Multi-Agent Transport Planning

J. Zutt [a] C. Witteveen [a b]

[a] Faculty EEMCS, Delft University of Technology,
P.O. Box 5, 2600 AA Delft, {J.Zutt, C.Witteveen}@ewi.tudelft.nl.
[b] Center for Mathematics and Computer Science, P.O. Box 94079,
NL-1090 GB Amsterdam, C.Witteveen@cwi.nl.

Abstract

We discuss a distributed transport planning problem with competitive autonomous actors that carry out time-constrained pick-up delivery orders from customers. The agents have to find conflict-free routes to execute a series of orders they have accepted. Hatzack and Nebel [2] were the first to suggest that finding such a conflict-free schedule can be reduced to solving a job shop scheduling problem. In this paper, we extend their approach in several ways. First of all, we use an iterative traffic-aware planning procedure to determine feasible routes and schedules, called HNZ-1, using a combination of route and schedule finding heuristics. That is, instead of using a static routing approach, we allow agents to *replan* their route if a preplanned route becomes unfeasible. Moreover, we adapt this (re)route and scheduling heuristic, to take into account *incidents* that might occur during the execution of orders. Two parameters used in this this HNZ-1 heuristic turn out to be important for the performance: the agent selection function and the resource block size. Results indicate that giving priority to agents that have more strict deadlines as well as scheduling with smaller resource block sizes improve the performance of the agents.

1 Introduction

We focus on the *operational* planning aspects of complex multi-agent (transportation) problems. Such problems are characterized by a set of autonomous transportation agents capable of carrying out orders for customers who want their cargo to be transported from a source location to a destination location within certain pickup and delivery windows. An agent carrying out an order in time will receive a positive reward and a negative reward (penalty) in case an order is not carried out within the time windows specified. The goal of each individual agent is to maximize its reward. This requires careful planning of routes to pickup and deliver the orders accepted. However, since the agents use a common restricted infrastructure where locations and connections have to be considered as resources

This research has been supported by the TNO-TRAIL project (16) *Fault detection and recovery in multimodal transportation networks with autonomous mobile actors* and by the Dutch Ministry of Economic affairs under the SENTER TSIT program *Cybernetic Incident Management* (TSIT2021).

having limited capacity, an agent is not completely free to plan its route. Due to the plans of other agents, it is quite likely that the set of resulting route plans is not conflict-free and the capacity of locations or connections is exceeded. These conflicts, however, can only detected if the plan is *scheduled*, i.e., it is exactly known which agent is where at what point of time. More precisely then the goal of each agent is to find a *conflict-free operational plan* that maximizes its reward. As already has been pointed out by Hatzack and Nebel (see [2]), however, finding optimal conflict-free routes for all the agents is an intractable problem. But they also suggested an easy way to find *approximate* solutions to the problem by *(i)* first letting each agent plan a shortest route to carry out its set of orders, without taking into account the routes of the other agents and then *(ii)* finding a suitable conflict-free schedule for the static routes thus determined. Such a schedule ensures conflict-free routes by ensuring that no infrastructure resource r (i.e. a location or road) is occupied by more than k agents at the same time if k is the capacity of r. As Hatzack and Nebel showed, conflict-free routes could be easily and efficiently obtained by applying a simple greedy selection heuristic that finds feasible schedules by selecting agents having minimal release dates first.[1] From their experiments with the method developed they concluded that the performance of this method is comparable to the performance of a human controller solving the same problem instances. Moreover, the simulation method turned out to be fast enough to use it for simulation purposes even for quite large problem instances.

In this paper, we improve their method in several respects. First of all, we do not base scheduling solutions on static routes of agents, but we apply an *iterative* algorithm allowing agents to replan their route if the scheduling procedure produces a schedule that does not suit them. This implies that the agents are allowed to apply some form of *replanning* if they are confronted with an unsuitable operational plan. The results of this iterative replanning algorithm are compared with the static routing method of Hatzack and Nebel. Next, we introduce *incidents* that can happen during execution of the plan of an agent and apply the iterative algorithm also in an on-line way to allow the agent to find an immediate repair plan for the remaining part of the plan of the agent. These methods are tested using a simulation tool developed and we discuss some experimental results.

The paper is organized as follows: in Section 2 we describe the elements of the multi-agent planning problem. Then, in Section 3, we focus on the operational planning algorithm. After that, we describe the set-up of the experiments in Section 4 together with their outcome. Finally, in Section 5, we present some conclusions and discuss possible future work.

2 Multi-agent planning problem

Building blocks A *transportation order* $o_j \in O$ is the request to pick-up freight at a certain source location s_j within a specified time-window $[t^s_{j,1}, t^s_{j,2}]$ and to deliver it at a specified destination location d_j within a time-window $[t^d_{j,1}, t^d_{j,2}]$.

[1] That is agents having orders with the earliest pickup time are scheduled first.

Time is assumed to be discrete, time points $t \in T$ are values in $T = \mathbb{Z}^+$. Associated with each order o_j there is a reward function $v_j : T \to \mathbb{Z}$ that takes positive values if the order is executed within its time-windows and negative values if executed outside it time-windows.

The *infrastructure* $I = (R, E, K, C)$ consists of a set R of *infrastructure resources* (e.g. roads, road segments, crossroads, parking space, etc.), a *connectivity* relation $E \subset R \times R$, a *capacity* function $K : R \to \mathbb{N}$ where $K(r)$ is the number of agents allowed to use resource r simultaneously and a *cost* function $C : R \to \mathbb{N}$, where $C(r)$ is the cost of using resource $r \in R$.[2] The orders $o_j \in O$ are executed by a set of *transport agents* using the infrastructure I. Each transport agent A_i owns a *transport resource*, representing e.g., a taxi-cab, truck, an autonomous guided vehicle (AGV) or other mobile entity. These transport resources can move around through the infrastructure I. Each transport resource r_j assigned to agent A_j has a maximum speed $maxspeed(r_j) \in \mathbb{N}$ and a loading capacity $cap(r_j) \in \mathbb{N}$.

Transport agents always have to claim the infrastructure resource r their transport resource is located at during time-window T. If r_1 and r_2 are infrastructure resources, $(r_1, r_2) \in E$ denotes that a transport resource claiming resource r_1 can claim resource r_2 next, while releasing the claim on resource r_1.

Planning The goal for each transport agent is to execute as many transportation orders as possible. At any point of time, the status of an order $o_j \in O$ is either *assigned*, i.e. an agent has been committed to carrying out o_j, or *free*. Each agent continuously checks the current set of free orders to find out which one suits best to add it to its existing set of orders, thereby extending the agent's current route with possible adaptations of its schedule. From that moment on, the order is assigned to the agent and the agent is committed to carry it out.[3]

Each agent plans a route to execute the orders it is committed to. Such a route $Rt_A = (r_1, r_2, \ldots, r_n)$ for agent A over the infrastructure I consists of a sequence of n resources such that for $i = 1, \ldots n-1, (r_i, r_{i+1}) \in E$. A schedule associated with Rt_A is a sequence $Sd_A = (s_1, s_2, \ldots, s_n)$ of time points s_i specifying the time agent A claims resource r_i for the first time. Note that this schedule implies that A uses r_j during the time-window $[s_j, s_{j+1})$, where s_{n+1} is assumed to be infinite.

Let resource r be claimed during a set of time-windows $\{[l_i, u_i)\}_{i=1}^{m}$. Since each resource $r \in R$ has a finite capacity $K(r)$, there is a conflict for r if there is a time point t such that t occurs in more than $K(r)$ time-windows $[l_i, u_i)$.

Incidents The pair (Rt_A, Sd_A) constitutes the agent A's initial executable plan. However, certain events (incidents) may occur that cannot be anticipated in advance, have a negative influence on plan execution and even might necessitate replanning. In this paper, we address two different types of incidents, namely, *road congestion* incidents and the *transport agent damage* incidents. Road conges-

[2] The cost function C specifies the minimal costs to claim a resource. For infrastructure resources this might be the minimal time needed to traverse a road (segment), for transport resources the costs for using them for a certain period.

[3] If more than one agent is investigating the suitability of an order o, the agent placing its commitment first, wins.

tion denotes an unexpected increase in traversal costs for a certain infrastructure resource within a certain time interval. Transport agent damage incidents represent malfunctioning transportation resources, resulting in a lower driving speed (possibly 0) during the specified time interval. Incidents are represented as four-tuples $(type, r, T, f)$ where *type* is the type of incident, resource r denotes the malfunctioning infrastructure or transport resource, time-window T specifies the interval in which the incident is effective and the fraction $0 \leq f \leq 1$ indicates the severeness of the incident. During time-window T, this fraction f increases the traversal time of an infrastructure resource (road congestion) or is multiplied with the normal speed of a transport resource (transport agent damage).

3 Operational planning algorithm

In this section we briefly discuss our HNZ-1 algorithm. It is based on the suggestion of Hatzack and Nebel [2] to transform computing a schedule for a fleet of vehicles to a Job Shop Scheduling problem with blocking. The HNZ-1 algorithm consists of a *routing* algorithm, a *scheduling heuristic* and a *rerouting function*.

Routing and scheduling In [2], only static routing and no particular method is discussed to obtain a route for a set of orders. In HNZ-1, initially, we run a traffic-aware shortest path algorithm. It takes into account previously computed information about the routes and schedules of the other agents, who store their plans in the infrastructure. To compute this shortest path we used the well-known label setting algorithm of Dijkstra [1].

Once the routes of the agents are known, the algorithm computes a feasible schedule for it to ensure a set of conflict-free operational routes. This scheduling algorithm operates on a greedy basis, where essentially two parameters are important: the *agent selection* function determining which agent will be selected for determining its schedule and a *block size* parameter determining which part of its route will be scheduled. Remember that an agent route essentially is a sequence of resources to be used. The block size parameter determines the number of resources for which the current schedule of the agent will be extended using the scheduling algorithm. These two steps are repeated until the whole group of agents has obtained a schedule for their complete plan. Once an agent A is selected to schedule it updates its schedule Sd_A according to the times as computed by the shortest path algorithm. Then, it commits the extra schedule times. These commitments might slow down other agents who were unlucky to be selected later by the agent scheduling function.

Rerouting In the HNZ-x algorithm[4], each transport agent gets x opportunities to change its route for each transportation order it executes. Immediately after an agent completes its schedule, it might decide to adapt its route, based on a comparison of the agent's current reward with a reward based on a new shortest

[4] In [4], we investigated this parameter and concluded that small values, e.g. $x = 1$, lead to good results. Therefore, in this paper we only consider HNZ-0 and HNZ-1.

path computation[5]. If the difference is greater than a given threshold value, the agent will decide to adapt its route. In consequence, other agents might also decide to reroute, because the agent who just changed its route might give room to another agent for obtaining a better route. To avoid that agents reroute for ever, the maximum number of reroute opportunities per order per agent is set to x in the HNZ-x algorithm. The latter has one exception: if an agent has used all its reroute opportunities, we give it one additional opportunity in case an incident disrupts the agent's regular plan execution. The autonomous agents only care about their own profits. However, they obey the scheduling algorithm and honestly wait for the agent selection function that defines when it is their turn. Thus, agents only let go of their individual optimal plans, if they are not selected in time by the agent selection function.

4 Experiments

Note that essentially the HNZ-algorithm is an algorithm scheme, containing two important parameters, viz. the agent selection function and the resource block size, that might heavily influence the performance of the resulting algorithm. It is the goal of this paper *(i)* to study the performance of the HNZ-1 algorithm, which uses two parameters: agent selection function and resource block size, and *(ii)* to measure the robustness of the resulting algorithms i.e., measuring the influence of different levels of incident densities on the performance. As a background comparison, we will use the performance of the HNZ-0 algorithms using the same parameters.[6]

Algorithmic parameter selection The first parameter of the HNZ-algorithm scheme is the *agent selection function*. Its goal is to select which agent's turn it is to schedule the next part of its plan. For each agent, the agent selection function computes a value and the agent with the smallest value wins. This agent is less bothered by future plans of other agents. We will investigate the following agent selection functions:

Random: agents are scheduled in a random order. This agent selection function provides a baseline.
Delays: the agent having to wait longest for other agents in the future can go first.
Deadlines: the agent with the most strict deadline for pickup/delivery can go first.
Penalties: the agent with the highest current plan penalty goes first.

The second parameter of the heuristic is the resource block size. The resource block size is the number of infrastructure resources the agent selected is to schedule now[7]. This parameter controls the greediness of the algorithm. If the agent

[5] Different from the previous one, because other agents have now committed to their schedules.
[6] Note that the algorithm used by Hatzack and Nebel [2] essentially is the HNZ-0 algorithm, although we will vary the two parameters mentioned.
[7] An agent schedules some extra resources if it ends up in a resource with a small capacity.

has unscheduled resources remaining in its plan, he has to be chosen again by the agent selection function to continue to schedule the remainder of its plan. We use the following blocksizes: 2, 4, 6 and infinite.

Problem instances Both algorithms, varying the parameter values, operated on the following set of problem instances. We used 10 different *infrastructures* consisting of 25 infrastructure resources: 10 of these infrastructure resources represent locations, 15 of them represent connections. Starting with a random tree (9 connections), to ensure the existence of a path between all pairs of location resources, we add 6 random connections. We ensured that at least some of the resources have sufficient capacity, to avoid deadlocks[8], the rest of the resources have capacity 1.

We included 10 different sets consisting of 75 transportation *orders* each. All have randomized source and delivery resources (the 10 location resources). We used a reward function that is 0 before and within the time-windows and linearly decreases when order o_j is picked up or delivered after the time-window closes at time t, i.e. $t^s_{j,2} - t$ or $t^d_{j,2} - t$ respectively.

We used two different sets of 30 *agents* located in random initial location resources.

Incidents Finally, the *incidents* are generated proportionally to the resources. We included three levels of incident densities in the experiments - a failure probability of 0, 0.1 and 0.2 respectively - and saved these incidents so that for later runs with other algorithms on the same problem instance we could reuse the incidents to obtain a fair comparison. The incidents are effective within a fixed small time-window. If the failure probability is 0.x, each resource is expected to malfunction x * 10% of the time, for both transport resources as infrastructure resources.

Results The results of the experiments are summarized in the Figures 1, 2 and 3. Figure 1 clearly shows one of the agent selection functions, viz. the function that looks at order deadlines, outperforms the other agent selection functions in all incident cases and both for HNZ-0 and HNZ-1. We observe the same trend for both with and without rerouting. Surprisingly, the other agent selection functions perform not better than the random selection function. Furthermore, the results seem to suggest that the differences between the selection functions become smaller if the number of incidents increases. Finally, note that in [4] we already showed that HNZ-1 outperforms HNZ-0. Here, we not only confirm these results, but also show that the same holds also when incidents do occur. Figure 2 clearly shows that performance decreases if the resource block size increases. Results are shown here for the random agent selection function, although the others show a quite

[8] We ensure transport resources only end their schedule in resources with sufficient capacity, such they cannot block the way for other agents for ever.

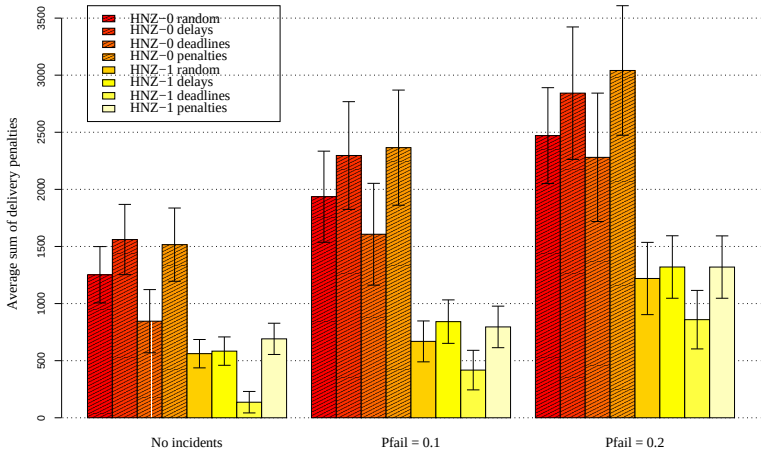

Figure 1: Different agent selection functions accompanied by 95% confidence intervals. Resource block size is fixed to 2 here. The vertical axis shows the total penalties, i.e. negative rewards. For data processing and plotting we use R [3].

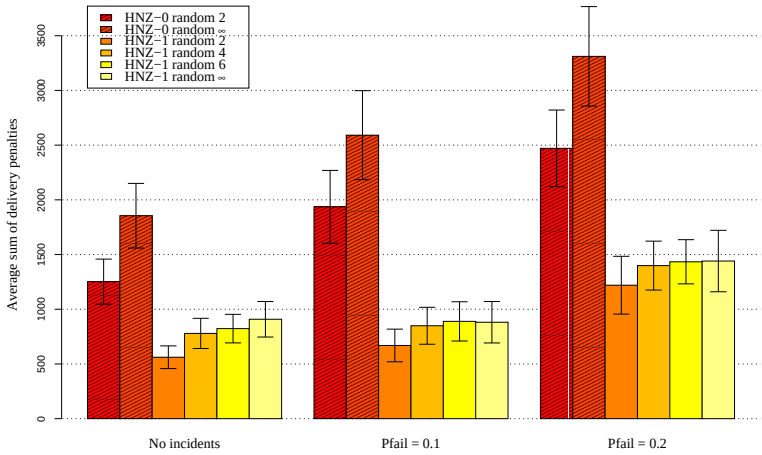

Figure 2: Performance for different resource block sizes.

similar trend. Under influence of incidents we see the same performance decrease, though the differences become smaller for the highest level of incidents.

Finally, Figure 3 shows a decrease in simulation time when the resource block size increases. Figure 2 and 3 together strongly indicate that a smaller block size improves performance at the cost of some extra computational investments.

5 Conclusion

In an earlier paper (cf. [4]) we discussed an extension of the Hatzack and Nebel approach by giving the agents a fixed number of opportunities to reroute. That

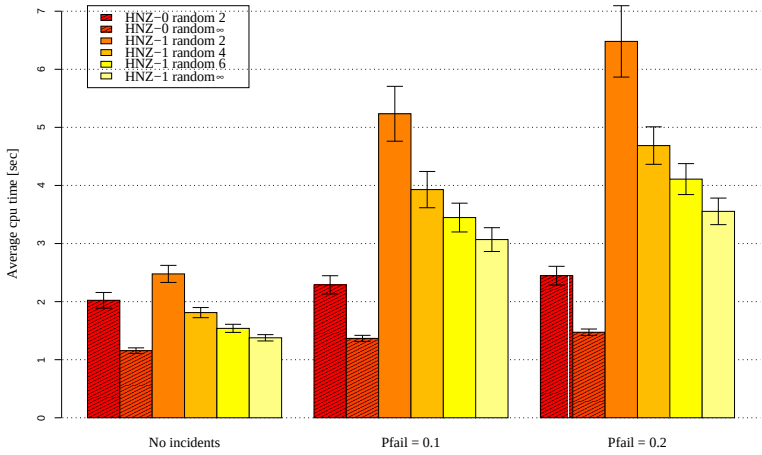

Figure 3: Average CPU-consumption for the different resource block sizes.

is, after the scheduling heuristic has assigned a schedule to the current route, an agent might conclude that perhaps an alternative route would offer him a better schedule. Based on small scale experiments we concluded there that a small number of opportunities sometimes might offer a significant improvement in overall performance. In this paper, we also included incidents in the experiments and we now infer the improvement is significant for any amount of incidents.

Furthermore, we refined the HNZ-1 algorithm scheme by distinguishing several parameters (agent selection and block size). We evaluated several agent selection functions and block sizes. Prioritizing agents with strict deadlines seems to give the best results, though we did not expect the data would suggest the difference is not significant anymore in both cases with incidents.

According to our intuition, a smaller block size resulted in better performance at the cost of some extra computational efforts.

References

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

[2] W. Hatzack and B. Nebel. Solving the operational traffic control problem. In A. Cesta, editor, *Proceedings of the 6th European Conference on Planning (ECP'01)*, 2001.

[3] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-00-3.

[4] J. Zutt, L.D. Aronson, R.P.J. van der Krogt, N. Roos, and C. Witteveen. Multi-agent transport planning. In H. Blockeel and M. Denecker, editors, *Proceedings of the Fourteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC'02)*, pages 387–394, October 2002.