be considered anomalous in most logistic application domains: at time t, two agents exchange resources. Although the resource capacity has never been exceeded, attempting this manoeuvre with real airplanes will undoubtedly result in a collision.

A second class of constraints is formulated to deal with situations in which the resource capacity constraint is insufficient to prevent agent collisions. In particular, we consider



(a) Situation at time t (b) Situation at time $t + \varepsilon$

Figure 3.3: The aircraft exchange resources just after time t.

three situations: (*i*) agents are not allowed to overtake each other, (*ii*) agents are not allowed to traverse a resource in opposing directions at the same time, and (*iii*) agents must maintain additional separation from each other, in the sense that there must always be one empty resource around each agent. The first two constraints are commonly relevant when resources are long and narrow, and there is no room for two agents to drive side-by-side. The third constraint is relevant when an agent needs more space than afforded by a single resource, for instance because it has very wide dimensions, or a very strong exhaust that requires agents to keep at least one resource between each other at all times.

The third class of constraints is defined on agent plans, rather than sets of agent plans. The constraints in this class represent choices that the agent designers can make with regard to desirable and undesirable properties of agent plans. One example of a potentially undesirable plan property is cyclicity: sometimes, the optimal agent plan (i.e., the shortest-time plan) requires an agent to visit one resource multiple times. If an agent designer is not only interested in arrival time but also in the distance an agent travels, then he or she can choose to forbid cyclic plans.

3.2.1 Simultaneous resource exchanges

In this section we will discuss constraints to prevent simultaneous resource exchanges between agents. The original constraint is due to Hatzack and Nebel [33]. They considered airport taxi route planning where each resource has unit capacity. Figure 3.3 depicts the type of situation they wish to prevent: two agents drive towards each other until both have reached the end of their respective resources, and then in one instant both agents find themselves at the start of the other's resource. If real aircraft would attempt to execute such plans, then a collision would result. This collision can be prevented by the following constraint (adapted from [33] to match our terminology):

$$\begin{aligned} \forall i \forall j \, (\langle r, [\_, t_1) \rangle, \langle r', [t_1, \_) \rangle) \sqsubseteq \pi_i \wedge \\ (\langle r', [\_, t_2) \rangle, \langle r, [t_2, \_) \rangle) \sqsubseteq \pi_j \rightarrow \\ t_1 \neq t_2 \end{aligned} \tag{3.1}$$

where the underscore symbol '_' stands for don't care, and $\sqsubseteq$ denotes a strict sub-sequence relation[4]. Equation 3.1 states that if there exists a plan π_i in which agent A_i travels from

[4]If sequence s is a strict sub-sequence of s', then s occurs in s' without any other elements interrupting

resource r to r' at time point t_1, then for all other agent plans π_j, it must hold that if A_j wants to travel in the other direction — from r' to r — then it must do so at some other time point $t_2 \neq t_1$.

In [103], Zutt considers a package-delivery application domain where resources can have a capacity for more than one agent. He allows simultaneous exchanges as long as the capacity of the *border* of the resource is not violated at any time. To formulate this constraint, he introduces the functions $stay : R \times T \to \mathbb{N}$ and $interchange : R \times R \times T \to \mathbb{N}$. The function $stay(r, t)$ returns the number of agents that are occupying resource r at time t, and that do not leave the resource at that time. The function $interchange(r, r', t)$ gives the number of agents that go from r to r' at time t, plus the number of agents that go from r' to r at time t. Zutt defines the following constraint:

$$\begin{aligned} &\forall (r, r') \in E_R, \forall t \in T : interchange(r, r', t) \leq \\ &\min\left(cap(r) - stay(r, t), cap(r') - stay(r', t)\right) \end{aligned} \tag{3.2}$$

Figure 3.4 shows two examples of exchanges that are allowed by equation 3.2, and two exchanges that are not allowed (note that Zutt does not distinguish between lane and intersection resources in his model). From figure 3.4(c) we can see that agents are allowed to exchange resources in a cyclic manner.

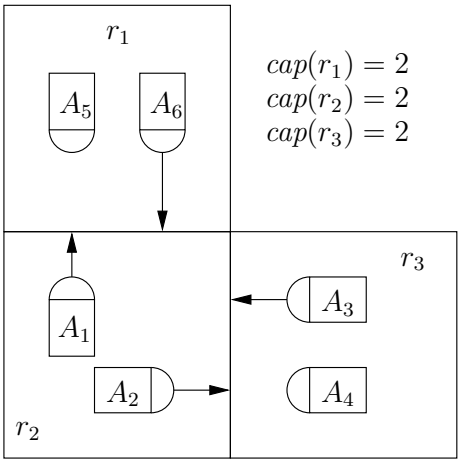
Both constraints 3.1 and 3.2 allow agents to simultaneously exchange resources if they form a cycle of three or more. Such plans can be executed only if agents move at exactly the right time; otherwise, the capacity of at least one of the resources will be violated. Even if (automated) agents could perfectly synchronize their movements, it might be safer if there is always (at least) one agent that moves to an empty resource; if for some reason perfect synchronization fails, this agent could move first into the empty resource. Then, the agent behind it can move into the vacated resource, and so on (see figure 3.5).

When we analyze the complexity of the multi-agent routing problem in section 3.3, we will see that the multi-agent routing problem is related to a number of puzzle problems (e.g. [46, 34]). In these puzzle problems there are *tokens* placed on the vertices of a graph, such that a vertex may contain at most a single token, and the objective is to move the tokens (by 'sliding them along the edges') to a particular goal configuration. One puzzle problem that we will use in section 3.3 is the Sliding Tokens problem.

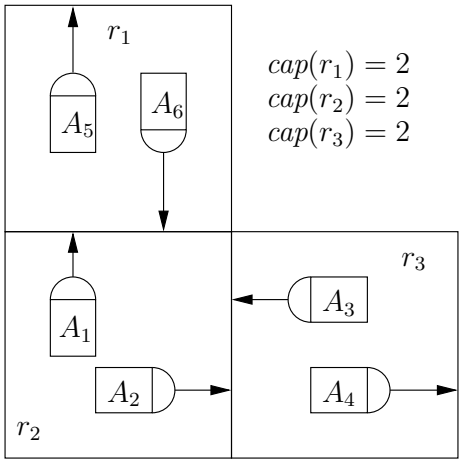
Definition 3.2.1 (Sliding Tokens (ST))**.** *Let $G = (V, E)$ be a graph. A* valid configuration *π is a placement of tokens on a subset $V_t \subset V$, such that no two adjacent vertices contain a token. A* move *consists of sliding a token to an adjacent, unoccupied vertex, such that the resulting configuration is also valid. The problem is: does there exist a sequence of moves that transforms an initial configuration π_0 into a goal configuration π_*?*

A solution to the Sliding Tokens problem is to move the tokens *one by one*; hence, a solution cannot require simultaneous movement of the tokens. If we want to transform a multi-agent route planning solution to a sequence of moves for the Sliding Tokens problem (for details, see section 3.3), then we must be able to serialize simultaneous

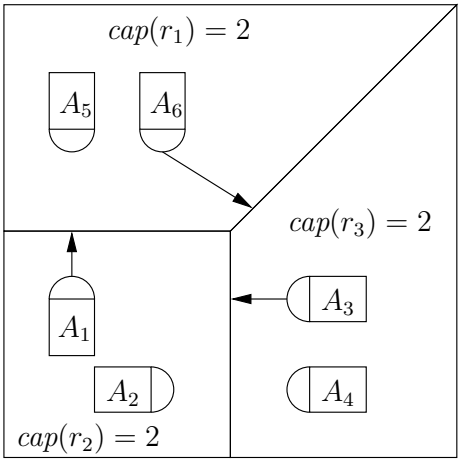
the sequence. For example, (b, c, d) is a strict sub-sequence of (a, b, c, d, e). Elsewhere in this thesis, we will also employ scattered sub-sequences, denoted by the symbol $\preceq$; the sequence (a, b, d) is an example of a scattered sub-sequence of (a, b, c, d, e).



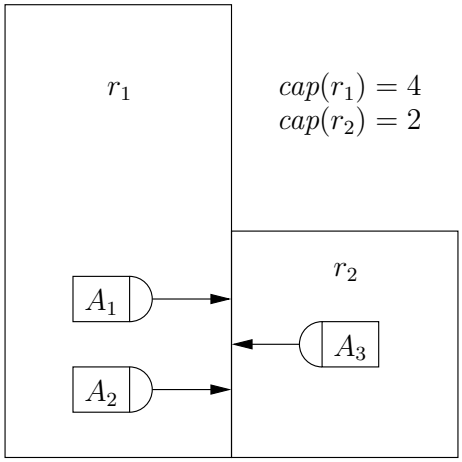
(a) Not allowed: $2 = interchange(r_1, r_2, t) \not\leq \min(2-1, 2-0) = 1$. The same holds for r_2 and r_3.



(b) Is allowed: $2 = interchange(r_1, r_2, t) \leq \min(2-0, 2-0) = 2$. The same holds for r_2 and r_3.



(c) Is allowed: $1 = interchange(r_1, r_2, t) \leq \min(2-1, 2-1) = 1$. The same holds for r_2 and r_3.



(d) Not allowed: $3 = interchange(r_1, r_2, t) \leq \min(4-0, 2-0) = 2$.

Figure 3.4: This figure shows some simultaneous exchanges that are allowed by constraint 3.2, and some that are not allowed. The arrows indicate the intention of the agents to change resources. All exchanges are assumed to occur at exactly the same time point t.

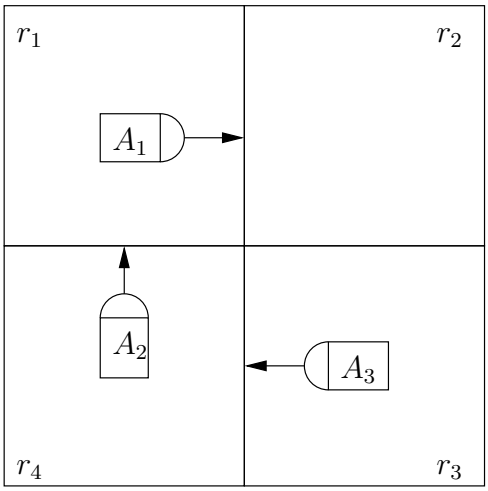


Figure 3.5: Agent A_1 does not need to synchronize with any agent to move to resource r_2.

movement from the multi-agent solution.

We will now present a stricter constraint that allows cyclic exchanges only if at least one of the resources in the cycle has capacity left over. Our constraint can be viewed as a *serialization* constraint, in the sense that we could move the agents to their new resources one by one, without ever exceeding the capacity of any resource. First, we define the set S_t of plan steps that start at time t, given a set of agent plans $\Pi = \{\pi_1, \ldots, \pi_n\}$.

$$S_t = \{\langle r, [t, \_)\rangle \in \pi \mid \pi \in \Pi\}$$

Next, we can define a dependency relation E_{S_t} between elements of S_t

$$E_{S_t} = \{(\langle r_i, [t, \_)\rangle, \langle r_j, [t, \_)\rangle) \mid (\langle r_i, [\_, t)\rangle, \langle r_j, [t, \_)\rangle) \sqsubseteq \pi \wedge \langle r_i, [t, \_)\rangle \in \pi'\}$$

The definition of E_{S_t} states that if $(\sigma, \sigma') \in E_{S_t}$, then plan step σ will enter at time t the resource that will be vacated at time t by the agent with plan step σ'. The serialization constraint states that if there is a cyclic exchange in E_{S_t}, then one of the resources in the cycle must have at least one unit of capacity left over. Let $C = (\sigma_1, \ldots, \sigma_m)$ be a cycle in E_{S_t}, then:

$$\exists i : \sigma_i = \langle r, [t, \_)\rangle \in C \text{ s.t. } \lambda(r, t^-) < cap(r) \tag{3.3}$$

where $t^- = \lim_{\varepsilon \downarrow 0} t - \varepsilon$ (t^- thus stands for the point in time just prior to the resource exchanges, when all agents are still in their 'starting resources'). In case there are only unit-capacity resources, then equation 3.3 specifies that E_{S_t} must be acyclic.

Note that we can verify in polynomial time whether equation 3.3 holds, despite the fact that E_{S_t} may contain an exponential number of cycles: we can simply select an arbitrary cycle in E_{S_t}, decide whether it contains a resource that has capacity left over, and remove the corresponding plan step from S_t. We can use this idea to translate a set of parallel agent plans (i.e., plans in which some resource exchanges occur at the same time) to a single multi-agent plan, in which only one agent moves to a different resource at any point in time.

Algorithm 2 Serialize parallel plans

Require: set of agent plans $\Pi = \{\pi_1, \ldots, \pi_{|\mathcal{A}|}\}$, and $S = \bigcup_{i=1,\ldots,|\mathcal{A}|} \bigcup_{j=1,\ldots,|\pi_i|} \sigma_{i,j}$.
Ensure: sequence of plan steps π such that $\forall i \in \{1, \ldots, |\mathcal{A}|\} : \pi_i \preceq \pi$.

```
1: for all t : S_t ≠ ∅ do
2:     determine(S_t, E_{S_t})
3:     while E_{S_t} contains cycle C do
4:         σ ← ⟨r, [t, _)⟩ ∈ C s.t. λ(r, t⁻) < cap(r)
5:         S_t ← S_t − σ
6:         append(π, σ)
7:         update(S_t, E_{S_t})
8:     S'_t ← topologicalSort(S_t, E_{S_t})
9:     append(π, S'_t)
10: return π
```

Algorithm 2 iterates over all resource transitions in the plans of the agents. For a particular time step t, there can be multiple resource transitions S_t. If the dependency relation E_{S_t} contains a cycle C, then we break this cycle by removing one transition σ that moves to a resource in which, just prior to the transitions at time t, there is at least one unit of capacity unused (line 4). We remove this transition from the set S_t of transitions at time t (line 5), add it to the sequential plan (line 6), and update the sets S_t and E_{S_t} (line 7) to reflect that transition σ has been processed. After all cycles have been removed from E_{S_t}, we can order the elements in S_t (line 8) in such a way that transition number i can move to a partially empty resource in case transitions $1, \ldots, i-1$ have been processed. Note that although the dependency relation E_{S_t} can contain an exponential number of cycles, the while-loop in line 3 runs for at most $O(S_t)$ iterations, because in every iteration we remove an element from the set S_t.

3.2.2 Resource traversal constraints

In this section we discuss three constraints. The first specifies that a resource may only be used in one direction at the same time, to rule out the possibility of oncoming traffic. The second constraint prevents agents from overtaking each other on a lane resource. The third constraint actually assumes unit-capacity resources, and specifies that an agent should have an empty resource around it at all times. The first two constraints are common in most AGV applications: lane resources can often hold more than one AGV, but there may not be room for two AGVs to drive side by side. Even if there is room for two AGVs next to each other, it will rarely be safe to allow both overtaking, and oncoming traffic

at the same time. The third constraint occurred in the Airport (taxi route) planning domain from the 2004 International Planning Competition, which included the constraint that an aircraft may not follow another too closely if the other's engines are running (i.e., while it is driving on the taxiways).

Simultaneous bidirectional lane traversal

If a resource r is a bidirectional lane connecting intersections v and w (i.e., $\{v, w\} \in E$), then agents can travel both from v to w via r, and also from w to v. As an additional constraint, we can formulate that traffic on a resource is only allowed in one direction at the same time. This constraint can be needed if there is not enough room for the agents to pass each other on the resource. Hence, if there is an agent that occupies r in the interval $[t, t')$, and this agent is going from v to w, then no agent may be going from w to v along the *same resource* r, between t and t'. This implies that if two agents have overlapping occupation intervals on r, then they must enter r from the same intersection:

$$\begin{aligned}(\langle v, \_\rangle, \langle r, \tau\rangle, \langle w, \_\rangle \sqsubseteq \pi \wedge \\ \langle w, \_\rangle, \langle r, \tau'\rangle, \langle v, \_\rangle \sqsubseteq \pi') \rightarrow \\ \tau \cap \tau' = \varnothing\end{aligned} \tag{3.4}$$

Note that if intervals τ and τ' from equation 3.4 meet, e.g. $\tau = [2, 6)$ and $\tau' = [6, 10)$, then a simultaneous resource exchange takes place at time 6 between resources r and w. If any of the constraints from section 3.2.1 holds, forbidding simultaneous exchanges, then τ and τ' cannot meet.

Catching is one thing[5]

If we forbid overtaking on (lane) resources, then an agent's entry and exit times into and out of a resource are constrained by two agents: the agent that will enter the resource directly before it, called the *leading* agent, and the agent that will come directly after it, called the *trailing* agent. Consider the following example.

Suppose we have a resource r with a capacity of 3, and we have the following two agents that are planning to make use of this resource: A_1 with plan step $\langle r, [20, 50)\rangle$, and A_2 with plan step $\langle r, [50, 70)\rangle$ (see figure 3.6). If a third agent A_3 can enter resource r from time 30 onwards, then it will drive between agents A_1 and A_2; agent A_1, which enters the resource first, is A_3's leading vehicle, whereas A_2, which enters the resource last, is the trailing vehicle.

If we do not allow overtaking, then A_3 is not completely free in planning its traversal of r, despite the fact that there is never a shortage of capacity. For example, if agent A_3 intends to travel through r in the interval $[30, 45)$, it would overtake agent A_1. The entry and exit times of agent A_3 are governed by the following equations:

[5] passing is another, as Murray Walker was wont to say. A good Dutch translation due to Olav Mol would be: "Erbij is één, er voorbij is twee".

Figure 3.6: Agent A_1 is leading A_2.

$$\begin{aligned} \text{entry}(A_1) < \text{entry}(A_3) < \text{entry}(A_2) \\ \text{exit}(A_1) < \text{exit}(A_3) < \text{exit}(A_2) \end{aligned}$$

The general overtaking constraint specifies that if two plan steps make use of the same resource, then the one that starts earlier should also finish earlier:

$$\begin{aligned} \langle r, [t_1, t_2) \rangle \in \pi \wedge \langle r, [t_3, t_4) \rangle \in \pi' \rightarrow \\ (t_1 > t_3 \wedge t_2 > t_4) \vee (t_3 > t_1 \wedge t_4 > t_2) \end{aligned} \tag{3.5}$$

Equation 3.5 ensures that agents exit resources in the order that they entered them, and as a result it allows agents to execute their plans without overtaking manoeuvres taking place.

Minimum separation

For safety reasons agents must often maintain some degree of separation. For instance, if two agents travel on the same resource, they could be required to maintain a minimum distance, or a minimum time gap. A more rigorous approach is to require that the resources connected to the agent's current resource are all empty. This constraint was implemented in the airport taxiway planning problem of the International Planning Competition of 2004 [91] (in a restricted form — only the resource behind the agent should be empty, and only if the agent is currently moving), to model minimum exhaust separation distances. We can formulate the following constraint to ensure that all resources surrounding an agent are empty:

$$\langle r, \tau \rangle \in \pi \wedge \langle r', \tau' \rangle \in \pi' \wedge (r, r') \in E_R \rightarrow \tau^+ \cap \tau'^+ = \varnothing \tag{3.6}$$

where τ^+ indicates the closed interval $[t, t']$ of interval $\tau = [t, t')$.[6] In constraint 3.6 we do not specify that at most one agent may occupy a resource at any time (it would clutter up the equation), but we do always assume that resources have unit capacity if constraint 3.6 holds.

[6] This is required for a smooth PSPACE-completeness proof in section 3.3.

3.2.3 Agent plan properties

In this section we discuss three constraints on agent plans. The first constraint is concerned with the initial and final locations of the agents. The two basic options are: (*i*) agents enter and leave the infrastructure, or (*ii*) they must always occupy a resource. The second and third constraints are both concerned with cycles in plans. The second constraint forbids agents to turn around on a single resource. Turning around on a resource may be viewed as undesirable because it can disrupt traffic on the resource, but the constraint can also arise out of the physical limitations of the agents and the resources, for example if there is insufficient space to turn. The third constraint simply forbids an agent to visit one resource more than once in its plan.

Agent entry and exit into infrastructure

In our model we assume a single start location and a single destination location for each agent. Once an agent has reached its destination location, it is done. Depending on the application domain, the agent can then either leave the infrastructure, or remain on the infrastructure. In the latter case, the agent will continue to occupy a resource until the end of the planning horizon (i.e., of the other agents). In the literature, the *idle vehicle positioning* problem [8] considers what to do with such an agent. One consideration is to let the agent move to a resource in the infrastructure where it does not obstruct other agents. Another possibility is that the agent might receive a new transportation order, and to move to a part of the infrastructure where the next package is most likely to originate.

In Zutt's thesis [103], the assumption is made that all start and destination locations are parking places (resources with infinite capacity), so the agent does not have to move. A domain in which agents leave the infrastructure is the airport taxi routing domain (e.g., from the International Planning Competition of 2004 [91, 35]) where an agent takes off from the infrastructure after traversing the final resource in its plan (a runway). The situation regarding the agent's start location is similar. When the agent starts planning, it need not be in the infrastructure yet (in the airport domain, it should land first).

The constraint that we now formulate specifies that an agent should occupy a resource at all times. If t_1 is the start time of an agent's plan, t_2 the end time of the plan, $t = t_s$ the start time of the earliest agent, and $t = t_e$ the end of the planning horizon, then we have the constraint:

$$t_1 = t_s \wedge t_2 = t_e \tag{3.7}$$

Spinturn

When an agent enters a lane resource r, having entered from intersection $v \in V$, it stands to reason that it wants to reach the other end of r to be able to enter intersection $w \in V$. On the other hand, it is also possible that the agent merely entered r in order to let a more important agent use intersection v. In that case, the agent would want to turn around in r (in [49], Lee refers to the turnaround as a *spinturn*).

Given a plan $\pi = (\langle r_1, \tau_1 \rangle, \ldots, \langle r_m, \tau_m \rangle)$, the following constraint forbids spinturns:

$$\forall i \in \{1, \ldots, m-2\} : r_i \neq r_{i+2} \tag{3.8}$$

Equation 3.8 expresses that a resource may not be revisited after only one intermediate plan step. Note that the definition of $G_R = (R, E_R)$ ensures that a resource cannot be visited in two consecutive plan steps, as there is no connection in E_R from a resource to itself, i.e., G_R is loop free.

Acyclic plans

To find an optimal route plan, it is sometimes necessary for an agent to visit a resource more than once, i.e., to make a *cyclic* route plan, as illustrated in the following example.

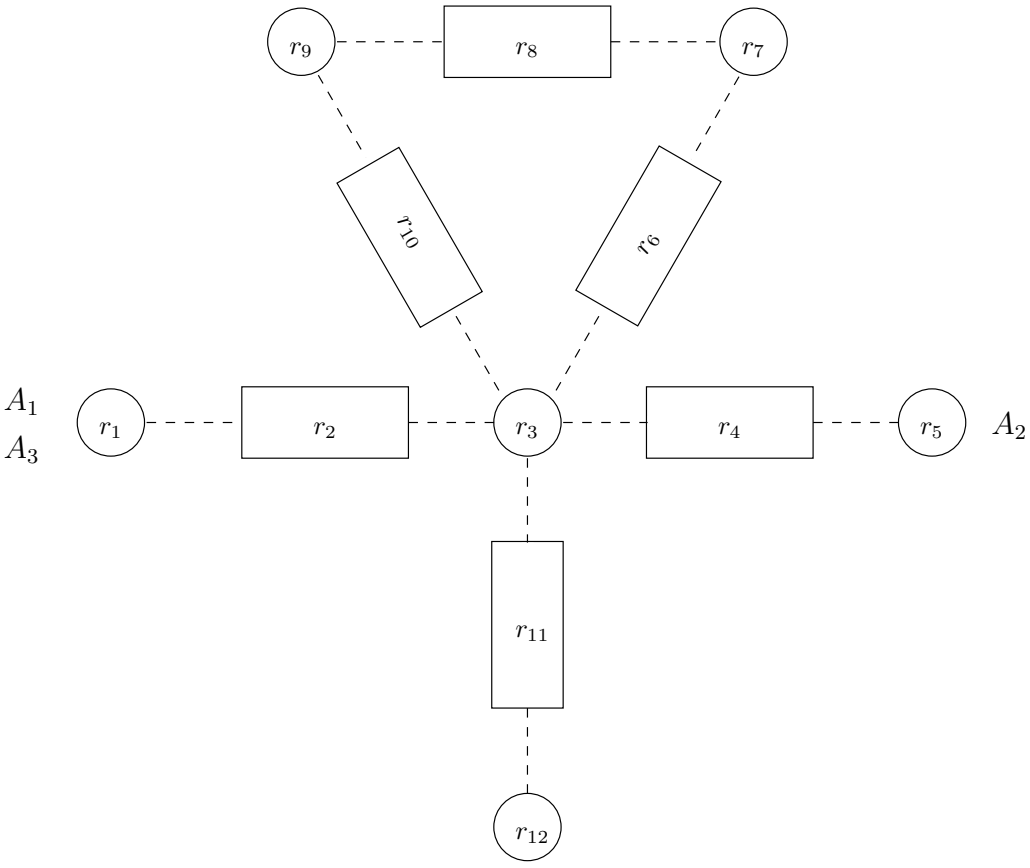


Figure 3.7: Resource graph where circles are intersections, rectangles are lanes, and dashed lines represent the successor relation E_R. There are three agents with respective start-destination locations: $A_1 : (r_1, r_5)$, $A_2 : (r_5, r_{12})$, and $A_3 : (r_1, r_{12})$.

Example 3.2.2. *Consider the resource graph of figure 3.7, and suppose that lane resources have a minimum travel time of 2, intersection resources have a minimum travel time of 1, and all resources have a capacity of 1. There are three agents: agent A_1 starts at resource r_1 and wants to go to r_5, A_2 with start location r_5 and destination r_{12}, and A_3 with start r_1 and destination r_{12}. Suppose that A_2 and A_3 have made the following plans:*

$$\pi_2 = \langle r_5, [4,5)\rangle, \langle r_4, [5,7)\rangle, \langle r_3, [7,8)\rangle, \langle r_{11}, [8,10)\rangle, \langle r_{12}, [10,11)\rangle$$
$$\pi_3 = \langle r_1, [6,7)\rangle, \langle r_2, [7,14)\rangle, \langle r_3, [14,15)\rangle, \langle r_{11}, [15,17)\rangle, \langle r_{12}, [17,18)\rangle$$

Note that for the purposes of this example, we have made agent A_3's traversal of r_2 longer than the minimum travel time.

Now A_1 should find a plan from r_1 to r_5, and suppose that it may not invalidate the plans of A_2 and A_3. Furthermore, suppose that plans should respect constraint 3.1, which means that no simultaneous resource exchanges may occur, but that constraint 3.7 need not hold, so A_1 can enter r_1 whenever it wants. The shortest plan from r_1 to r_5 is the following plan:

$$\pi_1 = \langle r_1, [0,1)\rangle, \langle r_2, [1,3)\rangle, \langle r_3, [3,4)\rangle, \langle r_4, [4,6)\rangle, \langle r_5, [6,7)\rangle$$

However, plan π_1 conflicts with agent A_2's plan π_2 on resource r_4. Hence, A_1 should wait for A_2 to clear intersection r_3. Agent A_1 could try the following plan, in which it waits in r_2 for agent A_2 to pass:

$$\pi_1' = \langle r_1, [0,1)\rangle, \langle r_2, [1,8)\rangle, \langle r_3, [8,9)\rangle, \langle r_4, [9,11)\rangle, \langle r_5, [11,12)\rangle$$

This plan, however, conflicts with the plan of A_3, which will enter r_2 at time 7. Agent A_1 is now left with two options: drive behind A_3, or 'wait' for A_2 to pass by making a cycle $r_3 - r_6 - r_7 - r_8 - r_9 - r_{10} - r_3$. In this example, the latter is the faster option, and it leads to A_1's optimal plan:

$$\pi_1^* = \langle r_1, [0,1)\rangle, \langle r_2, [1,3)\rangle, \langle r_3, [3,4)\rangle, \langle r_6, [4,6)\rangle, \langle r_7, [6,7)\rangle, \langle r_8, [7,9)\rangle,$$
$$\langle r_9, [9,10)\rangle, \langle r_{10}, [10,12)\rangle, \langle r_3, [12,13)\rangle, \langle r_4, [13,15)\rangle, \langle r_5, [15,16)\rangle$$

A disadvantage of a cyclic plan is that it often requires the agent to travel greater distances. In example 3.2.2, the optimal plan π_1^* covers more than twice the distance of the shortest path (which corresponds to the plan of driving behind agent A_3). In times when fuel consumption should be moderated, a quick but long plan may be undesirable. Forbidding cyclic plans can be a heuristic measure to stimulate shorter plans, since our definition of plan cost (definition 3.1.2) considers only time. We can forbid cyclic plans with the following constraint:

$$(\langle r, \tau\rangle \in \pi \wedge \langle r, \tau'\rangle \in \pi) \rightarrow \tau = \tau' \tag{3.9}$$

3.3 Problem complexity

In this section, we will prove that multi-agent route planning is strongly NP-hard[7], even if no additional constraints are taken into account. The proof consists of a reduction from the strongly NP-hard Flow Shop Problem with Blocking. If we consider the multi-agent routing problem under the additional constraints 3.7 (an agent must occupy a resource at all times), and 3.6 (an agent must keep an empty resource between itself and other agents), then the problem is even PSPACE-complete, subject to an assumption discussed in section 3.3.1.

First, we will define multi-agent route planning as the problem of finding a set of conflict-free agent plans with a minimum plan cost. A set of agent plans is conflict free if the resource load of any resource never exceeds its capacity, and if all the constraints from the model configuration are satisfied. A model configuration M can contain any of the constraints from section 3.2, from constraint 3.7 to constraint 3.9.

Definition 3.3.1 (Multi-Agent Route Planning (MARP)). *Given a resource graph $G_R = (R, E_R)$, a model configuration M, a set of agents $\mathcal{A}$ and for each agent $A_i \in \mathcal{A}$ a start location $r \in R$ and a destination location $r' \in R$, find a set $\Pi = \{\pi_1, \ldots, \pi_{|\mathcal{A}|}\}$ of agent plans, such that:*

1. $\forall t \forall r \in R : \lambda(r,t) \leq cap(r)$ *(resource load $\leq$ resource capacity),*
2. Π *satisfies the constraints in M,*
3. *the multi-agent plan cost $c(\Pi)$ is minimized.*

The multi-agent route planning (MARP) problem bears similarities to various *shop scheduling* problems. A shop scheduling problem consists of a set of *jobs*, in turn consisting of a set or sequence of *operations*, and each operation has to be processed on a particular *machine*. The relation to the multi-agent route planning problem is that jobs can be viewed as agents, and the machines as the resources of the infrastructure. A difference between many shop scheduling problems such as the Job Shop Scheduling Problem (see [26]) and multi-agent route planning is that the former specifies exactly which machines (resources) a job will make use of, whereas in multi-agent route planning, an agent is free to choose its route from start to destination(s). Below, we will transform a shop scheduling problem to the MARP problem in such a way, that the resulting resource graph leaves the agents no freedom to visit resources that are not part of the job description. In the Flow Shop Scheduling Problem, all operations of all jobs follow the same sequence of machines. Hence, we can create a resource graph for the MARP problem that consists of a single chain of resources.

An additional difference between 'regular' shop scheduling problems and multi-agent route planning is that an agent must occupy a resource at all times[8], whereas a job need not occupy a machine between the processing of one operation and the next. Therefore, we consider flow shop scheduling under the *blocking* constraint. The blocking constraint

[7] A problem is strongly NP-hard if it is still NP-hard if all numerical parameters are encoded in unary notation.

[8] Of course, without constraint 3.7, an agent need not occupy a resource before entering its start location or after reaching its destination.

specifies that a job, having completed processing on a machine, remains on the machine until the machine for its next operation becomes available. The flowshop scheduling problem with blocking was proved strongly NP-hard by Hall and Sriskandarajah [31], for problem instances with three or more machines. We will use the flowshop problem with blocking to prove the NP-hardness of the MARP problem.

Definition 3.3.2. *The* Minimum Flow Shop Scheduling Problem with Blocking *is given by:*

Instance *An ordered set* $P = (P_1, \ldots, P_m)$ *of processors, a set* $J = \{j_1, \ldots, j_n\}$ *of jobs, each job* $j_i \in J$ *consisting of a sequence* $o_{i,1}, \ldots, o_{i,m}$ *of operations, such that the* k^{th} *operation* $o_{i,k}$ *of job* j_i *must be processed on processor* P_k*, and for each operation* $o_{i,k}$ *there is a minimum processing time* $pt(i,k)$*.*

Solution *A schedule for the set of jobs* J*, i.e., a function* $s : \{o_{i,k} \in j_i \,|\, (j_i \in J) \wedge 1 \leq k \leq m\} \to T$ *that specifies the starting time of each operation on each processor, subject to the following constraints:*

1. *in the interval* $[s(i,k), s(i,k+1))$*,* $1 \leq i \leq n$ *and* $1 \leq k \leq m-1$*, processor* P_k *may only work on operation* $o_{i,k}$*,*
2. *operation* $o_{i,k+1}$*,* $1 \leq i \leq n$ *and* $1 \leq k \leq m-1$*, may not start before* $s(i,k) + pt(i,k)$*.*

Measure *The objective is to minimize the completion time of the schedule, i.e., to minimize* $\max_{i \in \{1,\ldots,n\}} s(i,m) + pt(i,m)$*.*

Proposition 3.3.3. *The multi-agent route planning problem with* $M = \varnothing$ *(no additional constraints) and makespan cost measure is strongly NP-hard.*

Proof. We will prove the strong NP-hardness of the multi-agent routing problem using a reduction from the minimum flowshop scheduling problem with blocking (definition 3.3.2) which was proved to be strongly NP-complete in [31] for a flow shop with three machines or more. Given an instance (P, J, pt), with $m = |P|$ and $n = |J|$, of the flowshop problem, we define the following transformation to the multi-agent routing problem.

We create a unit-capacity resource r_i for each processor $P_i \in P$, and we create a directed edge (r_i, r_{i+1}), $1 \leq i \leq m-1$, between every pair of consecutive resources. With each job $j_i \in J$ we associate an agent A_i with start location r_1 and destination location r_m. The agent travel time function at can be directly derived from the minimum processing times of the operations, i.e., for all $1 \leq i \leq n$ and $1 \leq k \leq m$, we have $at(A_i, r_k) = pt(i,k)$.

We will now show that if an instance of the flowshop problem with blocking has a schedule with completion time K, then such a schedule directly corresponds to a multi-agent route plan with completion time K: for all $1 \leq i \leq n$, if we have a flow shop schedule $(s(i,1), s(i,2), \ldots, s(i,m))$, then the plan for agent A_i is given by

$$\pi_i = (\langle r_1, [s(i,1), s(i,2))\rangle, \langle r_2, [s(i,2), s(i,3))\rangle, \ldots, \langle r_m, [s(i,m), s(i,m) + at(A_i, r_m))\rangle)$$

For the set of agent route plans, it is easy to verify that no resource is ever occupied by more than one agent at a time, because (i) in a solution of the flowshop problem, a

processor only operates on one job at a time, and (*ii*) because of the *blocking constraint* from the flowshop problem: if operation $o_{i,k}$ has been processed on processor P_k, then no other operation can start on P_k until the next operation of job j_i has started on the next processor P_{k+1}. It is also easy to verify that the makespan of the multi-agent route plan is equal to K.

We will now show that if the transformed multi-agent routing instance has a solution with makespan K, then there also exists a valid schedule for the original flowshop instance of size K. Again there is a direct correspondence between the solutions of the two problems. The start time $s(i,k)$ of operation $o_{i,k}$ is simply the start time of the k^{th} plan step of agent A_i. To understand that the obtained schedule is a valid flowshop schedule, we should note the following:

- As the resource graph is a directed chain of resources from r_1 to r_m, all agent plans follow this sequence of resources; no alternative routes are possible. Hence, the resource of the k^{th} plan step in any agent plan corresponds to processor P_k. The requirement that the k^{th} operation of any job is on processor P_k is thereby satisfied.

- As all resources have capacity one, there is at most one agent on any resource at any time in the multi-agent plan. Hence, in the corresponding flowshop schedule there is also at most one operation on any processor at one time.

- An agent A_i cannot exit a resource r_k earlier than its entry time plus the minimum travel time $at(A_i, r_k)$. Hence, no operation $o_{i,k}$ in the flowshop schedule can start before the preceding operation $o_{i,k-1}$ has finished.

We have shown that a solution for the flowshop problem corresponds to a solution for the routing problem and vice versa. Hence, the Flowshop Problem with Blocking can be reduced to the Multi-Agent Route Planning problem, which proves that MARP is strongly NP-hard. □

3.3.1 PSPACE-complete routing

If we consider the MARP problem under the additional constraints 3.7 (an agent must occupy a resource at all times) and 3.6 (agents must maintain one empty resource as separation), then it seems we can prove a stronger complexity result. Constraint 3.6 was inspired by the Airport planning problem from the 2004 edition of the International Planning Competition (more on the airport problem can be found in [91]). In Airport, there is a set of aircraft each with a start location and a destination location, and while taxiing the aircraft need to maintain a separation of one empty resource because of the powerful exhausts of jet engines. In [35], Helmert proved that deciding whether a solution exists for an Airport instance is PSPACE-complete. Hence, finding the shortest multi-aircraft taxi plan is also PSPACE-complete.

The proof in [35] consists of a reduction from the PSPACE-complete Sliding Tokens problem (see section 3.2). Within the proof, however, it is clear that the author makes use of a slightly different problem that we have defined below as the Distinguishable

Sliding Tokens problem. Helmert states that "*... only simple adjustments to the hardness proofs* [from Hearn and Demaine [34]] *are needed for the modified version*". However, what these simple adjustments are has not been reconstructed yet either by us or by the author[9].

In the sliding tokens problem (definition 3.2.1), we cannot distinguish one token from another. In routing, on the other hand, agents are unique, and we are only interested in 'goal configurations' where specific agents reach specific locations. Hence, we need to define the problem where each token has its own destination location.

Definition 3.3.4 (Distinguishable Sliding Tokens (DST))**.** *Let $G = (V, E)$ be a graph, and let $T = \{t_1, \ldots, t_k\}$ be a set of tokens. A* valid configuration *$\pi : T \to V$ is an assignment of tokens to vertices, such that no two adjacent vertices contain a token. A* move *consists of sliding a token to an adjacent, unoccupied vertex, such that the resulting configuration is also valid. The problem is: can we reach a goal configuration π_* from an initial configuration π_0?*

Assumption 3.3.5. *DST is PSPACE-complete.*

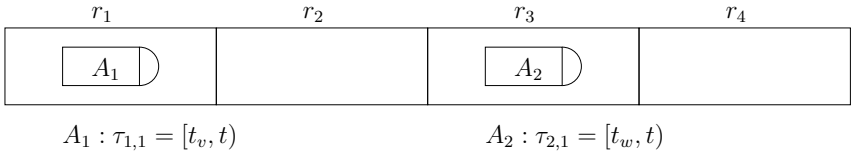
Proposition 3.3.6. *The multi-agent route planning problem with constraints 3.7 and 3.6 is PSPACE-complete under assumption 3.3.5.*

Proof. We will reduce DST to the multi-agent routing problem of definition 3.3.1. Given a DST graph $G = (V, E)$ and a set of tokens T, we create a resource r_v for every $v \in V$, and if $\{v, w\} \in E$, then we add the pairs (r_v, r_w) and (r_w, r_v) to E_R. Capacities and travel times for resources can be set to one. The set of agents $\mathcal{A}$ corresponds to the set of tokens T.

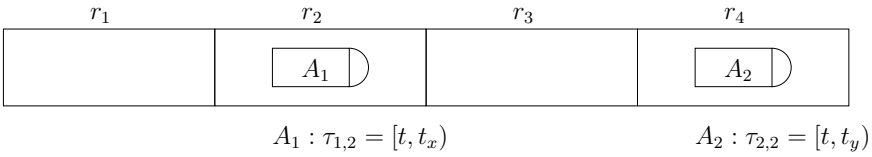
From a solution of the DST puzzle we can obtain a solution to the MARP problem. A solution to the DST puzzle consists of a sequence of token moves, such that a token never moves to a vertex that has a non-empty neighbour. Each move made for token t_i corresponds to a plan step for agent A_i. Since we are proving PSPACE-completeness of plan *existence* (rather than finding the shortest-time plan), we can construct a sequential multi-agent plan, in the sense that there is always only one agent performing a plan step. Hence, the plan step corresponding to token move $n+1$ in the DST puzzle can start after the plan step corresponding to move n (since travel times are 1, the agent should also be ready to move at that time).

From a solution of the routing problem, we can obtain a solution to the DST puzzle by transforming the parallel multi-agent solution to a sequential solution using algorithm 2. Constraint 3.6 ensures that there is no point in time t when two agents occupy adjacent resources. It is not hard to see that this holds if no resource transitions occur at t. To see that it also holds if there are resource transitions at t, consider figure 3.8. Figure 3.8 depicts resource transitions that are *not* allowed by constraint 3.6: given the occupation time $\tau_{1,2} = [t, t_x)$ of r_2 by A_1 and the occupation time $\tau_{2,1} = [t_w, t)$ of r_3 by A_2, we have $\tau_{1,2}^+ \cap \tau_{2,1}^+ = t$, which is not allowed, because r_2 is adjacent to r_3. Hence, under constraint 3.6, agent A_1 is only allowed to enter r_2 until after A_2 has left r_3.

[9] In personal communication with the author, he admitted that he could no longer recall which adjustments were required for the proof.



(a) Situation just before transition time t.



(b) Situation just after transition time t.

Figure 3.8: Two simultaneous resource transitions that are not allowed under constraint 3.6: A_2 should leave r_3 before A_1 is allowed to enter r_2.

To prove that MARP is in PSPACE, we can use the same line of reasoning that was used to demonstrate PSPACE membership of the Sliding Tokens puzzle in [34]. The state of the system, specifying the locations of all of the agents, can be encoded in a linear number of bits. Also, we can compute all possible agent moves from a given state in polynomial time. We can therefore nondeterministically traverse the search space, at each step choosing a move to make, maintaining the current state but not any of the previously visited states. Due to Savitch's Theorem [81], this NPSPACE algorithm can be converted into a PSPACE algorithm. □

3.4 The prioritized approach to MARP

Given the complexity results presented in this chapter, it is it unlikely that we can find optimal multi-agent route plans for problem instances of interesting sizes. In fact, in the previous chapter we discussed some centralized approaches from the literature, and saw that Desaulniers et al. [16] were able to solve instances with up to 4 AGVs. Other discouraging results were reported by Trug et al. [91] who investigated the application of general purpose planning systems to the airport domain from the planning competition, and found that many planners failed to find a plan for simple airports with less than 10 aircraft.

In the next chapter, we therefore present a prioritized approach to multi-agent route planning, in which agents plan in sequence, and the agent that is n^{th} to plan must respect the plans of the first $n-1$ agents. From a theoretical point of view, the prioritized approach has two disadvantages compared to the centralized approach. First, if we consider multi-agent routing with a model configuration that includes constraint 3.7, which states that an agent must occupy a resource at all times, then the prioritized approach to route planning is not complete. If two agents have opposite start destination locations (e.g., there is an

agent A_1 with start-destination pair (r, r') and an agent A_2 with start-destination pair (r', r)), then it is not possible for one agent to complete its route plan before the other has started planning, because it is unknown when the other agent will leave its start location. If constraint 3.7 does not hold — which means that agents can enter and leave the infrastructure — then the prioritized approach always finds a plan for each agent. The second disadvantage is that the cost of the multi-agent plan is not guaranteed to be a constant multiple of the cost of an optimal plan, as the next example demonstrates.

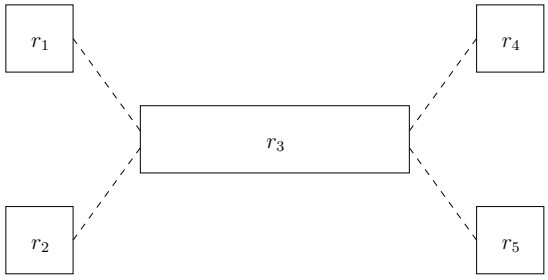


Figure 3.9: An infrastructure with potential bottleneck resource r_3.

Example 3.4.1. *Consider the infrastructure of Figure 3.9, and suppose that the set of agents is divided into two groups $\mathcal{A}_1$ and $\mathcal{A}_2$. All agents in $\mathcal{A}_1$ have their start location in r_1 and have r_4 as their destination location, whereas all agents in $\mathcal{A}_2$ start in r_5 and have r_2 as their destination location. All resources are assumed to have infinite capacity and are bidirectional, but traffic is only allowed in one direction at the same time. We finally assume that $tt(r_3) > tt(r_1) + tt(r_2) + tt(r_4) + tt(r_5)$.*

The optimal solution to the multi-agent routing problem is to let one group of agents plan before the other. In case the group $\mathcal{A}_1$ may plan first, then the last agent of $\mathcal{A}_1$ will arrive at resource r_4 at time

$$t_1 = tt(r_1) + tt(r_3)$$

At time t_1, agents from $\mathcal{A}_2$ will be able to enter r_3, and they will arrive at resource r_2 at time

$$t_2 = t_1 + tt(r_3)$$

If planning alternates between groups, then every agent has to wait for the previous agent to clear resource r_3. Consequently, the final agent will arrive at its destination at time

$$t_3 > |\mathcal{A}| \cdot tt(r_3)$$

Since $tt(r_3)$ is the most significant contribution to the travel times, the alternating solution is almost $\frac{|\mathcal{A}|}{2}$ times as bad as the optimal solution.

With regard to plan quality, the prioritized approach does allow us to find the optimal route plan for a single agent, given a set of plans from previous agents. In addition, the

prioritized approach is much faster, and in the next chapter we will present a single-agent route planning algorithm with a worst-case complexity that improves over the existing single-agent algorithm from Kim and Tanchoco [41]. Finally, we should note that the existence of the above $O(|\mathcal{A}|)$ example does not imply that the prioritized approach cannot do well in practice. As we shall see in our experiments (chapter 6), there are many types of infrastructures where the cost of the global plan is in fact quite low.

Chapter 4

Sequential Route Planning

In the previous chapter we saw that the multi-agent route planning problem is NP-hard, which means that finding optimal multi-agent route plans for realistically-sized instances will probably require too much computation time. In this chapter we will demonstrate that finding an optimal conflict-free route plan for a *single* agent can be done in polynomial time. The idea is that the plans of previous agents are assumed immutable: they have placed reservations (time slots) on resources corresponding to their passage over the infrastructure. Our approach is therefore a sequential one: agent n (i.e., the n^{th} agent to make a plan) must plan 'around' the plans of agents $1, \ldots, n-1$, but it is not concerned with the movements of agent $n+1$. Given the plans of agents $1, \ldots, n-1$, agent n can determine for each resource the time windows in which the resource can be entered without introducing a conflict with any other agent. We can construct a graph from the set of all *free time windows* on all resources, and an agent can use this graph to find a shortest-time, conflict-free route plan.

Previous research on route planning with free time windows (e.g. [41, 33]) focussed on finding the shortest route between a start location and a destination location. As an extension of our main algorithm (which also takes a single start location and a single destination location), we also present an algorithm that finds the shortest route along a (fixed) sequence of locations. The straightforward approach to the *multi-stage* route planning problem is to make route plans between successive locations, and to glue these plans together. However, as a result of the reservations in the system, this approach does not always yield a solution.

We will start this chapter by formalizing the notion of a free time window, in section 4.1. In chapter 3 we identified a number of constraints that can be imposed on the plans of the agents, for example if we wish to prevent overtaking. In section 4.1, we will show how each of these additional constraints can be encoded into free time windows. The idea of our approach is thus that, if agents only make use of free time windows, then all conflicts with other agents can be avoided. In section 4.2, we will discuss single-agent route planning when there is only a single destination resource. We will present a new single-agent algorithm that finds a shortest-time route using the graph of free time windows. In section 4.3, we will present a single-agent algorithm for the multi-stage route

planning problem. The algorithm is very similar to the single-destination algorithm, but it operates on a free time window graph that has an additional layer for each destination location.

4.1 Reservations and free time windows

Given a set of agent plans, the resource load of a resource tells us exactly when a resource is free to be used by other agents. During a time interval when the resource load is at least one less than the capacity, another agent may enter the resource. Figure 4.1 depicts the resource load for some resource r in the interval $[0, 10)$. The capacity of r is 3, and we see that during the intervals $[4, 6)$ and $[7, 8)$ this capacity is fully utilized. During the intervals $\tau = [0, 4)$, $\tau' = [6, 7)$, and $\tau'' = [8, 10)$ there is room for (at least) one more agent. However, if we assume that the minimum travel time of r is 2, then the interval $\tau' = [6, 7)$ is too short to be of use to any agent. Therefore, the only *free time windows* are τ and τ''.

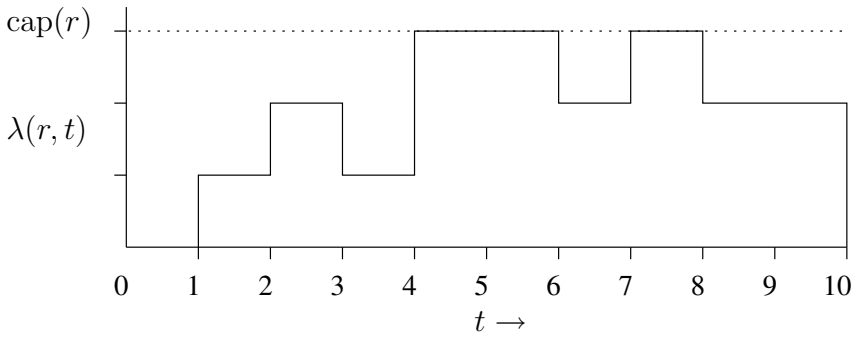


Figure 4.1: Resource load for resource r with capacity 3.

Definition 4.1.1 (Free time window)**.** *Given a resource-load function λ, a free time window on resource r is a maximal interval $f = [t_1, t_2)$ such that:*

1. $\forall t \in f : \lambda(r, t) < cap(r)$,
2. $(t_2 - t_1) \geq tt(r)$.

The above definition states that for an interval to be a free time window, there should not only be sufficient capacity at any moment during that interval (condition 1), but it should also be long enough for an agent to traverse the resource (condition 2). The set of all free time windows $F = (F_1, \ldots, F_{|R|})$ is partitioned into $|R|$ sets: one set of free time windows F_i for every resource $r_i \in R$. Note that the set of free time windows F_i on resource r_i is a vector $(f_{i,1}, \ldots, f_{i,m})$ of disjoint intervals such that for all $j \in \{1, \ldots, m-1\}$, $f_{i,j}$ precedes $f_{i,j+1}$.

Within a free time window, an agent must enter a resource, traverse it, and exit the resource. Because of the (non-zero) minimum travel time of a resource, an agent cannot enter a resource right at the end of a free time window, and it cannot exit the window at the start of one. We therefore define for every free time window f an *entry window* $\tau_{\text{entry}}(f)$ and an *exit window* $\tau_{\text{exit}}(f)$. The sizes of the entry and exit windows of a free time window $f = [t_1, t_2)$ on resource r are constrained by the minimum travel time of the resource:

$$\tau_{\text{entry}}(f) = [t_1, t_2 - tt(r)) \tag{4.1}$$

$$\tau_{\text{exit}}(f) = [t_1 + tt(r), t_2) \tag{4.2}$$

An agent that wants to go from resource r to resource r' should find a free time window for both of these resources. By definition 3.1.1 of an agent plan, the exit time out of r should be equal to the entry time into r'. Hence, for a free time window f' on r' to be *reachable* from free time window f on r, the *entry* window of f' should overlap with the *exit* window of f.

Definition 4.1.2 (Free time window graph)**.** *The* free time window graph *is a directed graph $G_F = (F, E_F)$, where the vertices are the set of free time windows, $F = \bigcup_{i=1}^{|R|} F_i$, and E_F is the set of edges specifying the* reachability *between free time windows. Given a free time window f on resource r, and a free time window f' on resource r', it holds that $(f, f') \in E_F$ if:*

1. $(r, r') \in E_R$,
2. $\tau_{\text{exit}}(f) \cap \tau_{\text{entry}}(f') \neq \varnothing$.

The free time window graph specifies exactly how an agent's possible movements are restricted by a set of other-agent plans. The free time windows represent the times at which resources may be visited by an agent, while the edges of the free time window graph specify the reachability between free time windows. In section 4.2, we will show how an agent can plan its optimal route by performing a search through the free time window graph.

In definition 4.1.1, we defined free time windows on the basis of the resource load: an agent may enter a resource if there is enough capacity left. In chapter 3, however, we saw that there can be other constraints that a set of agent plans must satisfy. We will now discuss how and if these constraints can be 'encoded' into the definition of a free time window.

4.1.1 Free time window and model configuration

Recall from chapter 3 that the model configuration consists of the set of constraints that a set of agent plans must satisfy (in addition to the resource capacity constraint). In this section, we discuss how the definition of a free time window should be extended (or not) for each of the constraints discussed in section 3.2.

Simultaneous resource exchanges

In chapter 3 we discussed the problem of agents exchanging resources at exactly the same point in time. To prevent simultaneous resource exchanges, we formulated a constraint to the following effect: constraint 3.3 states that if there is a cycle of agents exchanging resources, then at least one agent in the cycle must move to a resource where there is capacity left over. To forbid simultaneous exchanges in which all agents move to full resources, we must make a small change to the definition of the reachability between free time windows.

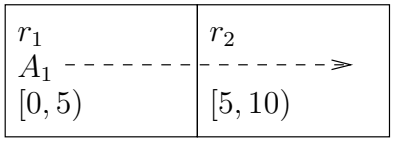


Figure 4.2: Agent A_1 with plan $\pi = \langle r_1, [0, 5)\rangle, \langle r_2, [5, 10)\rangle$

Using figure 4.2, we will show that if an agent makes a simultaneous exchange, then it makes use of two free time windows that *meet* (i.e., the end of one free time window is equal to the start of the next free time window). In figure 4.2, there is an agent A_1 that plans to travel from r_1 to r_2. A simultaneous exchange would occur if (i) there is another agent A_2 that wants to go from r_2 to r_1 (ii) agent A_1 *has* to leave r_1 before A_2 can enter it, and A_2 *has* to leave r_2 before A_1 can enter it (i.e., in the absence of other agents, both r_1 and r_2 should have unit capacity).

If we assume that A_1 is the first to make a plan, then the free time windows for agent A_2 are: on r_1, there will be one free time window $f_{1,1} = [5, \infty)$, and on r_2 there are two free time windows $f_{2,1} = [0, 5)$ and $f_{2,2} = [10, \infty)$. Agent A_2's simultaneous-exchange plan is given by: $\pi' = \langle r_2, [0, 5)\rangle, \langle r_1, [5, 10)\rangle$, which makes use of the free time windows $f_{2,1}$ and $f_{1,1}$. Clearly, the end of $f_{2,1}$ is equal to the start of $f_{1,1}$, and window $f_{2,1}$ meets window $f_{1,1}$. To ensure that no simultaneous resource exchanges can occur that violate constraint 3.3, we only need to require that for a free time window f' to be reachable from a free time window f, the intersection between $\tau_{\text{exit}}(f)$ and $\tau_{\text{entry}}(f')$ is *nonzero*.

Definition 4.1.3 (Serializable free time window graph)**.** *The* serializable free time window graph *is a directed graph $G_F = (F, E_F)$, where the vertices are the set of free time windows, $F = \bigcup_{i=1}^{|R|} F_i$, and E_F is the set of edges specifying the* reachability *between free time windows. Given a free time window f on resource r, and a free time window f' on resource r', it holds that $(f, f') \in E_F$ if:*

1. $(r, r') \in E_R$,
2. $|\tau_{\text{exit}}(f) \cap \tau_{\text{entry}}(f')| > 0$.

Simultaneous bidirectional lane traversal

We can prevent bidirectional lane traversal by maintaining two sets of free time windows for every (bidirectional) resource: one set for the 'upstream' direction and one for the

'downstream' direction. We can infer the direction of an agent plan step from the preceding plan step. Suppose we have a lane resource r that is connected to intersections v and w. If an agent plan contains the subsequence (v, r), then it will exit r at w[1] (see figure 4.3), and we can arbitrarily designate this the upstream direction; an agent plan with the subsequence (w, r) travels in the downstream direction.

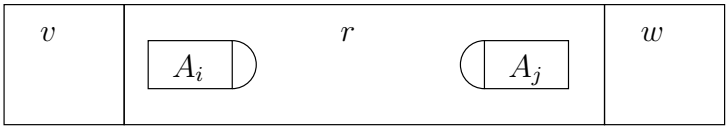


Figure 4.3: Agent A_1 will enter lane resource r from intersection v, and must therefore exit r via intersection w.

We introduce the augmented resource load function $\lambda^a : R \times R \times T$, such that $\lambda(r', r, t)$ specifies the number of agents that are on resource r at time t, and have come directly from resource r'.

Definition 4.1.4 (Directed free time window)**.** *Given an augmented resource-load function λ^a, and two intersection resources v and w connected by lane resource r, a free time window on resource r is the largest interval $f = [t_1, t_2)$ such that:*

1. $\forall t \in f : \lambda^a(v, r, t) < cap(r)$,
2. $\forall t \in f : \lambda^a(w, r, t) = 0$,
3. $(t_2 - t_1) \geq tt(r)$.

In definition 4.1.4, the free time window f is implicitly defined for the direction from v to r. To obtain the free time window in the other direction, we can exchange v and w.

Preventing overtaking

If we consider routing under constraint 3.5, then agents are not allowed to overtake each other. Hence, if an agent enters a resource before another, it should also exit the resource first. For this constraint, we need to redefine a free time window in terms of the plan steps of the leading and trailing agents. Let $\tau_{\text{lead}}(r, t)$ be the plan step of the first agent to enter resource r *before* time t, and let $\tau_{\text{trail}}(r, t)$ be the plan step of the first agent to enter resource r *after* time t. We will also assume that there is a minimum separation time δ between two successive agents. Hence, if an agent traverses a resource in the interval $[t_1, t_2)$, then no agent may enter the resource in $(t_1 - \delta, t_1 + \delta)$, and no agent may exit the resource during $(t_2 - \delta, t_2 + \delta)$. The following revised free time window definition prevents agents from overtaking each other.

[1] We ignore the possibility of spinturns here, but we can easily extend the results of this section to take them into account.

Definition 4.1.5 (Free time window no overtaking)**.** *Given a resource-load function λ, an intended entry time t^*, a leading-agent interval $\tau_{lead}(r, t^*) = [t_1, t_2)$, a trailing-agent interval $\tau_{trail}(r, t^*) = [t_3, t_4)$, and a minimum separation time δ, a free time window on resource r that does not allow overtaking is an interval $f = [t, t')$ such that:*

1. $t = t_1 + \delta$,
2. $t' = t_4 - \delta$,
3. $\forall t_i \in [t, t') : \lambda(r, t_i) < cap(r)$,
4. $(t' - t) \geq tt(r)$.

The entry and exit windows of f are specified by

$$\tau_{\text{entry}}(f) = [t, \min\{t' - tt(r), t_3 - \delta\}) \tag{4.3}$$
$$\tau_{\text{exit}}(f) = [\max\{t + tt(r), t_2 + \delta\}, t') \tag{4.4}$$

In case there is no leading vehicle, we can set $\tau_{\text{lead}}(r, t) = (-\delta, -\delta)$, and if there is no trailing vehicle, we can set $\tau_{\text{trail}}(r, t) = (\infty, \infty)$.

Free time windows with separation

Constraint 3.6 states that if an agent makes use of a resource, then all adjacent resources must be empty. This implies that if a resource is free to be used during a certain interval τ, then the resource load in all adjacent resources must be zero during τ. To ensure that this constraint holds, we can use the following revised free time window definition.

Definition 4.1.6 (Free time window with separation)**.** *Given a resource-load function λ, a free time window with separation on resource r is the largest interval $f = [t_1, t_2)$ such that:*

1. $\forall t \in f : \lambda(r, t) = 0$,
2. $\forall (r, r') \in E_R, \forall t \in f : \lambda(r', t) = 0$,
3. $\forall (r', r) \in E_R, \forall t \in f : \lambda(r', t) = 0$,
4. $(t_2 - t_1) \geq tt(r)$.

Definition 4.1.6 not only specifies that all connected resources must be empty, but the resource itself must also be empty for there to be a free time window. Recall from section 3.2.2 that if an agent's neighbouring resources must all be empty, then we assume that each resource can hold at most one agent.

Agent entry and exit into infrastructure

The first constraint from section 3.2 (constraint 3.7) states that an agent must occupy a resource at all times. This is not a constraint between agents, and it does not affect the definition of a free time window. It does, however, have implications for a sequential approach that makes use of free time windows. Suppose that we have two agents, such that the start location of one is the destination location of the other (see figure 4.4), and these locations have capacity one. One implementation of constraint 3.7 in route planning with free time windows is for each agent to reserve its start location for the interval $[0, \infty)$: this way, an agent can be assured that no other agent will require it to leave its start location until it has made its plan. Obviously, this does not allow us to find a solution for the agents in figure 4.4, even though a solution is not difficult to formulate. For example, agent A_1 could first move to r_3, and agent A_2 could first move to r_4. Then, their respective destination resources are both empty, and the agents can continue to their destinations, without encountering each other.

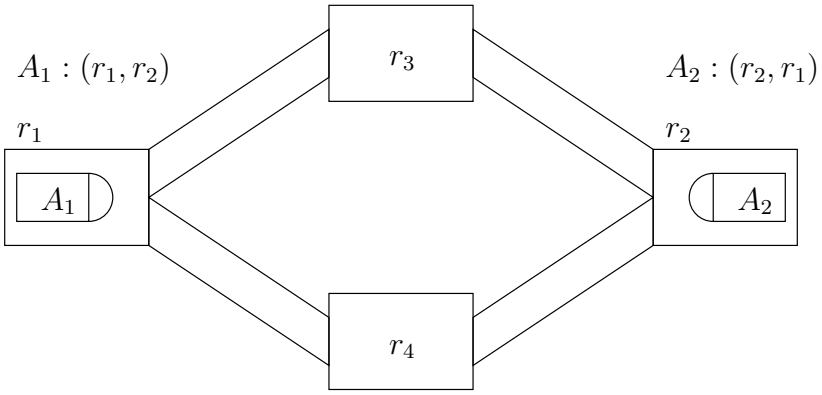


Figure 4.4: Two agents with opposite start and destination locations. Under constraint 3.7, the sequential method does not find a solution.

Acyclic plans and spinturn

The final two constraints of the model configuration, which forbid cyclic plans and spinturns respectively, cannot be encoded in free time windows or in the reachability between free time windows. The reason is that these are constraints on agent plans, and not constraints between agent plans. With regard to the latter type of constraints, these could easily be translated to time intervals when an agent is allowed to visit a resource, and when it is not allowed to do so. The single-agent constraints, however, must be checked during the planning process.

4.2 Route planning from A to B

In this section we will present an algorithm for the (single-agent) route planning problem in which an agent has a start location and a single destination location. If an agent finds itself alone on an infrastructure, then it can find the shortest path in space and time using a classical graph search algorithm like Dijkstra's algorithm [17] or A* [32]. The algorithm we will present later in this section can be seen as an adaptation of A*, so we will briefly discuss this algorithm here.

The A* algorithm maintains a list of partial routes from the start location to the destination location, and in each iteration it expands the most *promising* route to all neighbouring locations. A partial route p is the most promising if the sum of its cost $c(p)$ plus the estimated cost of reaching the destination, $h(p)$, is the lowest among all partial routes. The function $y(p) = c(p) + h(p)$ thus takes into account both actual cost and estimated cost[2]. In route planning, the heuristic function h can be e.g. the straight-line distance from the current location to the destination location.

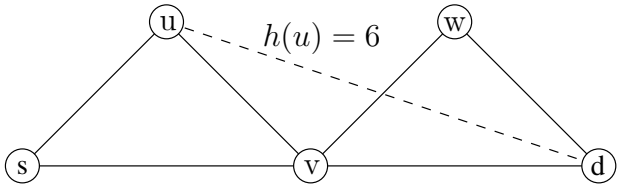


Figure 4.5: An infrastructure where all intersections (circles) have travel time 0, and all lanes have minimum travel time 4. The dashed line represents the straight-line distance between u and d.

Consider figure 4.5 for an example of how A* works. To keep the example simple, we allow travel times of 0 for the intersections, and lanes have minimum travel times of 4. Starting from the start location s, we have the initial plan $\pi_1 = \langle s, 0\rangle$. A* expands this plan to both adjacent intersections u and v, resulting in plans $\pi_2 = (\langle s, 0\rangle, \langle u, 4\rangle)$ and $\pi_3 = (\langle s, 0\rangle, \langle v, 4\rangle)$. In the next iteration, π_3 will be expanded: both plans have cost of 4, but $h(\pi_3) = 4$, whereas $h(\pi_2) = 6$. After the expansion of π_3, there are three partial plans:

$$\begin{aligned}\pi_2 &= (\langle s, 0\rangle, \langle u, 4\rangle); y(\pi_2) = 4 + 6\\ \pi_4 &= (\langle s, 0\rangle, \langle v, 4\rangle, \langle w, 8\rangle); y(\pi_4) = 8 + 4\\ \pi_5 &= (\langle s, 0\rangle, \langle v, 4\rangle, \langle d, 8\rangle); y(\pi_5) = 8 + 0\end{aligned}$$

The plan π_5 has the lowest y-value, so it will be considered for expansion in the next iteration. However, because π_5 is already a route to the destination d, the algorithm will return π_5. The A* algorithm will always return the optimal solution if the heuristic function h is *consistent* [78]. Russell and Norvig define a consistent heuristic as follows:

[2]The traditional A* notation is $f(p) = g(p) + h(p)$, but we will keep to our own notation.

A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$h(n) \leq c(n, a, n') + h(n') \tag{4.5}$$

Russell and Norvig also show that if a heuristic is consistent[3], then A* evaluates nodes with non-decreasing y-values. They give the following short proof.

Suppose that n' is a successor of n; then $c(n') = c(n) + c(n, a, n')$ for some a, and we have

$$y(n') = c(n') + h(n') = c(n) + c(n, a, n') + h(n') \geq c(n) + h(n) = y(n).$$

An important feature of A* is that, when a consistent heuristic is used, a node is expanded at most once. The moment a partial plan to node n is expanded, we have the guarantee that we have found a cheapest plan from a start node s to n. Since a node is expanded to all adjacent nodes, node n need never be expanded again in the search process. This implies the following:

If n is on a shortest path from s to d, then a shortest path from s to n can be expanded to a shortest path from s to d.

In case an agent is not alone on the infrastructure, and there are route plans from other agents that may not be changed, then an agent may need to consider more than just the shortest path to an intermediate location. The next example will show, for the same infrastructure used in figure 4.5, that the shortest route from s to v can *not* be expanded to the shortest route from s to d. In figure 4.6, we assume that all intersections have a minimum travel time of 2, all lanes have a minimum travel time of 4, and all resources have a capacity of 1. Agent A_1 has made the following plan to go from d to v: $\pi_1 = \langle d, [3, 5)\rangle, \langle vd, [5, 9)\rangle, \langle v, [9, 11)\rangle$ whereas agent A_2 wants to go from s to d.

The shortest plan from s to v (v is the intermediate location that all $s - d$ paths must pass through) is $\pi_2 = \langle s, [0, 2)\rangle, \langle sv, [2, 6)\rangle, \langle v, [6, 8)\rangle$. Note that this plan cannot be expanded along the lane vd that leads directly to d, since vd will be occupied by agent A_1 in the interval $[5, 9)$. Also, A_2 cannot wait in intersection v (assuming simultaneous exchanges are not allowed), because A_1 will enter this resource at time 9. Hence, if agent A_2 enters intersection v at time 6 (as in plan π_2), then it must exit v by time 9. This means that π_2 can only be expanded in the direction of w (or back to s). Expanding π_2 leads to the plan π_3:

$$\pi_3 = \langle s, [0, 2)\rangle, \langle sv, [2, 6)\rangle, \langle v, [6, 8)\rangle, \langle vw, [8, 12)\rangle, \langle w, [12, 14)\rangle, \langle wd, [14, 18)\rangle, \langle d, [18, 20)\rangle$$

An optimal plan for A_2 is the following:

$$\pi_4 = \langle s, [0, 2)\rangle, \langle sv, [2, 11)\rangle, \langle v, [11, 13)\rangle, \langle vd, [13, 17)\rangle, \langle d, [17, 19)\rangle$$

[3] A consistent heuristic is also *admissible*, which is to say that it never overestimates the cost of reaching the goal. An admissible heuristic is not necessarily consistent, although according to Russell and Norvig [78] it is uncommon to find an admissible heuristic that is not consistent.

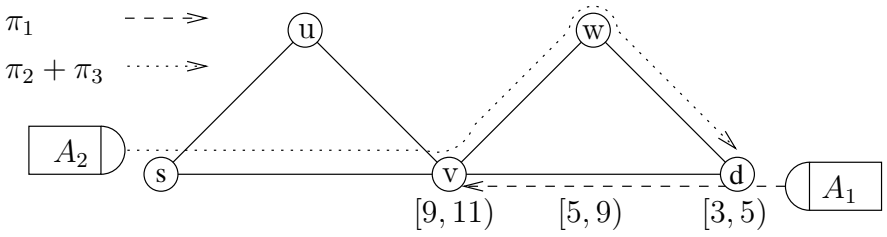


Figure 4.6: An infrastructure where all intersections (circles) have minimum travel time 2, and all lanes have minimum travel time 4. Agent A_1 travels from d to v (and subsequently leaves the infrastructure); agent A_2 wants to go from s to d.

The plan π_4 specifies that agent A_2 should wait in lane sv (or at least traverse it slowly in the interval $[2, 11)$) for A_1 to leave intersection v.

The example of figure 4.6 showed that in multi-agent routing, an agent may need to consider more than one route to a single resource: we saw that the shortest plan π_2 to v arrived at v at time 6, while plan π_4 arrived at v at time 11. The question is whether other plans to v should also be considered, for instance the plan $\pi_5 = \langle s, [0, 2)\rangle, \langle su, [2, 6)\rangle, \langle u, [6, 8)\rangle, \langle uv, [8, 12)\rangle, \langle v, [12, 14)\rangle$ that arrives at v at time 12. In this example, the answer is no. Resource v has two free time windows $f_1 = [0, 9)$ and $f_2 = [11, \infty)$. Plan π_4 makes use of f_2 from the earliest possible time, 11. Any other plan to enter v later than 11 can never lead to a better plan than (the expansion of) π_4. Suppose that plan π' reaches v at some time $t' > 11$. Any plan π'' that results from expanding π' can be simulated using π_4: we simply wait in intersection v from 11 to t', and then follow plan π''. Hence, we need to expand a free time window at most once (or rather, a plan that makes use of a free time window). In other words, if free time window f is on a shortest route from start location s to destination location d, then a shortest route from s to f can be expanded to a shortest route from s to d.

4.2.1 Algorithm specification

In this section, we present an algorithm that performs a search through the free time window graph using an adaptation of the A* algorithm. What results from this search is an agent plan that effectively moves from one free time window on one resource, to another free time window on the next resource; the free time windows associated with successive plan steps must be connected in the free time window reachability relation E_F. In each iteration of the algorithm, the most promising partial route — ending on some resource r with some associated free time window f — is expanded to all free time windows reachable from f.

The existence of a pair $(f, f') \in E_F$ does not guarantee that a plan π, ending in free time window f at some time $t \in \tau_{\text{exit}}(f)$, can be expanded to free time window f'. The reachability of f' from f implies that there exists a time point $t' \in \tau_{\text{exit}}(f) \cap \tau_{\text{entry}}(f')$, not that *all* time points in $\tau_{\text{exit}}(f)$ are also in $\tau_{\text{entry}}(f')$. Hence, when expanding a plan

that ends in window $f = [t_1, t_2)$ at time t to free time window f', we must verify that $[t, t_2) \cap \tau_{\text{entry}}(f') \neq \varnothing$ (see figure 4.7). We will write $\rho(r, t)$ to denote the set of free time windows reachable from resource r at earliest exit time t.

Definition 4.2.1 (Free time window reachability)**.** *Free time window f' is reachable from free time window $f = [t_s, t_e)$ (on resource r) at time t, denoted $f' \in \rho(r, t)$, if*

1. $(f, f') \in E_F$,
2. $t_s \leq t < t_e$,
3. $[t, t_e) \cap \tau_{\text{entry}}(f') \neq \varnothing$.

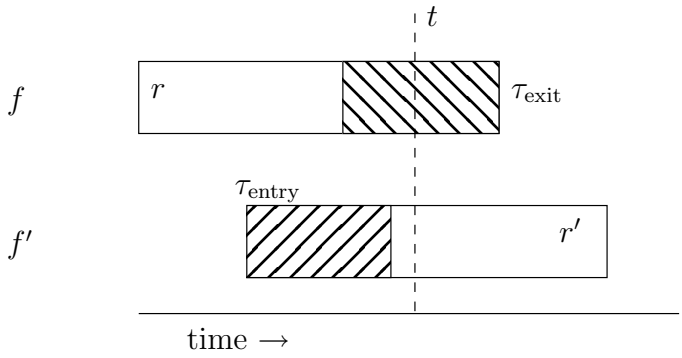


Figure 4.7: Free time window f' is reachable from window f (i.e., $(f, f') \in E_F$), but not from time point t (i.e., $f' \notin \rho(r, t)$).

The most promising partial plan π is the plan with the lowest value of $y(\pi) = c(\pi) + h(\pi)$, where $c(\pi)$ is the cost of the partial plan, and $h(\pi)$ is an estimate of the cost of completing π to the destination. We define the cost of a partial plan as the earliest possible exit time. For example, if a partial plan π enters a resource r at time t, then the earliest possible exit time out of r is given by $c(\pi) = \max(\text{start}(\tau_{\text{exit}}), t + tt(r))$. A choice of a consistent heuristic function might be the shortest path (without taking into account the presence of other agents) from the current resource to the destination resource. Hence, the estimated cost of a partial plan π (destination resource r') might be:

$$y(\pi) = c(\pi) + h(\pi) = \max(\text{start}(\tau_{\text{exit}}), t + tt(r)) + \text{shortestPath}(r, r')$$

In the following algorithm, we encode a plan π as a sequence of free time windows, and for each free time window, an entry time. For example, if a plan π ends in a free time window f on resource r, then we write $y(f)$ instead of $y(\pi)$. The free time window f is subsequently expanded to all reachable free time windows. The actual plan is only constructed (using backpointers) once the optimal route has been found.

In line 1 of algorithm 3, we check whether there exists a free time window on the start resource r_1 that contains the start time t. If there is such a free time window f, then

Algorithm 3 Plan Route

Require: start resource r_1, destination resource r_2, start time t; free time window graph $G_F = (F, E_F)$, open $= \varnothing$, closed $= \varnothing$

Ensure: shortest-time, conflict-free route plan from (r_1, t) to r_2.

```
 1: if ∃f [f ∈ F | t ∈ τ_entry(f) ∧ r_1 = resource(f)] then
 2:     mark(f, open)
 3:     entryTime(f) ← t
 4: while open ≠ ∅ do
 5:     f ← argmin_{f'∈open} y(f')
 6:     mark(f, closed)
 7:     r ← resource(f)
 8:     if r = r_2 then
 9:         return followBackPointers(f)
10:     t_exit ← c(f)
11:     for all f' ∈ {ρ(r, t_exit) \ closed} do
12:         t_entry ← max(t_exit, start(f'))
13:         if t_entry < entryTime(f') then
14:             backpointer(f') ← f
15:             entryTime(f') ← t_entry
16:             mark(f', open)
17: return nil
```

in line 2 we mark this window as open, and we record the entry time into f as the start time t. We will refer to the set of all free time windows with status open as the *open list*, in accordance with A* terminology[4].

In line 5, we select the free time window f on the open list with the lowest value of $y(f)$. Recall that $y(f) = c(f) + h(f)$, where $c(f) = \text{entryTime}(f) + tt(r)$ in which r is the resource associated with f. In line 6, we mark the window f as closed, which means that we will never expand this free time window again. If the resource r associated with f equals the destination resource r_2, then we have found the shortest route to r_2. We return the optimal plan in line 9 by following a series of backpointers from the current free time window f to the start window. If r is not the destination resource, we prepare to expand the plan. First, in line 10, we determine the earliest possible exit time out of r as the cost of the partial plan: $c(f) = \text{entryTime}(f) + tt(r)$. Then, in line 11, we iterate over all reachable free time windows that are not closed.

When expanding free time window f to free time window f', we first determine the entry time into f' as the maximum of the earliest exit time out of resource r, and the earliest entry time into f'. We only expand the plan from f if there has been no previous expansion to free time window f' with an earlier entry time (initially, we assume that the entry times into free time windows are set to infinity). In line 14, we set the backpointer of the new window f' to the window f from which it was expanded. Then, we record the entry time into f' as t_{entry}, and we mark f' as open. Finally, it is possible that no

[4]Even though open is usually implemented as a *priority queue* rather than a list.

conflict-free plan exists, in which case we return nil in line 17.

Proposition 4.2.2. *If the heuristic function h is consistent, algorithm 3 always returns an optimal solution.*

Proof. We will prove that algorithm 3 always returns an optimal route plan (if one exists) by showing that prior to termination, there is always a free time window on the open list that is on an optimal route. Because any sub-optimal (partial or full) route plan has a higher y-value than a (partial or full) optimal plan, the optimal free time window on open will be expanded before a sub-optimal solution can be returned. Expansion of an optimal partial plan will result in another optimal (partial or full) plan.

1. *Introducing notation*
   Let π^* be an optimal route from r to r', starting from time t. We can characterize π^* as a sequence of free time windows with optimal entry times: $\pi^* = (\langle f_1, t_1^* \rangle, \ldots, \langle f_m, t_m^* \rangle)$, where f_1 is a free time window on the start resource r, t_1^* equals the start time t, and f_m is a free time window on resource r'. We will show that, prior to termination of the algorithm, there exists an open window $f_k \in \pi^*$ with optimal entry time t_k^*.[5]

2. *There exist optimal* closed *windows*
   Let Δ be the set of all closed windows from the optimal plan π^*, such that for all $f_i \in \Delta$, $t_i = \text{entryTime}(f_i)$ is optimal. Note that Δ is non-empty, since after the first iteration it contains the start window f_1 with entry time t.

3. *Optimal* closed *window f_{k-1} has been expanded to* open *window f_k on optimal route π^**
   Let f_{k-1} be the window from Δ with the highest index. Since f_{k-1} is closed, it must have been expanded to f_k, unless the window f_k were already closed. If the heuristic function h is consistent, then prior expansion of f_k is impossible, which we will now demonstrate. If f_k were closed, it would, by definition of Δ, have a sub-optimal entry time $t' > t_k^*$. We have:

   $$\begin{aligned} y(f_{k-1}) &= t_{k-1}^* + tt(r_{k-1}) + h(f_{k-1}) \\ &\leq t_k^* + tt(r_k) + h(f_k) \text{ (consistency)} \\ &< t' + tt(r_k) + h(f_k) \end{aligned}$$

   Hence, window f_{k-1} would be expanded before f_k, so f_k cannot be closed.

4. *Window f_k on optimal route has been entered with optimal entry time t_k^**
   We will now show that the expansion of f_{k-1} to f_k has resulted in the optimal entry time t_k^* into f_k. The entry time of a free time window is determined in line 12 of algorithm 3; there are two cases to consider:

   case a: t_{entry} equals the start of the free time window f_k. Since a window cannot be entered earlier than its start time, t_{entry} equals the optimal entry time t_k^*.

[5] Showing the existence of an open window on the optimal path is similar to lemma 1 in Hart et al. [32].

case b: $t_{\text{entry}} = t^*_{k-1} + tt(r_{k-1})$. Since t^*_{k-1} is optimal, so is t_{entry}.

Having shown that there exists an open window with the optimal entry time, it is easy to show that algorithm 3 can never return a sub-optimal solution. First, note that a consistent heuristic is also *admissible*, i.e., it never overestimates the cost of reaching the destination. This implies that at the destination resource, we have $h = 0$. A sub-optimal plan to the destination therefore has a larger y-value than any (partial or full) optimal plan; an optimal plan will thus be retrieved from the open list first.

Finally, note that algorithm 3 always terminates: there is only a finite number of free time windows, and in each iteration, one free time window is closed (that can never be opened again). Hence, if a solution exists, algorithm 3 always returns an optimal solution. □

4.2.2 Algorithm complexity

Algorithm 3 runs in polynomial time in the size of the free time window graph, although the exact complexity depends on the type of heuristic used, as well as the type of data structure used. Earlier we mentioned that using a consistent heuristic has the advantage of having to expand each free time window at most once.

Proposition 4.2.3. *Algorithm 3 runs in* $O((|E_F| + |F|)\log(|F|))$ *time, if the heuristic* h *is consistent.*

Proof. A free time window $f \in F$ can be retrieved from the open list at most once, so the while loop runs for at most $|F|$ iterations. Lines 5 and 6 constitute a removal of the cheapest element from the open list. If we assume that this list is a priority queue, implemented by e.g. a binary heap, then this operation costs $O(\log(|F|))$ time (see e.g. Kleinberg and Tardos [44]).

The for loop in line 11 could inspect every connection between two free time windows exactly once, so lines 12 to 16 can run at most $|E_F|$ times. Within the for-loop, line 13 checks whether the free time window f has been visited yet, and if so, if the entry was earlier. If there already exists a partial plan to f' with a *later* entry time, then the values for backpointer(f') and entryTime(f') must be overwritten (in lines 14 and 15 respectively). This constitutes a *decrease key* operation for the priority queue, which requires $O(\log(|F|))$ time for a binary heap implementation. □

In the proof of proposition 4.2.3, we assumed that the open list was implemented using a binary heap structure. By using a Fibonacci-heap data structure, we can improve the complexity of a *decrease key* operation from $O(\log(|F|))$ to $O(1)$ amortized time[6]. Hence, the complexity of algorithm 3 using a Fibonacci-heap (and a consistent heuristic) equals $O(|F|\log(|F|) + |E_F|)$ amortized time.

Another way to improve the complexity of algorithm 3 is to set the heuristic function to zero, so that $y(\pi) = c(\pi)$. Although the heuristic function aims to reduce the running

[6]Amortized analysis considers the average running time per operation over a worst-case sequence of operations. In this case, a decrease key operation may require $O(|F|)$ operations, but because it can be proved that this is required only in one out of $O(|F|)$ times, the amortized running time of this operation is $O(1)$.

time of the algorithm, it does require some 'administrative' operations. In particular, in lines 14 and 15, we may need to overwrite the backpointer and entry time of a free time window that has been reached previously. If there is no heuristic function, on the other hand, then we can prove that the first visit to a free time window is optimal, so no overwriting is necessary.

Corollary 4.2.4. *Algorithm 3 runs in $O(|E_F| + |F| \log(|F|))$ time, if the heuristic function h is zero.*

Proof. Let f be the free time window that is removed from the open list in line 5. We will prove that a free time window f' is expanded with the earliest possible entry time. The entry time is determined in line 12, and there are two cases to consider:

case a: The entry time t_{entry} equals the start of the free time window f'. This is the earliest possible entry time, because a window cannot be entered earlier than its start time.

case b: The entry time equals $t_{\text{entry}} = c(f) = y(f)$. Since the heuristic function $h = 0$ is consistent, we know that in all future iterations of the while loop, a free time window f'' that is retrieved from open has a value $c(f'') = y(f'') \geq y(f)$. Hence, the window f' that we are about to expand cannot be entered earlier than t_{entry}.

If the first visit to a free time window is the optimal one, then each free time window can be inserted into open at most once. Hence, lines 14 to 16 are executed at most $|F|$ times, and insertion of a new element into the open list requires $O(\log(|F|))$ time for a binary heap implementation. Lines 14 to 16 therefore contribute $O(|F| \log(|F|))$ to the complexity of algorithm 3. Lines 11 to 13 may still require inspection of every element in E_F, so these lines contribute $O(|E_F|)$. Algorithm 3 therefore runs in $O(|F| \log(|F|) + |E_F|)$ time if the heuristic function is always zero. □

In the remainder of this section, we will show that there are bounds on $|E_F|$, the size of the reachability relation. Also, we will show that under the 'acyclic planning constraint', we can relate the complexity of algorithm 3 to the sets $\mathcal{A}$, R, and E_R (agents, resources, and connections between resources).

The relation E_F can maximally consist of $|F \times F|$ tuples. However, because each $f \in F$ represents a time interval, and two free time windows can only be connected if their intervals overlap, the number of elements in E_F is much smaller than $|F \times F|$.

Lemma 4.2.5 (Interval reachability)**.** *Given free time windows f_i and f_{i+1} on resource r and free time windows f_j and f_{j+1} on resource r', the following equation always holds:*

$$\begin{aligned} & (f_i \cap f_j \neq \varnothing) \wedge (f_{i+1} \cap f_j \neq \varnothing) \\ \rightarrow \quad & f_i \cap f_{j+1} = \varnothing \end{aligned} \tag{4.6}$$

Proof. Consider the following four free time windows:

$$\begin{aligned} f_i &= [t_1, t_2) \\ f_{i+1} &= [t_3, t_4) \\ f_j &= [t_5, t_6) \\ f_{j+1} &= [t_7, t_8) \end{aligned}$$

Free time windows f_i and f_{i+1} are successive free time windows on resource r, and free time windows f_j and f_{j+1} are successive windows on resource r' (see figure 4.8 for an illustration). We assume that both free time windows on resource r overlap with the first free time window on resource r', and we will prove that this implies: $f_i \cap f_{j+1} = \varnothing$.

It follows from definition 4.1.1 that any two free time windows on the same resource have an empty intersection. Hence, the end time t_2 of f_i is smaller than the start time t_3 of f_{i+1}. Since f_{i+1} overlaps with f_j, we also have $t_3 \leq t_6$.

Finally, we have $t_6 < t_7$, because window f_j ends before f_{j+1} starts. We can now conclude that the end time t_2 of f is smaller than the start time t_7 of f_{j+1}. Hence, free time windows f_i and f_{j+1} are disjoint. □

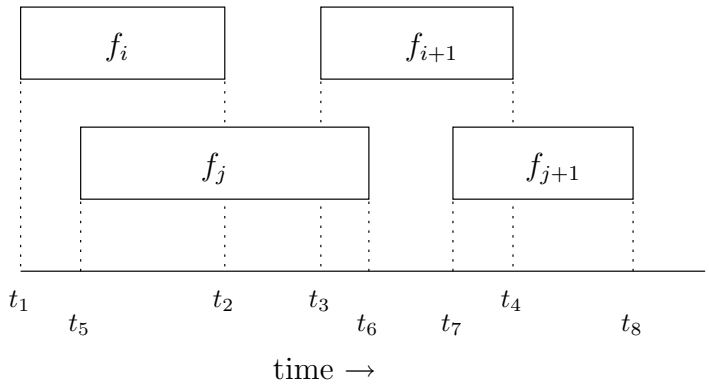


Figure 4.8: Interval f_i has no overlap with interval f_{j+1}

We can use lemma 4.2.5 to prove that there is a limited number of connections between free time windows on one resource and free time windows on another resource.

Proposition 4.2.6. *Given two resources r and r', and sets of free time windows F_r and $F_{r'}$, and let r-sum$(r, r') = |\,((F_r \times F_{r'}) \cap E_F)\,|$, then*

$$r\text{-}sum(r, r') \leq |F_r| + |F_{r'}| - 1 \tag{4.7}$$

Proof. Let $n = |F_r|$ and $m = |F_{r'}|$. We will prove the proposition with induction on $n + m$. For $n = 1$ and $m = 1$, the result is trivial. Suppose that the property holds for $k = n + m$. Let $k = n + m + 1$. We distinguish the following cases.

- $|F_r| = n + 1$. Let f_{n+1} be the last time window in F_r. Let $O_{r'} \subseteq F_{r'}$ be the set of all time windows having an overlap with f_{n+1} and let $p = |O_{r'}|$. If $p = 0$ then, clearly, r-sum$(r, r') \leq n + m - 1$. If $p > 0$, let f' be the first time window in $O_{r'}$. By Lemma 4.2.5, we have that r-sum is the number of tuples obtained from $(F_r - f_{n+1}, (F_{r'} - O_{r'}) \cup \{f'\})$ plus the number of tuples obtained from $(\{f_{n+1}\}, O_{r'})$. By induction hypothesis, the first number of tuples is at most $n + (m - p + 1) - 1$ and the second number is at most p. Hence, r-sum$(r, r') \leq n + (m - p + 1) - 1 + p = (n + 1) + m - 1$.

- $|F_{r'}| = m + 1$. Analogous to the previous case.

□

Corollary 4.2.7. *The size of the reachability relation E_F is bounded by*

$$|E_F| = O\left(|E_R| \cdot \max_{i \in \{1,\ldots,|R|\}}(F_i)\right)$$

Proof. For every $(r, r') \in E_R$, it is possible to create at most $2 \cdot \max_{i \in \{1,\ldots,|R|\}}(F_i)$ tuples in the reachability relation. □

Corollary 4.2.8. *Algorithm 3 run in $O(|\mathcal{A}||R| \log(|\mathcal{A}||R|) + |E_R||\mathcal{A}|)$ time, if constraint 3.9 holds.*

Proof. Under constraint 3.9, agents are not allowed to make cyclic plans. Hence, each resource can hold at most $|\mathcal{A}| - 1$ reservations (i.e., the reservations of the *other* $|\mathcal{A}| - 1$ agents), which can result in at most $|\mathcal{A}|$ free time windows per resource. Hence, $|F| \leq |R| \cdot |\mathcal{A}|$. From corollary 4.2.7, we know that $|E_F| \leq 2 \cdot |E_R| \cdot |\mathcal{A}|$. □

4.3 Route planning from A to Z

In this section we will present an algorithm for the single-agent, *multi-stage* route planning problem in which an agent has a start location and a *sequence* ϕ of locations to visit, rather than a single destination location. This *visiting sequence* must be visited in a fixed order, that is, if $\phi = (r_1, \ldots, r_m)$, then for any plan π it must hold that $\phi \preceq \text{resources}(\pi)$ (the visiting sequence is a sub-sequence of the resources in the plan).

The multi-stage routing problem can be relevant if an agent has multiple locations to visit, or if it has multiple transportation tasks. In airport taxi routing, single-stage routing often suffices; under normal weather conditions, an aircraft can taxi directly from the gate to the runway. However, wintry conditions sometimes require snow and ice to be removed from the aircraft, which means that an aircraft has to taxi from the gate to a *de-icing station*, and only then to the runway. In manufacturing, an Automated Guided Vehicle (AGV) may have a sequence of transportation orders to perform, and it must also make the occasional trip to the battery charging station in between orders. Even if an AGV has only a single transportation task, it cannot simply stop moving after delivering its final cargo, because it might get in the way of other agents. Hence, multi-stage routing is also relevant for the idle vehicle positioning problem (cf. [8]).

A related, but more general problem is the Traveling Salesperson Problem (TSP), in which there is a *set* (i.e., unordered) of locations that must be visited with minimum total cost. This generality comes at the cost of NP-completeness (see Garey and Johnson [26]), whereas the multi-stage routing problem can be solved in polynomial time. Also the generality afforded by the TSP is not required in all application domains; for instance, in the airport de-icing scenario, an agent need not consider route plans where the aircraft takes off prior to de-icing.

In the Multiple Destination Routing problem (MDR), there is a graph with one (or more) source locations and a set of destination locations, and the objective is to find a

minimum tree that connects the source(s) to all destinations (possibly subject to some additional constraints). The MDR problem is common in telecommunications networks (see e.g. [50]), where communication services must be delivered to multiple sources. An important difference with logistic route planning (and also the TSP problem), is that there need not be one particular route that all network packets follow. Instead, the source should be connected to the destinations using the minimum-cost sub-network.

4.3.1 Naive multi-stage algorithm

In classical shortest path planning research, multi-stage planning has never been a research topic, because there exists a trivial concatenation algorithm: given a visiting sequence $\phi = (n_1, \ldots, n_m)$, find the shortest path between all nodes n_i, n_{i+1}, and concatenate the resulting paths (in algorithm 4 below, we denote the concatenation operator by '$\frown$'). This results in a shortest path along ϕ, because of the property that we mentioned before: if n_{i+1} is on a shortest path from n_i to n_{i+2}, then a shortest path from n_i to n_{i+1} can be expanded to a shortest path from n_i to n_{i+2}.

In section 4.2, we already saw that this property does not hold if other agents have reserved their route plans on the infrastructure resources. In the following, we discuss another example that not only demonstrates that the concatenation approach is sub-optimal, but also that it is incomplete, i.e., it can fail to find a route plan even if one exists. For the sake of completeness, we first give a specification of the concatenation approach in our setting.

Algorithm 4 Multi-Stage Concatenation

Require: visiting sequence $\phi = (r_1, \ldots, r_m)$, start time t; free time window graph $G_F = (F, E_F)$.

1: $\pi \leftarrow \langle r_1, [t, t + tt(r_1))\rangle$
2: $i \leftarrow 1$
3: **while** $\pi \neq \text{nil} \wedge i < m$ **do**
4: $\quad t' \leftarrow c(\pi) - tt(r_i)$
5: $\quad \pi_{(i,i+1)} \leftarrow \text{planRoute}(r_i, r_{i+1}, t', G_F)$
6: $\quad$ **if** $\pi_{(i,i+1)} = \text{nil}$ **then**
7: $\quad\quad \pi \leftarrow \text{nil}$
8: $\quad$ **else**
9: $\quad\quad \pi \leftarrow \pi \frown \pi_{(i,i+1)}$
10: $\quad i \leftarrow i + 1$
11: **return** π

In figure 4.9, we see an infrastructure with three agents. We will discuss the planning problem of agent A_1, which has a visiting sequence $\phi_1 = (s, b, t)$. Agent A_2 wants to go from t to a (i.e., without needing to visit any intermediate resources), and agent A_3 wants to go from c to a. To show that the concatenation approach to multi-stage routing is sub-optimal, suppose that agent A_2 has already made a plan, but agent A_3 has not. Hence, agent A_1 only needs to take into account the plan of agent A_2, which is the following:

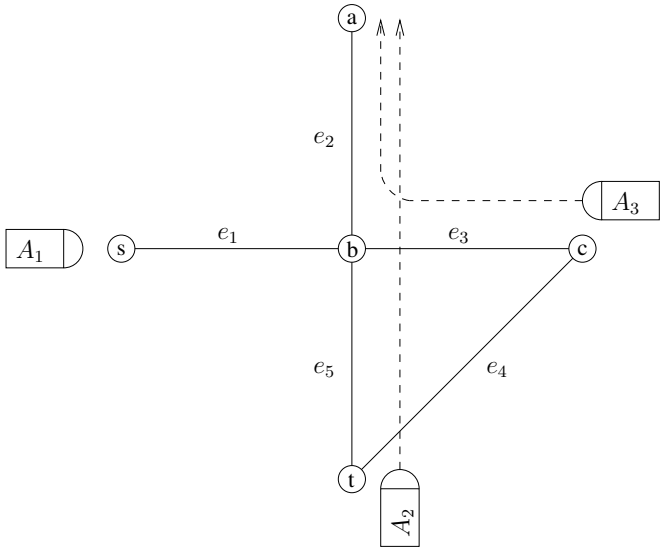


Figure 4.9: Infrastructure where intersections (circles) have minimum travel time 2, and lanes (lines) have minimum travel time 4. The visiting sequences of the agents are: $\phi_1 = (s, b, t)$, $\phi_2 = (t, a)$, and $\phi_3 = (c, a)$.

$$\pi_2 = \langle t, [2, 4)\rangle, \langle e_5, [4, 8)\rangle, \langle b, [8, 10)\rangle, \langle e_2, [10, 14)\rangle, \langle a, [14, 16)\rangle$$

Under the concatenation approach, agent A_1 first makes a plan from s to b. On intersection b, there are two free time windows: $f_{b,1} = [0, 8)$ and $f_{b,2} = [10, \infty)$. Agent A_1's plan can make use of $f_{b,1}$:

$$\pi_{1,1} = \langle s, [0, 2)\rangle, \langle e_1, [2, 6)\rangle, \langle b, [6, 8)\rangle$$

To make a plan from b to t (starting from resource b at time 6, in free time window $f_{b,1}$), agent A_1 has only one option: to take the detour e_3, c, e_4, t, because the direct route via e_5 and t is blocked by agent A_2 (we again assume that no simultaneous exchanges are allowed, see section 3.2.1):

$$\pi_{1,2} = \langle b, [6, 8)\rangle, \langle e_3, [8, 12)\rangle, \langle c, [12, 14)\rangle, \langle e_4, [14, 18)\rangle, \langle t, [18, 20)\rangle$$

Gluing together $\pi_{1,1}$ and $\pi_{1,2}$ results in a slower plan than A_1's optimal plan, which is to wait until agent A_2 has exited intersection b, and then to take the direct route:

$$\pi^* = \langle s, [0, 2)\rangle, \langle e_1, [2, 10)\rangle, \langle b, [10, 12)\rangle, \langle e_5, [12, 16)\rangle, \langle t, [16, 18)\rangle$$

To show that the concatenation approach is incomplete, suppose that in addition to agent A_2, agent A_3 has also made a plan:

$$\pi_3 = \langle c, [4, 6)\rangle, \langle e_3, [6, 10)\rangle, \langle b, [10, 14)\rangle, \langle e_2, [14, 18)\rangle, \langle a, [18, 20)\rangle$$

Again there are two free time windows on resource b: $f_{b,1} = [0, 8)$, and $f_{b,2} = [14, \infty)$. Agent A_1's plan to resource b is the same as before: the plan π_1 that arrives in b at time 6. However, when agent A_1 tries to make a plan from b (at time 6, in $f_{b,1}$) to t, it can only go 'up' in the direction of resource a, or back towards the start s. If we assume that an agent is not allowed to turn around in a resource (see section 3.2.3), then the latter option is not allowed, and the former does not lead to the destination t either. In this case, A_1's only option is to wait until both agents have cleared intersection b, but the concatenation approach does not find this option.

4.3.2 Algorithm specification

In the single-stage routing algorithm, a partial plan is completely characterized by the last free time window and the associated entry time: together they determine how the partial plan can be expanded. In multi-stage routing, we should also know which resources in the visiting sequence have been reached yet. For example, if a partial plan ends in the final resource of the sequence, but not all other stages have been visited yet, then we should continue expanding the plan. This also means that we may have to visit a free time window more than once. Consider the infrastructure in figure 4.10, where we have a visiting sequence $\phi = (r_2, r_4, r_1)$. Any valid multi-stage plan will have to pass intersection r_3 twice. In case there are no reservations in the system yet, then each resource has a single free time window, and we need to allow more than one visit to the free time window on resource r_i.

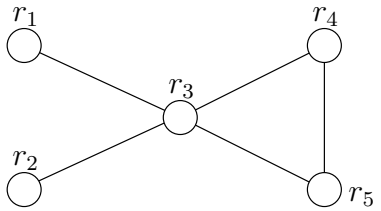
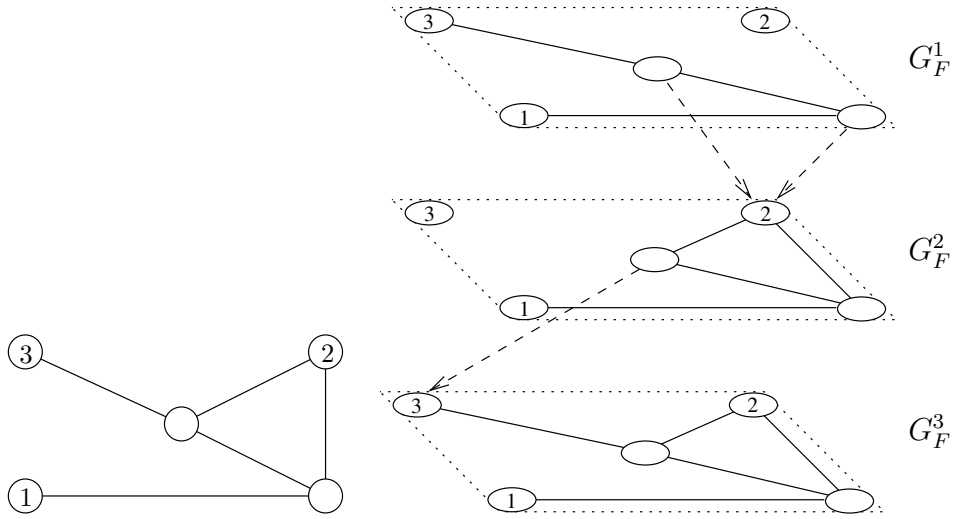


Figure 4.10: Given visiting sequence $\phi = (r_2, r_4, r_1)$, resource r_3 must be visited twice.

The solution we present in this section is to make $|\phi|$ copies of each free time window. In this way we can guarantee that, having visited the first k resources in the sequence, all free time windows with stage number k are visited at most once. We change the reachability relation E_F in such a way that the free time window graph consists of layers: free time windows with stage number k can only reach (and be reached from) other free time windows with stage number k. The exception, of course, is for free time windows on 'stage' resources: free time window f with stage number k is connected to f' with stage number $k + 1$ if $(f, f') \in E_F$, and the resource associated with f' is stage $k + 1$.

Figure 4.11 illustrates this *augmented reachability relation*, and the layered free time window graph that results from it. Figure 4.11(a) shows a simple infrastructure in which



(a) Infrastructure with three stages (b) Free time window graph with three layers

Figure 4.11: A visiting sequence of size three results in a free time window graph with three layers.

three intersections marked 1, 2, and 3 are the three stages. In the free time window graph, the free time window associated with stage 2 is not connected to any free time window in layer 1 (hence, we could leave that window out of the graph). Instead, the free time windows on the neighbouring resources are connected to the free time window with stage number 2. We define the augmented reachability relation E_F^a as follows, given a visiting sequence $\phi = (r_1, \ldots, t_m)$:

$$E_F^a = \{(\langle f, k\rangle, \langle f', k\rangle) \,|\, (f, f') \in E_F \wedge \text{resource}(f') \neq r_{k+1}\} \cup$$
$$\{(\langle f, k\rangle, \langle f', k+1\rangle) \,|\, (f, f') \in E_F \wedge \text{resource}(f') = r_{k+1}\}$$

As in the single-stage case, the existence of a pair $(\langle f, k\rangle, \langle f', k'\rangle) \in E_F^a$ does not guarantee that a plan π, ending in free time window f, stage k, at some time $t \in \tau_{\text{exit}}(f)$, can be expanded to free time window f' with stage k'. Hence, if $\tau_{\text{exit}}(f) = [t_1, t_2)$, then we must check whether $[t, t_2) \cap \tau_{\text{entry}}(f') \neq \varnothing$. We will write $\rho^a(r, t_{\text{exit}}, k)$ to denote the set of free time windows reachable from resource r at earliest exit time t, given the current stage number k.

Definition 4.3.1 (Augmented free time window reachability)**.** *Free time window f' at stage k' is reachable from free time window $f = [t_s, t_e)$ (on resource r) at time t, stage k, denoted $\langle f', k'\rangle \in \rho^a(r, t, k)$, if*

1. $(\langle f, k\rangle, \langle f', k'\rangle) \in E_F^a$,

2. $t_s \leq t < t_e$,

3. $[t, t_e) \cap \tau_{\text{entry}}(f') \neq \varnothing$.

Algorithm 5 Multi-Stage Plan Route

Require: visiting sequence $\phi = (r_1, \ldots, r_m)$ such that successive resources are different, start time t, free time window graph $G_F = (F, E_F)$, $\mathsf{open} = \varnothing$, $\mathsf{closed} = \varnothing$
Ensure: shortest-time, conflict-free plan π, such that $\phi \preceq \text{resources}(\pi)$.

```
1: if ∃f [f ∈ F|t ∈ τ_entry(f) ∧ r_1 = resource(f)] then
2:     mark(⟨f, 1⟩, open)
3:     entryTime(f, 1) ← t
4: while open ≠ ∅ do
5:     ⟨f, k⟩ ← argmin_{⟨f',k'⟩∈open} y(f', k')
6:     mark(⟨f, k⟩, closed)
7:     r ← resource(f)
8:     if k = |φ| then
9:         return followBackpointers(f, k)
10:    t_exit ← c(f, k)
11:    for all ⟨f', k'⟩ ∈ {ρ^a(r, t_exit, k) \ closed} do
12:        t_entry ← max(t_exit, start(f'))
13:        if t_entry < entryTime(f', k') then
14:            backpointer(f', k') ← ⟨f, k⟩
15:            entryTime(f', k') ← t_entry
16:            mark(⟨f', k'⟩, open)
17: return nil
```

Lemma 4.3.2. *Given a visiting sequence $\phi = (r_1, \ldots, r_m)$, an optimal route plan can be constructed without $\langle f_{k,j}, k-1\rangle$, for all $j \in \{1, \ldots, |F_k|\}$.*

Proof. Let $\langle f, k-1\rangle$ be the element retrieved from the open list in some iteration of algorithm 5. By expanding to $\langle f', k\rangle$ (where $\text{resource}(f') = r_k$), this path can no longer reach any free time window with stage number $k-1$ or smaller. This is not a restriction, however, since any free time window reachable from f' in E_F is reachable in E_F^a.

It can occur, of course, that we cannot expand from $\langle f', k\rangle$ to some $\langle f'', k''\rangle$ in case the latter already exists on open or on closed. However, this implies that another route π' has reached $\langle f'', k''\rangle$ with a better-or-equal entry time, and there is no need to expand $\langle f'', k''\rangle$ again. □

Corollary 4.3.3. *If the heuristic function h is consistent, algorithm 5 always returns the optimal solution.*

Proof. In proposition 4.2.2, we showed the optimality of algorithm 3. Since algorithm 5 is essentially the same algorithm that operates on a different graph structure, we do not repeat the proof here. In algorithm 3, the nodes of the graph are free time windows and the edges are formed by the reachability relation E_F; in algorithm 5, the nodes of

the graph are ⟨free time window, stage number⟩ tuples, and the edges are given by the augmented reachability relation E_F^a. Finally, lemma 4.3.2 showed that the optimal route can be found using E_F^a. □

The complexity of algorithm 5 is $O(|\phi|)$ times the complexity of algorithm 3: there are $|\phi| \cdot |F|$ ⟨free time window, stage number⟩ tuples, and the augmented reachability relation E_F^a is around $|\phi|$ times as large as the reachability relation E_F.

Proposition 4.3.4. *If the heuristic h is consistent, then algorithm 5 runs in $O(|\phi| \cdot \log(|F|) \cdot (|E_F| + |F|))$ time.*

Proof. The analysis of the computational complexity of algorithm 5 follows the same lines as the analysis for algorithm 3. In the main while loop, every ⟨free time window, stage number⟩ tuple can be retrieved from the open list at most once; in the for loop, every element of the augmented reachability relation might be inspected. □

4.4 Concluding remarks

In this chapter, we presented a sequential approach to the multi-agent route planning problem. Agents plan one by one, and the plans of previous agents must be respected. Given a set of agent plans, we can derive for each resource the set of *free time windows*, representing the time intervals during which a resource can be used without creating a conflict with any of the previous plans. The edges in the *free time window graph* specify when a free time window on one resource can be reached from a free time window on another.

We have presented an algorithm for the single-destination route planning problem that performs a search through the free time window graph. Our algorithm finds a route plan for a single agent that is both optimal and conflict-free. The computational complexity is $O(|\mathcal{A}||R| \log(|\mathcal{A}||R|) + |E_R||\mathcal{A}|)$ (where $\mathcal{A}$ is the set of agents, R the set of resources, and E_R are the connections between resources), which is a significant improvement over the $O(|\mathcal{A}|^4|R|^2)$ algorithm presented by Kim and Tanchoco [41].

Finally, we presented the *multi-stage* route planning problem, in which an agent must visit a fixed sequence of destination locations. It turns out that we can adapt our single-destination algorithm if we add layers to the free time window graph, one for each additional destination location.

In the next chapter, we will look at the execution of route plans, and in particular how we can apply re-scheduling in case unexpected incidents in the environment disrupt the original agent plans.

Chapter 5

Priority-Based Schedule Repair

In the previous chapter we presented an algorithm that allows us to find a conflict-free set of agent route plans. Absence of conflict is only guaranteed, however, as long as the agents are able to follow their plans to the letter. A small deviation from the plan, such as an AGV slowing down to avoid a collision with a human that steps into the lane, can already cause a deadlock situation. For example, if an agent enters a lane resource late, then it might encounter another agent head-on, after which neither agent can make any progress. In this chapter, we will present ways of dealing with unexpected incidents that threaten to disrupt the execution of the agents' plans.

An *agent plan* (definition 3.1.1) prescribes in detail the actions of an agent: for every point in time, the agent plan specifies which resource the agent should be driving on. Any inaccuracy in the agent's driving behaviour may cause it to fall behind or get ahead of its plan. Alternatively, an agent might suffer some breakdown which means it has to wait until it has been repaired. Both of these events can be classified as *vehicle* incidents, which is the type of incident that we consider in this chapter. In [103], Zutt also considers *infrastructure incidents*, such as a road becoming temporarily unavailable, but we do not consider these in our research. Our focus on vehicle incidents means that we can restrict ourselves to *schedule* repair actions, i.e., repair actions that only change the timing of plans, and still always be able to repair plans. Of course, it is possible that allowing re-routing would result in more efficient repaired plans.

If we abstract away from the specific time points in a plan, all that remains of a single-agent plan is a sequence of resources to visit. From a multi-agent plan (i.e., a conflict-free set of single-agent plans), we can derive for each resource the order in which agents will use the resource. This order in which agents visit a resource can be interpreted as the *priority* of the agents on the resource. In case of vehicle incidents, we can adjust the timing of the agent plans such that this priority does not change. If we maintain the original priorities of the agents, then it is not hard to show that each agent can still reach its destination location, albeit at a later time. If we increase the priority of a non-delayed agent over a delayed agent, we may achieve a more efficient execution of the set of agent plans. However, by making changes to the priority of the agents, it is possible (though not inevitable) that a deadlock situation is created. For example, in figure 5.1, agent A_1

has the higher priority on resources r_1, r_2, and r_3, but it is somewhat delayed. If agent A_2 is granted a higher priority on r_3, and A_2 immediately enters r_3, then the agents will meet head-on somewhere between resources r_1 and r_3, and unless A_2 drives back, they will end up in a deadlock situation.

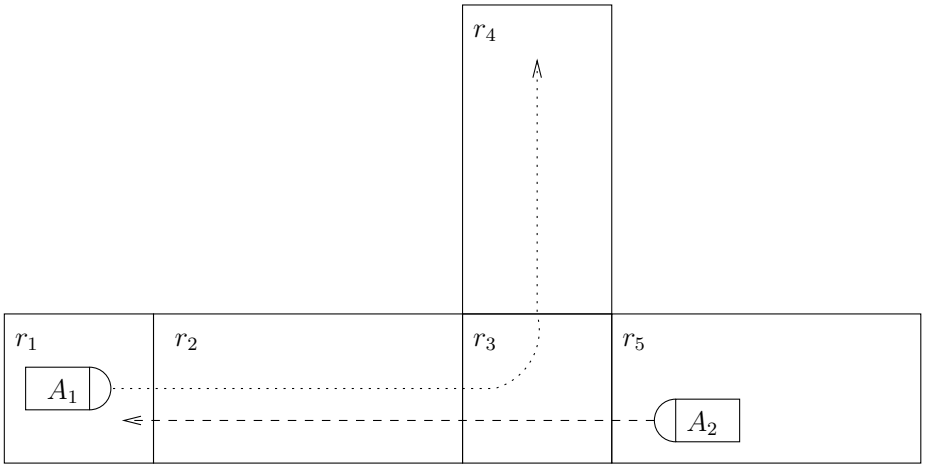


Figure 5.1: Suppose that A_1 has a higher priority on r_3, but A_2 is poised to enter. If the priority of A_2 is increased on resource r_3, then sooner or later A_1 and A_2 will meet head-on.

To judge whether a priority change is safe, we present a graph structure that contains a cycle if and only if a priority change leads to a deadlock. In addition, we present an algorithm that can perform safe priority changes. We will start this chapter by describing a plan execution model for our primary application domain, the airport taxi routing scenario.

5.1 Plan execution

In the airport taxi routing problem, the following constraints on agent plans must hold. First, no simultaneous resource exchanges are allowed between agents (constraint 3.3). Second, no overtaking is allowed (constraint 3.5), since the large wingspans of aircraft prevent agents from taxiing side by side. For the same reason, taxiways may be used in only one direction at the same time (i.e., no simultaneous bidirectional travel is allowed — constraint 3.4). Fourth, agents are neither allowed to nor capable of turning around on a taxiway, so spinturns are not allowed (constraint 3.8). Of course, these constraints should not only hold during the planning phase, but also during the execution of plans.

If agents never incur any delay, then conflict-free execution of conflict-free plans is trivial. The following example shows what can happen if one agent is delayed, and another agent proceeds as if nothing has changed.

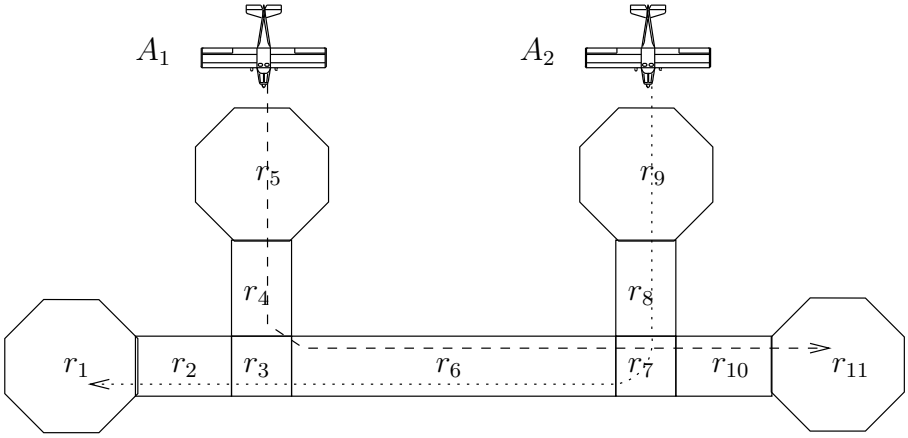


Figure 5.2: Two aircraft agents A_1 and A_2 with hangars r_{11} and r_1 as respective destinations.

Example 5.1.1. *Consider a small airport with aircraft agents A_1 and A_2, in figure 5.2. A_1 starts at hangar (resource) r_5 and wants to go to r_{11}, while A_2 wants to go from hangar r_9 to hangar r_1. The agents make the following plans, to the effect that A_1 is allowed to pass along the taxiway straight (r_3, r_6, r_7) first:*

$$\begin{aligned}\pi_{A_1} =&\langle r_5, [0,2)\rangle, \langle r_4, [2,4)\rangle, \langle r_3, [4,5)\rangle, \langle r_6, [5,10)\rangle,\\ &\langle r_7, [10,11)\rangle, \langle r_{10}, [11,13)\rangle, \langle r_{11}, [13,15)\rangle\end{aligned} \tag{5.1}$$

$$\begin{aligned}\pi_{A_2} =&\langle r_9, [0,2)\rangle, \langle r_8, [2,11)\rangle, \langle r_7, [11,12)\rangle, \langle r_6, [12,17)\rangle,\\ &\langle r_3, [17,18)\rangle, \langle r_2, [18,20)\rangle, \langle r_1, [20,22)\rangle\end{aligned} \tag{5.2}$$

Hence, A_2 plans to stay on taxiway r_8 in the interval $[2,11)$, just long enough to let A_1 pass.

Suppose that during plan execution, A_1 stalls his engine, and he departs with a delay of 5. Then, he will be on taxiway r_6 in the interval $[10,15)$. Clearly, this interferes with the plan of A_2, who will be on r_6 from 12 until 17. Agent A_1 and A_2 will therefore meet each other on taxiway r_6, and since the agents cannot turn around or drive backwards, they will end up in a deadlock situation.

What happened in example 5.1.1 was that the *priority* that the agents had agreed upon during planning, was violated during plan execution. After the planning phase, we know for each resource in which order it will be visited by the agents. If we maintain this order during plan execution, then no deadlocks can occur [58]. In example 5.1.1, A_2 should therefore not only wait until time 11 before entering taxiway r_7, it should also wait until A_1 has exited this resource.

During plan execution, an agent must therefore verify whether its turn has come to enter the resource. We write $p(r, \sigma_{i,j}) = n$ if the j^{th} plan step $\sigma_{i,j}$ from agent A_i is the

n^{th} plan step on resource r (note that resource r is also the j^{th} resource to be visited by agent A_i, so we sometimes write $r_{i,j}$). If resource r has been entered $n-1$ times (not necessarily by different agents, in case cyclic plans are allowed), then it is now the turn of agent A_i with plan step $\sigma_{i,j}$. In addition, for a lane resource with a capacity greater than 1, there must be enough capacity left over before the agent may enter.

Definition 5.1.2 (Resource entry permission)**.** *An agent A_i is allowed to start its next plan step $\sigma_{i,j} = \langle r, \tau \rangle$, such that $p(r, \sigma_{i,j}) = n$, by entering resource r at time t if:*

1. *$n-1$ plan steps have been started on this resource so far,*
2. *the number of agents on r at time t is at least one less than $cap(r)$.*

The second condition in definition 5.1.2 ensures that the capacity of the resource is not violated. Even if it's the agent's turn to enter, it may be that the resource is still filled to capacity with other agents. The first of these agents (because overtaking is not allowed) must exit the resource before the next agent can enter. Definition 5.1.2 also ensures that no simultaneous exchanges can occur, as long as the original set of plans is also simultaneous-exchange free.

Proposition 5.1.3. *If resource entry only occurs if all conditions from definition 5.1.2 are satisfied, then constraint 3.3 is always satisfied.*

Proof. Constraint 3.3 forbids two kinds of simultaneous exchanges: the first kind involves two resources, and the second kind involves a cycle of three or more resources, where each resource is full at the time of the exchange.

case I: Consider a possible exchange between two resources r and r', with agent A_i going from r to r' and agent A_j going in the opposite direction. To avoid a simultaneous exchange at the level of *plans*, either agent A_i, or A_j must be the first to enter on *both* resources. W.l.o.g. suppose that agent A_i has a higher priority on both resources. Then, a simultaneous exchange is prevented by condition 1 from definition 5.1.2: it is not the turn of agent A_j to enter resource r until agent A_i has exited both resources.

case II: If an exchange involves three or more resources, such that all resources are already full prior to the exchange, then no agent can enter its next resource, since the second condition of definition 5.1.2 requires that at least one unit of capacity is left in the resource that is to be entered. Hence, no exchange can occur that would violate constraint 3.3.

□

To prevent overtaking during plan execution, we simply assume that an agent can see the agent driving in front of it on the same lane resource. If the agent in front is driving slower, then an agent simply matches its speed to that of the slower agent. Also, we assume that an agent is always able to slow down in case there is a slower agent in front of it, so no collisions will occur. In fact, we assume that no acceleration or deceleration is required, and an agent can immediately travel at its desired speed.

In case an agent has a clear road in front of it, then it can determine its own speed. We will now discuss two viable methods to determine an agent's speed. Suppose an agent enters a resource r at time t, and its plan specifies that the exit time should be t'. The first method is to choose a constant speed such that the agent arrives at the end of the resource at exactly time t'. Hence, the agent's speed will be $s = \frac{x}{t'-t}$, where x is the distance of resource r. If s is negative or greater than the maximum speed (which implies that the entry time t was later than planned), then the agent will travel at its maximum speed. The second method is that the agent always travels at its maximum speed.

Using the second method, an agent might arrive at the end of resource r before its planned time t'. However, if the agent constructed its plan using algorithm 3 from chapter 4, then it cannot enter the next resource earlier than t' unless other agents are ahead of schedule. If the other agents also used algorithm 3 to construct their plans, then they cannot be ahead of schedule, either. The reason why an agent cannot enter a resource earlier than its planned entry time can be found in the proof of proposition 4.2.2: algorithm 3 always finds the earliest possible entry time into a resource (into a free time window, actually). Entering the resource earlier than the planned entry time would either require the agent to drive faster than its maximum speed, or it would enter the resource outside a free time window, thereby violating at least one of the conditions of resource entry (definition 5.1.2). If we change the priorities of agents during planning, however, then it *is* possible for an agent to enter a resource earlier than the planned entry time. In example 5.1.1, if agent A_2 has its priority increased for all resources (r_5, r_4, r_3), then it can enter resource r_5 at time 4, which is much earlier than the planned entry time 11. Therefore, we assume that agents always drive at their maximum speed.

5.1.1 Incidents in plan execution

We will discuss plan repair in the context of *agents that are delayed*. Delays are primarily caused by unexpected incidents, which we define to be events that temporarily immobilize an agent. So, if an agent suffers an incident, then it has to wait for some period of time before it can start moving again (when we discuss experimental results in chapter 6, we will discuss specifics of the duration of an incident, and the frequency with which they occur). Of course, an agent can also incur delay indirectly, because another agent has suffered an incident. In particular, an agent might be stuck behind an agent suffering an incident, or, as in example 5.1.1, an agent has to wait at the entry of a resource, because it is waiting for a delayed agent to exit the resource first.

In [103], Zutt also considers *resource incidents*, in which the maximum speed of a resource is temporarily reduced to a value between (and including) zero and the maximum speed. Like our vehicle incidents, the resource incidents occur without any agents having prior knowledge of the incident. In case of a resource incident, Zutt shows that it can be beneficial for agents to plan a new route, i.e., making use of different resources. In our research, we do not take the option of re-routing into consideration.

5.2 Planstep-Priority Graph

Changing the priority of the agents during plan execution can often be beneficial. The question is whether we can change the priority of the agents without introducing deadlocks. We will extend example 5.1.1 to show that changing the priority at every opportunity can lead to a deadlock.

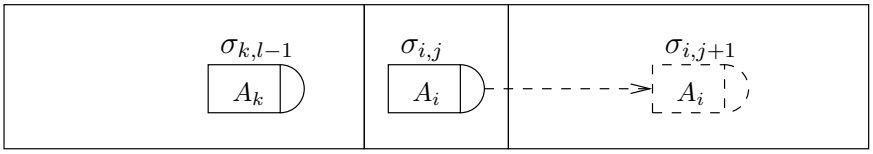
Example 5.2.1. *Consider again example 5.1.1, where agent A_2 reaches resource r_7 at time 4. According to its plan, it should wait until time 11 before it may enter, by which time agent A_1 should have exited r_7. However, since agent A_1 is delayed, it is decided (by some agent authorized to change priorities) to increase the priority of A_2 on r_7. Thus, A_2 enters r_7, as all conditions from definition 5.1.2 have been met. Agent A_2 subsequently reaches r_6 at time 5, and again A_2 is given priority over agent A_1, so A_2 proceeds along r_6. At time 10 agent A_2 reaches r_3, but by that time, it is occupied by agent A_1, which entered it at time 9, 5 time units behind schedule. This time, agent A_2 cannot be granted the highest priority on r_3, because it is already occupied by A_1. In fact, A_2 is already the next agent to enter; the problem is that the agents cannot make progress without performing a simultaneous exchange, which is impossible under definition 5.1.2 of resource entry permission.*

From example 5.2.1, it follows that the priority of agent A_2 should only be increased if A_2 gets the highest priority on all of the resources r_7, r_6, an d r_3. This sequence of resources can be thought of as a *corridor* of resources that A_2 may only enter if no other agent will enter from the other end. In section 5.3, we will discuss deadlock-free priority-changing algorithms, but first we will present a graph that predicts exactly which configuration of agent priorities will lead to a deadlock, and which will not. We will use this graph in our algorithm to verify that a proposed priority change will keep the system free of deadlocks.

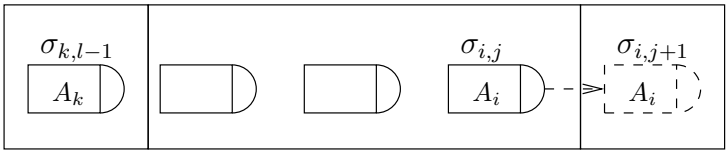
The Planstep-Priority Graph (PPG) that we will define below consists of the plan steps of the plans of all agents. There is an edge (σ, σ') in the PPG if plan step σ must precede plan step σ'. Under our model of plan execution, there are two reasons why one plan step must precede another. The first is that an agent A_i must perform its plan step $\sigma_{i,j}$ before it can start the next step in its plan, $\sigma_{i,j+1}$. The second reason is that an agent needs to obtain entry permission into the next resource (definition 5.1.2).

The first condition of definition 5.1.2 is that an agent should wait its turn before it may enter a resource. To start plan step $\sigma_{i,j}$ on resource r, agent A_i should wait for agent A_k with plan step $\sigma_{k,l}$, such that $p(r, \sigma_{i,j}) = p(r, \sigma_{k,l}) + 1$. Hence, there will be an edge $(\sigma_{k,l}, \sigma_{i,j})$ in the Planstep-Priority Graph. To determine the precedence relation between plan steps resulting from condition two from definition 5.1.2, see figure 5.3 for the two cases of a lane resource and an intersection resource. In figure 5.3, we see that an agent A_k cannot start its next plan step $\sigma_{k,l}$ until after agent A_i has made room on agent A_k's next resource. Hence, agent A_i must start its next plan step $\sigma_{i,j+1}$ before agent A_k can make any progress.

In the following definition, we assume a set $\Pi = \{\pi_1, \ldots, \pi_m\}$ of plans. To single out a particular plan step from a particular plan, we mean that $\sigma_{i,j} = \langle r_{i,j}, \tau_{i,j} \rangle$ is the j^{th} plan step from the i^{th} plan in Π. Resources are referred to in the same manner: resource $r_{i,j}$ is the resource associated with plan step $\sigma_{i,j}$.



(a) The intersection must be cleared by A_i before agent A_k can enter.



(b) Agent A_i must exit the lane to free up capacity for agent A_k.

Figure 5.3: Agent A_i must start its enabling plan step $\sigma_{i,j+1}$ before agent A_k can start its plan step $\sigma_{k,l}$.

Definition 5.2.2 (Enabling plan step)**.** *A plan step $\sigma_{i,j+1}$ is the* enabling plan step *of plan step $\sigma_{k,l}$ if:*

$$r = r_{i,j} = r_{k,l}, \textit{and} \tag{5.3}$$

$$\begin{aligned} &p(r, \sigma_{k,l}) = p(r, \sigma_{i,j}) + cap(r) \wedge \\ &\forall m, n \quad [p(r, \sigma_{i,j}) < p(r, \sigma_{m,n}) < p(r, \sigma_{k,l}) \rightarrow r_{i,j-1} = r_{m,n-1} = r_{k,l-1}] \end{aligned} \tag{5.4}$$

To understand equation 5.4, it is useful to distinguish between the case of unit capacity resources (such as intersections), and resources with capacity two or greater. For the case of unit capacity resources, it simply states that the priority number of step $\sigma_{k,l}$ is one higher than step $\sigma_{i,j}$; the second part of the equation is irrelevant, because there are no plan steps $\sigma_{m,n}$ on resource r with a priority between $\sigma_{i,j}$ and $\sigma_{k,l}$ (which means that the the implication is true because the antecedent is false). For the case of lane resources with a capacity of two or more, the second part of the equation states that all agents that enter r after $\sigma_{i,j}$ but before $\sigma_{k,l}$ must travel in the same direction as A_i and A_k, that is, they must all enter resource r from the same intersection: $r_{i,j-1} = r_{k,l-1} = r_{m,n-1}$. Only then can the situation of figure 5.3(b) arise, where the lane resource is full with agents, and agent A_k with step $\sigma_{k,l}$ can enter the lane once agent A_i enters its next resource.

Definition 5.2.3 (Planstep-Priority Graph)**.** *Given a set of agent plans Π, the* Planstep-Priority Graph *(PPG) is a directed graph $G_S = (S, E_S)$, where the set of vertices is given by:*

$$S = \bigcup_{i=1}^{|\Pi|} \bigcup_{j=1}^{|\pi_i|} \sigma_{i,j} \tag{5.5}$$

That is, there is a vertex for every plan step in every plan. The set of edges is made up

of three parts $E = E_1 \cup (E_2 = E_{2,1} \cup E_{2,2}) \cup E_3$*, where*

$$E_1 = \{(\sigma_{i,j}, \sigma_{i,j+1})\} \tag{5.6}$$

$$E_{2,1} = \{(\sigma_{i,j}, \sigma_{k,l}) \mid r = r_{i,j} = r_{k,l} \wedge p(r, \sigma_{k,l}) = p(r, \sigma_{i,j}) + 1\} \tag{5.7}$$

$$E_{2,2} = \{(\sigma_{i,j+1}, \sigma_{k,l+1}) \mid r = r_{i,j} = r_{k,l} \wedge p(r, \sigma_{k,l}) = p(r, \sigma_{i,j}) + 1\} \tag{5.8}$$

$$E_3 = \{(\sigma_{i,j}, \sigma_{k,l}) \mid \sigma_{i,j} \textit{ is the enabling plan step of } \sigma_{k,l}\} \tag{5.9}$$

Equation 5.6 expresses that there is an edge in E_1 for two successive plan steps of the same plan. Equation 5.7 specifies that if two plan steps $\sigma_{i,j}$ and $\sigma_{k,l}$ from different plans π_i and π_k succeed each other on a resource, then there is an edge in $E_{2,1}$. Because there is no overtaking, plan step $\sigma_{i,j}$ must also *finish* before $\sigma_{k,l}$. Hence, there is an edge in $E_{2,2}$ between the respective successor plan steps $\sigma_{i,j+1}$ and $\sigma_{k,l+1}$ (equation 5.8). Finally, equation 5.9 expresses that an enabling plan step must precede the 'enabled' plan step.

We will shortly prove that a deadlock will occur if and only if there is a cycle in the Planstep-Priority Graph. First, we give our definition of a deadlock.

Definition 5.2.4 (Deadlock)**.** *A* deadlock *is a cycle of agents waiting for each other, where each agent is unable to start its next plan step.*

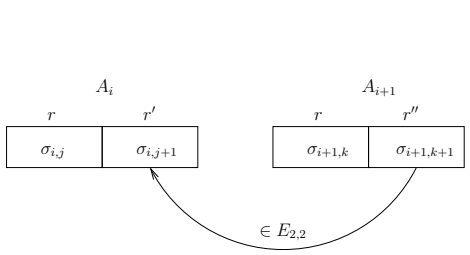
In the literature (see Zöbel [102] for an overview of deadlock literature), four conditions are mentioned that must hold for a deadlock to occur [12], and it is easily shown that these conditions can also hold for agents performing logistical tasks:

1. **Mutual exclusion condition:** Resources cannot hold an infinite number of agents at the same time.
2. **Wait for condition:** Agents occupy a resource while they wait for the next resource in their plan to become available.
3. **No preemption condition:** Agents cannot be removed from their resources; instead, they must drive to their next resource on their own.
4. **Circular wait condition:** When two agents are facing each other, then each is waiting for the other to vacate its resource.

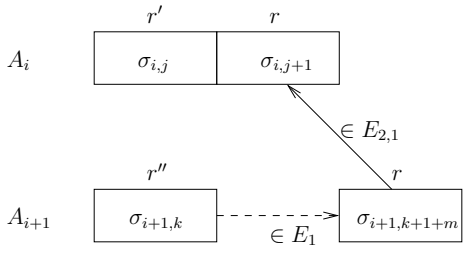
Proposition 5.2.5. *A deadlock occurs if and only if there is a cycle in the Planstep-Priority Graph.*

Proof. Case I: deadlock → cycle in PPG. A deadlock is a cycle $C = (A_1, \ldots, A_m)$ of agents waiting for each other. To prove that this agent cycle implies a cycle in the PPG, we will associate a plan step with every $A_i \in C$, and we will show that for any pair of consecutive agents A_i and A_{i+1}, there is a path in the PPG between their associated plan steps.

Consider some $A_i \in C$ that is currently executing a plan step $\sigma_{i,j}$. The plan step we will associate with A_i in our proof is $\sigma_{i,j+1}$, the *next* step in its plan, which it is unable to start because A_i is waiting for agent A_{i+1}. There are two ways in which A_i can be waiting for A_{i+1}.



(a) A_i is behind A_{i+1} on the resource r. The start of the next step of A_{i+1} must precede the start of the next step of A_i.



(b) A_i is waiting for A_{i+1}, which has one or more plan steps to complete before it can enter r_k.

Figure 5.4: Partial planstep-priority graphs involving agent A_i, which is waiting for A_{i+1}.

Case I.a: A_i is waiting behind A_{i+1}. Note that both agents are currently on the same resource r. Because there is no overtaking, A_{i+1} must exit r before A_i. Hence, the *next* plan step $\sigma_{i+1,k+1}$ of A_{i+1} must precede the next plan step $\sigma_{i,j+1}$ of A_i. By definition 5.2.3, there is an edge $(\sigma_{i+1,k+1}, \sigma_{i,j+1}) \in E_{2,2}$ (note that this edge is in the opposite direction of the agent cycle C). See figure 5.4(a).

Case I.b: A_i is waiting to enter the next resource. From definition 5.1.2, we know that there are two conditions that must be fulfilled before an agent may enter a resource: (i) it must be the agent's turn to enter, and (ii) there must be sufficient capacity to enter. In case the first condition does not hold, agent A_i is waiting for A_{i+1} to *enter* resource r; in case the second condition does not hold, A_i is waiting for A_{i+1} to *exit* the resource.

Case I.b.1: condition (i) If A_i is waiting to enter r, and A_{i+1} hasn't entered yet, then there is a sequence of plan steps $(\sigma_{i+1,k+1}, \ldots, \sigma_{i+1,k+1+m})$ within the plan of A_{i+1}, such that $m \geq 0$, and $\sigma_{i+1,k+1+m}$ is A_{i+1}'s plan step on r (see figure 5.4(b)). In case $m > 0$, then there are edges $(\sigma_{i+1,k+l}, \sigma_{i+1,k+l+1}) \in E_1, 1 \leq l \leq m$, between every pair of consecutive plan steps.

Agent A_i must be the direct successor of A_{i+1} on r; otherwise, the entrance of A_{i+1} would not end the waiting of A_i. This means that there is an edge $(\sigma_{i+1,k+1+m}, \sigma_{i,j+1}) \in E_{2,1}$.

Case I.b.2: condition (ii) Agent A_i can only enter r until after the exit of A_{i+1} has freed up a unit of capacity on r. The fact that A_i is waiting for capacity implies that the resource is currently full (either with one agent, or with multiple agents travelling in the same direction as A_i will be). Because there is no overtaking, the first agent to exit must be the agent that entered first (of the agents currently on r). Hence, if $p(r, \sigma_{i,j+1}) = n$, then $p(r, \sigma_{i+1,k}) = n - cap(r)$. From Definition 5.2.2, we can now infer that there is an edge $(\sigma_{i+1,k+1}, \sigma_{i,j+1}) \in E_3$.

We have shown that if agent A_i is waiting for A_{i+1}, then there is path from the (next) plan step $\sigma_{i+1,k+1}$ of A_{i+1} to the (next) plan step $\sigma_{i,j+1}$ of A_i. Hence, the agent cycle

$C = (A_1, \ldots, A_m)$ implies a cycle (in the opposite direction) in the planstep-priority graph.

Case II: cycle in PPG → deadlock. Let C be a cycle in the PPG, and let A_i be the first agent to reach a plan step $\sigma_{i,j}$ of C. Note that it is impossible to create a cycle in PPG using only arcs in E_1. This implies that A_i's first plan step in C must be preceded (in C) by a plan step $\sigma_{k,l}$ from a different agent A_k. Hence, $(\sigma_{k,l}, \sigma_{i,j}) \in \{E_2 \cup E_3\}$.

We will now explain why $\sigma_{i,j}$ cannot start before $\sigma_{k,l}$, by distinguishing between the cases $(\sigma_{k,l}, \sigma_{i,j}) \in E_2$, and $(\sigma_{k,l}, \sigma_{i,j}) \in E_3$.

Case II.a $(\sigma_{k,l}, \sigma_{i,j}) \in E_2$**:** There are two sub-cases to consider. The first is that agents A_i and agent A_k are on the same resource while executing their previous plan steps, $\sigma_{i,j-1}$ and $\sigma_{k,l-1}$ respectively. Since no overtaking is allowed, agent A_i cannot exit before A_k, and no progress is possible. In case A_i and A_k are not on the same resource, then A_i is waiting to enter the next resource, but agent A_k with plan step $\sigma_{k,l}$ should enter first. According to definition 5.1.2, agent A_i does not have permission to enter the next resource, and no progress is possible.

Case II.b $(\sigma_{k,l}, \sigma_{i,j}) \in E_3$**:** Because $\sigma_{k,l}$ is the enabling plan step of $\sigma_{i,j}$, agent A_k should start $\sigma_{k,l}$ to free capacity for agent A_i to be able to start its next plan step $\sigma_{i,j}$. Hence, condition 2 of definition 5.1.2 is not met until $\sigma_{k,l}$ has started. Hence, agent A_i is not allowed to enter its next resource, and no progress is possible.

From the above cases it follows that $\sigma_{k,l}$ must start before $\sigma_{i,j}$ can start. At the same time, because of the cycle C, $\sigma_{i,j}$ must also start before $\sigma_{k,l}$ can start. This implies that none of the agents associated with the plan steps in C can make progress, which means we have a deadlock situation. □

5.3 Priority-changing algorithms

In this section we will discuss two algorithms that increase the priority of an agent without introducing a deadlock. The idea behind both algorithms is that an agent A_i arrives at a resource r and finds that agent A_j should enter the resource before it. Hence, agent A_j is delayed, whereas A_i is not (or both agents are delayed, but agent A_j more so). By increasing the priority of A_i over A_j (and over other agents between A_j and A_i) on this and subsequent resources, we hope to reduce delay. Agent A_i will certainly benefit from the priority change, and so might agents that succeed A_i in other parts of the infrastructure. Agent A_j might incur even more delay, however, and so will agents that have to wait for A_j (and are unable to increase their priority). In chapter 6, we conduct experiments to evaluate the effects of priority changes.

The first algorithm we will discuss is from Maza and Castagna [60]. Their algorithm is based on the following theorem, which they prove in their paper (we adapted the theorem to fit with our notation):

> A vehicle A_i can inherit the greatest priority on the nodes of the path from r to r' without inducing conflicting priorities, if and only if all the vehicles having to cross this path before A_i are outside the path from r to r'.

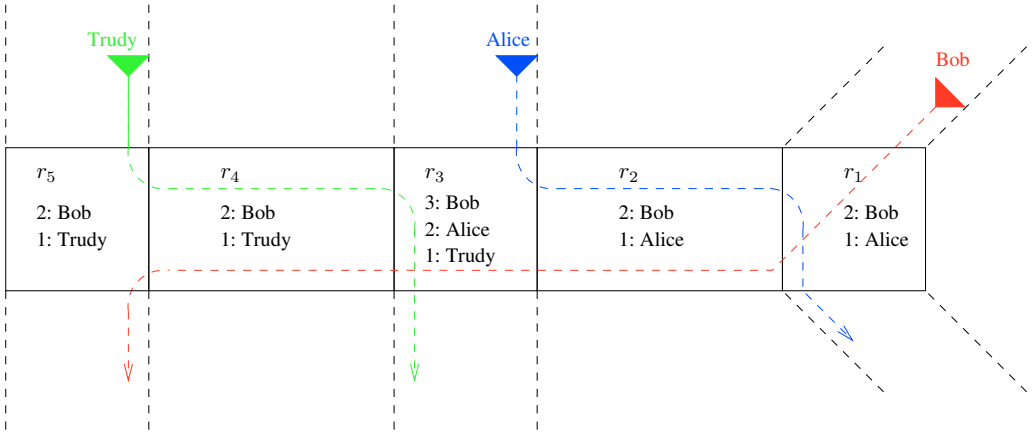


Figure 5.5: Agent Bob can increase its priority over both Alice and Trudy.

The path from r to r' is a sub-sequence of the resources in the plan of agent A_i. Maza and Castagna's algorithm amounts to determining the path from r to r' for which the agent wants to obtain the highest priority. The first resource r is presumably the resource that the agent wants to enter next. In any case, it is a resource where the agent does not have the highest priority. The final resource r' is simply the first resource in its plan (after r) where the agent currently has the highest priority. Agent A_i can be granted the highest priority on all resources between r and r', if and only if all of these resources are currently empty. A more formal description of the algorithm and a proof of correctness can be found in [60].

A disadvantage of the algorithm from Maza and Castagna is that it not only increases the priority of an agent over delayed agents, but also over agents that need not be delayed, as the following example illustrates.

Example 5.3.1. *In figure 5.5 we see an infrastructure with three agents Bob, Alice, and Trudy. According to their plans, Bob is the last agent to enter each of the resources $r_1, \ldots, r_5$. However, Bob is ready to enter r_1 whereas Alice is delayed, so Bob wants to increase his priority. The algorithm from Maza and Castagna specifies that Bob should get the highest priority on all resources $r_1, \ldots, r_5$. This means that Bob should also go before Trudy on resources r_3, r_4, and r_5, even though Trudy may not be delayed. Note that if Bob would only increase his priority over Alice, then a deadlock-free execution would still be possible.*

Example 5.3.1 brings to light another disadvantage of Maza and Castagna's algorithm: if Trudy were already on e.g. resource r_4, then no priority change would be made, even if only changing with Alice is certainly a possibility. Hence, some opportunities to increase priority are not identified by the algorithm of Maza and Castagna. We therefore present a new algorithm that identifies more opportunities for changing the priority, and it limits

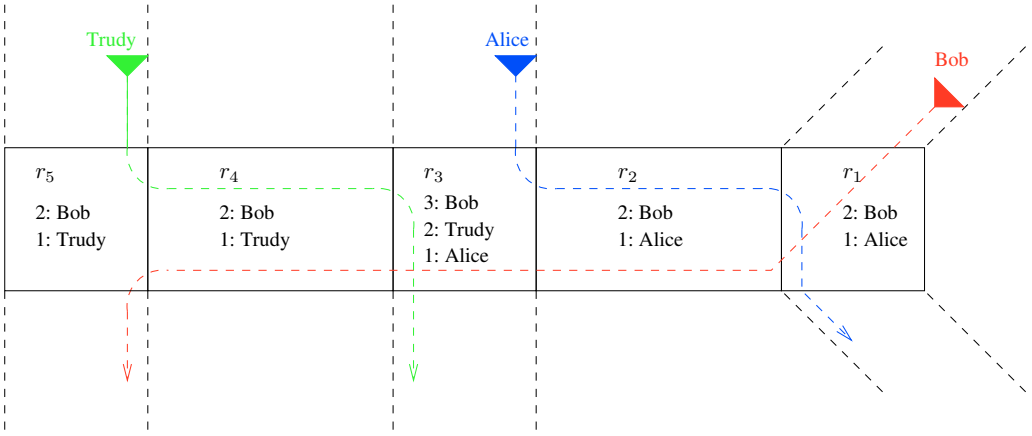


Figure 5.6: To increase his priority over Alice, agent Bob must also increase his priority over Trudy.

priority increases over non-delayed agents[1].

Our algorithm is based on the idea of a *corridor* of resources (as mentioned before): a maximal, uninterrupted sequence of resources that an agent A_i shares with a delayed agent A_j. For each of the resources in the corridor, A_i must increase its priority over A_j if a deadlock is to be prevented. In example 5.1.1, we already saw that A_2 should increase its priority over A_1 for all of the resources r_5, r_4, and r_3. In example 5.3.1, figure 5.5, agent Bob shares the corridor r_1, r_2, r_3 with agent Alice. Of course, it is not always the case that agents A_i and A_j are the only ones to make use of the resources in their corridor. Example 5.3.2 shows how we should deal with other agents.

Example 5.3.2. *Figure 5.6 shows the same situation as in figure 5.5, with one subtle difference: on resource r_3, Alice has now a higher priority over Trudy. Hence, if Bob wants to increase his priority over Alice and no other agents make priority changes, then Bob should also increase his priority over Trudy. What happens in our algorithm is that upon encountering Trudy in resource r_3, she is added to the list of delayed agents. Consequently, Bob now also has to take into account the corridor he shares with Trudy, consisting of resources r_3, r_4, and r_5.*

[1] Unfortunately, we learned of the existence of Maza and Castagna's own alternative algorithm [61] until after our comparison with their original effort. Their alternative algorithm is also aimed at increasing the number of priority changes, and it also makes use of the concept of a shared corridor between agents. A difference between Maza and Castagna's alternative algorithm and the algorithms discussed here is that the former decreases the priority of delayed agents, rather than increasing the priorities of non-delayed agents.

5.3.1 Algorithm description

Algorithm 6 assumes an agent A_i that is executing its plan π_i. It is about to start its next plan step $\sigma_{i,k}$ when it notices that there is still a set A of agents that should enter $r_{i,k}$ (the resource associated with plan step $\sigma_{i,k}$) before it. To make safe priority changes, A_i will step through its plan to determine the corridor it shares with each of the delayed agents. If none of the delayed agents have entered their respective corridors yet, then it is safe for agent A_i to increase its priority over the delayed agents.

The end of the corridor that A_i shares with some agent A_j is the first resource $r_{i,l}$ where A_j is no longer before A_i on $r_{i,l}$. Then, A_j is removed from the set of delayed agents. Once this set is empty, A_i knows that it is safe to increase its priority over all of the delayed agents and for all of the corridors that it recorded during the run of the algorithm. Of course, it can also happen that an agent is added to the set of delayed agents, as explained in example 5.3.2. If A_j is a delayed agent on resource $r_{i,k}$, and agent A_l (that was not previously in the set of delayed agents) is between A_j and A_i on $r_{i,k+1}$, then A_l is added to the set of delayed agents, starting from $r_{i,k+1}$.

Algorithm 6 Increase Agent Priority

Require: agent A_i, plan $\pi_i = (\sigma_{i,1}, \ldots, \sigma_{i,n})$, Planstep-Priority Graph PPG
Ensure: give agent A_i the highest priority on the next resource in its plan π_i, in case this does not create a deadlock

```
 1: k ← nextPlanStepNumber(π_i)
 2: r ← r_{i,k}
 3: n ← entryCounter(r)
 4: A ← {A_j ∈ 𝒜 | n < p(r, σ_{j,x}) < p(r, σ_{i,k})}
 5: M ← ∅
 6: deadlock ← false
 7: while A ≠ ∅ ∧ ¬deadlock do
 8:     r ← r_{i,k}
 9:     A_m ← min_{(A_j ∈ A)} p(r, σ_{j,x})
10:     A ← {A_j ∈ 𝒜 | p(r, σ_{m,x}) ≤ p(r, σ_{j,y}) < p(r, σ_{i,k})}
11:     for all A_j ∈ A do
12:         if locatedAt(A_j, r) then
13:             deadlock ← true
14:             continue
15:     M ← M ∪ ⟨r, A⟩
16:     k ← k + 1
17: if ¬deadlock then
18:     for all ⟨r, A⟩ ∈ M do
19:         increasePriority(A_i, A, r, PPG)
20:     if cycle in PPG then
21:         rollbackPriorityChanges(M, PPG)
```

In line 1, the function nextPlanStepNumber returns the number of the plan step that agent A_i is about to start. Line 2 determines the resource r associated with this plan

step, and in line 3 the function entryCounter returns the number of plan steps that have been started on resource r so far. The set of delayed agents for resource r is determined as follows, in line 4: it consists of those agents with a plan step $\sigma_{j,x}$ on resource r that hasn't been started yet, and should start before $\sigma_{i,k}$, the next plan step of agent A_i. Line 5 initializes a *map* that will hold ⟨resource, delayed agent set⟩ tuples.

The while loop of line 7 iterates over all the steps of A_i's plan, until either there are no more delayed agents, or it turns out that no priority change is possible. First, in line 8, we determine the resource associated with the plan step of this iteration, $\sigma_{i,k}$. Then, in line 9, we find the agent A_m that has the highest priority on r for all delayed agents. In line 10, we determine the new set of delayed agents by finding out which agents have a plan step on r between A_m and A_i. Hence, in this line it is possible that new agents are added to A, and other agents are removed from A.

Then, for each delayed agent A_j, we check whether A_j is currently driving on r using the function locatedAt(A_j, r), in line 12. If this is the case, then A_j has entered the corridor it shares with A_i, and no priority increase is possible for agent A_i. If all delayed agents are not on r yet, then this resource and the current set of delayed agents are added to the map M, in line 15. In line 16, we increment A_i's plan step pointer k. In case the while loop has exited with the flag deadlock still false, then we perform the priority changes, in line 19.

The corridor concept is useful in identifying safe priority changes, but it is not foolproof, as we can find examples in which the priority changes proposed by algorithm 6 can lead to a deadlock. Therefore, in line 20, we perform a check to see if the priority changes have left the Planstep-Priority Graph in a cyclic state. If a cycle is found, we must roll back the priority changes, in line 21. We will now discuss an example of a deadlock situation that can occur if we run algorithm 6 without checking for cycles in the PPG.

Example 5.3.3. *In figure 5.7, we see an infrastructure with five agents Bob, Alice, Trudy, Ken, and Charles. Bob is the second agent to enter resource r_1, but since Alice is delayed, Bob wants to increase his priority. On resource r_2, Bob is the fourth agent to enter r_2, but algorithm 6 only requires him to increase his priority over Alice. On resource r_3, on the other hand, Bob should also go ahead of Trudy, who is set to enter r_3 after Alice but before Bob.*

As a result of these priority changes, Trudy will wait for Bob to enter r_3. In turn, agent Charles cannot enter r_4 before Trudy, and on resources r_5 and r_2 agent Ken has to wait for Charles. However, Bob has to wait for Ken on resource r_2, because he only increased his priority over Alice. The priority changes of agent Bob have therefore created a circular wait: Ken waits for Charles, Charles for Trudy, Trudy for Bob, and Bob for Ken.

Proposition 5.3.4. *The run-time complexity of algorithm 6 is $O(|\mathcal{A}||R| + |S|)$, where S is the set of all plan steps.*

Proof. The while loop of line 7 runs for at most $|R|$ iterations, and within this loop there is a for-loop (line 11) that iterates over the set of delayed agents, in $O(|\mathcal{A}|)$ time. Other computations within the while loop do not require more than $O(|\mathcal{A}|)$ time.

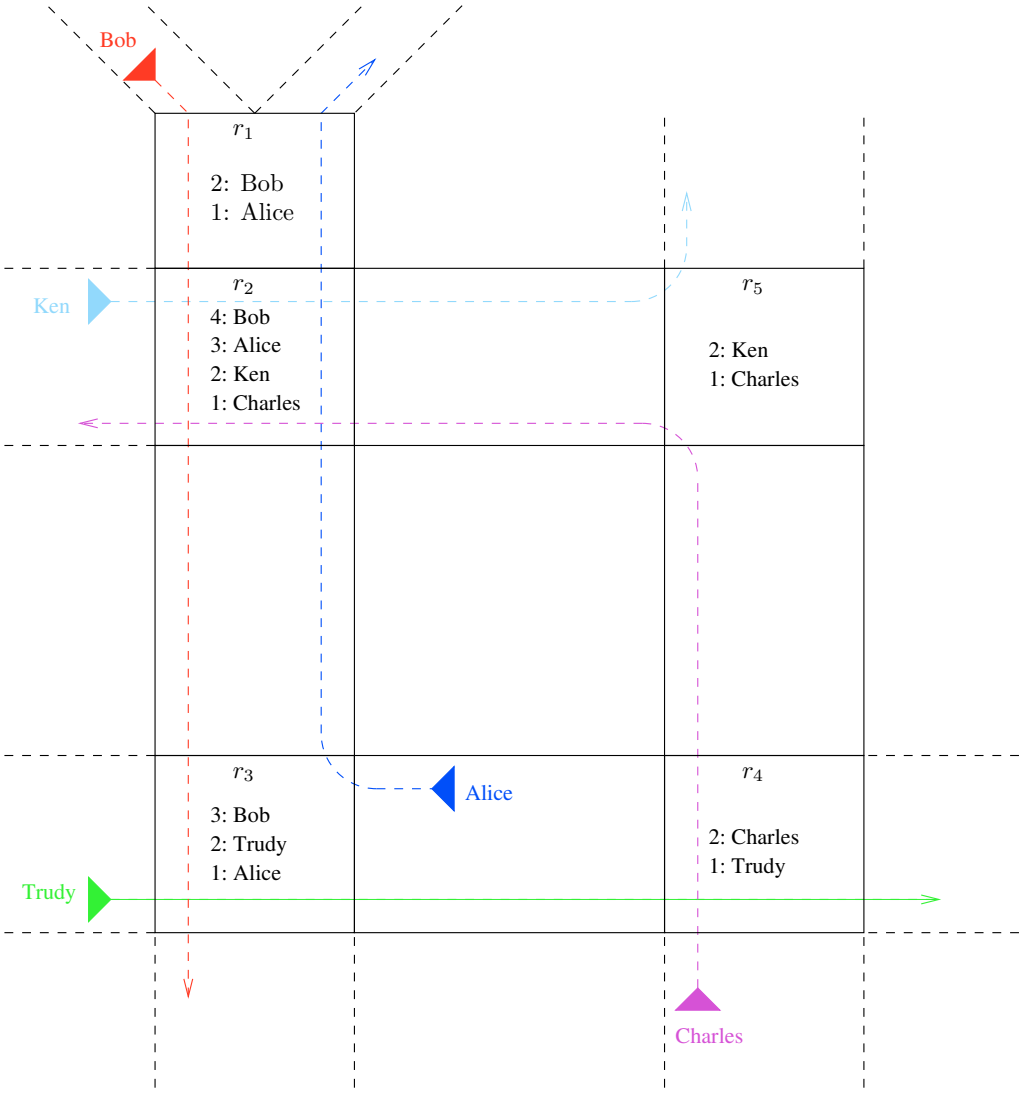


Figure 5.7: If agent Bob increases his priority over Alice, then a deadlock will occur.

In line 20 we have to check whether the Planstep-Priority Graph contains a cycle. Cycle-detection in a graph requires linear time in the number of vertices of the graph, and the set of vertices of the PPG is the set S of plan steps. Hence, cycle detection can be done in $O(|S|)$ time, for a total run-time complexity of $O(|\mathcal{A}||R| + |S|)$. □

5.3.2 Extending the IAP algorithm

With our IAP algorithm, we aim to construct an algorithm that improves over Maza and Castagna's priority-changing algorithm by (*i*) only increasing an agent's priority over delayed agents, and by (*ii*) relaxing the set of constraints that must hold for a priority change to be allowed. From example 5.3.3 above, we can infer that these two aims are sometimes conflicting.

In example 5.3.3 and figure 5.7, agent Bob modestly increases his priority only over Alice, and not over agents Ken and Charles (in resource r_2), who are not delayed, as far as Bob knows. However, Bob could increase his priority over all agents — which is what Maza and Castagna's algorithm would do — because all of the resources in Bob's plan are currently empty. Hence, the stricter rule of Maza and Castagna's algorithm can sometimes find a conflict-free priority change when our IAP algorithm does not find any solution.

We will now show how we can extend algorithm 6 to encompass the algorithm of Maza and Castagna. The idea behind the extension is to reduce the 'modesty' of the agent in algorithm 6, in case that modesty leads to a cycle in the Planstep-Priority Graph. Effectively, the modesty of the agent can be regulated by line 9, in which we specify the agent A_m with the lowest priority number (i.e., A_m is the agent with the highest priority) that we want to increase the priority over: in line 10, we derive the set of delayed agents as all the agents that have to enter the resource r between A_m and agent A_i, the agent requesting the priority increase. A difference between our algorithm and Maza and Castagna's algorithm is that in the former algorithm, the agent A_m is simply the agent that currently has the highest priority on resource r (i.e., the next agent to enter r), whereas in IAP there may be several agents that will enter r before A_m will enter.

We can reduce the modesty of the agent A_i by iteratively lowering the priority number of the agent A_m, until we conclude that the proposed priority change does not create a cycle in the Planstep-Priority Graph, or until we conclude that with the reduced modesty, no priority change is possible. The latter situation can occur if we find that A_m already occupies one of the resources in A_i's path (see line 12). Algorithm 7 below is an extension of the IAP algorithm that includes a modesty-level parameter α.

Algorithm 7 differs from algorithm 6 in two ways. First of all, we make use of the modesty parameter α in line 10, and second, we return a value of 0 or -1. A return-value of 0 indicates that the algorithm returned either after successfully performing a priority change, or after concluding that no priority change is possible. The algorithm returns -1 if the proposed priority change resulted in a cycle in the PPG. These return values might be used by another algorithm, such as algorithm 8 below.

Algorithm 8 simply calls the E-IAP algorithm until the latter returns 0, which happens in case a priority change has been made, or no priority change proved possible.

Algorithm 7 Extended Increase Agent Priority (E-IAP)

Require: agent A_i, plan $\pi_i = (\sigma_{i,1}, \ldots, \sigma_{i,n})$, Planstep-Priority Graph PPG, modesty-level parameter α

Ensure: give agent A_i the highest priority on the next resource in its plan π_i, in case this does not create a deadlock

```
1: k ← nextPlanStepNumber(π_i)
2: r ← r_{i,k}
3: n ← entryCounter(r)
4: A ← {A_j ∈ 𝒜 | n < p(r, σ_{j,x}) < p(r, σ_{i,k})}
5: M ← ∅
6: deadlock ← false
7: while A ≠ ∅ ∧ ¬deadlock do
8:     r ← r_{i,k}
9:     A_m ← min_{(A_j ∈ A)} p(r, σ_{j,x})
10:    A ← {A_j ∈ 𝒜 | max(0, p(r, σ_{m,x}) − α) ≤ p(r, σ_{j,y}) < p(r, σ_{i,k})}
11:    for all A_j ∈ A do
12:        if locatedAt(A_j, r) then
13:            deadlock ← true
14:            continue
15:    M ← M ∪ ⟨r, A⟩
16:    k ← k + 1
17: if ¬deadlock then
18:    for all ⟨r, A⟩ ∈ M do
19:        increasePriority(A_i, A, r, PPG)
20:    if cycle in PPG then
21:        rollbackPriorityChanges(M, PPG)
22:        return -1
23: return 0
```

Algorithm 8 Iterative Increase Agent Priority

Require: agent A_i, plan $\pi_i = (\sigma_{i,1}, \ldots, \sigma_{i,n})$, Planstep-Priority Graph PPG

Ensure: give agent A_i the highest priority on the next resource in its plan π_i, in case this does not create a deadlock

```
1: α ← 0
2: value ← −1
3: while value < 0 do
4:     value ← E-IAP(A_i, π_i, PPG, α)
5:     if value = −1 then
6:         α ← α + 1
```

With each call, the value of α is incremented. For a value of $\alpha = 0$, algorithm E-IAP equals algorithm IAP, whereas for sufficiently high values of α, E-IAP produces the same results as the algorithm by Maza and Castagna.

In this section, we showed how we can reduce the modesty of a priority-requesting agent by also allowing priority changes over agents that are not delayed. If an agent's modesty is fully reduced, we obtain Maza and Castagna's algorithm in the sense that an agent always requests the highest priority for all of his remaining resources. Although algorithm 8 is therefore more general than algorithm 6, there are three reasons not to recommend algorithm 8 over algorithm 6 unreservedly:

1. Increasing an agent's priority over other non-delayed agents may be unfair or otherwise undesirable.

2. The number of priority changes 'gained' by algorithm 8 could be small; this should be verified empirically.

3. The effects of a priority change on the delays of the agents — especially in case only an 'immodest' priority change is feasible — is unclear and should be investigated.

Algorithm 8 therefore constitutes an interesting method of extending the generality of algorithm IAP to encompass the algorithm from Maza and Castagna, but for practical purposes (also considering CPU time usage) algorithm IAP may be the better choice, and therefore we use it in our experiments in chapter 6.

5.4 Concluding remarks

In this chapter we showed how unexpected incidents can disrupt the execution of route plans. In particular, we considered incidents where vehicles are temporarily immobilized, which can cause congestion in case overtaking is not allowed. Moreover, if some agents are delayed and others are not, then oblivious execution of the original set of agent plans can result in a deadlock situation, for example if two agents meet each other head-on on a lane resource.

Fortunately, simple schedule repair techniques exist that ensure that all agents can reach their destinations safely. These repair techniques make use of the concept of the *priority* of the agents: from the original set of plans, we can derive the priority of each agent on each resource, and by maintaining these priorities during plan execution, we can ensure deadlock-free operation. However, maintaining priorities implies that sometimes one has to wait for a delayed agent. If we try to increase the priority of an agent in favour over one or more delayed agents, we have to be careful not to introduce any deadlocks.

The first algorithm to increase agent priority is from Maza and Castagna [60], but to ensure deadlock-free execution, the conditions under which priority changes are allowed are very strict. Moreover, their algorithm increases the priority of an agent not only over delayed agents, but also over other agents that happen to make use of the same resources. We presented an alternative priority-changing algorithm that only increases priority over delayed agents, and the conditions under which we allow a priority change

are more lenient, so we expect that our algorithm can perform more priority changes than Maza and Castagna's algorithm. In the next chapter, we will present experiments that should confirm this expectation. Perhaps more importantly, we also investigate whether these priority changes can reduce the total delay in the system, and whether more changes equals less delay.

Chapter 6

Usability of Prioritized Route Planning

In the previous chapter we showed how deadlock-free execution of multi-agent route plans can be ensured by maintaining the priority of the agents at each of the resources. What is harder to prove is that multi-agent route plans can be executed with little delay, in case unexpected incidents make the original plans (schedules) impossible to adhere to. In this chapter we will therefore present experiments that investigate the relation between agent delay and the frequency and severity of vehicle incidents (incidents that temporarily immobilize agents).

An agent can incur delay because it suffers an incident, but also because of the mechanism that is used to prevent deadlocks. To prevent deadlocks, an agent may not enter a resource before all higher-priority agents have entered it. If any of the higher-priority agents are delayed, then an agent must wait. In the previous chapter we also presented two mechanisms that can increase the priority of non-delayed agents, so they do not have to wait for delayed agents with a higher priority. Having its priority increased will reduce an agent's delay, but it may increase the delay of other agents. Therefore, we will investigate the impact that priority changes have on the global delay.

The reason for choosing a planning approach to the multi-agent routing problem (rather than a reactive approach that chooses an agent's next action only on its view of the current situation) is that higher-quality (or lower-cost) agent plans can be obtained. In chapter 3, we introduced two ways to measure multi-agent plan cost. The first measure is to sum the plan costs of all agents, where the cost of one agent plan is the difference between its start time and its finish time; the second measure is the *makespan*, i.e., the difference between the earliest agent start time and the latest agent finish time. The prioritized planning approach that we present in this thesis allows us to find an optimal (minimum-cost) plan for a single agent, given the plans of higher-priority agents. However, in chapter 3 we also showed that a set of individually optimal route plans, obtained using the prioritized planning approach can result in a multi-agent plan with a cost of $O(|\mathcal{A}|)$ times the cost of an optimal multi-agent plan.

In this chapter we will investigate how the global plan quality differs between randomly assigned priorities. The smaller the difference between the best and the worst assignment of priorities found, the more confidence we have that an arbitrary priority assignment will result in a global plan that is not much more costly than an optimal global plan — under the assumption that the optimal *priority* leads to a close-to-optimal global plan. We will see that the impact of the priority ordering on the global plan quality depends not only on the type of infrastructure the agents travel on, but also on the distribution of start and destination locations over the infrastructure.

This chapter is organized as follows. In section 6.1 we will analyze the *robustness* of multi-agent route plans in the face of unexpected incidents. First, we present an experiment in which no priority changes are allowed, in the setting of taxi route planning on a (model of a) real airport. Not allowing any priority changes can be viewed as the most basic way to deal with incidents, and we investigate how much delay agents incur using this baseline method. Second, we will experiment with the two algorithms from chapter 5 that increase the priority of non-delayed agents, and see whether they constitute an improvement over the baseline, no-priority-changes method. In section 6.2, we will investigate the effect of different agent orderings on the global plan quality. A first batch of experiments is run on the airport infrastructure, a second batch of experiments is run on random infrastructures, to investigate the relation between global plan quality, agent priority ordering, and infrastructure characteristics.

6.1 Robustness of route plans

Recall from chapter 1 that we define the robustness of a set of plans in terms of the delay that agents suffer (when unexpected incidents disrupt the execution of their plans) *given* a set of available plan repair techniques. In this section we wish to evaluate the robustness of agent route plans, and how it depends on:

1. the particular plan repair technique used,
2. the topology of the infrastructure,
3. the frequency and severity of incidents.

In the previous chapter we discussed three repair techniques; all three techniques modify only the timing of the agent plans, so we can term them *schedule repair* techniques[1], since the sequences of resources to be visited remain unchanged. All three repair techniques use the priority of the agents on the resources, as explained in chapter 5 (for example in the concluding section 5.4). The simplest repair technique, developed by Maza and Castagna [58], is to maintain the priorities during plan execution, even if some agents are delayed. This means that an agent is only allowed to enter a resource once its turn has come, and it may have to wait for delayed agents that have a higher priority. The second repair algorithm, also from Maza and Castagna [60], aims to reduce the delay of

[1]The difference between planning and scheduling — and hence between plan repair and schedule repair — is that planning is often associated with choosing actions, and scheduling with locating actions in time (cf. [86]).

agents that are on time, by increasing their priority over agents that are already delayed. Of course, the latter agents may therefore experience more delay because of their lowered priority, but our expectation is that the total delay in the system is reduced when a non-delayed agent is given priority over a delayed one. The third repair algorithm allows more priority changes, and we therefore expect it to result in lower delay values.

This section is organised as follows: in section 6.1.2 we evaluate robustness using only the simplest repair mechanism that does not make priority changes. As such, these experiments should rebut the claim from Le-Anh and De Koster [47] that "*a small change in the schedule may destroy it completely*"[2], although one can assume that this remark was made in ignorance of the existence of the simple schedule repair mechanism from Maza and Castagna. In section 6.1.2 we evaluate robustness not only for "small changes" in the schedule, but also for more frequent and severe incidents. We also investigate whether the robustness of agent plans depends on the infrastructure topology. Among the types of infrastructures we consider are random networks, grid-like networks, and we also have an airport infrastructure. In section 6.1.3, we investigate whether we can improve the performance of the multi-agent system by using priority-changing algorithms. Of course, this section is also dedicated to the comparison between our priority-changing algorithm and the one from Maza and Castagna. First, in section 6.1.1, we describe our experimental setup, and we describe the type of incidents we consider.

6.1.1 Experimental setup

Figure 6.1 illustrates the setup of our experiments. Each agent A_i has a start location s_i and a destination location d_i, and they use algorithm 3 from chapter 4 to find a conflict-free route plan. We let the agents plan in an arbitrary order, each agent respecting the plans of the previous agents. After an agent has made its route plan, the free time window graph is modified to reflect that the set of free time windows for the next agent has changed. After all agents have made their plans, we randomly generate vehicle incidents for each of the agents, by determining for each plan step whether the agent will suffer an incident while performing that plan step. We generate incidents according to two parameters: the incident *rate* and the incident *duration*. The incident duration is simply the number of seconds that an incident lasts, i.e., how long an agent must stand still. The incident rate is a value between zero and one, and it determines the frequency with which agents receive incidents. If the value of the incident rate parameter is 0.1, then an agent can expect to suffer one incident for every 10 plan steps[3]. The values for incident duration and incident rate are fixed for a single experiment run, in the sense that all incidents last equally long, and all agents are equally likely to suffer incidents.

Once each agent has a plan and a set of incidents, we can start the simulator in which each agent tries to execute its plan. The simulator works by dividing time into small steps of e.g. 0.1 seconds, and in each time step, each agent has the opportunity to perform a small part of its plan. For example, if the agent plans to move along resource r with a speed of 10 metres per second, then in one time step it will advance one metre. An agent's

[2] Although they incorrectly stated that Kim and Tanchoco remarked this of their approach [41].

[3] The number of plan staps in a plan depends on the size of the infrastructure. On the Schiphol airport infrastructure, some plans have up to 100 plan steps, while for the randomly generated infrastructures plans can have up to 50 steps.

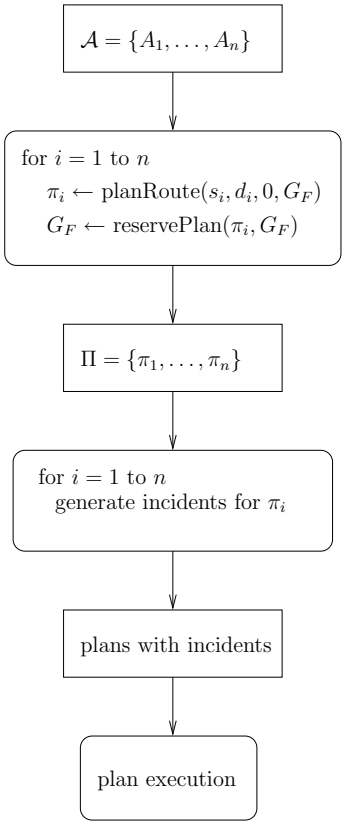


Figure 6.1: The setup of our experiments.

movements are restricted, however, by the conditions specified in chapter 5: an agent may not overtake, or collide with other agents, and it may not enter a resource when it is not yet its turn or when the resource is full. This means that if there is an agent directly in front of the agent, then the agent cannot go faster than the leading agent. If the leading agent is suffering an incident and standing still, then the agent will also have to stop. Eventually, all agents will reach their destination locations, and there we can measure how much the agent has been delayed with regard to its original schedule. We will now discuss how we measure delay.

Measuring delay

As mentioned before, we only consider *vehicle incidents*: incidents that immobilize a vehicle for a certain period of time. Also, because we do not allow overtaking in our experiments, when one agent suffers a temporary breakdown, the agents behind it must also wait for the first agent's recovery. We say that an agent accumulates *incident delay*

either when it is suffering an incident, or when it's stuck behind an agent that is accumulating incident delay. There is a second source of delay, however, and it stems from the mechanism that is used to prevent deadlocks. An agent accumulates *mechanism delay* if it is waiting to enter a resource and it is not allowed to enter yet because a higher-priority agent should enter the resource first. An agent also incurs mechanism delay if it is stuck behind an agent that is accumulating mechanism delay. The *total delay* of an agent is its finish time minus its planned finish time. As incident delay and mechanism delay are the only two sources of delay, we have the following inequality.

$$\text{total delay} \leq \text{incident delay} + \text{mechanism delay}$$

The total delay can be less than the sum of the incident delay and mechanism delay in case the agent has the opportunity to make up some time. To see how an agent can make up time, consider the following example: an agent A_1 plans to traverse resource r in the interval $[10, 70)$, while the minimum travel time of resource r is 20. Hence, A_1's planned traversal of r is slower than the fastest traversal of r, e.g. because it has to wait until time 70 before it may enter the next resource in its plan. As a result, the agent has 40 time units of slack in its traversal of resource r, that it might use to make up for delays. Suppose for example that during the execution of its plan the agent suffers an incident at time 40, when it is halfway along resource r. If the incident lasts for 30 time units, then it can get going again at time 70. To make up time, the agent will traverse the remainder of the resource at full speed. Because there is still half the resource to go, the agent needs 10 more time units to reach the end of r, at time 80. Hence, the agent's total delay is 10, even though it has an incident delay of 30.

In our experiments, we have focussed on the mechanism delay. Because we do not consider the possibility of re-routing in this thesis, it is unlikely that we can reduce the incident delay: if an agent suffers an incident, or an agent in its path suffers an incident, there is nothing the agent can do to avoid the delay. The mechanism delay, however, is a measure of how effective the priority-maintaining approach is. Maintaining priorities is one way of preventing deadlocks, but the *cost* of this approach is the mechanism delay. If this delay is prohibitively high, then route planning in combination with maintaining priorities is not a viable approach to the multi-agent motion problem in dynamic environments. The total delay is not a good measure of the viability of the approach for the same reason that incident delay is not.

6.1.2 Robustness under fixed priorities

We will start this section with an experiment of airport taxi routing; later, we will see if the results obtained for the taxi routing case also hold for agent route planning on random infrastructures. For the taxi routing experiments, we used a model of Amsterdam Schiphol Airport, an infrastructure consisting of around a thousand resources (see figure 6.2), including gates, runways, taxiways, and aprons (the tarmac in front of the gate). Of the aforementioned resources, we treat taxiways and aprons as lane resources, and gates and runways as intersections. The reason we model a runway as an intersection is that only one agent may land or take off at the same time.

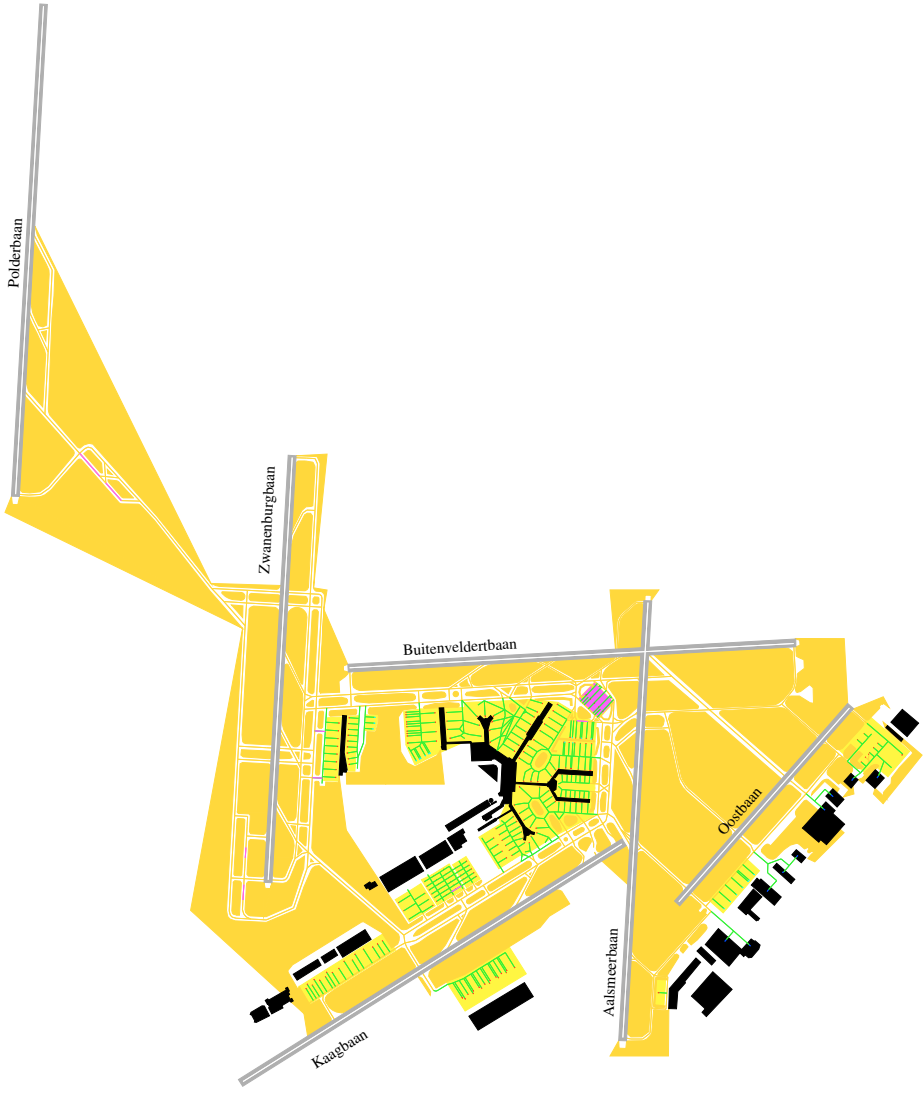


Figure 6.2: The infrastructure of Amsterdam Airport Schiphol, which consists of 1016 resources.

rate	duration	
	30s.	120s.
0.01	LL	LH
0.1	HL	HH

Table 6.1: Combinations of values for incident rate and duration.

All agents in our experiments have a runway as a start location, a gate as a first destination location, and a runway as a second and final destination location. From figure 6.2, we can see that Schiphol has six runways, of which the smallest, the Oostbaan, is rarely used in practice. A common runway configuration at Schiphol is either to have two runways open for arrival and one for departure (during an arrival peak), or to have two runways open for departure and one for arrival (during a departure peak). A *mixed-mode* runway configuration, in which a runway can be used both for arrival and for departure, is not used at Schiphol. Although a mixed-mode operation allows for slightly higher runway utilization, it involves more complicated air traffic control [99].

In our experiments, each runway can be used in any of the four modes arrival, departure, mixed, or closed. Hence, we allow any kind of runway configuration, as long as there is at least one runway open for departure, and one for arrival (it may be the same runway). The determination of the runway configuration is part of the generation of a random problem instance (i.e., we determine both a random runway configuration together with random start and destination locations for the agents). We used between 100 and 400 agents for our experiments, each with a preferred starting time of zero. However, because each agent must start its plan at an arrival runway, only the first agents to plan can actually start at or close to their intended start time.

One experiment run is characterized by the following parameters: (*i*) the number of agents, (*ii*) the incident rate, and (*iii*) the incident duration. For the number of agents, we go from 100 to 400 agents in 41 steps. As Schiphol typically processes around 600 aircraft per day, trying to handle 100 aircraft at the same time will lead to a mildly congested airport, while handling 400 aircraft at the same time should result in a very congested airport. For the incident rate and duration parameters, we mainly used the values shown in table 6.1. The length of an agent's plan can vary from around 25 to 100 plan steps (depending on the runway configuration, of course). With an incident rate of 0.01, we can expect some but not all agents to suffer an incident, while an incident rate of 0.1 will practically guarantee that all agents will have one or more incidents. To put the incident duration into context, an agent plan can last as little as five minutes to as long as an hour. Agents with an hour-long plan are typically those with a low priority (they have to plan around the movements of other agents), and those that have both a remote arrival runway and a remote destination runway. For each combination of the parameters, we performed 50 runs. Hence, we conducted a total of 8200 experiments.

Schiphol results

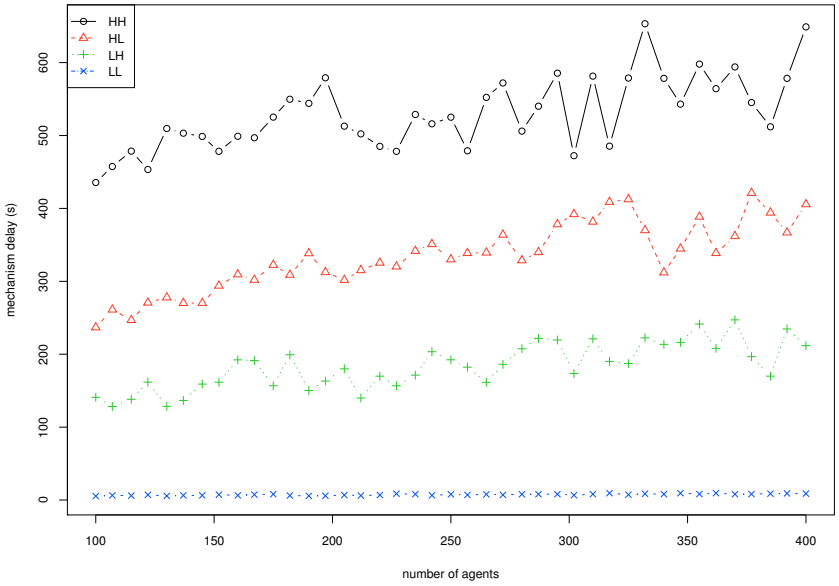
In figure 6.3, we see how the mechanism delay depends on the number of agents in the system, for each of the four incident-parameter combinations. One point in the

graph represents the average value of 50 experiment runs with the same combination of parameters; the value of one experiment run, in turn, is the average of the mechanism delay of all (100 to 400) agents in the experiment. Figure 6.3(a) shows the *absolute* mechanism delay per agent, which slowly increases as the number of agents in the system increases. Figure 6.3(b) shows the *relative* mechanism delay, which for agent A_i is the mechanism delay from A_i divided by $c(\pi_i)$, the cost of agent A_i's plan. The relative mechanism delay decreases with the number of agents in the system, which shows that the average plan length increases as the number of agents in the system increases.

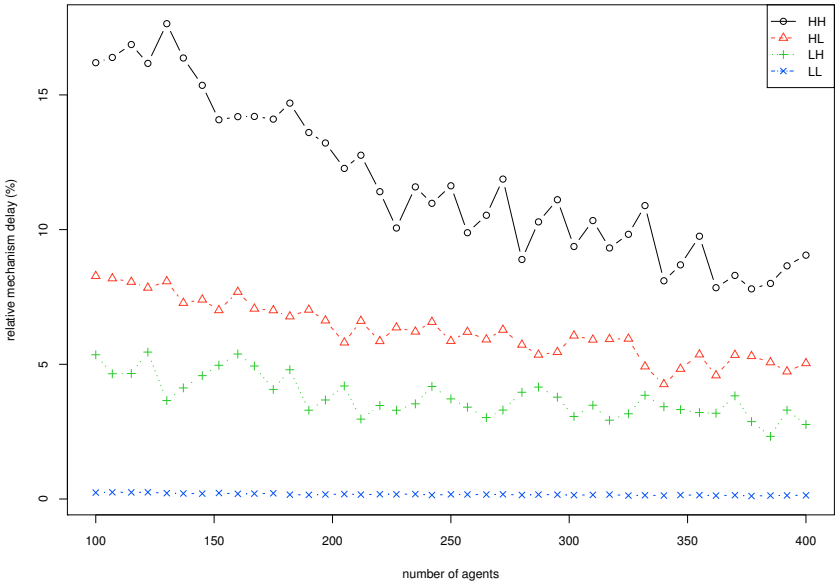
From figure 6.3 we can draw a number of conclusions. First of all, for a small number of short incidents (incident rate = 0.01, incident duration = 30 seconds) there is hardly any mechanism delay. Hence, there is no evidence to suggest that a small change in the schedule can destroy it completely. Also, it is not just mechanism delay that is low. If we look ahead to figure 6.5(a), we see that the total delay (i.e. the delay that each agent has at its destination location), is even lower than the mechanism delay. Hence, any mechanism delay and incident delay that agents incur on the way will have been mostly made up by the time they reach their respective destination locations. In other words, for a small number of small incidents, any delays are almost fully absorbed by the slack in the agent plans.

The second conclusion that we can draw from figure 6.3 is that if either the incidents are long (duration = 90s.), or if they occur frequently (rate = 0.1), then the mechanism delay is around 2 to 5 minutes, although this still only amounts to 3 to 8 percent of an agent's plan length. Third, for a large number of long incidents (rate = 0.1, duration = 90 seconds), the mechanism delay is significantly higher, though on average never more than 18% in this set of experiments. A fourth conclusion that we can draw from the zigzag motion of the HH-line is that the mechanism delay is much less predictable for a large number of long incidents. This conclusion is confirmed by figure 6.4, which displays the 95% confidence intervals for the data points of figure 6.3(b). The meaning of the confidence intervals is that the average mechanism delay per agent will fall within the specified interval with a probability of 95%. As can be seen from figure 6.4, the confidence intervals are quite large.

Figure 6.5 shows how incident delay, mechanism delay, and total delay relate to each other. In case of a small number of short incidents (figure 6.5(a): rate = 0.01, duration = 30s.), the total delay is almost zero, while the incident delay and mechanism delay are somewhat higher. As mentioned before, the slack in the plans of the agents almost fully absorbs the delays encountered along the way. If either the incident duration is increased (figure 6.5(b)), or the incident rate is increased (figure 6.5(c)), then the mechanism delay is significantly higher than either the incident delay or the total delay. Moreover, for these settings the mechanism delay is the least predictable of all four incident-parameter combinations. In both figures 6.5(b) and 6.5(c), the total delay is about equal to the incident delay. In case of a large number of long incidents (figure 6.5(d): rate = 0.1 and duration = 90s.), the mechanism delay is about the same as the incident delay, but the total delay is significantly higher than either, although not as high as the sum of both. The fact that the total delay is higher than both incident delay and mechanism delay does not mean that we can conclude that agents are unable to exploit the slack in their plans as efficiently as in the other settings, because the total delay time that is made up



(a) Absolute mechanism delay per agent



(b) Relative mechanism delay per agent

Figure 6.3: Mechanism delay per agent on the Schiphol airport infrastructure, for the following parameter values: LL = (0.01, 30s.), LH = (0.01, 90s.), HL = (0.1, 30s.), HH = (0.1, 90s.).

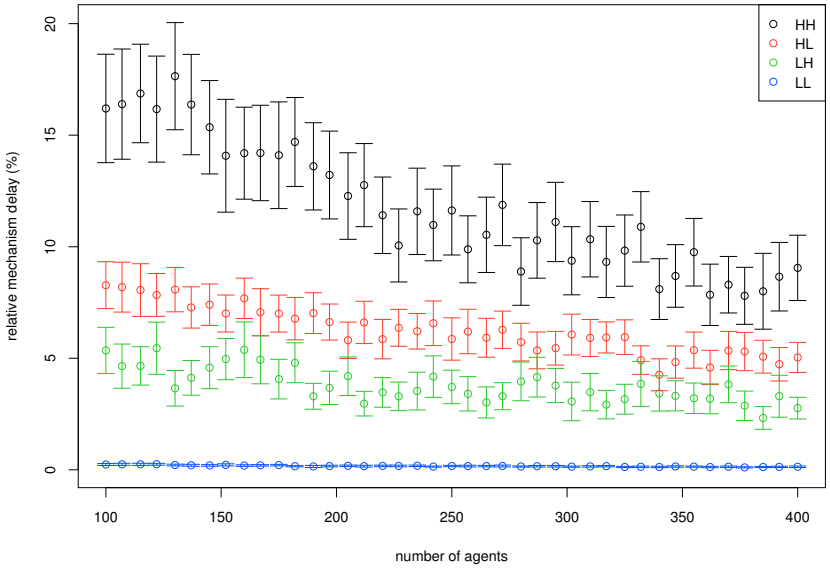


Figure 6.4: 95% Confidence intervals for the relative mechanism delay.

for is still the largest in figure 6.5(d). For example, if we compute the sum

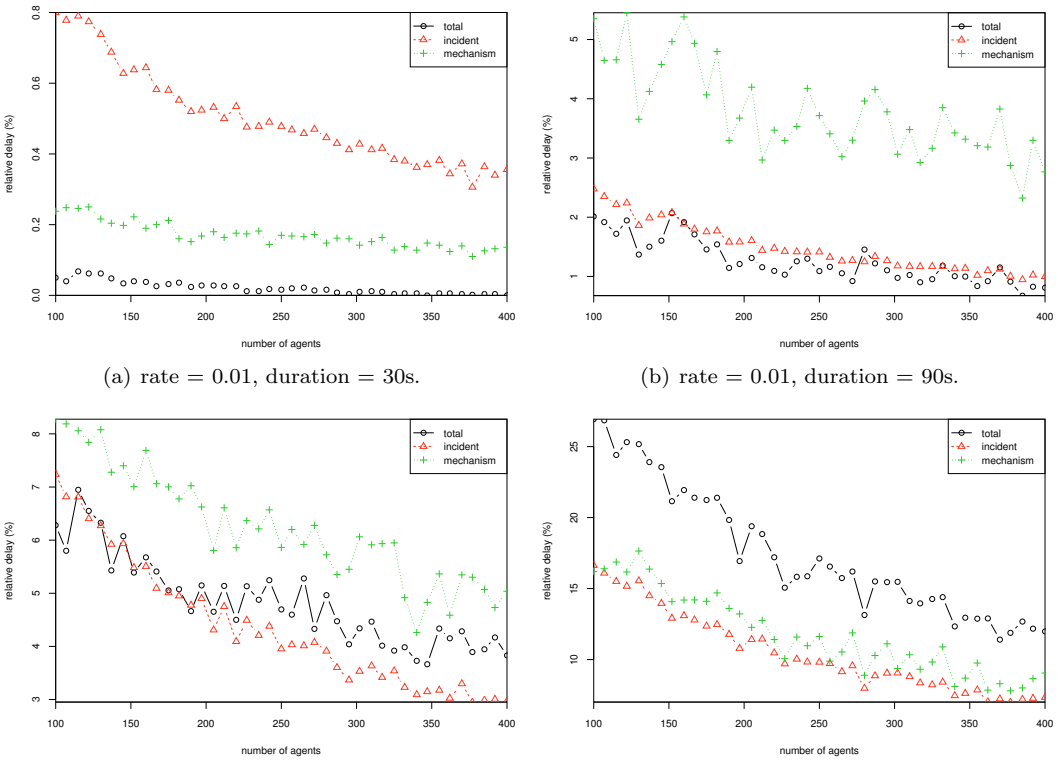
$$\text{recovered delay} = (\text{incident delay} + \text{mechanism delay}) - \text{total delay}$$

for each sub-figure at 100 agents, then we obtain 6.5(a) $\approx 1\%$, 6.5(b) $\approx 6\%$, 6.5(c) $\approx 6\%$, and 6.5(d) $\approx 7\%$ of the average agent plan length.

In the previous figures, we have only considered experiments with either a very low incident rate (0.01) or a very high incident rate (0.1). In figure 6.6, we show a set of experiments in which the number of agents was fixed at 250, the incident duration at 90 seconds, and the incident rate varied from 0.01 to 0.2, in twenty steps. The most interesting result from figure 6.6 is that the mechanism delay initially rises quickly, but that it starts to level out when the incident rate hits 5%. One reason for this might be that as incidents are more common, all agents either have to wait behind a broken-down agent, or suffer an incident themselves. Hence, the original multi-agent schedule more or less gets pushed back in time by the incidents. The total delay does continue to increase, more or less keeping pace with the incident delay.

Results for random graphs

The results on the Schiphol infrastructure are very promising, and now we will investigate whether these results also hold for other infrastructures. In this section, we will present experiments that use infrastructures that we randomly generated. We generated three types of random infrastructures, that differ in the way nodes (intersections) are connected (via lanes). First of all, we created 'fully random' graphs, by first creating a random



(a) rate = 0.01, duration = 30s.

(b) rate = 0.01, duration = 90s.

(c) rate = 0.1, duration = 30s.

(d) rate = 0.1, duration = 90s.

Figure 6.5: Relative total delay, incident delay, and mechanism delay on Schiphol airport infrastructure for 250 agents and various incident-parameter settings.

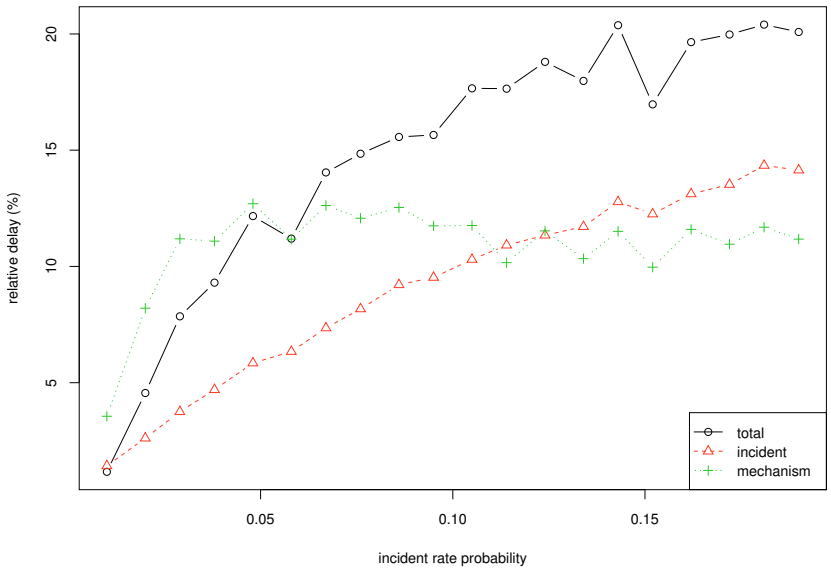


Figure 6.6: Relative delay for the Schiphol infrastructure for varying incident rates, with 250 agents and an incident duration of 90 seconds.

spanning tree[4], and then adding edges between randomly chosen, as yet unconnected nodes (until the desired number of edges in the graph has been reached). Second, we created two-dimensional lattice infrastructures in which each intersection is connected to four other intersections (the lattice networks are basically toroidal grids, with the additional feature that the length of the lanes can vary). Third, we created small-world networks (cf. [43]), which are the same as lattice networks, except that each node has one additional connection to a randomly chosen long-distance node.

To determine minimal travel times for each of the edges in the random graphs, we used the open-source software graphviz[5] to create a two-dimensional layout. For example, figure 6.7 displays a 'fully' random graph[6] on 15 nodes and 30 edges. From the length of the edges in the layout, we determined the travel time of the corresponding lane resource. We set the length of the median-length edge to 150 metres, and with a maximum agent speed of 40 kilometres per hour (we used the same agent speed for the Schiphol experiments), this results in a minimum travel time of 13.5 seconds for the median-length lane resource.

For each type of infrastructure, we created 100 different graphs: 100 random graphs on 180 nodes and 300 edges, 100 lattice graphs with 144 nodes, and 100 small-world networks

[4]We create a random spanning tree by iterating through the set of nodes, and in iteration i we connect the node with index i with a randomly chosen node with index smaller than i.

[5]`http://graphviz.sourceforge.net/`

[6]From now on, we will refer to fully random graphs simply as random graphs, and to the other graphs as lattice networks and small-world networks.

also with 144 nodes. Each infrastructure consists of around 400 to 450 resources (intersections and lanes together). As in the Schiphol experiments, the number of agents varied from 100 to 400, the incident duration was 30 or 90 seconds, and the incident rate was 0.01 or 0.1. Performing 50 runs for each combination of the parameters resulted in 8200 experiments for each of the infrastructure classes. To assign a particular infrastructure to experiment number x, the number of the graph was chosen as $x \mod 100$.

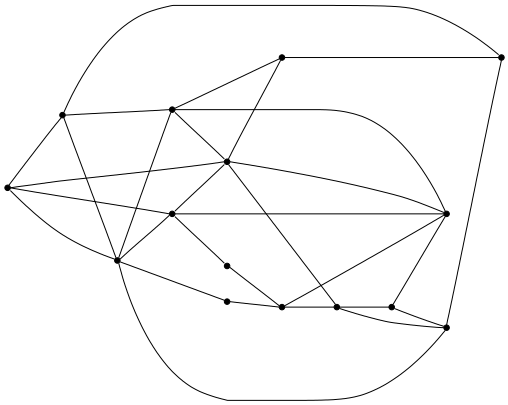
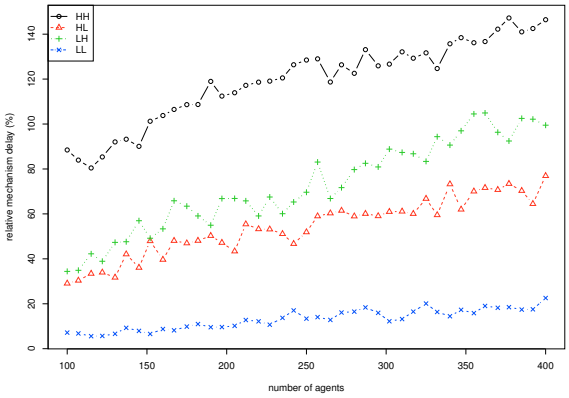


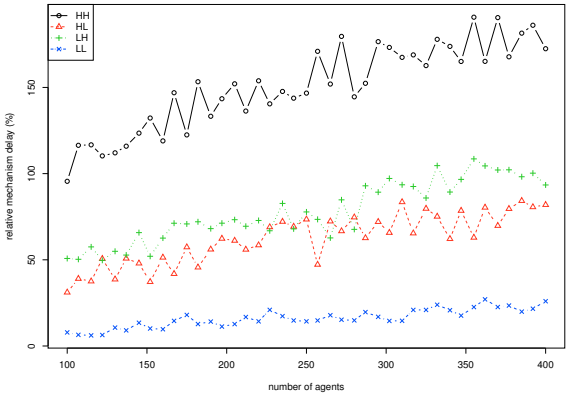
Figure 6.7: A random graph consisting of 15 nodes and 30 edges.

Figure 6.8 clearly shows that the mechanism delay for random graphs is much higher than for the Schiphol experiments. In case of a small number of short incidents (rate = 0.01 and duration = 30s.), the mechanism delay is still quite low: around 5% to 15% for random graphs and lattice networks, and around 5% to 25% for small world networks. The mechanism delay for the high incident setting (rate = 0.1 and duration = 90s.) is almost always over 100% of average plan length, and for small-world networks even in the range of 200%. If we look at the *absolute* mechanism delay, in figure 6.9 for random graphs, we see that it is still much shorter than the absolute mechanism delays for the Schiphol experiments: the mechanism delay for random graphs is rarely more than 100 seconds, whereas for the Schiphol experiments it is sometimes as much as 600 seconds. Obviously, the plan lengths for random graphs are much smaller, with an average agent plan lasting around 100 seconds, compared to the 5 minutes to one hour that agents required for Schiphol route plans. This difference in plan lengths cannot solely be explained by the size of the infrastructure in terms of resources — Schiphol has around 1000 while the random graphs around 450 — but is probably also the result of the fact that the start and destination locations for agents in the Schiphol experiments are runways, which are mostly situated at the edge of the infrastructure.

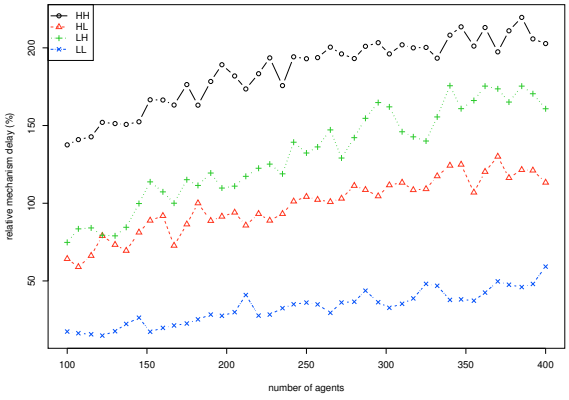
The difference in plan lengths between the Schiphol experiments and the random-graph experiments means that a 90-second delay is relatively more severe in the latter set of experiments. For example, if an agent has to wait at an intersection because a higher-priority agent has suffered an incident, then the agent will have a mechanism delay of almost 100%, unless it manages to make up for the loss of time during the execution of



(a) Random graphs with 180 nodes and 300 edges



(b) 2D Lattice networks



(c) Small-world networks

Figure 6.8: Relative mechanism delay for random graphs (a), lattice networks (b), and small-world networks (c).

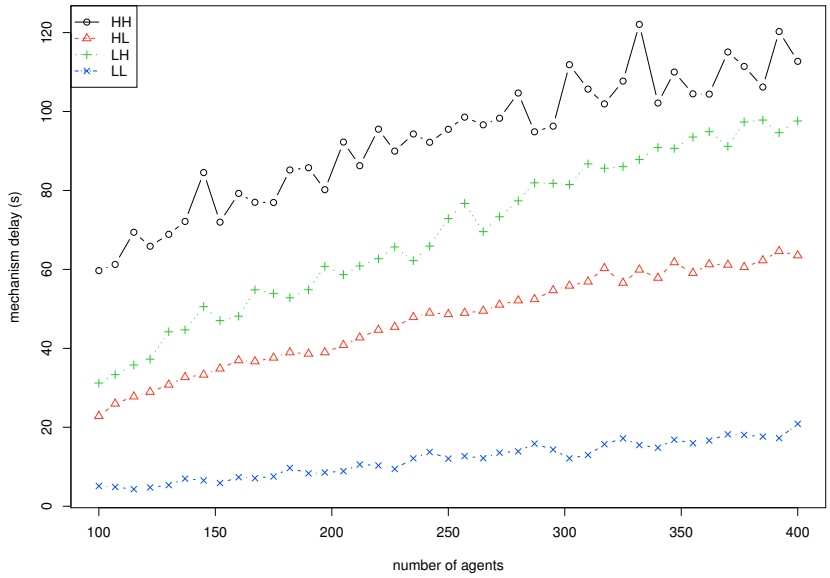


Figure 6.9: Absolute mechanism delay for random graphs on 180 nodes and 300 edges.

the remainder of its plan.

Another key difference between the Schiphol experiments and the experiments on random graphs is that in the latter (almost) all agents enter the infrastructure at the same time, whereas agents arrive (land) one by one in the Schiphol experiments. Hence, in the Schiphol experiments, there are fewer agents simultaneously in the infrastructure, at least at the beginning and the end of the experiments, when agents gradually arrive at and depart from the infrastructure. To see what happens if we spread the agents over time in the random-graph experiments, we ran another set of experiments on the small-world infrastructures, where the start time of each agent was randomly determined by:

$$t = \mathsf{random}(10 \cdot |A|)$$

where random returns a random number, uniformly distributed between 0 (inclusive) and $10 \cdot |A|$ (exclusive). In figure 6.10 we can see that the relative mechanism delay has indeed drastically reduced from a maximum of about 200% to around 70%. The reduction in relative mechanism delay cannot be completely explained by the smaller number of agents that simultaneously make use of the resource, however, because the relative mechanism delay does not increase with the number of agents in the system (in figure 6.10, as well as in figure 6.3(b) of the Schiphol experiments). Instead, we expect that because of the spread of the start time, the plans of the agents more easily organize into *flows*. With a flow we mean that a number of subsequent agents use lane resources in the same direction (and at the same time), which might be more efficient than a lane resource being used in alternating directions. In the Schiphol experiments, such a flow of agents arises very naturally, not just because of the spread of agents over time, but also

because many agents will be going in similar directions: from a shared arrival runway to a gate located somewhere at the center of the infrastructure, to a shared departure runway (of course, not all agents need to have the same arrival and destination runways).

To conclude this section on experiments with fixed agent priorities, we can state that incidents have a relatively modest impact on the mechanism delay in the Schiphol experiments. Moreover, for a small number of short incidents, what little mechanism delay there is, is almost completely made up by the time the agents reach their destination locations (i.e., there is hardly any delay with regard to the original set of plans). The more disappointing results for infrastructures of different topologies (random graphs, small-world networks and lattice networks) indicate that the Schiphol infrastructure has at least two characteristics that are beneficial to our experiments: agents are somewhat spread over time, and because of the particular distribution of start and destination locations, agents can easily organize themselves into flows. In the next section, we will investigate whether we can improve the performance on the other topologies by changing the priorities of the agents.

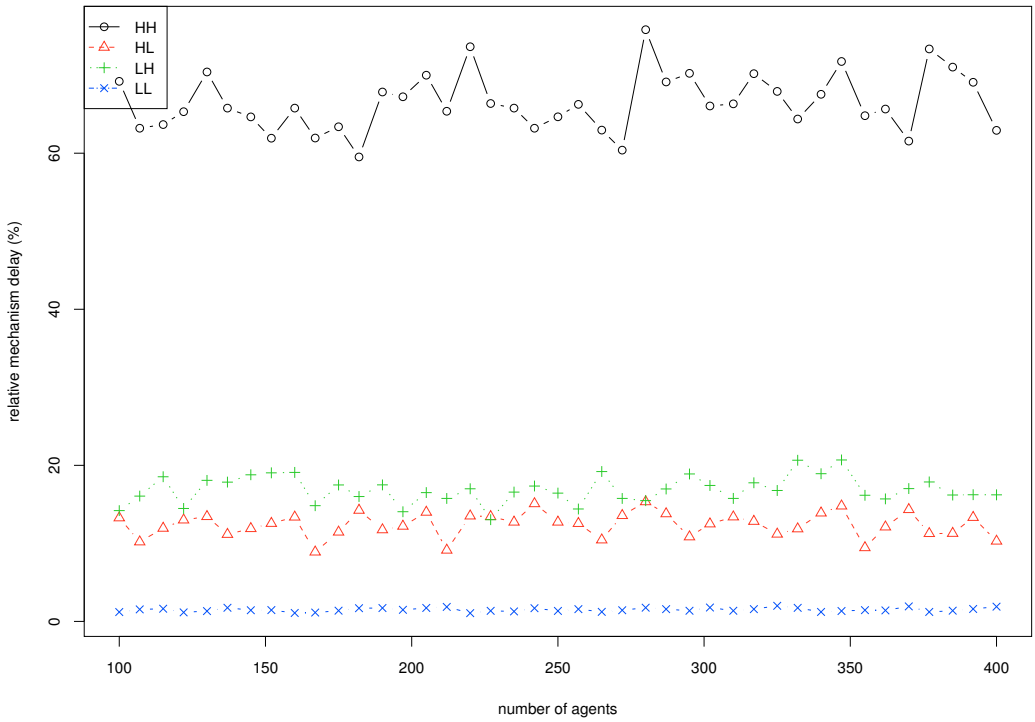


Figure 6.10: Relative mechanism delay for a small world infrastructure, with each agent's start time a random number between 0 and $10 \cdot |A|$.

6.1.3 Robustness with flexible priorities

The idea behind making priority changes is that an agent that is on time does not have to wait for delayed agents, but can continue execution of its plan. If the agent that has its priority lowered is 'sufficiently delayed', then it need not incur any additional delay on account of its low priority. Hence, we expect that priority changes in favour of timely agents will reduce the average delay in the system. Our algorithm 6 has been designed to perform more priority changes than Maza and Castagna's algorithm, so we first compare the number of priority changes performed by both algorithms. Next, we compare the average delay per agent that results from the priority changes.

Recall from section 5.3 that Maza and Castagna's algorithm grants an agent the highest priority on all the remaining resources in its plan (or until the resource where it already has the highest priority), and this is only allowed if all of those resources are currently unoccupied. Their algorithm is named RVRAA, which stands for the Robust Vehicles Routing Ahead Algorithm. Their algorithm that maintains priorities is called RVWA, although they intentionally[7] or unintentionally forget to mention what the acronym stands for (in a later paper [61], they reveal that RVWA stands for Robust Vehicle Waiting Algorithm[8]).

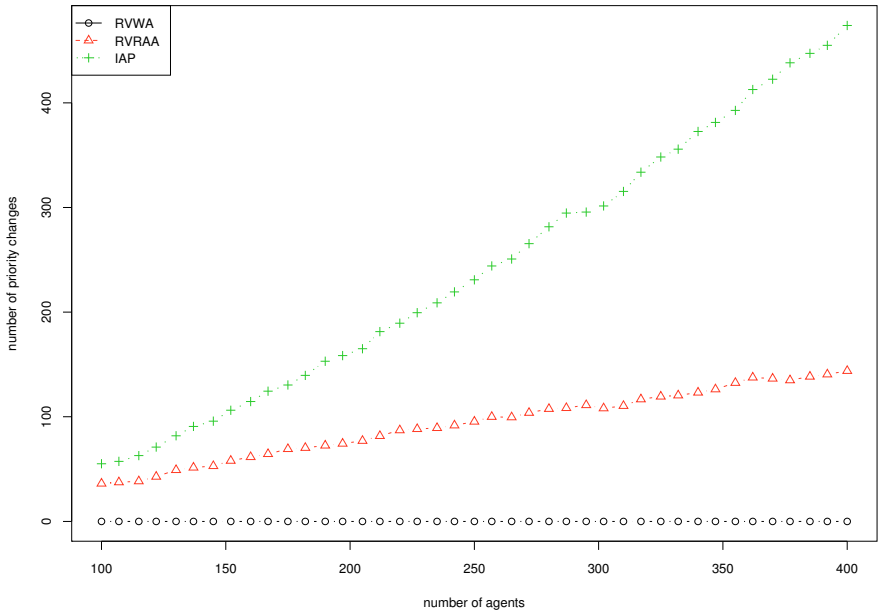
The setup of the experiments in this section is the same as before. The number of agents varied from 100 to 400, the incident duration was 30 or 90 seconds, and the incident rate was 0.01 or 0.1 (see table 6.1 from section 6.1.2). For each combination of the parameters, we generated 50 random instances, consisting of both the infrastructure to be used and the set of agent plans, and the set of incidents that the agents will encounter. We ran each random instance with all three schedule repair algorithms RVWA, RVRAA, and IAP (our algorithm 6). For the first set of experiments, we used random graphs on 180 nodes and 300 edges.

Figure 6.11(a) shows the number of priority changes for each of the three algorithms we consider, averaged over the four incident settings from table 6.1 (in section 6.1.2). The RVWA line is always at zero, of course, because it is the algorithm that does not perform any priority changes. The number of priority changes seems to increase linearly with the number of agents in the system, both for the RVRAA algorithm and for our IAP algorithm, but our IAP algorithm manages many more priority changes. To avoid clutter, we did not display the number of priority changes by both algorithms for each of the four incident settings separately, but anyway the results are the same: the IAP algorithm performs significantly more priority changes for all choices of the incident parameters.

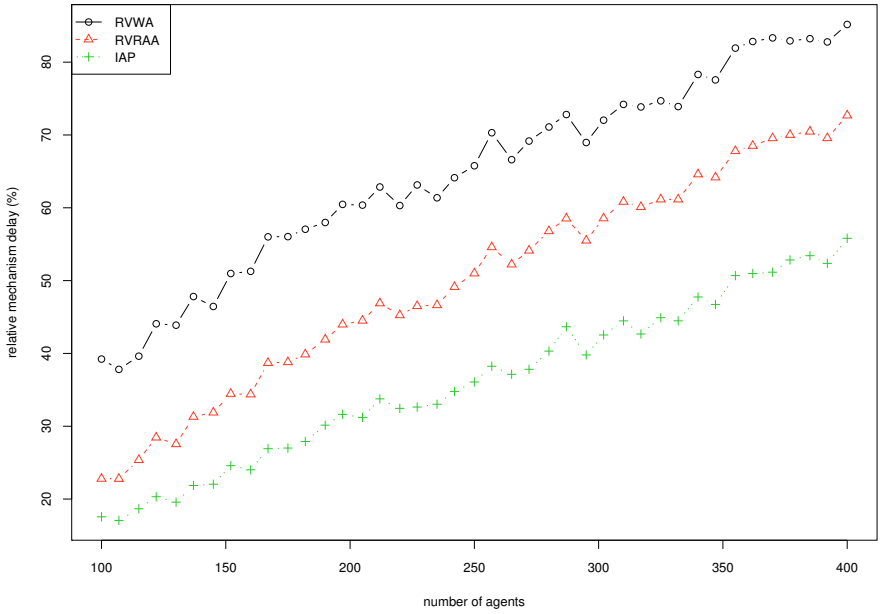
Figure 6.11(b) shows the average mechanism delay per agent for each of the three schedule repair algorithms on random graphs with 180 nodes and 300 edges. The figure shows that the application of IAP results in 10% to 15% reduction in mechanism delay compared to the application of RVRAA; the RVRAA algorithm in turn manages to produce 10% to 15% less mechanism delay than the RVWA algorithm. Hence,

[7] In the case of 1997 computer game MDK, for example, the mystery surrounding the meaning of the acronym became part of the game's advertising campaign. Among the candidate meanings were Murder, Death, Kill (in homage to *Demolition Man*), Massive Dollops of Ketchup, Mother's Day Kisses, Mission: Deliver Kindness, and Mohicans Discover Knitting.

[8] Unfortunately, we discovered this paper [61] too late to implement their new RVDA Robust Vehicle Delaying Algorithm and include it in the experiments described in this chapter.



(a) Number of priority changes



(b) Relative mechanism delay

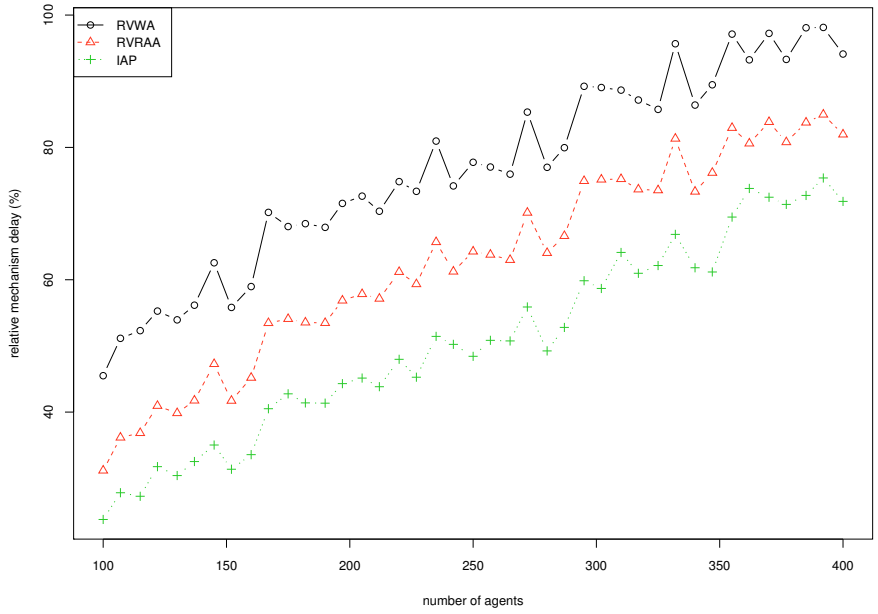
Figure 6.11: Comparison of the algorithms IAP, RVRAA, and RVWA on random graphs of 180 nodes and 300 edges, averaged over different incidents settings.

figure 6.11(b) suggests that making more priority changes results in a lower mechanism delay. At the same time, figure 6.11(b) shows that there is not a linear relation between the number of priority changes and the delay reduction achieved. For example, at 400 agents RVRAA makes around 100 priority changes more than RVWA, while IAP makes almost 400 changes more than RVRAA. The reduction in delay that IAP achieves over RVRAA is the same, however, as the reduction in delay that RVRAA achieves over RVWA. If we look at the performance of the three algorithms for each of the incident settings separately, then the relative performances of the algorithms stay the same: IAP outperforms RVRAA, which in turn outperforms RVWA.

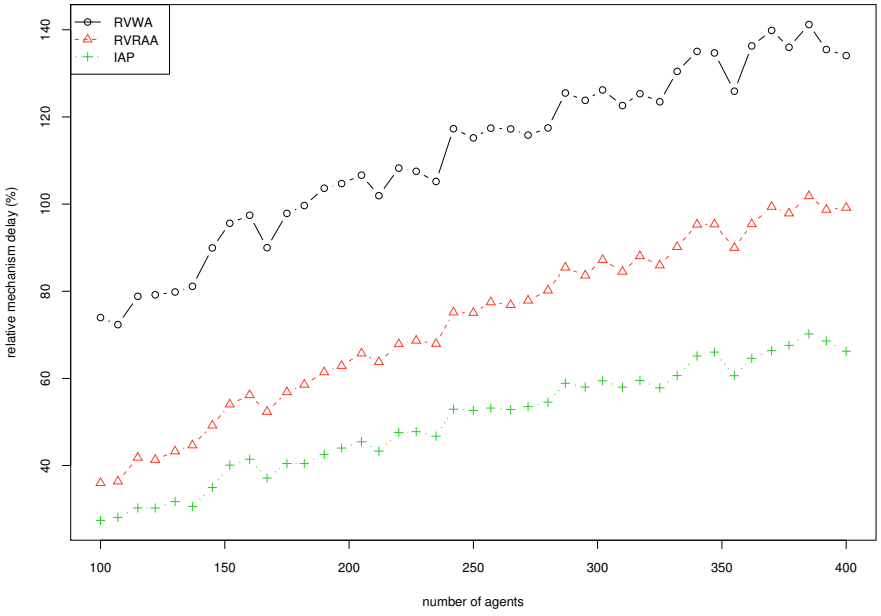
Figure 6.12 shows the relative mechanism delay for two other types of infrastructures, namely two-dimensional lattice networks and small-world networks. The general picture is the same: application of IAP results in the least mechanism delay, while application of RVWA results in the most application delay. For lattice networks, the differences in delay between the three algorithms are about the same as for random networks: IAP produces 10% to 15% less delay than RVRAA, which in turn produces 10% to 20% less delay than RVWA. On small-world networks, for which the delay resulting from RVWA is by far the highest, the reduction in mechanism delay is much greater. The gap between IAP and RVRAA can be as much as 30%, while the difference between RVRAA and RVWA can be as much as 50%.

For the sake of completeness, we show in figure 6.13 the performance of the algorithms on small-world networks for the following incident settings: LL = (0.01, 30s.); HL = (0.1, 30s.); HH = (0.1, 90s.). In the legend of the figure, algorithm RVWA is referred to by number 1 (so the application of RVWA in the incident setting HH is denoted by HH.1, for example), RVRAA has number 2, and IAP has number 3. The figure shows that the relative performances of the three algorithms are the same for each of the three incident settings: IAP outperforms RVRAA, which in turn outperforms RVWA. The difference between IAP and RVWA is such that the performance of IAP on incident setting HH is the same as the performance of RVWA on incident setting HL. It should be noted, however, that the relative mechanism delay for IAP on the highest incident level still reaches 100% at 400 agents. In fact, application of IAP cuts the mechanism delay almost in half for the experiments on small-world networks, but it still cannot approach the performance of plain RVWA on the Schiphol infrastructure.

To conclude this section on priority-changing algorithms, we can say that allowing priority changes enables us to reduce delay. Moreover, our IAP algorithm, which performs the most priority changes, also results in the greatest reduction in delay. At the same time, we saw that we cannot expect the same benefit from every priority change. In fact, sometimes a priority change can even increase the delay in the system, especially if an agent increases its priority over multiple agents. In figure 6.14, for example, agent Charles is ready to enter resource r_4, but agents Bob, Alice, and Trudy are scheduled to enter the sequence of resources (r_2, r_3, r_4) first. If we increase the priority of Charles so that he can enter r_4 first, then all three other agents have to wait for him. If we assume that it does not take Alice, Bob, and Trudy much more time to traverse (r_2, r_3, r_4) together than it does Charles to traverse these resources, then increasing the priority of Charles will increase the delay from this example almost threefold.



(a) 2D Lattice networks



(b) Small-world networks

Figure 6.12: Relative mechanism delay resulting from the application of algorithms IAP, RVRAA, and RVWA, averaged over different incidents settings.

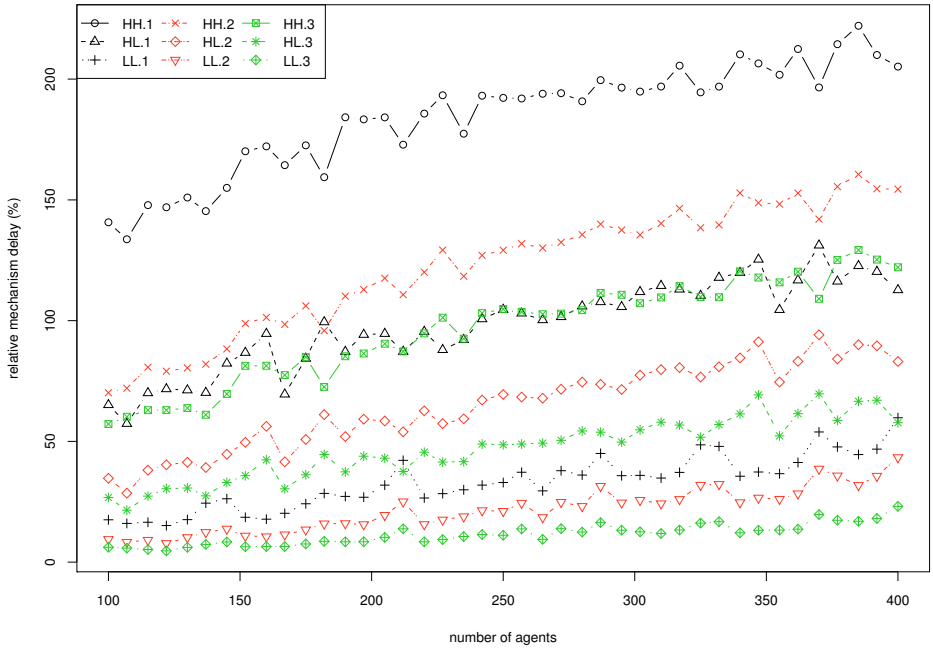


Figure 6.13: Relative mechanism delay on small-world networks resulting from the application of algorithms IAP (number 3 in the legend), RVRAA (number 2), and RVWA (number 1), for incident settings LL = (0.01, 30s.), HL = (0.1, 30s.), HH = (0.1, 90s.).

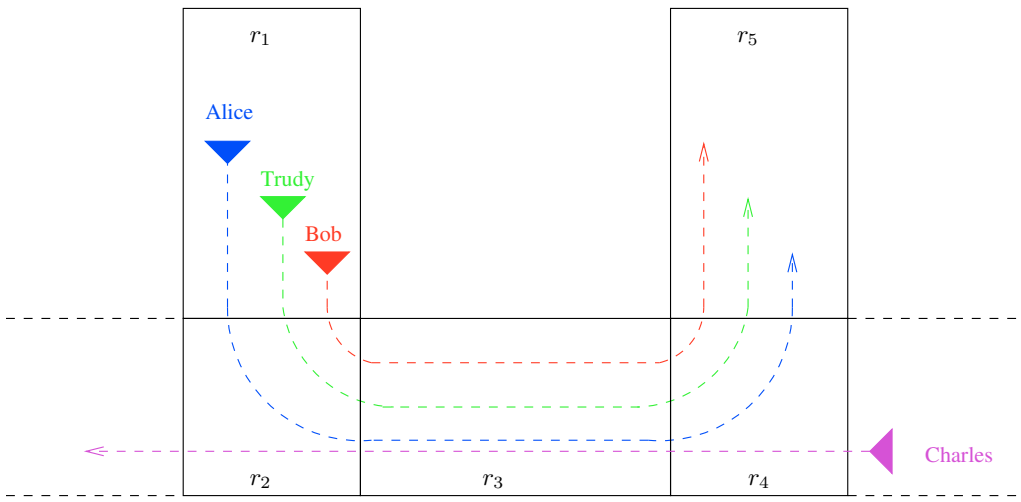


Figure 6.14: If agent Charles is granted the highest priority for the resources (r_2, r_3, r_4), this will result in more delay than if the other agents go first.

As figure 6.14 illustrates, it might be useful to make a distinction between identifying the possibility of a safe priority change, and performing the actual priority change. Deciding to increase an agent's priority can be done on the basis of heuristic information, that should predict whether a priority change is globally beneficial or not. Information that might prove useful is how much an agent is delayed, and how much time the delayed agent still needs to reach the resource that the non-delayed agent is waiting to enter. The more an agent is delayed, the more likely it seems that global delay will decrease if a non-delayed agent is granted the higher priority.

6.2 Priorities and global plan quality

So far in this thesis, we have assumed that agents plan in sequence, but which particular sequence to plan in has been left unspecified. In some domains, an ordering of the agents is given simply because the agents *arrive* in sequence, but in other domains agents must be assigned a priority. Assigning priorities arbitrarily is justified in case the worst-case ordering does not result in a significant increase of global plan cost. Otherwise, with an exponential number of possible agent orderings to choose from, some heuristic function should be used to choose a reasonable assignment of priorities.

As we have seen in section 3.4, there exist cases where the cost of the worst ordering is $O(|\mathcal{A}|)$ times as much as the cost of the best ordering. Obviously, this constitutes a significant increase in global plan cost, so choosing an arbitrary ordering may not be a suitable approach for all application domains. We will first investigate how global plan quality differs for different agent orderings in our airport taxiway routing problem. Finally, we will repeat this experiment on different classes of random networks, that vary in their average node degrees (the number of edges connected to a node, on average).

6.2.1 Agent ordering assumption

For the experiments in sections 6.2.2 and 6.2.3, we will make the following assumption: the optimal agent ordering, combined with optimal single-agent planning, will result in a multi-agent plan with a close-to-optimal cost. Figure 6.15 depicts a situation where the optimal set of priorities will not lead to an optimal plan. It shows an agent A_1 that wants to go from left to right, and an agent A_2 that wants to go from right to left. The infrastructure contains two directed lanes, both of which can be used for a short detour (the travel time of the directed resources is assumed to be longer than the travel time of the other lanes). In the optimal multi-agent plan, either agent A_1 or agent A_2 must make a detour. However, in a prioritized approach this multi-agent plan will never be constructed. Suppose w.l.o.g. that agent A_1 is allowed to plan first. It will choose its optimal plan, which is to make no detour. When it is agent A_2's turn to plan, the reservations of A_1's plan force it to wait before entering its start location, because it cannot take the 'detour lane' close to its start location, which can only be used in the opposite direction.

Whether the optimal priority assignment is guaranteed to result in an optimal plan is still a partially open question. If we assume an infrastructure with only undirected resources, then it is much harder to find an example where a prioritized approach does

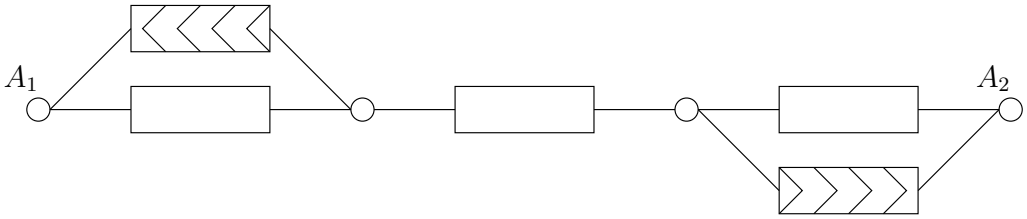


Figure 6.15: In a prioritized approach, neither agent will make use of the detours, which means that the lower-priority agent can only enter its start location when the other agent has passed.

not yield an optimal solution. One example was presented in [89], however that example does not distinguish between lane and intersection resources.

6.2.2 Schiphol experiments

For the first set of experiments, we considered taxi route planning on Schiphol airport. This time, we consider three classes of start-destination pairs for the agents. The first class is the realistic case where either the start location is a runway and the destination location is a gate, or the start location is a gate and the destination location is a runway. Of the five runways in use[9], two could be used solely for take-off, and three could be used only for landing. The second class of start-destination locations is similar to the first, except that here all runways can be used both for landing and for take-off[10]. The third class of start-destination pairs is that all locations are chosen randomly.

In our experiments, we compare the difference between the best priority assignment and the worst. We experimented with 600 agents, and three times 100 different sets of start-destination pairs. For each of those 100 problem instances, we tried 100 random agent orderings. The result of one experiment is the difference between the best priority assignment and the worst. If this difference turns out to be small, then this provides strong evidence that the prioritized approach will result in low-cost multi-agent plans: an arbitrary priority assignment is close to the best assignment, which in turn is assumed to be close to an optimal multi-agent plan.

Figure 6.16 shows six box plots[11]. One box plot is a summary of 100 experiments, where one experiment is: given 600 agents and for each agent a start and destination

[9]From figure 6.2 it can be seen that Schiphol has six runways, but the smaller Oostbaan is rarely used.

[10]Recall that at Schiphol airport, the simultaneous operation of a runway for landing and take-off (called *mixed-mode* operation) is not in use.

[11]To read the boxplots: the box contains all data points between the first *quartile* (the 'lowest' 25% of the data points) and the third quartile. The *whiskers* extend to the farthest data points that are not considered *outliers*, which are data points that are farther removed from the median than 1.5 times the *interquartile range* (the height of the box); outliers are represented by circles. Our boxplots have notches around the median; the meaning of these is that if the notches of two boxplots do not overlap, then this is 'strong evidence' that the median of the respective data sets differ [11].

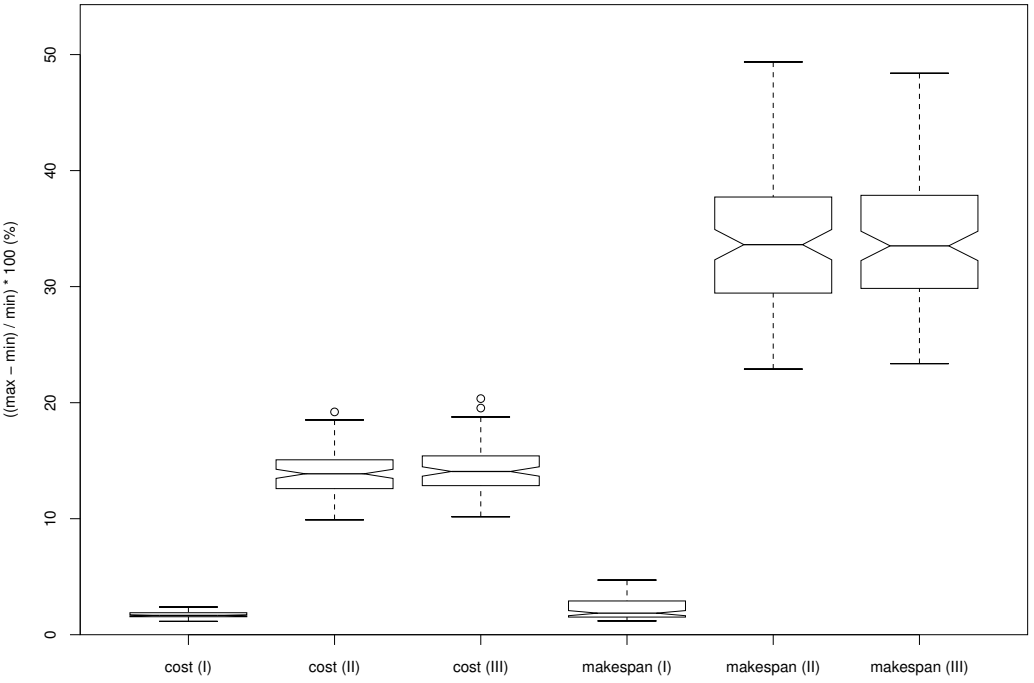


Figure 6.16: Difference between the best and the worst agent ordering (in terms of cost or makespan), for (I) realistic runway configurations, (II) mixed-mode runway configurations, (III) random start and destination locations.

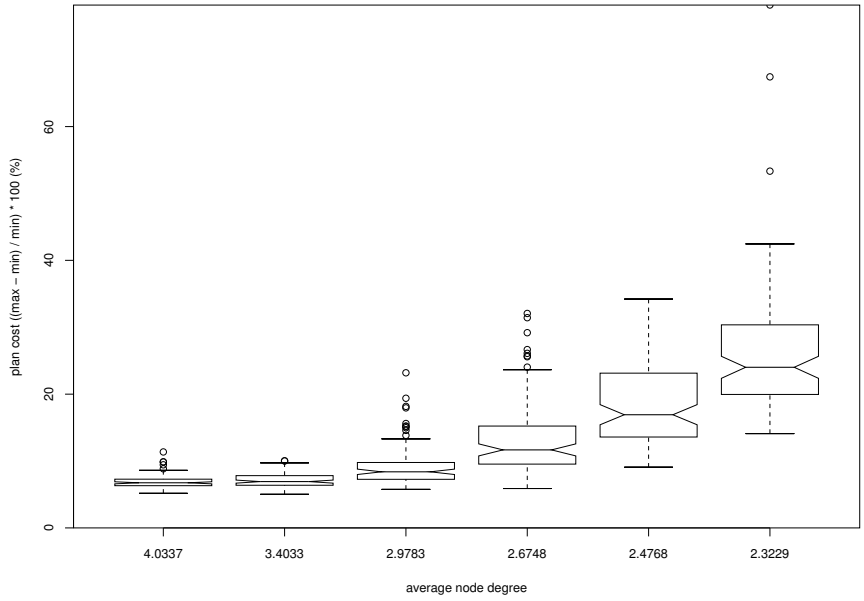
location, try 100 different agent orderings to find a multi-agent plan, and record the difference between the best plan and the worst plan. We consider difference in multi-agent plan cost according to two different measures. The first, labeled 'cost' on the horizontal axis of figure 6.16, is the summation of agent plan costs (in accordance with definition 3.1.2 of agent plan cost); the second is the makespan of a multi-agent plan, which is the time that the latest agent finishes. The first conclusion we can draw is that the agent ordering has little impact on the global plan cost in case start-destination pairs belong to the first class of instances (in which agents travel between gates and runways and runways are not used in mixed mode). As a result of the planning process, *flows* of agents are created on the infrastructure, in the sense that many agents travel in the same direction. Because multiple agents can occupy a taxiway if they travel in the same direction, there is little interference between the agents, especially between those that share a runway as a starting location. In case agents travel from a different gate to the same resource, their paths must *merge* into a flow. Apparently, this merging is achieved without much difficulty by the prioritized planning approach.

In case agents are allowed to use a runway in different directions, or if start and destination locations are distributed randomly over the infrastructure, then the flow of agents can be disrupted. For example, if there is a single long taxiway that leads to a runway (which is the case for e.g. the Polderbaan, see figure 6.2) then no agent can approach the runway for take-off as long as a landing aircraft has not exited this 'access' taxiway. However, the maximum inefficiency in terms of summed agent plan cost is still only around 20% of the best priority assignment; an arbitrary priority assignment can therefore can be expected to be closer to the best agent ordering. Figure 6.16 also shows that the makespan cost measure is more sensitive to the priority assignment. This is not surprising, however: if only one or two agents are very late, then this will not result in a high summed agent plan cost, but it will directly result in a high makespan.

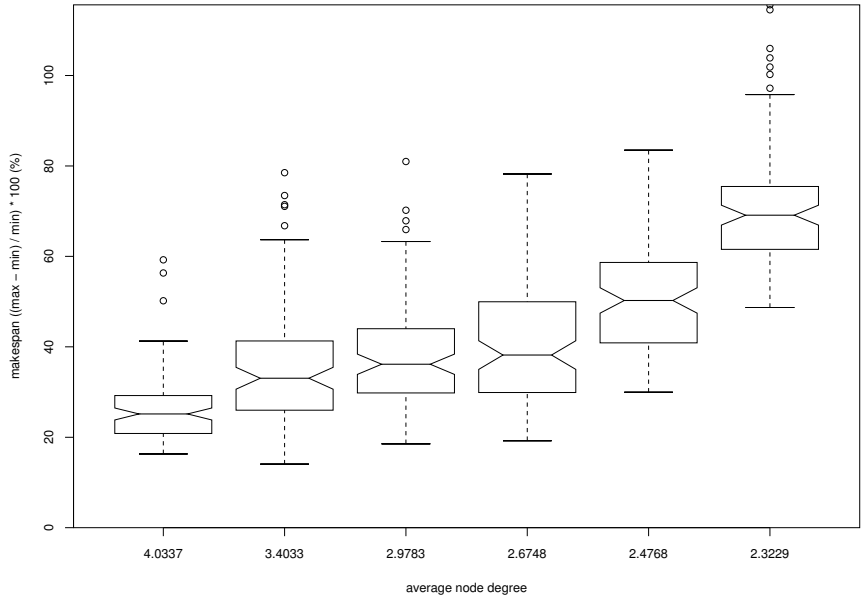
6.2.3 Experiments on random infrastructures

The above experiments on the Schiphol infrastructure showed a distinct drop in performance of the prioritized planning approach in case start and destination locations are not grouped in a particular fashion. We therefore repeated the above experiments on sets of random infrastructures, with randomly chosen start and destination locations. We generated several classes of random infrastructures, which differ in the average node degree (the average number of lanes that an intersection is connected to). The average node degree ranges from around 4, to a little over 2. An average node degree of four corresponds to having about twice as many edges as there are nodes. Our choice for the maximum factor of 2 was motivated by the research into random *planar* networks by Denise et al. [15], who found that for approximately maximum random planar networks, the number of edges is around twice the number of nodes. Having planar networks makes sense in a transportation setting, but unfortunately constructing them is rather involved (cf. [6]), so we constructed ordinary random networks. An average node degree of around 2 corresponds to having roughly an equal number of nodes and edges, which constitutes a sparse but connected graph.

The most obvious conclusion that can be drawn from figure 6.17 is that the difference



(a) multi-agent plan cost



(b) makespan

Figure 6.17: Differences between best and worst priority assignments in terms of makespan or multi-agent plan cost, for random networks with an average node degree from 4.0 to 2.3.

between the best and the worst agent ordering increases as the average node degree decreases (note that there is no scale on the horizontal axis in figure 6.17 — the labels simply represent the average node degree for the infrastructures in one set of experiments), both in terms of summed agent plan cost and in terms of makespan. We believe this can be explained in terms of the densities of the infrastructures. For infrastructures with high average node degree, each intersection is connected to many lanes, and there are many paths between any two locations. If one path is congested, then the route planning algorithm from chapter 4 will find an alternative route. For infrastructures with a low average node degree, there is often only a single reasonable path between two locations. Such a path can become a bottleneck especially in the case it is used in both directions, because then only one agent can make use of it at the same time. Indeed, the worst agent ordering can produce many situations illustrated in example 3.4.1, with a path being used by agents travelling in alternating directions.

To compare the results of figure 6.17 with the experiments on the Schiphol infrastructure in figure 6.16, the Schiphol infrastructure has an average node degree of around 4. The results for the random infrastructures with average degree 4 and the Schiphol infrastructure are indeed comparable, although there are a couple of outliers in the random infrastructure experiments that did not appear in the Schiphol experiments.

6.3 Concluding remarks

In this chapter we empirically evaluated the *usability* of the prioritized multi-agent route planning approach, i.e., how well the approach should lend itself to application in realistic problem domains. We evaluated both the robustness in the face of unexpected incidents, and how the cost of the global plan depends on the order in which the agents plan. In both experiments, the best results were obtained using a realistic airport infrastructure (of Amsterdam Airport Schiphol).

With regard to the cost of the global plan, the suitability of the airport infrastructure is straightforward to explain. We think that in order to obtain low-cost multi-agent plans, agents should form *flows* over the infrastructure: if agents travel in the same direction along a path of resources, then the resource utilization is much higher than if the travel direction of a path changes with each subsequent agent. For airport infrastructures, such flows will be established naturally due to the location of the start and destination locations of the agents: each agent starts at a runway and then goes to a gate, or an agent goes from a gate to a runway. Especially for the case of Schiphol airport, where runways are only used either for landing or for taking off, many agents travel in the same direction, and agents that arrive do not interfere much with agents that depart. As a result, it does not matter much to the global plan cost in which order we let the agents make their plans. For random infrastructures, the organization of agents into flows is more easily achieved in case an infrastructure has many alternative paths connecting any two locations. We believe this is the reason why infrastructures with a high average node degree are less sensitive to the priority ordering of the agents than infrastructures with a low average node degree.

With regard to the robustness of multi-agent plans in case of unexpected incidents, an important result was that small disturbances have little impact on the operation of the

multi-agent system. In case many (and severe) incidents occur, the average delay increases (and becomes more unpredictable), but at 20% of plan length it is still within bounds for the airport experiments. On other types of infrastructures (we tested small-world networks, two-dimensional lattice networks, and random graphs), the average mechanism delay on the highest incident level was between 100% and 200% of plan length. Fortunately, by applying schedule repair algorithms we can often cut the mechanism delay in half. The organization of agents into flows also seems beneficial for the robustness of a set of agent plans. For the Schiphol experiments, these flows occur not only because of where the start and destination locations of the agents are situated, but also because the arrival and departure of the agents are spread over time (because only one or two agents can land or take off at the same time). If we also spread the arrival times of agents in random infrastructures, then we also achieve a reduction in mechanism delay, presumably due to an increase in the formation of agent flows.

Chapter 7

Conclusions

This thesis aimed to answer the question,

> Can *agents*, operating on a shared *infrastructure*, find *conflict-free* route plans that are both *efficient* and *robust* under changing circumstances, given limited *computational resources*?

In this chapter, we will evaluate whether the contributions of this thesis provide a satisfactory answer to the research question, and in which areas future work is still required.

7.1 Finding conflict-free route plans

When planning a route to perform a transportation task, an agent must ensure that it will not come into conflict with other agents that operate on the same infrastructure. We model the infrastructure as a set of resources representing the roads and intersections that the agents travel on. Each resource has a capacity that specifies the maximum number of agents that may occupy the resource at the same time. The agents should ensure that they come up with a set of route plans such that the planned usage of any resource never exceeds its capacity.

In many application domains, the resource capacity constraint alone is not sufficient to ensure conflict-free plan execution. In our model of multi-agent routing, we therefore define other constraints that a set of agent route plans may also be required to satisfy. For example, we may require that agents are not allowed to overtake each other on a resource; or, a (road) resource that allows bidirectional traffic may only be traversed in one direction at the same time.

We have presented a *prioritized* approach to route planning, in which each agent is assigned a (unique) priority, and the agent with the highest priority may plan first, followed by the agent with the second-highest priority, and so on. When the agent with the highest priority has decided on its route plan, it places reservations on the resources in its plan for the duration of its intended traversal of the resources. The second agent to plan must respect these reservations, in the sense that it is not allowed to place any reservations that would exceed the capacity of any resource at any time (or violate any

of the other constraints in force). Hence, the agent with the n^{th} highest priority must plan around the reservations of the first $n-1$ agents. To do so, it need not consider all of the different reservations on each of the resources it encounters. Instead, prior to planning, it is possible to derive the set of *free time windows* for each resource in the infrastructure. A free time window constitutes an interval during which an agent can use a resource without causing a conflict with any of the higher-priority agents.

We can find a conflict-free route plan by performing a search through the *graph* of free time windows. If we find the shortest path in the free time window graph, then we have found a route plan that avoids conflicts with all of the higher-priority agents. Moreover, this plan is time-optimal, in the sense that no route plan can be found that arrives at the agent's destination resource earlier.

7.2 Efficiency of route plans

We have presented an algorithm that finds the shortest-time route plan for a single agent, given a set of plans from higher-priority agents. This does not guarantee, however, that the *set* of agent plans is optimal. In fact, a simple example can show that even with optimal single-agent planning, the multi-agent plan (simply the union of the single-agent route plans) can be $O(|\mathcal{A}|)$ times as costly as an optimal multi-agent plan (where $\mathcal{A}$ is the set of agents).

We have presented a set of experiments designed to evaluate how an arbitrarily chosen assignment of priorities might influence the cost of the multi-agent route plan. To this end, we tried 100 different priority assignments for each problem instance, and recorded the difference between the best and the worst priority assignment. Interpreting this difference as an estimation of the worst-case priority assignment, we found that the impact of the agent ordering is influenced by a number of factors.

First of all, an experiment with taxi route planning on a model of a real airport (Amsterdam Airport Schiphol) showed the importance of the distribution of the start and destination locations of the agents. With all agents either travelling from a runway to a gate, or from a gate to a (different) runway, there is little chance that the agents will meet head-on. The increase in plan cost for the worst priority assignment was less than 5% over the best priority assignment. However, if agents have random start and destination locations, the increase in multi-agent plan cost was at most 20%.

A second factor that influences the performance of prioritized planning is the density of the infrastructure. A second set of random infrastructures showed that if the (average) number of roads connected to intersections (the average node degree) increases, then the increase in plan cost for the worst priority assignment goes down. For random infrastructures with an average node degree of around 4, the worst priority assignment had an increase in plan cost of around 10%, whereas for infrastructures with an average node degree of 2.3, the worst priority assignment was at most 80% more expensive. The reason for this performance difference is that for sparse graphs, there is often only a single path between two locations; if subsequent agents use that path in opposite directions (and no simultaneous bidirectional travel is allowed), then very inefficient behaviour ensues. If there are two paths between locations, then one can be used for agents in one direction, and the other can be used by agents travelling in the other direction.

A direction of future work is to find heuristic functions to determine efficient priority assignments. Our experiments show that agents lose much time waiting for agents travelling in the opposite direction, so a heuristic function could be aimed at creating *flows* of agents travelling in the same direction.

7.3 Robustness of route plans

The study of the impact of unexpected incidents on plan execution is especially important for multi-agent routing, since a small delay of one agent can, if all other agents have zero delay, create a deadlock situation. Fortunately, Maza and Castagna [58] described a procedure that can prevent delays from creating deadlocks. As a result of planning, it is possible to infer for each resource which agents will use it, and in which order (specifying the priorities of the agents *on this resource*). If we maintain this resource priority during the execution of the agents' plans — meaning that agents can only enter the resource in the order that they planned to — then the delays of the agents will not lead to a deadlock.

Maintaining resource priorities implies that an agent must sometimes wait for a higher-priority agent that has some delay. The robustness of a set of route plans can therefore be measured in terms of the delay agents suffer while waiting for permission to enter a resource. Our experiments in taxi route planning show that in case there are only a small number of short incidents, then agents hardly suffer any additional delay. In case there are many long incidents, our experiments in taxi route planning show that the additional delay amounts to at most 20% of average plan length.

A different set of experiments conducted using random infrastructures (including grid-like infrastructures, small-world infrastructures [43], and 'fully random' infrastructures) also showed that delay is low for a small number of short incidents. For a large number of long incidents, however, additional delays of around 100% were frequently reached. One reason for the difference in delay between these experiments and the airport taxi routing experiments is that in the latter set, the lengths of the plans of the agents are much longer, so one long incident (incident durations were the same for the airport experiments and the random-infrastructure experiments) has less impact in a relative sense. A second reason why the results are so favourable for the airport experiments is that the locations of the start and destination points of the agents allow them to organizes themselves into flows (for example, agents travelling from a runway to a gate are going in more or less the same direction and will not interfere much with each other). It seems to be that flows of agents are not as vulnerable to delays as agents that use resources in alternating directions.

We managed to reduce the delay by up to 50% by making use of an algorithm that increases the priority of a non-delayed agent. Maza and Castagna [60] first presented an algorithm that allows an agent, which is waiting to enter a resource, to increase its priority over the agents that it is waiting for. The solution presented by Maza and Castagna is to grant an agent the highest priority for all of its remaining resources (i.e., it will be the first agent to enter these resources), which will not lead to a deadlock if and only if these resources are currently empty of agents. We proposed an alternative algorithm that only increases an agent's priority over those agents that are truly delayed, allowing us to perform more priority changes. Our experiments indicate that our mechanism

leads to lower delay than Maza's algorithm, for all incident 'levels' that we investigated. As future work, we should also compare our algorithm to Maza and Castagna's own alternative priority-changing algorithm [61], which we did not discover in time to include in our experiments.

Our experiments indicate that, on the whole, making more priority changes will reduce delay. At the same time, it is easy to show that not all priority changes will reduce delay, and some can even increase it. Whether a particular priority change will reduce delay or not should be the subject of further research. One possibility is to develop a heuristic function that should estimate the merit of each potential priority change. This heuristic could be based on e.g. the estimated time that a delayed agent needs to reach a particular resource in its plan (to judge how much longer a non-delayed agent would have to wait).

7.4 Computational complexity of route planning

Finding an optimal set of route plans is a computationally hard problem. In its most basic form, the problem is strongly NP-hard even for instances with only three resources; in its hardest form (i.e., when a set of agent plans must satisfy a particularly difficult set of constraints), the multi-agent routing problem is PSPACE-complete.

Finding an optimal route plan for a single agent, given a set of plans from higher-priority agents, *can* be achieved in polynomial time. An existing algorithm from the literature [41] reports an $O(|\mathcal{A}|^4|R|^2)$ worst-case running time (where $\mathcal{A}$ is the set of agents and R is the set of infrastructure resources). We have presented an algorithm with an improved complexity of $O(|\mathcal{A}||R|\log(|\mathcal{A}||R|) + |E_R||\mathcal{A}|)$, where E_R is the set of lane resources in the infrastructure (the 'edges' in the infrastructure graph).

7.5 Answering the research question

This thesis has provided a tentative "yes" to the question "can agents, operating on a shared infrastructure, find conflict-free route plans that are both efficient and robust under changing circumstances, given limited computational resources?" First of all, we have made progress both in the areas of computational complexity and robustness: our new single-agent, context-aware route planning algorithm has a significantly better worst-case running time complexity, and our new schedule repair algorithm achieves a substantial reduction in delay caused by unexpected incidents in the environment.

With regard to the efficiency of route plans, our single-agent route planning algorithm ensures a locally (i.e., for the agent itself) optimal solution, but not a globally optimal one. Our experiments did reveal that there are certain conditions under which the individually optimal agent plans form a multi-agent plan that looks close to optimal. In particular, in case the agent plans form flows of agents over the infrastructure, then the multi-agent plan cost is often low. Further research into the cost of the multi-agent plan can therefore be directed at stimulating the *emergence* of agent flows. One possibility, as mentioned above, is to come up with a set of agent priorities that should achieve this.

7.6 Outlook

So far, we have identified directions of future work that can be viewed as direct continuations of the research described in this thesis: we can try to improve global plan quality by finding better agent planning orders, and we can develop heuristics to guide us in determining when we should increase the priority of non-delayed agents. In addition, we will now present some ideas for future work into research directions that are relatively unexplored, more challenging, and more *exciting*!

The first new research question is whether we can do away with the timing of the plans. Currently, the route plans that we produce specify exactly where an agent is at each point in time. This is convenient, because it allows us to predict exactly when an agent will finish its transportation task. On the other hand, such a plan specification may seem too precise in a dynamic environment, where small disturbances might cause the execution of the plans to have a slightly different timing. In addition, our results on the Planstep Priority Graph from chapter 5 show that we do not need to know the exact timing of agent plans to ensure deadlock-free system operation. Instead, we only need to decide in which order the agents will visit the various resources. Hence, we might conceive of a planning method that only gives an agent priorities on the resources it will use, in such a way that the set of agent plans is deadlock free. To execute such a set of agent plans, we only need Maza and Castagna's priority maintenance idea, possibly extended with a priority changing algorithm to fine-tune performance. The main motivation for pursuing this line of research would be the hope that we can produce a faster route planning algorithm.

The second question is whether we can find other ways to improve the quality of the global plan. One possibility is to let coordination between the agents play a bigger role. The prioritized approach we presented in this chapter promotes the *autonomy* of the agents: an agent may make use of any resource, during any time interval, as long as no conflict is created. This also means that an agent is not in any way guided towards plans that are beneficial to others, for instance if there is a choice between equivalent plans. Additional constraints or guidelines — imposed before or during the planning process (cf. [92]) — could be used to steer an agent in a good direction. Another possibility to steer the agents in the right direction might be to construct a small percentage of the agent plans centrally before letting the other agents plan as usual. The initial set of plans could form one or more flows, to which the other agents would add their plan, thereby strengthening the flows. Note that this is actually a kind of Stackelberg game, in which the *leader* first chooses an action, then one or more self-interested *followers* choose theirs (cf. [45]). The leader will choose its action in such a way that the best response of the followers, having observed the action of the leader, will lead to an outcome that is desirable for the leader.

A final idea might be to conduct experiments in more realistic application domains. In this thesis we have argued for the usability of (the prioritized approach to) multi-agent route planning, and we have shown that e.g. we can still use route plans when incidents disrupt the execution of plans. But to answer questions like: "can we really reduce delay at Schiphol airport", or "can a human air traffic controller or AGV dispatcher improve his or her performance using a MARP decision support system", or "how should we integrate MARP with other problems like idle vehicle management, (transportation-)task

allocation, battery management, etc.", we not only need more comprehensive simulation models, we also need to open up our relatively sheltered simulation environment to human users.

Bibliography

[1] S. Akella and S. Hutchinson. Coordinating the motions of multiple robots with specified trajectories. In *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*, pages 624–631, Washington DC, May 2002.

[2] M. S. Aktürk and H. Yilmaz. Scheduling of automated guided vehicles in a decision making hierarchy. *International Journal of Production Research*, 34(2):577–591, 1996.

[3] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[4] M. Beaulieu and M. Gamache. An enumeration algorithm for solving the fleet management problem in underground mines. *Comput. Oper. Res.*, 33(6):1606–1624, 2006.

[5] M. Bennewitz and W. Burgard. Finding solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2–3):89–99, November 2002.

[6] M. Bodirsky, C. Gröpl, and M. Kang. Generating labeled planar graphs uniformly at random. *Theoretical Computer Science*, 379(3):377–386, June 2007.

[7] A. J. Broadbent, C. B. Besant, S. K. Premi, and S. P. Walker. Free ranging AGV systems: promises, problems, and pathways. In *2^{nd} International Conference on Automated Materials Handling*, pages 221–237, 1985.

[8] G. Bruno, G. Ghiani, and G. Improta. Dynamic positioning of idle automated guided vehicles. *Journal of Intelligent Manufacturing*, 11(2):209–215, April 2000.

[9] J. Canny, B. Donald, J. Reif, and P. Xavier. On the complexity of kinodynamic motion planning. In *29^{th} Annual Symposium on Foundations of Computer Science*, pages 306–316, White Plains, NY, USA, October 1988.

[10] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *28^{th} Annual Symposium on Foundations of Computer Science*, pages 49–60, 1987.

[11] J. M. Chambers, W. S. Cleveland, and P. A. Tukey. *Graphical Methods for Data Analysis*. The Wadsworth statistics/probability series. Duxbury Resource Center, 1983.

[12] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, 1971.

[13] A. I. Corréa, A. Langevin, and L.-M. Rousseau. A scheduling and conflict-free routing problem solved with a hybrid constraint programming/mixed integer programming approach. Technical report, GERAD, January 2005.

[14] A. I. Corréa, A. Langevin, and L.-M. Rousseau. Scheduling and routing of automated guided vehicles: A hybrid approach. *Computers and Operations Research*, 34(6):1688–1707, 2007.

[15] A. Denise, M. Vasconcellos, and D. J. Welsh. The random planar graph. *Congressus Numerantium*, 113:61–79, 1996.

[16] G. Desaulniers, A. Langevin, D. Riopel, and B. Villeneuve. Dispatching and conflict-free routing of automated guided vehicles: An exact approach. *International Journal of Flexible Manufacturing Systems*, 15(4):309–331, November 2004.

[17] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[18] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, 1993.

[19] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2(1):477–521, 1987.

[20] M. P. Fanti. A deadlock avoidance strategy for AGV systems modelled by coloured Petri nets. In *Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)*, 2002.

[21] P. Farahvash and T. O. Boucher. A multi-agent architecture for control of AGV systems. *Robotics and Computer-Integrated Manufacturing*, 20(6):473–483, December 2004.

[22] C. Ferrari, E. Pagello, J. Ota, and T. Arai. Multirobot motion coordination in space and time. *Robotics and Autonomous Systems*, 25(3):219–229, November 1998.

[23] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[24] S. Fujii, H. Sandoh, and R. Hozaki. 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, 1989.

[25] K. Fujimura and H. Samet. Motion planning in a dynamic domain. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, volume 1, pages 324–330, Cincinnati, Ohio, USA, May 1990.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W.H.Freeman, 1979.

[27] E. Gawrilow, E. Köhler, R. H. Möhring, and B. Stenzel. *Mathematics Key Technology for the Future*, chapter Dynamic Routing of Automated Guided Vehicles in Real-time, pages 165–177. Springer Berlin Heidelberg, 2007.

[28] S. S. Ge and Y. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 2002.

[29] Y. Guo and L. E. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA02)*, volume 3, pages 2612–2619, 2002.

[30] A. N. Habermann. Prevention of system deadlocks. *Communications of the ACM*, 12(7), 1969.

[31] N. G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, May-June 1996.

[32] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 4(2):100–107, 1968.

[33] W. Hatzack and B. Nebel. The operational traffic problem: Computational complexity and solutions. In A. Cesta, editor, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 49–60, 2001.

[34] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, October 2005.

[35] M. Helmert. New complexity results for classical planning benchmarks. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 52–61. AAAI Press, 2006.

[36] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, Winter 1984.

[37] J. Huang, U. Palekar, and S. Kapoor. A labeling algorithm for the navigation of automated guided vehicles. *Journal of Engineering for Industry*, 1993.

[38] B. H. Jeong and S. U. Randhawa. A multi-attribute dispatching rule for automated guided vehicle systems. *International Journal of Production Research*, 39(13):2817–2832, September 2001.

[39] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, Fall 1986.

[40] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[41] C. W. Kim and J. M. Tanchoco. Conflict-free shortest-time bidirectional AGV routeing. *International Journal of Production Research*, 29(1):2377–2391, 1991.

[42] S. H. Kim and H. Hwang. An adaptive dispatching algorithm for automated guided vehicles based on an evolutionary process. *International Journal of Production Economics*, 60(1):465–472, April 1999.

[43] J. M. Kleinberg. Navigation in a small world. *Nature*, 406(24):845, August 2000.

[44] J. M. Kleinberg and Éva Tardos. *Algorithm Design*. Pearson Education, 2006.

[45] Y. A. Korilis, A. Lazar, and A. Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Transactions on Networking*, 5(1):161–173, 1997.

[46] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In 25^{th} *Annual Symposium onFoundations of Computer Science*, pages 241–250, 1984.

[47] T. Le-Anh and R. B. De Koster. A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23, May 2006.

[48] C.-C. Lee and J. Lin. Deadlock prediction and avoidance based on Petri nets for zone-control automated guided vehicle systems. *International Journal of Production Research*, 33(12):3249–3265, December 1995.

[49] J. H. Lee, B. H. Lee, and M. H. Choi. A real-time traffic control scheme of multiple AGV systems for collision-free minimum time motion: a routing table approach. *IEEE Transactions on Man and Cybernetics, Part A*, 28(3):347–358, May 1998.

[50] Y. Leung, G. Li, and Z.-B. Xu. A genetic algorithm for the multiple destination routing problems. *IEEE Transactions on evolutionary computation*, 2(4):150–161, 1998.

[51] J. Lin and P.-K. Dgen. An algorithm for routeing control of a tandem automated guided vehicle system. *International Journal of Production Research*, 32(12):2735–2750, December 1994.

[52] S. G. Loizou and K. J. Kyriakopoulos. Closed loop navigation for multiple nonholonomic vehicles. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, pages 4240–4245, Taipei, Taiwan, September 2003.

[53] S. G. Loizou and K. J. Kyriakopoulos. Navigation of multiple kinematically constrained robots. *IEEE Transactions on Robotics*, 24(1):221–231, February 2008.

[54] T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, February 1983.

[55] P. Maes. Artificial life meets entertainment: Life like autonomous agents. *Communications of the ACM*, 38(11):108–114, 1995.

[56] B. Mahadevan and T. Narendran. Design of an automated guided vehicle-based material handling system for a flexible manufacturing system. *International Journal of Production Research*, 28(9):1611–1622, September 1990.

[57] W. L. Maxwell and J. A. Muckstadt. Design of automatic guided vehicle systems. *IIE Transactions*, 14(2):114–124, 1982.

[58] S. Maza and P. Castagna. Conflict-free AGV routing in bi-directional network. In *Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation*, volume 2, pages 761–764, Antibes-Juan les Pins, France, October 2001.

[59] S. Maza and P. Castagna. Robust conflict-free routing of bi-directional automated guided vehicles. In A. E. Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of the 2002 IEEE International Conference on Systems, Man and Cybernetics*, volume 7, Yasmine Hammamet, Tunisia, 2002. Piscataway, N.J.

[60] S. Maza and P. Castagna. A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. *Computers in Industry*, 56(7):719–733, 2005.

[61] S. Maza and P. Castagna. Sequence based hierarchical conflict-free routing strategy of bi-directional automated guided vehicles. In *Proceedings of IFAC (International Federation of Automatic Control) World Congress*, 2005.

[62] R. W. McHaney. Modelling battery constraints in discrete event automated guided vehicle simulations. *International Journal of Production Research*, 33(11):3023–3040, 1995.

[63] R. H. Möhring, E. Köler, E. Gawrilow, and B. Stenzel. *Conflict-free Real-time AGV Routing*, volume 2004 of *Operations Research Proceedings*, pages 18–24. Springer Berlin Heidelberg, 2004.

[64] R. Narasimhan, R. Batta, and H. Karwan. Routing automated guided vehicles in the presence of interruptions. *International Journal of Production Research*, 37(3):653–681, February 1999.

[65] D. Naso and B. Turchiano. Multicriteria meta-heuristics for AGV dispatching control based on computational intelligence. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(2):208–226, April 2005.

[66] T. Nishi, M. Ando, and M. Konishi. Experimental studies on a local rescheduling procedure for dynamic routing of autonomous decentralized agv systems. *Robotics and Computer-Integrated Manufacturing*, 22(2):154–165, April 2006.

[67] T. Nishi, M. Ando, M. Konishi, and J. Imai. A distributed route planning method for multiple mobile robots using lagrangian decomposition technique. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, volume 3, pages 3855–3861, Taipei, Taiwan, September 2003.

[68] T. Nishi, S. Morinaka, and M. Konishi. A distributed routing method for AGVs under motion delay disturbance. *Robotics and Computer-Integrated Manufacturing*, 23(5):517–532, October 2007.

[69] P. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robotmanipulators. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, volume 1, pages 484–489, Scottsdale, AZ, USA, May 1989.

[70] P. Ogren and N. E. Leonard. A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics*, 21(2):188–195, April 2005.

[71] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International Journal of Robotics Research*, 24(4):295, 2005.

[72] L. Qiu and W.-J. Hsu. A bi-directional path layout for conflict-free routing of AGVs. *International Journal of Production Research*, 39(10):2177–2195, 2001.

[73] S. Rajotia, K. Shanker, and J. Batra. A semi-dynamic time window constrained routeing strategy in an AGV system. *International Journal of Production Research*, 36(1):35–50, January 1998.

[74] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. *Journal of the ACM*, 41(4):764–790, 1994.

[75] J. H. Reif. Complexity of the mover's problem and generalizations. In 20^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 421–427, San Juan, Puerto Rico, October 1979.

[76] S. A. Reveliotis. Conflict resolution in AGV systems. *IIE Transactions*, 32(7):647–659, July 2000.

[77] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.

[78] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, second edition edition, 2003.

[79] S. R. S., K. Shanker, and J. Batra. Determination of optimal AGV fleet size for an FMS. *International Journal of Production Research*, 36(5):1177–1198, May 1998.

[80] M. Savchenko and E. Frazzoli. On the time complexity of conflict-free vehicle routing. In *Proceedings of the 2005 American Control Conference*, volume 5, pages 3536–3541, Portland, OR, USA, June 2005.

[81] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[82] J. Schwartz and M. Sharir. On the piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applications of Mathematics*, 4:298–351, 1983.

[83] V. Sharma, M. Savchenko, E. Frazzoli, and P. G. Voulgaris. Transfer time complexity of conflict-free vehicle routing with no communications. *The International Journal of Robotics Research*, 26(3):255–271, 2007.

[84] T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: a resolution-complete algorithm. *IEEE Transactions on Robots and Automation*, 18(1):42–49, February 2002.

[85] D. Sinriech and J. Tanchoco. Impact of empty vehicle flow on performance of single-loop AGV systems. *International Journal of Production Research*, 30(10):2237–2252, 1992.

[86] D. Smith, J. Frank, and A. J'onsson. Bridging the gap between planning and scheduling, 2000.

[87] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, volume 4, pages 3310–3317, San Diego, CA, USA, May 1994.

[88] F. Taghaboni-Dutta and J. M. Tanchoco. Comparison of dynamic routing techniques for automated guided vehicle system. *International Journal of Production Research*, 33(10):2653–2669, 1995.

[89] A. W. ter Mors, C. Yadati, C. Witteveen, and Y. Zhang. Coordination by design and the price of autonomy. *Journal of Autonomous Agents and Multi-Agent Systems*, on-line version, 2009.

[90] A. W. ter Mors, J. Zutt, and C. Witteveen. Context-aware logistic routing and scheduling. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 328–335, 2007.

[91] S. Trüg, J. Hoffmann, and B. Nebel. Applying automatic planning systems to airport ground-traffic control - a feasibility study. In *27^{th} Annual German Conference on AI, KI*, pages 183–197, 2004.

[92] J. Valk. *Coordination among autonomous planners*. PhD thesis, Delft University of Technology, December 2005.

[93] J. P. van den Berg and M. H. Overmars. Prioritized motion planning for multiple robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2217–2222, 2005.

[94] J. P. van den Berg and M. H. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[95] M. van der Heijden, M. Ebben, N. Gademann, and A. van Harten. Scheduling vehicles in automated transportation systems Algorithms and case study. *OR Spectrum*, 24(1):31–58, 2002.

[96] I. F. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, May 2006.

[97] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Transactions on Robotics and Automation*, 6(6), December 1990.

[98] C. W. Warren. Multiple robot path coordination using artificial potential fields. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, volume 1, pages 500–505, Cincinnati, Ohio, USA, May 1990.

[99] A. T. Wells and S. B. Young. *Airport Planning and Management.* McGraw-Hill, New York, 5$^{\text{th}}$ edition, 2004.

[100] G. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3(1):131–150, 1991.

[101] N. Wu and M. Zhou. Resource-oriented petri nets in deadlock avoidance of AGV systems. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA)*, pages 64–69, Seoul, Korea, May 2001.

[102] D. Zöbel. The deadlock problem: a classifying bibliography. *SIGOPS Oper. Syst. Rev.*, 17(4):6–15, 1983.

[103] J. Zutt. *Operational Transport Planning in a Multi-Agent Setting.* PhD thesis, Delft University of Technology, 2010. Forthcoming.

[104] J. Zutt and C. Witteveen. Multi-agent transport planning. In *Proceedings of the Sixteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'04)*, pages 139–146, Groningen, October 2004.

Summary

The world according to MARP

In multi-agent route planning (MARP), there is a set of agents each with a start location and a destination location on a shared infrastructure, and the goal of each agent is to reach its destination in the shortest possible time, without coming into conflict (e.g. because of a collision) with any of the other agents. The MARP problem is relevant in automated guided vehicle systems, with application domains in flexible manufacturing systems or at container terminals like Rotterdam or Singapore. Another application domain for MARP is airport taxi routing, where aircraft must taxi between runway and gate, while avoiding close proximity with other aircraft.

In the literature, a number of approaches to MARP exist. The first is to have one planner make a plan for all agents. Because of its complete overview, the centralized planner can in principle find an optimal set of agent route plans. Unfortunately, finding an optimal set of plans is intractable (as we show in this thesis[1]), and in practice optimal plans can only be found if there are not more than a handful of agents. A second approach, and the one we take in this thesis, is to let the agents plan one by one, each agent planning around the plans of the previous agents (in the sense that no conflict is introduced with any existing plans). In this *prioritized* approach (the agents plan in the order of their priority), it *is* possible to find an individually optimal route plan in reasonable time, although there is no guarantee that the combination of agent plans is also optimal. A third approach to MARP is to do very little planning (for example by choosing a fixed path between start and destination), and instead to check at every next step in the plan-execution phase whether it is safe to move forward, or whether a conflict will ensue. In this approach there are no guarantees for either local or global optimality, but it can be a convenient approach in dynamic environments, where unexpected changes can invalidate carefully crafted plans. If however there are no plans, then nothing can be disrupted, and any situation can be treated the same way. By contrast, the first two (planning) approaches to MARP require additional plan repair techniques to deal with unexpected incidents.

We present a model for MARP in which the infrastructure is modeled as a graph

[1] As far as we know, no previous complexity results exist for the problem of route planning over a known infrastructure. For robot motion planning planning problems, where agents can freely move around the available space (and are not restricted to move only along roads), the intractability *has* already been proved.

of *resources* (like roads and intersections); each resource has a capacity, which is the maximum number of agents that may occupy the resource at the same time. A route plan is a sequence of resources each with an occupation time interval, and the objective in MARP is to find a set of agent route plans such that the capacities of the resources are never exceeded. In our model, we also identify a number of additional constraints that have to be satisfied depending on the application domain. For example, we formulate constraints to forbid overtaking, constraints that forbid cyclic plans, and constraints that require a minimum separation between agents of at least one empty resource. Even if no additional constraints need to be satisfied, finding an optimal solution to the MARP problem is intractable.

Our route planning algorithms make use of *free time windows* on resources. A free time window is an interval during which an agent can make use of a resource without causing a conflict with any of the existing route plans. We search for a route plan by constructing a graph of the free time windows, and performing an adapted shortest-path search through this graph, which results in an optimal (shortest-time), conflict-free route plan for a single agent. Our contribution lies in the speed of our algorithm, which is much faster than previous optimal single-agent route planning algorithms. In addition, we also present a route planning algorithm that can find a conflict-free route plan along a sequence of locations (rather than from a start location to a single destination location), which is the first algorithm in its kind, as far as we know.

Planning approaches to MARP require additional mechanisms to deal with unexpected incidents during plan execution. In this thesis, we consider vehicle incidents that temporarily immobilize an agent (which can be caused by e.g. a human that steps into the path of an agent). If, after the slowdown, all agents including the delayed agent ignore the disruption, then a deadlock situation can arise. In the literature there exists a mechanism that can prevent deadlocks, and thus ensure that each agent can reach its destination safely, albeit with a possible delay. This mechanism is based on the *resource priority* that agents inherit after the planning process: from the set of agent plans, we can infer the order in which agents visit a resource (and the first agent to visit a resource has the highest resource priority). The deadlock prevention mechanism is simply to respect these resource priorities during plan execution, which means that agents sometimes have to wait for a delayed agent with a higher priority.

In some cases, there is no need to wait for a delayed agent, and the resource priority can be changed during plan execution without creating a deadlock situation. In the literature, one such priority-changing algorithm exists, but the conditions under which it allows a priority change are very strict, which limits the applicability of the algorithm. We developed a new priority-changing algorithm that allows changes in more situations, which therefore has the potential to achieve a bigger reduction in agent delay. Our algorithm makes use of a graph structure that can predict exactly which priority changes are safe, and which will lead to a deadlock.

In our experiments we evaluated the robustness of agent route plans, i.e., the property of plans to remain efficient, in terms of minimizing delay, even if unexpected incidents necessitate minor plan revisions. Above, we described three *schedule repair* algorithms[2]:

[2]We use the term *schedule* repair because only the timing of the plans is changed.

either fully respect the resource priorities during plan execution, or allow some priority changes using the existing priority-changing algorithm, or using our new algorithm. Respecting resource priorities can be seen as a baseline approach to incident handling: if it results in short delays, then no priority changes are necessary; if it would result in long delays, then we can seek to reduce this delay by changing some priorities at run-time.

In taxi route planning experiments on a model of Schiphol airport, the baseline method produced satisfactory results: in case there are only a few short incidents, then there is hardly any delay; in case there are many incidents of a long duration, then the average delay was still rarely more than 20% of the length of the agent plans. Experiments on other types of infrastructures (such as grid-like networks and small-world networks) also showed that delay is low for a small number of short incidents, but for a large number of long incidents, delays of around 100% were frequently reached. One reason why the results are so favourable for the airport experiments is that the locations of the start and destination points of the agents were not fully randomized, as aircraft agents travel between runways and gates. Hence, if aircraft agents use the same resource, then they are most likely heading in the same direction. Our experiments suggest that the most delay from incidents arises when agents that have to wait for each other travel in opposite directions. We also managed to reduce delay by changing priorities, and our algorithm, which produces the greatest number of priority changes, also managed the greatest reduction in delay.

A different set of experiments evaluated the cost of the multi-agent plan (which can be measured by e.g. recording the finish time of the latest agent) that results from sequential single-agent planning, for arbitrary agent priorities. Under the assumption that the optimal set of agent priorities will result in a close-to-optimal global plan, then measuring the difference between the best and the worst observed set of priorities gives an indication of the worst-case global plan cost that can be obtained using prioritized planning. In our experiments, we tried 100 different sets of agent priorities for a single problem instance, and for each type of infrastructure, we tried numerous instances. The best results were again obtained for the airport infrastructure, and again the locations of the start and destination points seem to be the main reason: because all agents travel in more or less the same direction, they manage to organize themselves into flows, regardless of which agent is allowed to plan first. For other types of infrastructures, it proved to be important to have multiple routes between any two locations. Otherwise, if e.g. one road connects two parts of the infrastructure, then this road can become a bottleneck if agents on either side of the road need to cross; arbitrarily assigned priorities can then lead to a situation where the travel direction of the resource alternates with every other agent, which is much less efficient than many agents using the resource at the same time in the same direction.

To conclude, we presented a prioritized approach to the multi-agent route planning problem. MARP has many important application domains like airport taxi routing and route planning for automated guided vehicles, and our new algorithm is fast enough to be applied to problem instances of realistic size. To deal with unexpected incidents, we have focussed on schedule repair methods, but in future work we can investigate how *plan* repair techniques (i.e., also allowing agents to choose a different route) can further reduce delay. Another direction of future research is how we can reduce the cost of

the global plan. Our route planning algorithms are optimal for a single agent, and the combination of individually optimal route plans sometimes leads to high-quality global plans — for example in the taxi route planning experiments, where the location of the start and destination points of the agents ensured that they travel in the same directions and therefore interfere little with each other. On other infrastructures, an arbitrary priority assignment can sometimes lead to a poor global plan. To improve global plan cost, one possibility is to find a better set of agent priorities. Another option is to investigate whether agents can coordinate before or during planning, such that an agent also takes into account the benefit of other agents during planning.

Samenvatting

De wereld volgens MARP

In *multi-agent route planning* (MARP) hebben we een verzameling agenten ieder met zijn eigen start- en bestemmingslocatie op een gedeelde infrastructuur, en het doel van iedere agent is om zijn bestemming zo snel mogelijk te bereiken, zonder in conflict te komen (bijvoorbeeld door een botsing) met één van de andere agenten. Het MARP probleem is relevant in systemen van *automated guided vehicles* (AGVs), met toepassingsdomeinen als flexibele productiesystemen of bij container terminals van de havens van bijvoorbeeld Rotterdam en Singapore. Een ander toepassingsdomein voor MARP is route planning voor taxiënde vliegtuigen, die van een gate naar een startbaan moeten of andersom, terwijl ze niet te dicht in de buurt mogen komen van andere vliegtuigen.

In de literatuur bestaan een aantal aanpakken voor het MARP probleem. De eerste aanpak is om één planner een plan te laten maken voor alle agenten. Vanwege diens volledige overzicht is het voor de centrale planner in principe mogelijk om een optimale verzamelingen routes te vinden. Helaas is het vinden van zo'n optimale verzameling plannen een ondoenlijk probleem (zoals we in dit proefschrift laten zien[3]), en in de praktijk lukt het slechts om optimale plannen te vinden voor probleem instanties met een beperkt aantal agenten. Een tweede aanpak, en degene die we in dit proefschrift hanteren, is om de agenten één voor één te laten plannen, zodat iedere agent om de plannen van de vorige agenten heen plant (in de zin dat geen conflict wordt geïntroduceerd met enig bestaand plan). In deze op prioriteiten gebaseerde aanpak (de agenten plannen op volgorde van prioriteit) is het wel mogelijk om een individueel optimaal plan te maken in redelijke tijd. Echter, er is geen garantie dat de combinatie van de plannen van de agenten ook optimaal is. Een derde aanpak voor het MARP probleem is om heel weinig daadwerkelijke planning uit te voeren (door bijvoorbeeld vantevoren vastgestelde paden te gebruiken) en in plaats daarvan om bij iedere volgende stap in de planexecutiefase te controleren of het veilig is om vooruit te gaan, of dat een conflict zal optreden. In deze aanpak zijn er geen garanties voor lokale of globale optimaliteit, maar het kan een handige aanpak zijn voor dynamische omgevingen, waar onverwachte veranderingen nauwkeurig uitgedachte plannen kunnen verstoren. Als er echter geen plannen zijn, kunnen die ook niet verstoord

[3]Voor zover wij weten bestaan er geen eerdere complexiteitsresultaten voor het routeplanningsprobleem over een gegeven infrastructuur. In de literatuur over het plannen van de bewegingen van robots, waarbij de agenten vrijelijk door de ruimte kunnen bewegen (en zich niet hoeven te beperken tot het rijden over wegen), is de ondoenlijkheid al wel aangetoond.

worden, en dan kan iedere situatie op dezelfde manier behandeld worden. Voor de eerste twee (plannings)aanpakken voor het MARP probleem, daarentegen, zijn additionele *plan repair* technieken nodig om met onverwachte incidenten om te kunnen gaan.

We presenteren een model voor MARP waarin de infrastructuur gemodelleerd wordt als een graaf van *resources* (zoals wegen en kruispunten); iedere resource heeft een capaciteit, wat het maximaal aantal agenten is dat tegelijkertijd van de resource gebruik mag maken. Een route plan is een sequentie van resources met voor iedere resource een tijdsinterval (dat aangeeft wanneer de agent zich op de resource zal begeven) en het doel in MARP is om voor iedere agent een routeplan te vinden, zodanig dat de capaciteiten van de resources nooit overschreden worden. In ons model onderscheiden we ook een aantal extra regels die afhankelijk van de applicatie ook in acht genomen moeten worden. Zo kan er een verbod zijn op inhalen, kunnen we cyclische plannen verbieden, en zijn er regels die ervoor zorgen dat er altijd minimaal één lege resource tussen twee agenten moet zitten. Ook als geen extra regels in acht genomen hoeven te worden is het ondoenlijk om een optimale oplossing te vinden voor het MARP probleem.

Onze routeplanningsalgoritmes maken gebruiken van de *vrije tijdsvensters* op resources. Een vrij tijdsvenster is een interval waarin een agent gebruik kan maken van een resource zonder een conflict te scheppen met enig bestaand plan. Het zoeken naar een routeplan kan dan gebeuren door een graaf van vrije tijdsvensters te maken, en een aangepast kortste-padalgoritme uit te voeren in deze graaf. Het resultaat van dit zoekalgoritme is een optimaal routeplan voor een individuele agent, dat geen conflicten bevat met andere plannen. Onze bijdrage ligt in de snelheid van het algoritme, want het is veel sneller dan eerdere optimale routeplanningsalgoritmes voor individuele agenten. We presenteren ook een algoritme dat een optimaal, conflictvrij plan kan vinden langs een opeenvolging van locaties (in plaats van een enkele start- en bestemmingslocatie); voor zover we weten is dit het eerste algoritme voor dit specifieke probleem.

Planningsaanpakken voor MARP hebben een extra mechanisme nodig om met onverwachte incidenten om te kunnen gaan tijdens de planexecutiefase. In dit proefschrift beschouwen we incidenten die een agent tijdelijk stil doen staan, bijvoorbeeld omdat een mens in de baan van de agent stapt. Als na het oponthoud alle agenten inclusief de vertraagde agent de vertraging negeren, dan kan een *deadlock* situatie optreden. In de literatuur bestaat een mechanisme dat zulke deadlocks kan voorkomen, en daardoor ervoor kan zorgen dat iedere agent toch zijn bestemming bereikt, zij het met een mogelijke vertraging. Dit mechanisme is gebaseerd op de *resourceprioriteiten* die de agenten na de planningsfase hebben verkregen: uit de verzameling van agentplannen kunnen we afleiden in welke volgorde een bepaalde resource door de agenten bezocht wordt; de eerste agent die een resource aandoet heeft per definitie de hoogste resourceprioriteit op die resource. Het mechanisme om deadlocks te voorkomen eist simpelweg dat deze resourceprioriteiten tijdens de executiefase worden gerespecteerd, wat wel inhoudt dat agenten soms moeten wachten op een vertraagde agent met een hogere prioriteit.

In sommige gevallen is het niet nodig om op een vertraagde agent te wachten en kan de resourceprioriteit tijdens de executiefase veranderd worden zonder dat een deadlock wordt geïntroduceerd. In de literatuur bestaat een algoritme om resourceprioriteiten aan te passen, alleen zijn de condities waaronder een verandering veilig worden geacht

tamelijk streng, zodat prioriteitsveranderingen in de praktijk niet vaak plaats kunnen vinden. Wij hebben een nieuw algoritme ontwikkeld dat prioriteiten kan veranderen in meer situaties, zodat het ook de mogelijkheid heeft om een grotere reductie in vertraging teweeg te brengen. Ons algoritme maakt gebruik van een graaf waarmee we precies kunnen voorspellen wanneer een prioriteitsverandering veilig is, en wanneer deze tot een deadlock zal leiden.

In onze experimenten evalueren we de robuustheid van agentplannen, waarmee we bedoelen de eigenschap van plannen om ook efficiënt te blijven (in termen van het minimaliseren van vertraging) als onverwachte incidenten kleine aanpassingen in de plannen noodzakelijk maken. Hierboven hebben we drie *schedule repair*[4] technieken besproken: of we respecteren de resourceprioriteiten, of we staan prioriteitsveranderingen toe door gebruik te maken van het algoritme uit de literatuur, of door ons eigen algoritme te gebruiken. Het respecteren van de resourceprioriteiten kan gezien worden als een basisaanpak: indien dit tot korte vertragingen leidt is het niet nodig om prioriteiten te veranderen; zou dit echter resulteren in lange vertragingen, dan kunnen we de vertaging proberen terug te dringen door tijdens de uitvoer van de plannen de resourceprioriteiten aan te passen.

In experimenten met het plannen van taxiroutes op een model van luchthaven Schiphol gaf de eenvoudige aanpak van prioriteiten respecteren bevredigende resultaten: indien er slechts een klein aantal korte incidenten zijn, is er niet of nauwelijks vertraging; in het geval er veel lange incidenten zijn, dan is de gemiddelde vertraging nog steeds niet meer dan 20% van de lengte van de plannen van de agenten. Experimenten met andere typen infrastructuren (zoals rasternetwerken en kleinewereldnetwerken (small-world networks)) laten ook zien dat de vertraging laag is bij een klein aantal korte incidenten, maar voor een groot aantal lange incidenten kan de vertraging al snel oplopen tot 100% van de planlengte. Een reden waarom de uitkomsten in het voordeel uitvallen van de experimenten op de luchthaven is dat de begin- en eindpunten van de agenten op Schiphol niet volledig willekeurig zijn vastgesteld, omdat de vliegtuigen altijd tussen gate en startbaan reizen of andersom. Dit betekent dat als agenten van dezelfde resource gebruik maken, dat ze hoogstwaarschijnlijk in dezelfde richting rijden. Onze experimenten suggereren dat de meeste vertraging onstaat als de agenten die op elkaar moeten wachten in verschillende richtingen rijden. Het is ons ook gelukt om de vertraging terug te brengen door de resourceprioriteiten aan te passen. Ons nieuwe algoritme, dat veruit de meeste prioriteitswijzigingen wist te bewerkstelligen, realiseerde ook de grootste reductie in vertraging.

We hebben ook experimenten uitgevoerd om de globale kosten van de verzameling agentplannen (die kosten kunnen bijvoorbeeld gemeten worden door te kijken naar de eindtijd van de laatste agent) te evalueren, indien de agenten in een willekeurige volgorde hun plannen maken. Onder de aanname dat de optimale agentvolgorde resulteert in een bijna optimaal globaal plan, dan zal het meten van het verschil tussen de beste en de slechtste volgorde een indicatie geven van het slechtst mogelijke plan dat je kan krijgen met sequentieel plannen. In onze experimenten hebben we voor iedere probleeminstantie 100 verschillende volgordes geprobeerd, en we hebben verscheidene instanties gebruikt voor ieder type infrastructuur. De beste resultaten werden wederom behaald op de

[4]We gebruiken de term *schedule* omdat we alleen de tijdsaspecten van de plannen aanpassen.

Schiphol infrastructuur, en ook hier was de ligging van de begin- en eindpunten weer doorslaggevend: aangezien veel agenten in min of meer dezelfde richting rijden, kunnen ze zichzelf in stromen organiseren, ongeacht welke agent eerst zijn plan mag maken. Voor andere typen infrastructuren bleek het belangrijk dat tussen een willekeurig paar locaties meer dan één verbinding bestaat. Is dat niet het geval, dan is het bijvoorbeeld mogelijk dat als een weg twee delen van de infrastructuur verbindt, dat die weg een knelpunt wordt als agenten aan beide zijden van de weg naar de andere kant moeten; willekeurig bepaalde prioriteiten kunnen er dan toe leiden dat de weg afwisselend in de ene en de andere richting gebruikt wordt, wat veel minder efficiënt is dan wanneer meerdere agenten de weg tegelijkertijd in dezelfde richting zouden gebruiken.

We hebben in dit proefschrift een sequentiële aanpak voor het MARP probleem gepresenteerd. MARP heeft veel belangrijke toepassingsgebieden zoals taxiroutes plannen op een luchthaven, en routes plannen voor AGVs, en ons nieuwe algoritme is snel genoeg om problemen van realistische grootte aan te kunnen. Om onverwachte incidenten aan te pakken hebben we onze aandacht gevestigd op schedule repair technieken, maar in toekomstig werk zouden we kunnen onderzoeken in hoeverre we vertraging verder kunnen terugdringen door ook *plan* repair technieken te gebruiken (dat wil zeggen dat we agenten ook zouden toestaan om een andere route te kiezen). Een andere richting voor verder onderzoek is het proberen terug te brengen van de kosten van het globale plan. Onze algoritmes zijn optimaal voor een individuele agent, en de combinatie van agentplannen leiden soms tot globale plannen met lage kosten — bijvoorbeeld in onze experimenten met het plannen van taxiroutes, waar de ligging van de begin- en eindpunten ervoor zorgt dat agenten ongeveer in dezelfde richting rijden en dus elkaar niet teveel hinderen. Op andere typen infrastructuren kan een willekeurige planningsvolgorde echter tot gevolg hebben dat het globale plan hoge kosten heeft. Eén mogelijkheid om die kosten te verlagen is om een betere planningsvolgorde voor de agenten te vinden. Een andere mogelijkheid is om te onderzoeken of we de agenten voor of tijdens de planning kunnen coördineren, zodat een agent tijdens de planning ook rekening houdt met de belangen van andere agenten.

Acknowledgements

Doing a PhD can be something of a solitary exercise. Not only are researchers in general encouraged to do something that no-one else does, during a PhD one should also demonstrate his or her ability to perform scientific research independently of others. Nevertheless, I have been fortunate to have had two fruitful and enjoyable partnerships during my four-and-a-half year PhD project.

First of all, my Almende colleague Xiaoyu Mao and I were pretty much inseparable, scientifically speaking of course, for the first 18 months of our joint project (which was a two-person project, aimed at improving mobility between academia and industry; we were both working at Almende, while I was supervised by Cees Witteveen from TU Delft, and Xiaoyu was supervised by Nico Roos from University Maastricht). Together we tread the precipitous path from vague problem description to concrete problem, and along the way we programmed interesting if scientifically irrelevant demos. Eventually, and inevitably, our scientific paths diverged: I settled on multi-agent route planning, while Xiaoyu decided to study scheduling of airport ground handling tasks.

My initial attitude towards multi-agent route planning was simply to use what was already there, but together with my TU Delft colleague Jonne Zutt we quickly found that existing methods could be improved upon, and from there my research got properly started, while his got an extra boost. I think that our competition over who could implement the fastest route planning algorithm benefited both our theses, while our competition over the chess board distracted only a little from our thesis writing...

Finally, a special thanks to Cees Witteveen, whose continuing support I have had for my entire academic career. Also, thanks to my colleagues in Almende for a pleasant and relaxed working environment (and I won't embarrass us all here by mentioning Friday evening Starcraft + pizza sessions), to Alfons Salden for a different view on my research, and finally to Jeroen Valk: I have 10.000 reasons to be grateful.

Curriculum Vitae

Adriaan Willem ter Mors was born in Zoetermeer, the Netherlands, on March 29, 1980. His early childhood saw him build civilizations that stood the test of time, build sprawling railroad empires, and rack up a number of Formula 1 World Championship titles that would make Micheal Schumacher eat his heart out. After that it all went downhill a little bit.

In 1998 he received his *VWO-diploma* with honours, after which he started his academic career at Delft University of Technology, where he studied computer science (*Technische Informatica*). In the course of his study, he enjoyed various positions as student-assistant (to finance his expensive cycling hobby), helping students with programming problems in languages like Java, Prolog, and assembler. In 2002 he joined the Collective Agent Based Systems (CABS) group, headed by Cees Witteveen, to work on his graduation project. Supervised by Cees and Jeroen Valk, a PhD student at the time, he studied coordination in multi-agent systems, and in particular the problem of finding a minimal set of constraints that allow the agents to act completely independently of each other in the subsequent planning phase. Adriaan not only received his master's degree in computer science with honours, his master's thesis also won the award for the best Master's thesis in (technical) computer science in the Netherlands in 2005.

After his master's study, he did a nine-month stint as an associate researcher on a project on developing information systems to aid emergency responders in disaster situations, before joining Almende to do a PhD on multi-agent route planning. Currently, he is continuing his research on agents, route planning, and coordination at the TU Delft.

SIKS dissertations series

1998

1 Johan van den Akker (CWI[5]) *DEGAS - An Active, Temporal Database of Autonomous Objects*

2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*

3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

4 Dennis Breuker (UM) *Memory versus Search in Games*

5 Eduard W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999

1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated Modelling of Quality Change of Agricultural Products*

2 Rob Potharst (EUR) *Classification using Decision Trees and Neural Nets*

3 Don Beal (UM) *The Nature of Minimax Search*

4 Jacques Penders (UM) *The Practical Art of Moving Physical Objects*

5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

6 Niek J.E. Wijngaards (VU) *Re-Design of Compositional Systems*

7 David Spelt (UT) *Verification Support for Object Database Design*

8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*

2000

1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*

2 Koen Holtman (TU/e) *Prototyping of CMS Storage Management*

3 Carolien M.T. Metselaar (UvA) *Sociaal-organisatorische Gevolgen van Kennistechnologie; een Procesbenadering en Actorperspectief*

4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

[5]Abbreviations: SIKS - Dutch Research School for Information and Knowledge Systems; CWI - Centrum voor Wiskunde en Informatica, Amsterdam; EUR - Erasmus Universiteit, Rotterdam; KUB - Katholieke Universiteit Brabant, Tilburg; KUN - Katholieke Universiteit Nijmegen; OU - Open Universiteit; RUL - Rijksuniversiteit Leiden; RUN - Radboud Universiteit Nijmegen; TUD - Technische Universiteit Delft; TU/e - Technische Universiteit Eindhoven; UL - Universiteit Leiden; UM - Universiteit Maastricht; UT - Universiteit Twente, Enschede; UU - Universiteit Utrecht; UvA - Universiteit van Amsterdam; UvT - Universiteit van Tilburg; VU - Vrije Universiteit, Amsterdam.

5 Ruud van der Pol (UM) *Knowledge-Based Query Formulation in Information Retrieval*

6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*

7 Niels Peek (UU) *Decision-Theoretic Planning of Clinical Patient Management*

8 Veerle Coupé (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*

9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*

10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*

11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*

2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*

3 Maarten van Someren (UvA) *Learning as Problem Solving*

4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*

6 Martijn van Welie (VU) *Task-Based User Interface Design*

7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*

8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*

9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice BRAHMS: a Multiagent Modeling and Simulation Language for Work Practice Analysis and Design*

11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002

1 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*

2 Roelof van Zwol (UT) *Modelling and Searching Web-based Document Collections*

3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*

4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*

5 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments Inhabited by Privacy-Concerned Agents*

6 Laurens Mommers (UL) *Applied Legal Epistemology; Building a Knowledge-based Ontology of the Legal Domain*

7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*

8 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*

10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*

11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

13 Hongjing Wu (TU/e) *A Reference Architecture for Adaptive Hypermedia Applications*

14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*

17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003

1 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*

2 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*

3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

4 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*

5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law – A Modelling Approach*

6 Boris van Schooten (UT) *Development and Specification of Virtual Environments*

7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*

8 Yong-Ping Ran (UM) *Repair-Based Scheduling*

9 Rens Kortmann (UM) *The Resolution of Visually Guided Behaviour*

10 Andreas Lincke (UT) *Electronic Business Negotiation: Some Experimental Studies on the Interaction between Medium, Innovation Context and Cult*

11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

12 Roeland Ordelman (UT) *Dutch Speech Recognition in Multimedia Information Retrieval*

13 Jeroen Donkers (UM) *Nosce Hostem – Searching with Opponent Models*

14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*

15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*

16 Menzo Windhouwer (CWI) *Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouse*

17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*

18 Levente Kocsis (UM) *Learning Search Decisions*

2004

1 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*

2 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*

3 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*

4 Chris van Aart (UvA) *Organizational Principles for Multi-Agent Architectures*

5 Viara Popova (EUR) *Knowledge Discovery and Monotonicity*

6 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*

7 Elise Boltjes (UM) *Voorbeeld$_{IG}$ Onderwijs; Voorbeeldgestuurd Onderwijs, een Opstap naar Abstract Denken, vooral voor Meisjes*

8 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale Politiële Gegevensuitwisseling en Digitale Expertise*

9 Martin Caminada (VU) *For the Sake of the Argument; Explorations into Argument-based Reasoning*

10 Suzanne Kabel (UvA) *Knowledge-rich Indexing of Learning-objects*

11 Michel Klein (VU) *Change Management for Distributed Ontologies*

12 The Duy Bui (UT) *Creating Emotions and Facial Expressions for Embodied Agents*

13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*

14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*

15 Arno Knobbe (UU) *Multi-Relational Data Mining*

16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*

17 Mark Winands (UM) *Informed Search in Complex Games*

18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*

19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*

20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*

2005

1 Floor Verdenius (UvA) *Methodological Aspects of Designing Induction-Based Applications*

2 Erik van der Werf (UM) *AI techniques for the game of Go*

3 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*

4 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*

5 Gabriel Infante-Lopez (UvA) *Two-Level Probabilistic Grammars for Natural Language Parsing*

6 Pieter Spronck (UM) *Adaptive Game AI*

7 Flavius Frasincar (TU/e) *Hypermedia Presentation Generation for Semantic Web Information Systems*

8 Richard Vdovjak (TU/e) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*

9 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*

10 Anders Bouwer (UvA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*

11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search*

12 Csaba Boer (EUR) *Distributed Simulation in Industry*

13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*

14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*

15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*

16 Joris Graaumans (UU) *Usability of XML Query Languages*

17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*

18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*

19 Michel van Dartel (UM) *Situated Representation*

20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*

21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006

1 Samuil Angelov (TU/e) *Foundations of B2B Electronic Contracting*

2 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*

3 Noor Christoph (UvA) *The role of metacognitive skills in learning to solve problems*

4 Marta Sabou (VU) *Building Web Service Ontologies*

5 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*

6 Ziv Baida (VU) *Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling*

7 Marko Smiljanic (UT) *XML schema matching – balancing efficiency and effectiveness by means of clustering*

8 Eelco Herder (UT) *Forward, Back and Home Again - Analyzing User Behavior on the Web*

9 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*

10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*

11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*

12 Bert Bongers (VU) *Interactivation - Towards an e-cology of people, our technological environment, and the arts*

13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*

14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change*

15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*

16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*

17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*

18 Valentin Zhizhkun (UvA) *Graph transformation for Natural Language Processing*

19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*

20 Marina Velikova (UvT) *Monotone models for prediction in data mining*

21 Bas van Gils (RUN) *Aptness on the Web*

22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*

23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*

24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*

25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*

26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*

27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*

28 Borkur Sigurbjornsson (UvA) *Focused Information Access using XML Element Retrieval*

2007

1 Kees Leune (UvT) *Access Control and Service-Oriented Architectures*

2 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality: A Formal Approach*

3 Peter Mika (VU) *Social Networks and the Semantic Web*

4 Jurriaan van Diggelen (UU) *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*

5 Bart Schermer (UL) *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*

6 Gilad Mishne (UvA) *Applied Text Analytics for Blogs*

7 Natasa Jovanovic' (UT) *To Whom It May Concern - Addressee Identification in Face-to-Face Meetings*

8 Mark Hoogendoorn (VU) *Modeling of Change in Multi-Agent Organizations*

9 David Mobach (VU) *Agent-Based Mediated Service Negotiation*

10 Huib Aldewereld (UU) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*

11 Natalia Stash (TU/e) *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*

12 Marcel van Gerven (RUN) *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*

13 Rutger Rienks (UT) *Meetings in Smart Environments; Implications of Progressing Technology*

14 Niek Bergboer (UM) *Context-Based Image Analysis*

15 Joyca Lacroix (UM) *NIM: a Situated Computational Memory Model*

16 Davide Grossi (UU) *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*

17 Theodore Charitos (UU) *Reasoning with Dynamic Networks in Practice*

18 Bart Orriens (UvT) *On the development and management of adaptive business collaborations*

19 David Levy (UM) *Intimate relationships with artificial partners*

20 Slinger Jansen (UU) *Customer Configuration Updating in a Software Supply Network*

21 Karianne Vermaas (UU) *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*

22 Zlatko Zlatev (UT) *Goal-oriented design of value and process models from patterns*

23 Peter Barna (TU/e) *Specification of Application Logic in Web Information Systems*

24 Georgina Ramírez Camps (CWI) *Structural Features in XML Retrieval*

25 Joost Schalken (VU) *Empirical Investigations in Software Process Improvement*

2008

1 Katalin Boer-Sorbán (EUR) *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*

2 Alexei Sharpanskykh (VU) *On Computer-Aided Methods for Modeling and Analysis of Organizations*

3 Vera Hollink (UvA) *Optimizing hierarchical menus: a usage-based approach*

4 Ander de Keijzer (UT) *Management of Uncertain Data - towards unattended integration*

5 Bela Mutschler (UT) *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*

6 Arjen Hommersom (RUN) *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*

7 Peter van Rosmalen (OU) *Supporting the tutor in the design and support of adaptive e-learning*

8 Janneke Bolt (UU) *Bayesian Networks: Aspects of Approximate Inference*

9 Christof van Nimwegen (UU) *The paradox of the guided user: assistance can be counter-effective*

10 Wauter Bosma (UT) *Discourse oriented Summarization*

11 Vera Kartseva (VU) *Designing Controls for Network Organizations: a Value-Based Approach*

12 Jozsef Farkas (RUN) *A Semiotically oriented Cognitive Model of Knowlegde Representation*

13 Caterina Carraciolo (UvA) *Topic Driven Access to Scientific Handbooks*

14 Arthur van Bunningen (UT) *Context-Aware Querying; Better Answers with Less Effort*

15 Martijn van Otterlo (UT) *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains*

16 Henriette van Vugt (VU) *Embodied Agents from a User's Perspective*

17 Martin Op't Land (TUD) *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*

18 Guido de Croon (UM) *Adaptive Active Vision*

19 Henning Rode (UT) *From document to entity retrieval: improving precision and performance of focused text search*

20 Rex Arendsen (UvA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met een overheid op de administratieve lasten van bedrijven*

21 Krisztian Balog (UvA) *People search in the enterprise*

22 Henk Koning (UU) *Communication of IT-architecture*

23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*

24 Zharko Aleksovski (VU) *Using background knowledge in ontology matching*

25 Geert Jonker (UU) *Efficient and Equitable exchange in air traffic management plan repair using spender-signed currency*

26 Marijn Huijbregts (UT) *Segmentation, diarization and speech transcription: surprise data unraveled*

27 Hubert Vogten (OU) *Design and implementation strategies for IMS learning design*

28 Ildiko Flesh (RUN) *On the use of independence relations in Bayesian networks*

29 Dennis Reidsma (UT) *Annotations and subjective machines- Of annotators, embodied agents, users, and other humans*

30 Wouter van Atteveldt (VU) *Semantic network analysis: techniques for extracting, representing and querying media content*

31 Loes Braun (UM) *Pro-active medical information retrieval*

32 Trung B. Hui (UT) *Toward affective dialogue management using partially observable markov decision processes*

33 Frank Terpstra (UvA) *Scientific workflow design; theoretical and practical issues*

34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*

35 Benjamin Torben-Nielsen (UvT) *Dendritic morphology: function shapes structure*

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*

2 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*

3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*

4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*

5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality*

6 Muhammad Subianto (UU) *Understanding Classification*

7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*

8 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*

9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*

10 Jan Wielemaker (UvA) *Logic programming for knowledge-intensive interactive applications*

11 Alexander Boer (UvA) *Legal Theory, Sources of Law & the Semantic Web*

12 Peter Massuthe (TU/e, Humboldt-Universtät zu Berlin) *Operating Guidelines for Services*

13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*

14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*

15 Rinke Hoekstra (UvA) *Ontology Representation - Design Patterns and Ontologies that Make Sense*

16 Fritz Reul (UvT) *New Architectures in Computer Chess*

17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*

18 Fabian Groffen (CWI) *Armada, An Evolving Database System*

19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*

20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*

21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*

22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*

23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*

24 Annerieke Heuvelink (VU) *Cognitive Models for Training Simulations*

25 Alex van Ballegooij (CWI) *"RAM: Array Database Management through Relational Mapping"*

26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*

27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*

28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*

29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*

30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*

31 Sofiya Katrenko (UvA) *A Closer Look at Learning Relations from Text*

32 Rik Farenhorst and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*

33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*

34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*

2 Ingo Wassink (UT) *Work flows in Life Science*

3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*

4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*

5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*

6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*

7 Wim Fikkert (UT) *A Gesture interaction at a Distance*

8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*

9 Hugo Kielman (UL) *Politiële gegevensverwerking en Privacy, Naar een effectieve waarborging*

10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*