Dealing with Uncertainty in Operational Transport Planning

Jonne Zutt, Arjan van Gemund, Mathijs de Weerdt, and Cees Witteveen

Abstract An important problem in transportation is how to ensure efficient operational route planning when several vehicles share a common road infrastructure with limited capacity. Examples of such a problem are route planning for automated guided vehicles in a terminal and route planning for aircraft taxiing at airports. Maintaining efficiency in such transport planning scenarios can be difficult for at least two reasons. Firstly, when the infrastructure utilization approaches saturation, traffic jams and deadlocks may occur. Secondly, incidents where vehicles break down may seriously reduce the capacity of the infrastructure and thereby affect the efficiency of transportation. In this chapter we describe a new approach to deal with congestion as well as incidents using an intelligent infrastructure. In this approach, infrastructural resources (road sections, crossings) are capable of maintaining reservations of the use of that resource. Based on this infrastructure, we present an efficient, context-aware, operational transportation planning approach. Experimental results show that our context-aware planning approach outperforms a traditional planning technique and provides robustness in the face of incidents, at a level that allows application to real-world transportation problems.

Jonne Zutt
Department of Software Technology, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands, e-mail: j.zutt@tudelft.nl

Arjan van Gemund
Department of Software Technology, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands, e-mail: a.j.c.vangemund@tudelft.nl

Mathijs de Weerdt
Department of Software Technology, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands, e-mail: m.m.deweerdt@tudelft.nl

Cees Witteveen
Department of Software Technology, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands, e-mail: c.witteveen@tudelft.nl

1 Introduction

Transportation is one of the strongest growing activities in our society, and as a result, there is an increasing need for improving the transportation process of public as well as freight transportation. One way to meet these efficiency demands is to automate significant parts of the transportation process. For example, in 1988, European Container Terminals realized the first ever robotized container terminal where completely controlled automated guided vehicles drive around 24 hours a day. Another initiative is the to be realized fully automated underground logistic system at Amsterdam airport, connecting the flower market of Aalsmeer, the airport, and a new rail terminal to one another. Also in public transportation automation has been applied. Already in 1939, General Motors presented a vision of "driver-less" vehicles moved under automated control at that year's World's Fair in New York. A fully automated highway was initially examined by General Motors during the late 1970s. Due to the advances in computing technologies, microelectronics, and sensors in the 1980s, the University of California program Partners for Advanced Transit and Highways has carried out significant research and development efforts in highway automation since the 1980s. In Europe, automation has been applied in public transportation as well. For example, in Paris, the underground railway is partly automated. Other examples of automation are car navigation systems that not only are capable to advise and assist, but also, maybe in the near future, will be used to monitor and partially direct human driver behavior.

One of the main promises of automation in public and freight transportation is to help in detecting and resolving possible *conflicts* (e.g., collisions) between large amounts of vehicles moving on a rather restricted infrastructure (highways, railways, or air corridors). For example, on-board computers help in detecting obstacles, assist the driver if sight is limited, and give advice if accidents are about to occur.

Another main feature of automation, however, is that it might also assist in *preventing* such conflicts to occur by careful *planning* of transportation activities, where route choices are determined and communicated in advance. One significant property of planning in transportation is that it occurs in a *distributed* way, without centralized control. Since, usually, every vehicle in a transportation process is planning its transportation independently from the other, conflicts have to be detected locally and solved in real-time. The traditional solution to preventing these conflicts between plans is as taken in our everyday traffic: a set of operational conflict-resolution rules such as traffic rules (keep right), traffic lights (semaphores), and dynamic traffic guidance systems ensure effective conflict resolution when the plans are *executed*, i.e., when the previously planned route choices are followed.

The problem with this traditional approach is that, first of all, due to the high load on road section resources, in reality travel times as planned are nearly unpredictable, even if no incidents occur. This means that, normally, the quality of plans cannot be guaranteed as vehicles are dependent upon the plans of others. In addition, operational conflict rules might not always be successful: the high load may easily lead to deadlocks, where a number of vehicles is waiting for one another without any

progress. This, for example, might happen if four vehicles approach an intersection at the same time from different directions. Furthermore, as a consequence of this type of conflict resolution, the infrastructure is often used inefficiently: sometimes vehicles are waiting, while a short detour could have prevented this.

To solve these problems, we need to deal with conflict resolution not during the *execution* of the individual transportation plans, but during the *planning* phase. Therefore, a more sophisticated approach would be to split the problem into a first phase, in which routes are found for each of the vehicles individually, and a second phase, in which feasibility of the collection of routes is ensured, *before* these individual plans are actually executed. In fact, this form of plan conflict resolution can be considered as a form of *plan repair*, modifying individual plans if they are in conflict. As a representative of this approach, Broadbent et al. [3] employ a simple shortest path algorithm to find a set of initial routes. In case of *catching-up* conflicts, some vehicles are slowed down; for *head-on* conflicts, an alternative route is found that does not make use of the road at which the conflict occurred. Broadbent's algorithm can be used both on unidirectional and bidirectional infrastructures, but in the latter case it need not find the optimal solution. Like Broadbent et al., the approach proposed by Hatzack and Nebel [8] can also be considered as such a two-step approach. Compared with earlier approaches, however, Hatzack and Nebel use a more refined model of the infrastructure by considering parts of the infrastructure (such as lane segments and intersections) as resources having a limited capacity. Once the individual routes have been chosen, conflict resolution is then modeled as a *job-shop scheduling* problem with blocking to ensure that the capacity constraints on the resources are not violated.

Instead, however, of using such a two-step approach to distributed transportation planning, the approach in this chapter aims at a full *integration* of the route planning and the conflict-resolution process. To this end, we assume the existence of an automated, distributed, intelligent infrastructure that can be used by planners to realize a personal but so-called *context-aware* traffic plan. The basic idea here is that infrastructural resources, like road sections and crossings, maintain a list of reservations (time windows) of the use of that resource. While making their transportation plans, vehicles query a resource for its availability during some time interval and make time-window reservations for the use of that resource. Hence, individual vehicles are capable of planning a route in such a way that the influence of all plans of other vehicles have been taken into account, i.e., they are able to construct a *context-aware* route plan.

In this chapter, firstly, we elaborate upon the above idea for context-aware planning, because such a method can guarantee that traffic plans do not suffer from inherent unpredictability, since context-aware planning methods take into account conflicts in the planning phase instead of the execution phase. In particular, we not only review different existing variants of context-aware planning methods that may be used given such an intelligent infrastructure, we also present an algorithm that is optimal for one vehicle, given the reservations of the others, and computationally more efficient than any of its predecessors. We compare the performance of this im-

proved context-aware planning method to a traditional planning method that solves conflicts in the plan execution phase.

Secondly, we address the distributed transportation problem when incidents *do* occur. In our case, we consider incidents to be events that cannot be anticipated in advance and that have a negative influence on the planned activities, such as a collision, or a vehicle with a motor that is malfunctioning. Note that if incidents occur, even our context-aware route planning algorithm again cannot guarantee conflict-free execution of individual transportation plans and we are forced, as in the traditional approach, to resolve conflicts in the execution phase. As we will show, however, we are able to revise plans to take the consequences of incidents into account by making some adaptations of the context-aware route planning algorithm. We show that also under incident conditions, a context-aware route planning approach performs better than resolving conflicts in the execution phase.

These contributions are presented in this chapter as follows. We first introduce our model of an infrastructure, vehicles, and requests, and a description of the types of solutions we expect (Section 2). Then we discuss some existing work on context-aware approaches, and introduce our improved method in Section 3. In Section 4 we propose two different ways to extend context-aware methods to deal with incidents. Thereafter, we evaluate these methods by running a number of simulations in a synthetic grid infrastructure, as well as in a realistic network of the Dutch national airport near Amsterdam.

2 A framework for distributed operational transport planning

To discuss the context-aware route planning algorithms that we compare with the traditional approaches, we first need to specify the building blocks of our approach to distributed transportation planning. In our framework, we distinguish a *transportation network* consisting of resources, *transportation requests (tasks)*, and *transport agents* who make *transportation plans* for vehicles to serve the transportation requests using the transportation network.

Transportation network A transportation network, or *infrastructure* is a tuple (R,E,K,D,S). Here, each of the n (infrastructure) *resources* $r \in Rr$ represents space that can be occupied by the transport agents, e.g., a road, a road segment, part of an intersection or a parking space. The directed *connectivity* relation $E \subseteq R \times R$ defines which infrastructure resources a transport agent can traverse to from a given infrastructure resource. The *distance* function $D : R \to \mathbb{R}^+$ gives, for each resource $r \in R$, the positive non-zero distance of traversing r.

For all infrastructure resources $r \in R$, the *capacity* function $K : R \to \mathbb{N}$ specifies the number $K(r)$ of agents that can use resource r simultaneously, and the *speed* function $S : R \to \mathbb{R}^+$ specifies the maximum allowed driving speed $S(r)$ at resource r. Note that this maximum is defined independently from the vehicle using the resource.

Transportation requests A transportation request or *task* j is modeled by a tuple $(f_j, s_j, \tau_j^s, d_j, \tau_j^d, \pi_j)$ and specifies the request to pick up some amount of freight f_j to a certain source location $s_j \in R$ within a specified time window $\tau_j^s = [t_{j,1}^s, t_{j,2}^s]$ and to deliver it at a specified destination location d_j within a time window $\tau_j^d = [t_{j,1}^d, t_{j,2}^d]$. Associated with each request j there is a *reward function* $\pi_j : W \times W \to \mathbb{R}$, where W denotes a set of time windows. This reward is maximized if the request is executed within its time windows, and will typically be (much) smaller if one or both of the time windows of the transportation request are violated.[1]

Transport agents The transportation agents A are the transport planners. Each transport agent owns exactly one vehicle. The task of a transport agent is to compute a plan for the tasks (transportation requests) it accepted and then to execute this plan. Intuitively, each agent will try to maximize the rewards associated to its transportation requests, while minimizing the (traversal) costs. We assume that similar to infrastructure resources, for agents A there is a a *capacity* function $K : A \to \mathbb{N}$, specifying the maximum load capacity. If J_t is the set of transportation requests loaded by an agent $a \in A$ at time t, the constraint $\sum_{j \in J_t} f_j \leq K(a)$ (i.e., the sum of all loaded freight is smaller than the capacity of the vehicle) always holds. Likewise, the *speed* function $S : A \to \mathbb{R}$ specifies the maximum driving speed $S(a)$ of the vehicle of agent a. Furthermore, at each point $t \in T$ in time, each transport agent $a \in A$ claims exactly the infrastructure resource r at which agent a resides at time t. If r_1 and r_2 are infrastructure resources with $(r_1, r_2) \in E$, and agent a holds a claim at infrastructure resource r_1, it should claim resource r_2 after having traversed resource r_1, but before (or when) releasing its claim on resource r_1.

Transportation plans Each agent $a \in A$ plans a route to execute the transportation requests it has been assigned. Such a route $Rt_a = (r_1, r_2, \ldots, r_k)$ for agent a is represented as a sequence of resources such that resources r_i and r_{i+1} are connected to each other, i.e., $(r_i, r_{i+1}) \in E$ for $1 \leq i < k$. Accompanying this route Rt_a, the schedule Sd_a of agent a provides information on when each of these resources in Rt_a are claimed. Schedule $Sd_a = (t_1, t_2, \ldots, t_k)$ is a sequence of time points, where t_i specifies the time at which agent a claims resource r_i. This implies that agent a uses r_i during the time window $[t_i, t_{i+1})$ for $1 \leq i < k$ and uses resource r_k during time window $[t_k, \infty)$. Obviously, at any time, the route and schedule have the same length, i.e., $\forall a \in A : |Rt_a| = |Sd_a|$.[2]

Although every individual agent plan might be feasible, it may be the case that the set of all agent plans is not feasible, because some of these plans may be in *conflict*. Somewhat simplifying[3] we say that there is a conflict in the set of all agent plans

[1] Allen's interval algebra [1] defines that a time-interval τ' is during time-interval τ if and only if time window τ' does not start before time window τ and does not end at a later time.

[2] We use $|S|$ to denote the cardinality of a set S.

[3] To define all types of conflicts that might occur is somewhat more involved. The reader is referred to the PhD thesis of Zutt [20] for a complete overview.

if there is a subset $A' \subseteq A$, a resource $r \in R$, and a time t, such that at time t, every agent $a \in A'$ has a claim on resource r and $K(r) < |A'|$. If there is no such conflict, the set of agent plans is said to be conflict free.

The ultimate goal of solving a distributed transportation planning problem based upon this framework is to come up with a set of conflict-free route plans before these plans are executed. We describe such a route planning method in the next section.

3 Operational transport planning methods

In this section, we first briefly discuss some traditional approaches to operational transport planning. Then, we give the state-of-the art in *context-aware* routing, where conflicts between agents are removed during planning and before execution. At the end of this section, we propose a more efficient context-aware routing algorithm that we will also use in our experiments.

3.1 Conventional transportation planning approaches

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

Using such an approach in our framework, the simplest way to determine the route for an agent, neglecting the presence of other agents on a given infrastructure, is to have each agent plan a shortest path from its current resource location to its destination resource. A basic *shortest-path algorithm*, such as Dijkstra [6], can be used by the agents to achieve this.

The resolution of operational conflicts is then done by defining resource usage rules that prioritize vehicles when they enter an intersection at the same time. These resource usage rules can be based on static aspects related to the importance of the task, or on dynamic aspects, such as who arrives first. Examples of such static resource usage rules are:

- task-priority: when the rewards for executing tasks are not constant, agents that are executing tasks with a high reward should be given priority. For example, an ambulance, police, or fire brigade in action could precede regular traffic;
- vehicle-priority: each type of vehicle is assigned its own priority level. For example, buses and trucks can be assigned higher priority than personal vehicles.

Examples of dynamic resource usage rules are:

- first-come-first-served: for example, in the USA at a four-way-stop road crossing, the vehicle that enters a crossroad first gains the highest priority;

- longest-waiter-first: during plan execution, agents possibly have to wait at several occasions. This rule is similar to the previous one, but sums the waiting times of multiple crossroads;
- urgent-deadline-first: when agents are executing tasks with delivery deadlines, the agent that has the least slack should go first.

As remarked above, the problem with these traditional approaches is that travel times become almost *unpredictable*: an agent must at least have some knowledge of what the other agents are doing to know how this will affect its own plan. Even more important is the possibility that *deadlocks* occur, which means that the agents will not even be able to execute their plans. Finally, more knowledge about each others actions can improve the agent's decision making, which in turn may improve the performance of all agents.

More in general, the traditional approach can be compared with a pure *generate-and-test* approach to solving transportation planning problems: first, the routes are generated and then a test approach is followed, which detects conflicts during execution (e.g.,, the resource requested is already occupied) and then finds a solution to the problem (by using priorities). The resource-usage rules described above are much more effective, however, if one could push the test approach into the route generator. This implies that agents should use more conflict information available during the generation stage and hence make better decisions. This is the essence of the *context-aware routing* approach, which we propose as an alternative to the traditional approach that, from now on, will be used as a *baseline approach* (UNINFORMED) in comparison with the context-aware approach.

3.2 Context-aware route planning

Context-aware routing can be viewed as a multi-agent extension to single-agent shortest-path algorithms such as Dijkstra [6]. Typically, however, shortest-path algorithms do not take into account the plans of other agents while searching for a shortest time path from a source to a destination location. Taking into account the effect of other plans has some important consequences. For example, when a path to a node has been found in Dijkstra's shortest path algorithm, it is known that the current path to this node is the shortest, and the algorithm does not need to consider any other paths leading to this node. In context-aware routing, however, the first (and shortest) path to reach a resource is not necessarily the one that will result in the shortest path from the source to the destination, via the current resource. Consider the example in Figure 1. From the first (and only) free time window (a period without any reservations) on start resource r_s, we can reach both free time windows on resource r_1 (which is on a direct path to the destination resource r_d). However, from the first free time window on r_1, $f_{1,1} = [0,2]$, we cannot reach any free time window on r_d, because on the destination resource there is a reservation until time 5. Hence, we must go from r_s to r_1 at time 4 (assuming travel times of 1 for all resources, by time 4 we will have had enough time to traverse r_s). Then, we can

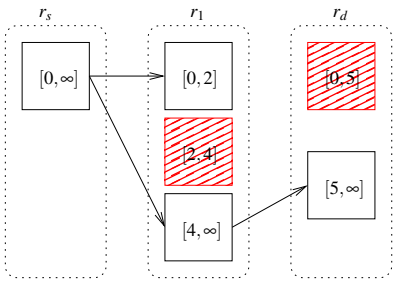


Fig. 1 This graph represents the time windows at three resources. The shaded boxes denote time intervals reserved by other agents. The traversal time for each resource is one. The first arrival at resource r_1, at time 1, will not lead to a shortest path to destination resource r_d (in fact, a path to r_1 should first return to r_s since r_1 has been reserved from 2 to 4). Instead, we must consider the path that visits r_1 during its second free time window, which starts from 4.

leave r_1 at time five, entering r_d at time 5, at the start of the free time window on the destination resource. This clearly shows that in context-aware routing we sometimes need to consider more than one visit to a resource due to reservations made by other agents.

Hence, a context-aware routing algorithm has to take into account not only whether a next resource is reachable from the current one, but also whether there is a suitable *free time window* on it, in which no reservations have been made by other agents. Consequently, for each location a set of disjoint free time windows is computed from all known reservations that exactly specifies the time windows at which the load of the location is smaller than its capacity (see Figures 2 and 3). Furthermore, a reachability relation is defined that specifies which free time windows at one location can be reached from which free time windows at another location (similar to arcs in a normal graph). When a shortest-path algorithm is used in this graph, each individual plan is optimal given that the reservations of all other agents do not change.

To illustrate this approach, let us look at the example depicted in Figure 2. The task is to route from source location r_s to destination location r_d starting at time $t = 0$, either by traveling via location r_1 or via location r_2, while taking into account the specified reservations of other agents. At first sight, it seems quicker to traverse via location r_1, because location r_2 is already reserved up to time $t = 6$. However, from location r_2 the journey cannot continue because of the reservation $[0,5)$ in location r_d, and the agent is not allowed to wait in location r_1 due to the reservation $[2,8)$ over there. Hence, the agent must wait in resource r_s until time $t = 8$ if it desires to travel the upper route. The free time-window graph gives somewhat more information. There is only an arc from location r_1 to location r_d from the free time window $[8,\infty)$, so one can immediately infer that the upper route has costs 9, and thus the lower route with costs is quicker with costs 7 (assuming the all locations have a traversal time of 1).

This single-agent source-destination routing computes a shortest path (in time) for an agent to traverse from its initial location to a destination location. However, in transportation planning, agents have to look further. They can be assigned multiple tasks, and hence have to create a plan to visit multiple locations one after another. The sequence of loading and unloading locations that an agent has to travel to is referred to as the *visiting sequence* of that agent. The route for a visiting sequence can be found by computing a shortest path between any two locations using context-aware routing, and by including the loading or unloading time at each location.

Unfortunately, finding a globally optimal plan (minimizing the sum of the costs) for all visiting sequences of all vehicles is NP-hard [13], so we cannot expect to find an efficient (polynomial time) algorithm for this problem. We will therefore restrict ourselves to an efficient algorithm that does not find optimal solutions. The idea is to apply a context-aware routing method each time an agent takes up a new transportation request. For a transportation request from a loading to an unloading location all possible ways of inserting these two locations in a visiting sequence are considered as long as the loading takes place before the unloading and the capacity of the vehicle is not exceeded. The agents insert such transportation requests in the order in which they arrive and without interleaving the planning with other agents. This causes the resulting plans to be usually sub-optimal. We will illustrate this sub-optimality by two examples. First, due to the arbitrary ordering: suppose that agent a_1 plans earlier than agent a_2 in this ordering of agents, and that agents a_1 and a_2 share some infrastructure resources in their routes. If an optimal plan requires that agent a_2 precedes agent a_1 in at least one of these shared infrastructure resources, this plan might not be found (in the case that agent a_1 reaches the infrastructure resource earlier than agent a_2). Second, due to the absence of interleaved planning: if agent a_1 first has to precede agent a_2, but later has to take priority in any optimal plan, then such a plan also cannot be found, because the agents create a complete plan for all of their transportation requests at once when it is their turn.

We would like to point out that there are a number of approaches related to this approach, in which route planning problems consist of finding a free path of resources from the origin to a destination, taking into account reservations that have been made by other agents using the same infrastructure. For example, the algorithm proposed by Huang et al. [9] finds a path through the (graph of) free time windows on the resources, rather than directly through the graph of resources. Huang et al.'s algorithm assumes unit capacity for all resources and is optimal both for unidirectional and bidirectional networks. Their algorithm runs in $O((n+m)^2 \log(n+m))$ for one vehicle, with n being the number of resources, and m the number of connections between the resources. Fujii et al. [7] combine the search through free time windows with a heuristic that calculates the shortest path from the current resource to the destination resource, assuming no other traffic. The solution method proposed should result in an optimal, polynomial-time algorithm, but the description of the algorithm is not entirely correct, and the time complexity of this algorithm is unknown. The work of Kim and Tanchoco [10] is similar to the work of Fujii et al., but their treatment of the problem and the analysis of their algorithm is more comprehensive. Kim and Tanchoco's algorithm finds the (individually) optimal solution

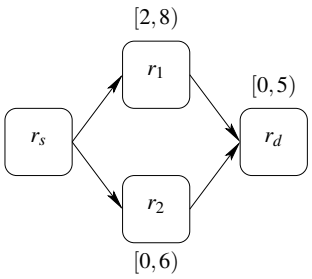


Fig. 2 Transport network. Arcs represent connections between locations. The time intervals are reservations of other agents.

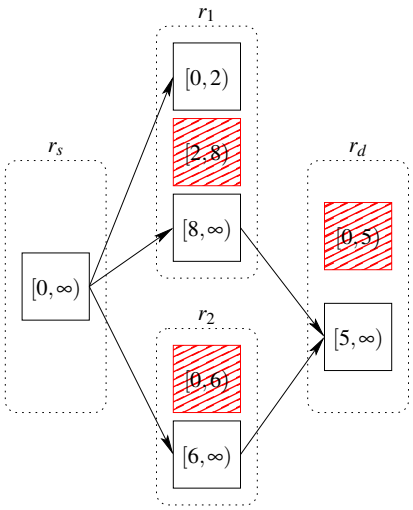
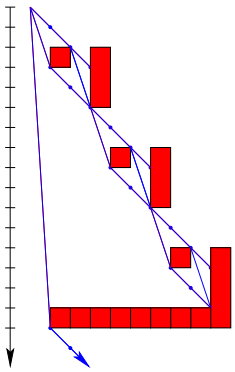


Fig. 3 Free time-window graph. Connections specify the reachability relation between free intervals.

for both uni- and bidirectional networks, and they give an $O(v^4n^2)$ time complexity for their algorithm, where v is the number of vehicles in the system, and n is the number of resources in the infrastructure. Because of this relatively high run-time complexity (especially given the limited computational resources of an early 90s PC), Taghaboni-Dutta and Tanchoco [18] developed an approximation algorithm that decides at every intersection to which resource to go next, based on the estimated traffic density of the resources from the current intersection to the destination. The authors show only a small loss of plan quality, and they claim that the algorithm consumes significantly fewer computational resources. However, the run-time complexity of this approximation algorithm is unknown, and we have not found any quantitative comparisons.

Finally, considering the approach by Hatzack and Nebel [8] in more detail, we see that their approach consists of two phases. In the first phase each agent computes a context-unaware shortest path from its current location to its destination location. In the second phase, the agents schedule their routes sequentially (each agent completes its reservations before the next agent), while preventing conflicts with of the other agents. At the end, all agents have a plan and the joint plan is guaranteed to be free of conflicts.

A drawback of their approach is that their algorithm uses backtracking. Since they do not make use of the idea that a free time window needs to be considered at most once, it is possible to construct examples in which the algorithm keeps backtracking through the same paths of time windows. Figure 4 depicts such an example in which this worst-case behavior is realized. This figure illustrates the reservations on the sequence of resources (horizontally from source resource s via



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

Fig. 4 The figure on the left illustrates how these difficult instances for the algorithm of Hatzack and Nebel are created. The vertical axis represents the progress of time. The horizontal axis represents the resources on the path from the source (on the left) to the destination (on the right). The rectangles represent already reserved time windows. The table on the right gives the time required to find a plan for different problem sizes n, and the number of recursive calls used by the algorithm of Hatzack and Nebel.

$1,2,3,\ldots$ to destination resource d); vertically, the progress of time is shown. Each resource is assumed to have a traversal time of 1. To create such an instance where the algorithm needs 2^n+1 updates, no more than the following $5n$ reservations are needed in a route $s,r_1,r_2,\ldots,r_{3n},d$ of $3n+2$ resources:

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

The table shown in Figure 4 shows that if such a structure of reservations occurs, the algorithm of Hatzack and Nebel will not be able to solve the instance within acceptable time. In the next section we propose an algorithm (Algorithm 1) that has no problem with these instances at all.

3.3 A more efficient context-aware routing algorithm

One of the disadvantages of the context-aware planning approach discussed above is its rather high run-time complexity; especially when the number v of vehicles is growing, this will prohibit an efficient context-aware route finding process. We therefore propose an alternative context-aware single-agent routing algorithm that is more efficient. This algorithm is embedded in the same approach for multiple agents discussed above, but has a time complexity that is much better than all of the

Require: start resource s, destination resource d, start time t.
Ensure: exit time from d for the shortest path from s to d.
1: **if** s is free from t to $t + D'(s)$ **then**
2: $Q \leftarrow \{(s, t + D'(s))\}$
3: **end if**
4: **while** $Q \neq \varnothing$ **do**
5: $(r_i, t_i) \leftarrow \text{pop}(Q)$
6: **if** $r_i = d$ **then**
7: **return** (r_i, t_i)
8: **end if**
9: **for all** neighboring resources r_j **do**
10: **for all** free time windows $[t_j, t'_j)$ of r_j **do**
11: **if** possible to go to this window from (r_i, t_i) **then**
12: $Q \leftarrow \text{insert}(Q, (r_j, \max\{t_i, t_j\} + D'(r_j)))$
13: remove all time windows up to t_j from the list of free time windows of r_j
14: **end if**
15: **end for**
16: **end for**
17: **end while**

Algorithm 1: Context-aware routing.

above-mentioned existing methods (cf. [15]). This context-aware routing method is based on Dijkstra's shortest path method.

Let us consider Dijkstra's algorithm for computing a shortest path from a given source node r_s to all other nodes in order of their distance from r_s. The nodes for which the shortest path to s is already known are maintained in a list R_s. In each iteration one of the neighbors j of the nodes in R_s is added to R_s. The shortest path to each neighbor j using only nodes in R_s is simply the minimum of the distance to a node i in R_s plus the length of the edge from i to j. These neighbors are stored in a priority queue sorted on this distance. For the neighbor with the minimum distance in this queue, it is known that there is no shorter path using nodes outside of R_s, because all these nodes are further away. This node is then added to R_s, and the distances of its neighbors in the priority queue are updated if they can be reached through this node.

Our context-aware routing algorithm differs from Dijkstra's shortest path method in a number of aspects. First, we apply this method on a network in which the costs are in the resources, not in the connections between the resources. Second, we apply this method to search for a shortest path in the free time-window graph, and we make sure that only the relevant part of this time-window graph is represented. Third, we ensure that all constraints on the neighbor (connectivity) relation of two infrastructure resources are explicitly checked. Fourth, we sort the priority queue on the exit time of a certain resource (instead of the shortest distance to a node). Fifth, for an agent a we use the time distance function $D'(r) = \frac{D(r)}{\min(S(a), S(r))}$ instead of just the length of the edges between nodes. This then boils down to Algorithm 1.

Let us briefly go over the main steps in this algorithm. In Line 2, we initialize the priority queue Q of free time windows (with priority on exit time) to the start

resource and the earliest exit time of this resource. In Line 5, we retrieve the time window (r_i, t_i) with the lowest cost exit time. To expand the current free time window, we consider in Line 9 and 10 all (resource, free time window) pairs that are reachable from (r_i, t_i). In Line 11, we check whether it is really possible for the agent to go from r_i with earliest exit time t_i through resource r_j within the interval $[t_j, t'_j)$. This check involves checking whether there is an exit time $t \leq t_i$ from r_i that is later than t_j and for which also $t + D(r_j) \leq t'_j$. Furthermore, it is ensured that this step does not involve a conflict with the reservations of other agents (see Section 2). If all constraints are met, this option is added to the priority-queue. Finally, we remove the free time windows up to t_j from resource r_j's set of free time windows, since we have already found the shortest path to this time window (since the exit time of r_i was the earliest). This is an important step, as it guarantees that we do not consider any free time window for expansion more than once.

To analyze the run-time complexity of this algorithm, we first place a bound on the number of time windows, using n for the number of resources in the infrastructure, and v for the number of vehicles. Assuming that vehicles visit each resource only a constant number of times, there are at most $O(nv)$ free time windows, since for each free time window there must be a reservation as well. Based on this observation, we can now bound the run-time complexity of this algorithm by $O(mv + nv\log(nv))$ where m is the number of connections in the infrastructure. This can be seen as follows. Each free time window is considered at most once in an iteration of the while loop (Line 4). Because there are $O(nv)$ free time windows in total, the while loop is executed at most nv times. Since Q is a priority queue, removing the smallest element from the list takes $O(\log nv)$ time. Lines 1–7 therefore contribute $O(nv\log(nv))$ to the complexity of the algorithm. Rather than looking at lines 9–13 in the context of the while loop, we observe that over the whole run of the algorithm, these lines will be executed at most once for each time window of each neighbor in the time window graph, i.e., $O(mv)$. Similarly, regarding line 12, we can see that this line will be executed at most once per time window, i.e., $O(nv)$. Since this inserting in a priority queue takes $O(\log(nv))$, the total run-time for this part of the algorithm is $O(mv + nv\log(nv))$. Hence, Algorithm 1 has a run-time complexity of $O(mv + nv\log(nv))$.

Thus, compared to earlier work, our approach is more efficient. For example, the run-time complexity of the (single-agent) free time-window graph routing algorithm of Kim and Tanchoco [10] is $O(v^4n^2)$, while our algorithm has a run-time of $O(mv + nv\log(nv))$. The reason for this difference is that their conflict detection procedure is inefficient. Kim and Tanchoco [10] did not make use of the fact that it is enough to consider only the direct successor and predecessor to check for catching-up conflicts (instead, they iterated through all present vehicles). Furthermore, they used a different framework in which they both had to check for conflicts in the locations, as well as on lanes. In our framework lanes are also modeled as resources with the same properties as locations.

4 Dealing with uncertainty

Thus far we have considered the transportation planning problem as a static problem. This is, of course, in reality not the case. Several sources of uncertainty can be distinguished [4]. Uncertainty can be caused by incidents, such as communication failures between automated guided vehicles and the system maintaining reservations, break-down of a mobile entity (engine failure), or failures in the transport network (e.g. due to traffic accidents). Uncertainty can also be caused by a modification of the transportation requests. For example, the arrival of a new transportation request renders a current plan infeasible.

Uncertainty, and especially incidents can be dealt with pro-actively or reactively. Pro-active methods attempt to create robust schedules, while reactive methods recover from incidents at the moment they occur. A typical pro-active approach is to insert slack in plans such that for example delays have (almost) no consequences, and new requests can easily be inserted, see, e.g. [5]. However, in case nothing unexpected happens, these plans take much longer than strictly required. Therefore, in the following, we focus on approaches that deal with uncertainty reactively.

In principle, uncertainty does not offer any additional problem to the traditional approach, since the operational conflict resolution methods are usually sufficient to handle conflicts due to new arrivals or incidents as well. For context-aware route planning systems, however, such unexpected changes are a serious threat. Of course we could add an operational conflict resolution system to context-aware routing to remove any conflicts during execution. However, then it easily might turn out that such systems are not better than the traditional route planning approaches regarding efficiency and predictability. Therefore, we propose to extend context-aware routing with the ability to deal with such conflicts by reconsidering the plans of the involved vehicles (and possibly some of the others) each time an unexpected event occurs.

We consider the following two possible improvements. In the *revising-priorities* method a number of selected vehicles reconsider only the timing of their reservations in order of their priority, which is determined by a heuristic function. The *revising-routes* method also examines alternative routes for the vehicles. After all, it might be the case that a vehicle is better off taking a detour if the heuristic function determines that this vehicle should wait for other vehicles.

4.1 Revising priorities

In context-aware routing, reservations are (in principle) permanent, and later requests take existing reservations into account. Consequently, the performance of the system depends on the order in which the vehicles request reservations. The idea of the REVISING-PRIORITIES method is to improve the performance by re-evaluating reservations at certain moments, for instance, when a vehicle has accepted a new transportation request, a transportation request has been modified, or when a vehicle is bothered by an incident on its path. This method usually leads to a revised

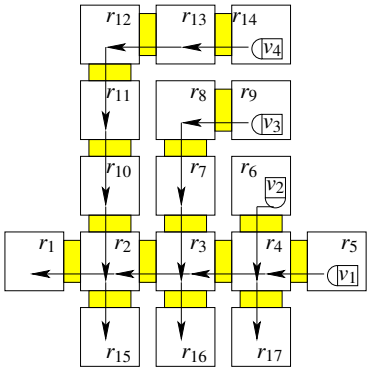


Fig. 5 Initially the plan of v_1 is $(r_5,[0,2),r_4,[2,4),r_3,[4,6),r_2,[6,7),r_1,[7,\infty))$. It thus has to wait at every resource on its route. With REVISING-PRIORITIES vehicle v_2 can give way to v_1. The plan of v_1 then is much more efficient, while v_2 has to wait only one additional time step.

Table 1 List of heuristics used by the REVISING-PRIORITIES and REVISING-ROUTES methods.

heuristic	description
random	for baseline comparison, chooses a random agent to go first
delays	agent with highest sum of expected delays goes first
deadlines	agent with least amount of slack before the deadline goes first
profits	agent with lowest expected profits goes first
wait	agent that waits longest to enter its current location goes first
task	agent that is assigned the task that has the highest reward goes first
inv task	the reversed ordering of the task heuristic, used to see the effect of bad versus good heuristics

ordering of the vehicles' reservations of certain infrastructural resources, hence the name. Below we describe a centralized version of this method where one scheduler controls the order in which the vehicles can make new reservations. Typically, the vehicle requesting rescheduling determines a group of involved vehicles, such as all vehicles that share at least one infrastructure resource with the requesting vehicle. This set of vehicles can quickly be determined by considering reservations for infrastructure resources in the plan of the requesting vehicle.

When the request to revise priorities is granted by the scheduler, all of the participating vehicles dispose of their current reservations, but maintain their route. In turn, these vehicles can request new reservations for the first part (block) of their remaining route. A heuristic function is used to determine the order in which this should be done. Examples of such heuristic functions are given in Table 1. This algorithm terminates when each vehicle in the group has a new schedule for each complete route.

To see that this method cannot only be used in case of incidents, but also has some potential to improve plans under regular operating conditions, consider the following example in Figure 5. If all resources traversals take one time step, vehicle v_1 has to wait an additional time step at each resource on its route for another vehicle.

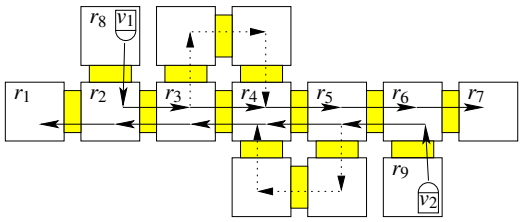


Fig. 6 When the route of v_1 is r_2, r_3, r_4, r_5, r_6 and the route of v_2 exactly the opposite, one of these vehicles will have to wait until the other one has completed its route. With REVISING-PRIORITIES one of them can take a small detour instead, resulting in a more efficient global plan.

If v_1 is given priority at only one of these resources, there is no need for v_1 to wait at later resources anymore, while it will cost another vehicle only one more time step.

4.2 Revising routes

Besides reconsidering the priorities of vehicles when making reservations, the REVISING-ROUTES method also allows vehicles to reserve a completely new route. In principle, this method is very similar to REVISING-PRIORITIES: first a selection of vehicles dispose of their reservations, and then each vehicle in turn requests new reservations for a part of their route. The difference is that in REVISING-ROUTES, a vehicle now also disposes of its route, and plans a new route when it is its turn. When planning a new route, vehicles do not consider a re-ordering of the visiting sequence, because finding an optimal visiting sequence is a very hard problem in itself, and takes too much time to compute.[4] However, reconsidering the routes between each pair of locations in a visiting sequence of a group of vehicles still gives much more flexibility than only reconsidering the reservations.

For example, when two vehicles plan to use the same sequence of resources, but in the opposite direction, a context-aware routing method lets one of the vehicles wait before entering this sequence (see Figure 6). REVISING-PRIORITIES can improve the average efficiency by a heuristic that determines which vehicle should wait. The REVISING-ROUTES method can further improve the resulting plans by allowing vehicles to take a detour. Especially when a certain resource is blocked for some time due to an incident, being able to compute a new route is a great advantage. We therefore expect this approach to result in better plans than REVISING-PRIORITIES, but the computation costs (in CPU time) are also higher, since the context-aware routing algorithm is now called as a subroutine for each pair of locations of each of the agents' vehicle sequences. In the next section an experimental evaluation is presented to indicate under which conditions which method is the best alternative.

[4] This problem is similar to the traveling salesman problem, which is NP-complete.

5 Experimental Evaluation

To evaluate the proposed context-aware routing algorithms, we perform simulations in two settings. First, we compare the context-aware routing (INFORMED) and its two improvements (REVISING-PRIORITIES and REVISING-ROUTES) to the baseline approach in which no intelligent infrastructure with reservations is used (UNINFORMED). This evaluation is done in a synthetic grid infrastructure. In this setting we also evaluate the effect of incidents on the context-aware methods that we propose. Second, we study the effect of using our improved context-aware routing method in a realistic situation, i.e., taxiing at Amsterdam airport. Here we approximate current practice with the reservation-based planning method of Hatzack and Nebel [8] and compare the resulting plans to the ones produced by our proposed approach.

5.1 Comparing context-aware methods on a grid network

For these experiments an 8×8-grid network is used with 32 vehicles and 192 requests. The problem instances are randomly generated in such a way that we know that an optimal solution exists, i.e., a solution for which all requests are served within the given deadlines.

In this section we consider the effect of the planning method and several incident settings (these are the independent variables). The *dependent variables* (or performance indicators) are the relative system reward and the CPU time required to finish the particular simulation. The reader is referred to [20] for a study on the influence of the request load, the size and the topology of the transport network, and the number of vehicles. The planning methods are divided into four categories: *(i)* baseline approach (also called UNINFORMED), *(ii)* context-ware routing (also called INFORMED), *(iii)* revising priorities, and *(iv)* revising routes.

For each request j, the agents receive a reward defined by the reward function π_j (see Section 2). The relative system reward is the ratio of the realized reward, divided by the maximum reward for all transportation requests. A relative reward of 1 means that the maximum possible reward is obtained for all of the transportation requests. This maximum reward cannot always be achieved, especially not if there are many requests or incidents.

All results presented have been obtained by making use of the transport planning simulator TRAPLAS, see [20]. The simulations are run on one processor, and a central controller was used. However, in this simulator, each vehicle and each resource has its own separate light-weight thread, which should make it relatively easy to re-use the source code for a distributed implementation on a real infrastructure. A free software environment for statistical computing and graphics called R [17] has been used to combine the output of TRAPLAS, to plot the graphs, and for all further data analysis. All experiments were done on the Distributed ASCI Supercomputer (DAS-3 [2]).

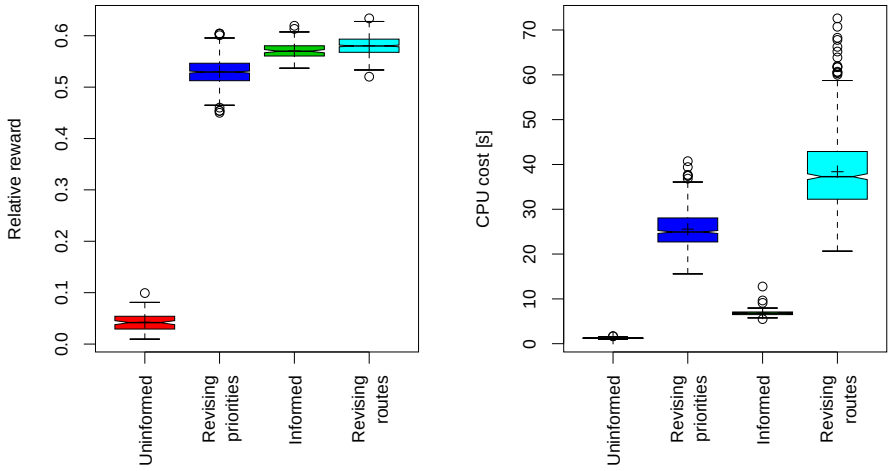


Fig. 7 The performance of the four methods. With one exception, all methods execute all transportation requests (but not all on time). The UNINFORMED method, however, executes only 48.5% on average. The '+' symbol indicates the mean (often coinciding with the median in the figure) and the '○' symbol is used for outliers.

5.1.1 Comparing the planning methods

Figure 7 shows a box-and-whisker plot [19] for the four different planning-method categories. The UNINFORMED and INFORMED plots are based on 100 samples (10 simulations and 10 different sets of 192 transportation requests); the REVISING-PRIORITIES and REVISING-ROUTES categories consist of 7 times more samples to test each of the 7 different heuristics listed in Table 1.

An analysis-of-variance study shows that the differences in the mean values between the methods are statistically significant. The first thing to note is that the UNINFORMED method has a much lower relative reward. This can be explained by the fact that this is the only method that was not able to execute all transportation requests. This is due to deadlocks that occur without any coordination between the agents. On average, the UNINFORMED method executed 48.5% of the transportation requests.

The INFORMED context-aware planning method makes use of an intelligent infrastructure that manages reservations. This method produces much better results. All transportation requests are executed successfully. The plot on the right-hand side in Figure 7 shows the sole strength of the UNINFORMED method; it is the cheapest in terms of CPU cost. In general, it can be seen that the CPU cost required by the planning methods increases if more information is considered. Although this might be an advantage for very large transportation instances, the INFORMED method is usually fast enough for practical instances.

A further observation is that the performance of the INFORMED method is in between the REVISING-PRIORITIES and REVISING-ROUTES methods. This is a bit sur-

prising, as we introduced REVISING-PRIORITIES as an improvement over the standard INFORMED context-aware planning method. However, this can be explained by the results for the seven heuristics (see Figure 8).

Most of the heuristics that we study perform no better than the random heuristic. The results for these heuristics significantly influence the average relative reward for REVISING-PRIORITIES in Figure 7. In addition, we can see that both the REVISING-PRIORITIES and the REVISING-ROUTES method have the best performance when using the *wait* heuristic (in this grid network with a relatively small number of requests and no incidents).

The good performance of the *wait* heuristic can be understood by appreciating the intuitive resemblance with the first-come-first-served heuristic [11], which works so well in scheduling. The *wait* heuristic aims to minimize the waiting time of transport resources, by giving priority to the longest waiting vehicle. Hence, the *wait* heuristic attempts to increase the throughput of transportation requests, which increases the performance of the agents.

5.1.2 Studying the influence of incidents

As discussed in Section 4, there are many types of uncertainty in operational transport planning. The focus regarding uncertainty in this chapter is on uncertainty caused by incidents. We model such incidents by a time period (the repair time) during which a certain resource or vehicle endures a slow-down between 0 (no slow-down) and 1 (full stop). This model can capture all kinds of real-life situations ranging from regular traffic jams (delays of infrastructural resources for certain periods of the day) to unexpected vehicle break-downs (full stop for some time until repaired). To evaluate the effect of incidents on the proposed context-aware methods, we generate situations ranging from a few incidents with a low impact to many incidents with long repair times and a high impact. An incident is generated in analogy to the concept of mean-time-between-failure, which is in our case based on an exponential distribution with a failure probability between 0 and 0.2, and the repair time is drawn from a normal distribution with mean 400 and standard deviation 50. We have chosen to create a relatively small number of incidents, but with quite a significant repair time, because this brings out the differences between planning methods more explicitly [14].

The impact of all incidents in a generated problem instance is expressed by the *incident level*. This incident level is the sum over all incidents of the impact of that incident times the duration of the incident, and varies in our case between 0 and 323981 seconds (about half a week). For six scenarios within this range we compute the relative reward of the INFORMED context-aware routing method, REVISING-PRIORITIES, and REVISING-ROUTES. For the latter two we use the *wait* heuristic, since this heuristic performs best according to our earlier experiments.

Figure 10 shows that the two methods that have been superior so far, i.e., REVISING-PRIORITIES and REVISING-ROUTES, now clearly outperform the standard context-aware method when the incident level is high, with a relative reward

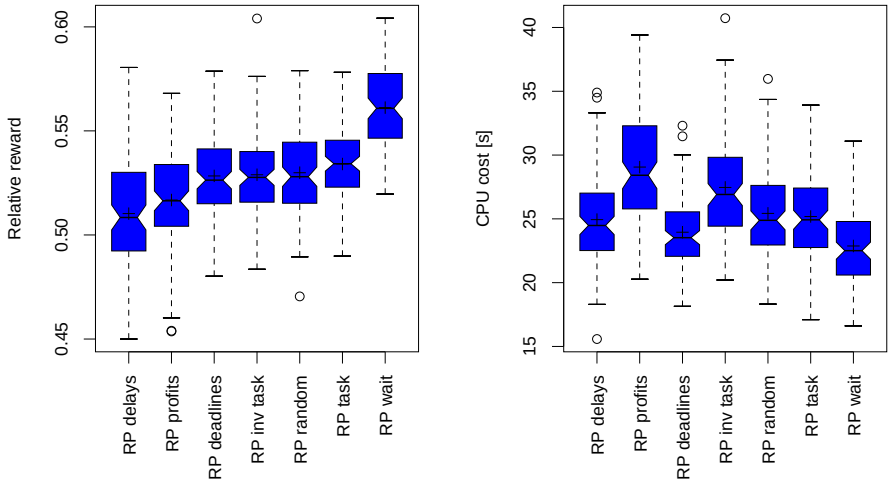


Fig. 8 The performance of each of the heuristics in combination with the REVISING-PRIORITIES method. The '+' symbol indicates the mean and the '○' symbol is used for outliers.

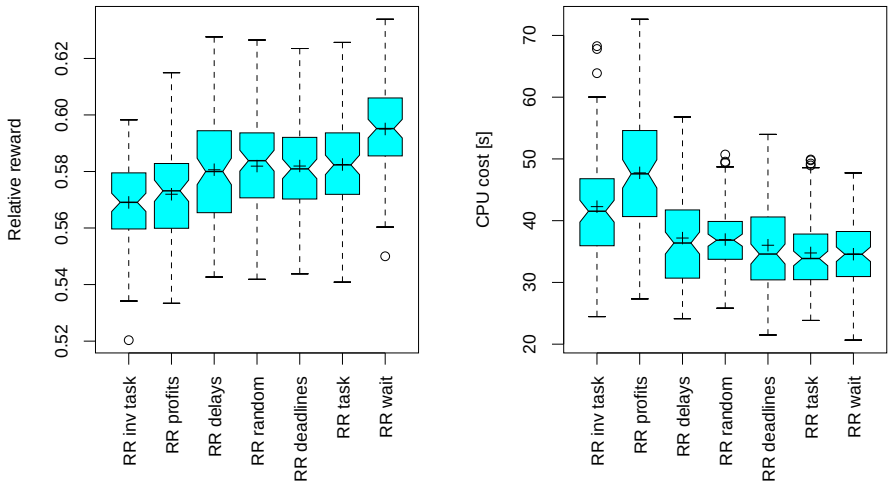


Fig. 9 The performance of each of the heuristics in combination with the REVISING-ROUTES methods. The '+' symbol indicates the mean and the '○' symbol is used for outliers.

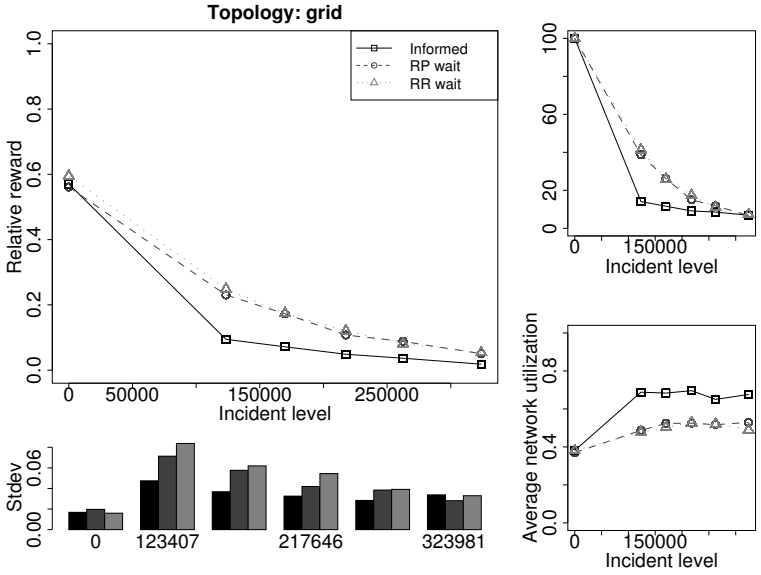


Fig. 10 The relative reward for three planning methods on a 8×8 grid network decreases when the incident level increases (with a fixed number of 192 requests).

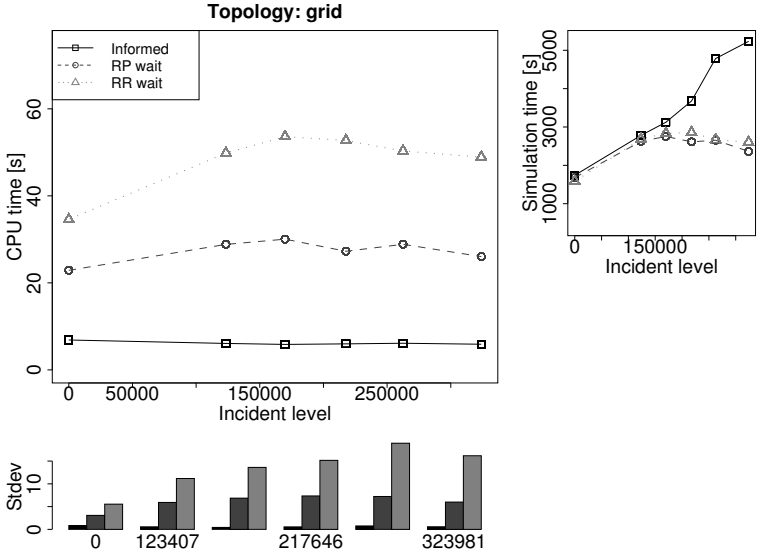


Fig. 11 The CPU costs (in time) required by the selection of planning methods on grid networks; the incident level is increasing and the request load is fixed to 192 requests.

that is almost twice as high. The performance of all methods degrades when the incident level is increased. The differences of the means of the performance of the INFORMED and the other two methods are statistically significant for all six scenarios.

Next, let us consider the CPU time and the simulation time of these runs (see Figure 11). As expected, we can again clearly observe that REVISING-ROUTES requires significantly more computation time than REVISING-PRIORITIES, and both are significantly outperformed in this respect by INFORMED. However, the execution of the plans takes much longer for INFORMED in cases with a high incident level, since some vehicles may encounter multiple incidents on their planned route, without being able to plan around them (either in time or in space).

5.2 Taxiing at Amsterdam airport

Besides these experiments based on a synthetically generated test set, we evaluate our methods also on a real-life transport network. For this, we conduct experiments on the Amsterdam airport network (Schiphol) in the Netherlands consisting of 1016 infrastructure resources (see Figure 12 for a map of the center part of this network).[5] In this transportation network, the taxiing problem of aircrafts (on the ground) plays an a crucial role in the performance of the whole airport [15]. The usual sequence of an airplane after touch-down is to first taxi to a gate, then wait for services, such as cleaning, boarding, safety checks, and possibly a visit to a de-icing station. Currently approximately 300 airplanes per day go through this process (a number that is increasing), and thus efficient and robust routing methods are required.

The goal of the airplane taxiing experiments is to compare the current practice in a realistic setting to our context-aware routing method. Current practice is comparable to the approach of Hatzack and Nebel [8], in which first routes are planned, and then the use of resources is scheduled in order to prevent conflicts.

To make a fair comparison between these two approaches, the current practice as well as two of our context-aware routing methods are used to compute a route for the same start-destination pair, given the *same set of prior reservations* on the infrastructure. For each of such a set of reservations, we measure the average time and the average quality of finding a conflict-free path for 20 randomly chosen start-destination pairs, one of which is actually reserved for the next round. We repeat this 3000 times, reserving 3000 routes in the end. First, we study a setting where the sets of prior reservations are constructed using the algorithm of Hatzack and Nebel [8]. Then, we do the same measurements for sets of reservations based on the plans obtained by our context-aware approach.

Running our experiments, we have discovered that the context-aware routing algorithm (Algorithm 1) considered visiting the same part of the infrastructure twice, for example to step aside to let another airplane pass. This is illustrated in Figure 12.

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

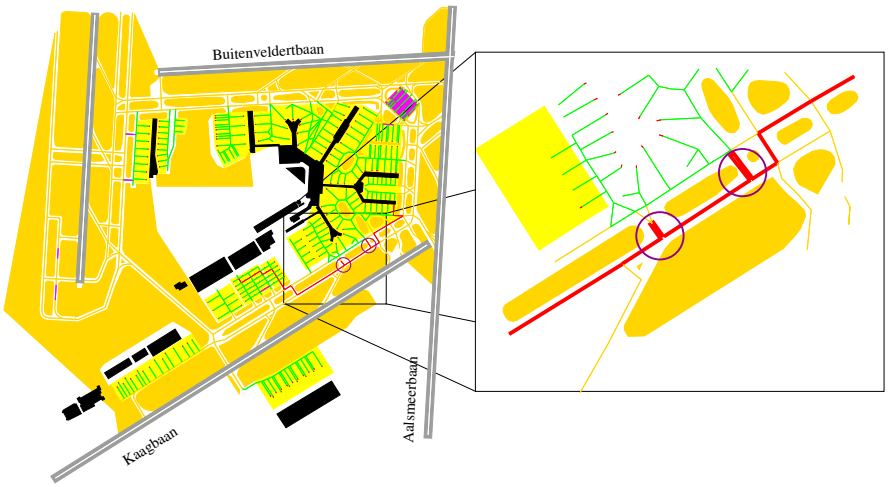


Fig. 12 If cycles are allowed and there are many reservations of other agents present in the network, the context-aware algorithm often produces plans with a cycle. An example of a produced shortest path is highlighted by the thick line, and the side-steps are indicated by the two circles. Such plans are not considered by the acyclic version of the routing algorithm.

Since this is considered undesirable, we also introduce a simple modification of the context-aware routing algorithm in which such cycles are not allowed. For this *acyclic* variant of the algorithm, in Line 11 of Algorithm 1, the time windows to be considered do not only need to be possible to go to, but they also may not already occur in the plan of the agent. This thus requires some additional administration as well.

In the following experiments, we thus compare not only the context-aware routing algorithm (INFORMED) to the plans produced by the two-stage method of Hatzack and Nebel [8], but also the context-aware routing method that excludes cycles in the plans ("No cycles").

From Figure 13, it can be concluded that the context-unaware approach of the algorithm of Hatzack and Nebel is so fast, the context-aware algorithms look slow by comparison. A closer look reveals that the context-aware algorithms are still quite fast, as a solution (for one additional request) is found on average within two tenths of a second. Also, the 95% confidence intervals are reasonably small, so this performance is reasonably stable. With regard to the different variants of the context-aware routing methods, it can be seen that the no-cycles variant is significantly faster than the other, despite the fact that checking for absence of cycles is not a very cheap

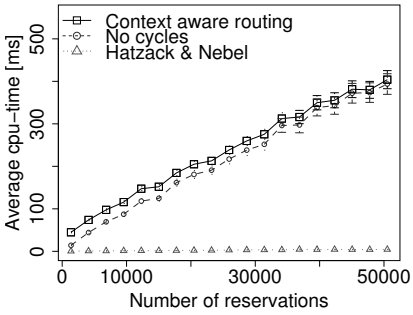


Fig. 13 Planning for one additional request takes more CPU time when there are more resource reservations in case these reservations have been made using Hatzack and Nebel [8].

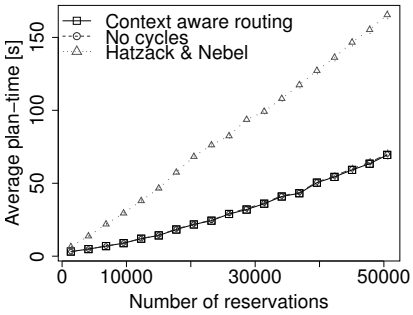


Fig. 14 The average end-times of plans are later when there are more resource reservations, but there is a significant difference between context-aware routing and Hatzack and Nebel [8].

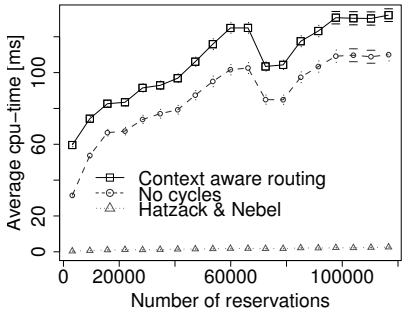


Fig. 15 Planning for one additional request takes also more CPU time when there are more resource reservations in case these reservations have been made using context-aware routing.

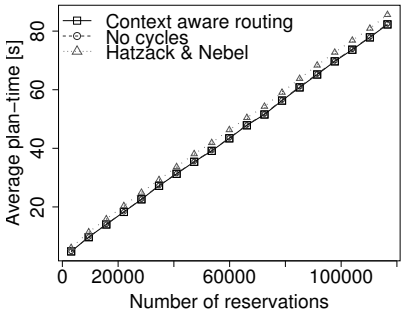


Fig. 16 The average end-times of plans are later when there are more resource reservations (confidence intervals too small to display).

operation. This is due to the fact that the no-cycles variant can ignore all routes where a resource is visited twice.

In Figure 15, a clear notch is visible, just before 80,000 reservations, which shows the CPU cost (in time) is not growing monotonically for an increasing number of reservations in the transport network. Intuitively, the more reservations there are, the more difficult it is to search a path through the transport network. However, after adding certain reservations, the search might become easier all of a sudden, because a "difficult" part of the network does not have to be searched through anymore (due to a reservation now prohibiting this). The exact position of such a notch depends on the transport network and the order in which agents create the reservations for traveling to their desired target locations.

Looking at the end-times of the generated plans, the plans generated by the no-cycles variant are of equal quality as those generated by Algorithm 1. When the set of reservations is produced by Hatzack and Nebel [8], the context-aware methods both succeed in finding a much more efficient plan by routing around bottlenecks

(Figure 14). When the set of reservations of all other agents is produced by the context-aware routing method, however, the plan made by the algorithm of Hatzack and Nebel [8] for one additional agent is only slightly longer (in time), i.e., about 3 to 4% (Figure 16). In this case, there are apparently much shorter queues for bottlenecks than in the case when the route for all agents is fixed on forehand (as done by Hatzack and Nebel).

6 Conclusions and future research

In this chapter we have presented an alternative to the traditional approach to operational transport planning where conflicts are prevented during execution. The proposed method relies on an infrastructure that maintains reservations of its use at specific times, and can thereby resolve conflicts during planning. This ensures predictable travel times and thereby significantly reduces the uncertainty in transportation. Based on this intelligent infrastructure, we proposed three new context-aware route planning algorithms that either have a better run-time complexity than earlier work, or are much more sophisticated in finding alternatives in case of incidents or bottlenecks. These algorithms have been evaluated in a synthetic grid network (with and without incidents), and in an airplane taxiing simulation of Amsterdam airport. We have shown that methods using the reservation information of infrastructure resources outperform local (traffic) coordination rules, and that performance increases even more when vehicles' reservations or even parts of their routes are rescheduled in case of incidents. This latter method can deal with incidents and is still fast enough (in the airport-taxiing experiments finding an additional route can be done with less than a second), and gives the best results in terms of plan quality.

Apart from studying the behavior of this method experimentally in other realistic infrastructures and under other incident conditions, there are quite a number of other interesting directions for future work. For example, in the context of multiple requests and the possibility of incidents, a method in which all agents optimize the length of their own plan (by reserving required resources as early as possible) may not always lead to the best result. Possibly introducing some slack in the plan may lead to better overall results. This idea can definitely be exploited when agents plan to arrive too early at their destination. The proposed method does not take this into account and just lets the agent wait at the last resource until it is the right time to deliver. Secondly, the context-aware method introduced in this chapter allows for a distributed implementation to potentially allow for better scalability and reliability. It would be interesting to evaluate whether these promises can be fulfilled by performing some additional experiments. Another direction is to optimize not just the length of an agent's plan, but to allow for multiple objectives. For this, the work presented could be integrated with a multi-objective routing package called Samcra [12, 16]. Finally, we believe there is still some room for improving the solution quality of the resulting operational transport plans, since this is an NP-hard problem and our algorithms are polynomial. Repeatedly (re)running the context-aware

routing algorithm for a set of agents can thus not always give optimal results, even though our context-aware routing algorithm for a single agent is both optimal and very efficient.

References

[1] Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
[2] Bal, H., Bhoedjang, R., Hofman, R., Jacobs, C., Kielmann, T., Maassen, J., van Nieuwpoort, R., Romein, J., Renambot, L., Rühl, T., et al.: The distributed ASCI supercomputer project. ACM SIGOPS Operating Systems Review **34**(4), 76–96 (2000)
[3] Broadbent, A.J., Besant, C.B., Premi, S.K., Walker, S.P.: Free ranging AGV systems: Promises, problems and pathways. In: Hollie, R.H. (ed.) Proceedings of the 2nd International Conference on Automated Guided Vehicle System, pp. 221–237 (1985)
[4] Davenport, A.J., Beck, J.C.: A survey of techniques for scheduling with uncertainty (2000). Unpublished manuscript
[5] Davenport, A.J., Gefflot, C., Beck, J.C.: Slack-based techniques for robust schedules. In: Proceedings of the Sixth European Conference on Planning (ECP-2001), pp. 7–18 (2001)
[6] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
[7] Fujii, S., Sandoh, H., Hozaki, R.: A routing control method of automated guided vehicles by the shortest path with time-windows. In: Production Research: Approaching the 21st Century, pp. 489–495 (1989)
[8] Hatzack, W., Nebel, B.: Solving the operational traffic control problem. In: Cesta, A. (ed.) Proceedings of the 6th European Conference on Planning (ECP'01) (2001)
[9] Huang, J., Palekar, U., Kapoor, S.: A labelling algorithm for the navigation of automated guides vehicles. Journal of Engineering for Industry **115**(3), 315–321 (1993)
[10] Kim, C.W., Tanchoco, J.M.A.: Conflict-free shortest-time bidirectional AGV routeing. International Journal on Production Research **29**(12), 2377–2391 (1991)
[11] Kruse, R.L.: Data Structures & Program Design. Prentice-Hall, Inc., Upper Saddle River, New Jersey (1984)
[12] Kuipers, F., Mieghem, P.V.: Conditions that impact the complexity of qos routing. IEEE/ACM Transaction on Networking **13**(4), 717–730 (2005)
[13] Lenstra, J., Kan, A.: Complexity of vehicle routing and scheduling problems. Networks **11**(2), 221–227 (1981)

[14] Maza, S., Castagna, P.: A performance-based structural policy for conflict-free routing of bi-directional automated guided vehicles. Computers in Industry **56**, 719–733 (2005)
[15] Mors, A.W.T., Zutt, J., Witteveen, C.: Context-aware logistic routing and scheduling. In: Boddy, M., Fox, M., Thiebaux, S. (eds.) Proceedings of the 17th International Conference on Automated Planning and Scheduling, pp. 328–335. AAAI Press (2007)
[16] P. Van Mieghem, H.D.N., Kuipers, F.: Hop-by-hop quality of service routing. Computer Networks **37**(3-4), 407–423 (2001)
[17] R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2007). URL http://www.R-project.org. ISBN 3-900051-07-0
[18] Taghaboni-Dutta, F., Tanchoco, J.: Comparison of dynamic routing techniques for automated guided vehicle system. International Journal of Production Research **33**(10), 2653–2669 (1995)
[19] Tukey, J.W.: Exploratory Data Analysis. Addison Wesley (1977)
[20] Zutt, J.: Operational transport planning in a multi-agent setting. Ph.D. thesis, Delft University of Technology (2009, forthcoming)