Public Transp (2010) 2: 249–268
DOI 10.1007/s12469-010-0033-6

ORIGINAL PAPER

Real-time train driver rescheduling by actor-agent techniques

Erwin J.W. Abbink · David G.A. Mobach · Pieter J. Fioole · Leo G. Kroon · Eddy H.T. van der Heijden · Niek J.E. Wijngaards

Published online: 12 November 2010

Abstract Passenger railway operations are based on an extensive planning process for generating the timetable, the rolling stock circulation, and the crew duties for train drivers and conductors. In particular, crew scheduling is a complex process.

After the planning process has been completed, the plans are carried out in the real-time operations. Preferably, the plans are carried out as scheduled. However, in case of delays of trains or large disruptions of the railway system, the timetable, the rolling stock circulation and the crew duties may not be feasible anymore and must be rescheduled.

This paper presents a method based on multi-agent techniques to solve the train driver rescheduling problem in case of a large disruption. It assumes that the timetable and the rolling stock have been rescheduled already based on an incident scenario. In the crew rescheduling model, each train driver is represented by a driver-agent. A driver-agent whose duty has become infeasible by the disruption starts a recursive task exchange process with the other driver-agents in order to solve this infeasibility. The task exchange process is supported by a route-analyzer-agent, which determines whether a proposed task exchange is feasible, conditionally feasible, or not feasible. The task exchange process is guided by several cost parameters, and the aim is to find a feasible set of duties at minimal total cost.

The train driver rescheduling method was tested on several realistic disruption instances of Netherlands Railways (NS), the main operator of passenger trains in the

E.J.W. Abbink (✉) · P.J. Fioole · L.G. Kroon
Netherlands Railways, NSR Logistics Innovation, P.O. Box 2025, 3500 HA, Utrecht, Netherlands
e-mail: erwin.abbink@ns.nl

D.G.A. Mobach · E.H.T. van der Heijden · N.J.E. Wijngaards
D-CIS Lab, Thales Research & Technology NL, P.O. Box 90, 2600 AB, Delft, Netherlands

L.G. Kroon
Rotterdam School of Management, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR, Rotterdam, Netherlands

Netherlands. In general the rescheduling method finds an appropriate set of rescheduled duties in a short amount of time. This research was carried out in close cooperation by NS and the D-CIS Lab.

1 NS train driver rescheduling

The railway operations of Netherlands Railways (Nederlandse Spoorwegen or NS) are based on an extensive planning process, consisting of three phases: timetable planning, rolling stock scheduling, and crew scheduling. The crew scheduling process supplies each train with a train driver and with sufficient conductors. In the past years, NS has successfully applied novel Operations Research models to significantly improve the planning process for which NS received the Edelman Award 2008 of INFORMS, see Kroon et al. (2008).

After the planning process, the daily plans are carried out in the real-time operations. Preferably, the plans are carried out exactly as scheduled. However, in the real-time operations, the plans have to be updated continuously in order to deal with delays of trains and larger disruptions of the railway system. A disruption may be due to an accident, or a breakdown of infrastructure or rolling stock. On the Dutch rail network (more than 5000 daily trains), on average 3 disruptions of a route occur per day. Delays occur even more frequently: On average 450 trains experience one or more delays (>3 minutes) per day. These delays lead to about 10 cancelled train services per day. The current methods and techniques are very useful for generating the initial daily schedules, yet their calculation time usually takes multiple hours, making them unfit for direct application in real-time rescheduling purposes. In this paper we focus on an actor-agent based approach for *real-time rescheduling of train drivers*.

The crew scheduling process at NS is very complex, see Abbink et al. (2005). NS train drivers operate from 29 crew bases. Each day a driver carries out a number of tasks, which means that he/she operates a train on a trip from a certain start location and start time to a certain end location and end time. The trips of the trains are defined by the timetable. Train drivers can use *positioning* trips to travel to the starting location of driving tasks. In addition, *stand-by* tasks are defined and assigned to *stand-by* train drivers: these can be used to resolve rescheduling problems. The tasks of the train drivers have been organized in a number of duties, where each *duty* represents the consecutive tasks to be carried out by a single driver on a single day. Each duty starts in a crew base, and a hard constraint is that the duty ends at the same crew base within a limited period of time. Also several other constraints must be satisfied by the duties, such as the presence of a meal break at an appropriate time and location, and an average working time per crew base of at most 8 hours. Initially in the planning process, duties are anonymous, which means that the allocation of drivers to duties is still needed. The latter is handled by the creation of crew rosters, which describe the sequence of duties that are carried out by the individual drivers on consecutive days.

The total number of train drivers is about 3000. Each day, about 1000 duties are carried out by the drivers. Furthermore, at any moment in time, the number of active duties at that moment is about 300. Due to cancellations and delays of trains or rescheduling of the rolling stock a number of duties of train drivers may become infeasible. An infeasibility of a duty is due to a time conflict (often caused by delays) and/or a location conflict (often caused by cancelled train services). In both cases, a conflict occurs between two consecutive tasks in the duty. Dispatchers are responsible for rescheduling tasks among train drivers so that the trains are staffed adequately. Dispatchers are organized into five regions, where they are responsible for rescheduling train drivers who currently reside in their region. Often dispatchers need to perform task-rescheduling actions, which they can handle given the available time and resources. Typically, about five minutes are spent to resolve a single inconsistent duty. Frequently, rescheduling problems are left ‘open ended’ for later resolution by other dispatchers (often in another region). In larger disruptions some trains simply cannot be driven as dispatchers are busy rescheduling train-drivers, causing additional delays for passengers.

The main objectives of the train-driver rescheduling research system described in this result-oriented paper are to explore the effectiveness and suitability of a decentralized, actor-agent based approach to crew rescheduling. Another objective is to determine whether multi-agent technology is sufficiently mature to be used in a real-world decision support system.

This paper is structured as follows. Section 2 describes the main actor-agent design paradigm and elements of the rescheduling system. Section 3 describes the implementation of the rescheduling research system, followed by a description of results in Sect. 4. This paper is concluded by a discussion and comparison with related work in Sect. 5.

2 Related work

Traditionally, crew scheduling problems are approached using Operations Research techniques. Real-time crew *rescheduling* however is a relatively new area of research.

Potthoff et al. (2010) and Veelenturf et al. (2009) present an algorithm to reschedule NS’ crew when a disruption occurs. They study the same problem as presented in this paper. Their model is a set covering model, and their algorithm is based on column generation techniques combined with Lagrangian heuristics from Operations Research. To handle the very large number of duties in practical instances, they first define a core problem, with a limited number of duties. If some tasks remain uncovered in the solution of the core problem, they perform a neighbourhood exploration to improve the solution by adding additional duties to the core problem. Computational experiments with real-life instances show that their method is capable of producing good solutions within a couple of minutes of computation time. The model of Veelenturf et al. (2009) is an extension of the model of Potthoff et al. (2010), since it allows trains to be slightly delayed if this leads to better crew schedules.

Rezanova and Ryan (2010) present a similar solution approach for a railway crew rescheduling problem. They formulated the problem also as a set partitioning problem with side constraints. Their approach uses depth-first search in a branch-and-price tree. The solution method starts with a small set of duties around the disruption, containing only duties in a limited time period with delayed, canceled or re-routed tasks. As long as some constraints are violated while solving the LP-relaxation of the model, the set of duties is extended.

Walker et al. (2005) describe a model for simultaneously rescheduling the timetable and the crew duties. In the timetabling part, the departure time of a train can be chosen within certain time windows. Constraints on duty length and task sequencing ensure that the departure times are chosen such that only the meal break rule is possibly violated in some of the selected duties. Breaks are added to the duties during the branching process. A conflict free timetable is obtained by branching on the priority decisions between pairs of trains.

To our knowledge, no research has been published on agent-based crew rescheduling applications in the railway domain. Shibghatullah et al. (2006) propose an agent-based framework for bus crew scheduling including crew reassignment. The paper provides an overview of the potential advantages of agent-based approaches (e.g., modelling individual preferences, more suited for partial, on-demand rescheduling), but lacks further details of the proposed framework.

Jiang and Xie (2009) describe a multi-agent model for simulating train movements in a mass rail transit system. However, in their model, the agents represent the trains, and the crews are not considered. They illustrate their model by an example of two rail transit lines in Shanghai.

Tranvouez and Ferrarini (2006) present a multi-agent based approach to disruption management in the supply chain domain. A distinction is made between partial and complete rescheduling, as well as periodic and event-driven rescheduling. It is argued that partial, event-driven rescheduling appears to increase schedule stability. Our rescheduling approach can also be classified as partial, event-driven rescheduling. Schedule stability is also a desirable characteristic in the railway crew rescheduling domain, given the fact that initial crew schedules are already optimized for efficiency, and changes should be minimized.

Mao et al. (2007) recognize the need for short-term operational planning and scheduling methods in the domain of airport resource scheduling, and present an agent-based approach based on two coordination mechanisms: decommitment penalties and a Vickrey auction mechanism. The coordination approach used in this paper is based on a combination of similar mechanisms: The driver-agent interaction protocol has auction-like properties (agents report costs (i.e. bid) for taking over tasks), and decommitment penalties are determined based on increasing commitment levels.

In literature, coordination approaches based on negotiation concepts are often divided in cooperative and non-cooperative (self-interested) approaches. Although driver-agents in our model can in some respect be considered as self-interested agents (driver preferences are included in the agent's cost function), the agents cooperate to achieve the global goal of resolving disruptions, and agents do not engage in direct competition.

3 Design

In this section the main principles underlying the actor-agent based train-driver rescheduling process are introduced. First, the applied design paradigm is introduced. After this, the two main elements of the rescheduling system are described.

3.1 Actor-agent paradigm

The actor-agent paradigm, see Iacob et al. (2009), explicitly recognizes both human actors and artificial agents as equivalent team members, each fulfilling their respective roles in order to reach the team objectives. The involved agents have quasi cognitive capabilities that are complementary to (as opposed to mimicking) human cognition. Actor-agent teams and communities are hybrid collectives of human experts ("actors") and agents with complementary cognitive capabilities which are focused on a certain problem and reach a structural and functional complexity that matches the size and nature of the problem as well as possible.

The actor-agent based design process provides the system with several useful global system characteristics. First, the decentralized approach in which agents use local knowledge, world views, and interactions, contributes to an open system design. This openness facilitates easy reconfiguration and/or adaptation to changing system requirements. Second, combining humans and agents within the system design allows for integrating them at their appropriate abstraction levels.

3.2 Rescheduling principle

The main principle for the train-driver rescheduling application is to model the solution based on 'levels of responsibility' in the dispatching and rescheduling process:

- Human dispatchers at the strategic/management level,
- Human train drivers at the level of defining and guarding their personal interests, and
- Their respective agents at the level of implementing the strategic/management decisions and resolving actual schedule conflicts.

In our approach driver-agents represent driver-actors in the rescheduling process. In the event of a disruption, all driver-agents are informed of the disruption details (i.e. cancelled/delayed train services). The driver-agent(s) directly affected by a disruption (i.e., the disrupted train service is associated with a task in their driver's schedule) assume the role of *team leader*. Each team leader starts a team-configuration process in order to resolve its respective schedule conflicts.

The main principle underlying the actor-agent based rescheduling process is that of *task-exchange*: Each team is extended with additional team members able to take over tasks from agents already participating in the team. A driver-agent may be able to take over tasks without affecting other tasks already present in its schedule, for example replacing a positioning trip with a task (*unconditional* takeover). However, in most cases, a driver-agent will only be able to take over tasks by replacing existing tasks in its schedule (*conditional* takeover). This will then lead to a new set of (conflicting) tasks to be taken over by another driver-agent. In Fig. 1, an example of this process is shown: Driver-agent 1 is informed that task B–C is cancelled, which makes it infeasible for him to operate task C–D. Driver-agent 1 becomes a team leader and asks who can take-over task C–D. Driver-agent 2 conditionally replaces task F–D with task C–D from driver-agent 1. He uses a positioning trip F–C to get to the start location of task C–D. Driver-agent 2 asks who can take-over task F–D. Driver-agent 1

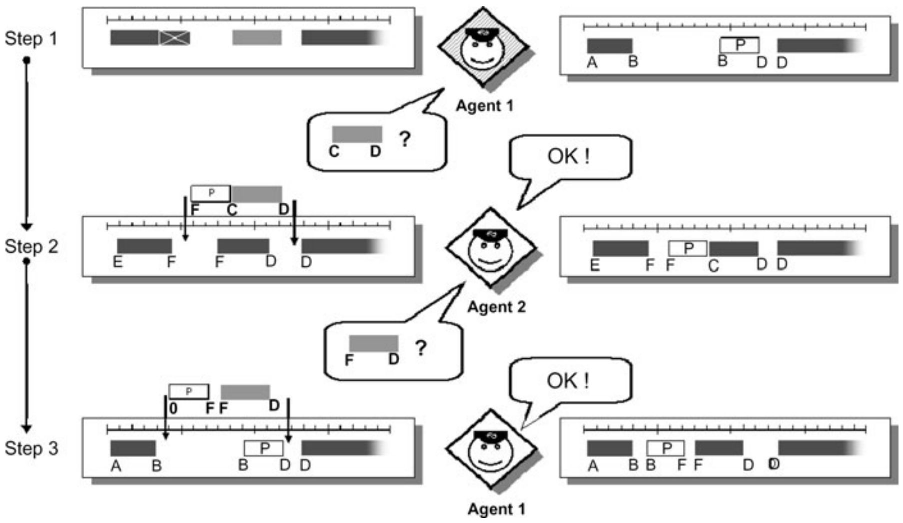
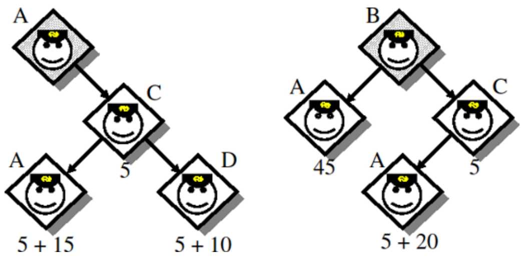


Fig. 1 The task takeover process

Fig. 2 Two example task exchange teams consisting of three agents



now takes over task F–D from driver-agent 2 unconditionally, using a positioning trip from B to F. This example results in a task exchange, involving two unique train drivers.

In case a driver-agent has determined that tasks can be taken over, a cost-function is applied to determine the costs associated with the takeover: costs are assigned to various aspects such as the amount of overtime introduced, replacement of meal breaks, etc. Subsequently the set of new conflicting tasks of this agent (if any) is announced to the other driver-agents. This leads to a recursive addition of layers of team members to the team, resulting in a team consisting of multiple task-exchange configurations, originating at the team-leader.

In Fig. 2, two team configurations are shown, consisting of four driver-agents (A, B, C and D): In the left team, driver-agent A is team leader, and starts the task-exchange process. The same driver-agent participates as a team member in its own team, as well as (in two different configurations) in the team led by driver-agent B: This feature allows for any driver-agent to participate in possible task-exchange configurations and thus to facilitate the solution process to find the best task-exchange configurations for each team. Driver-agents can withdraw themselves from teams and team-configurations based on the commitment levels in the task-exchange protocol, ensuring local and global consistency when final team-configurations are determined. Details of this protocol are beyond the scope of this paper and can be found in Mobach et al. (2009).

The team extension process is considered complete when a configuration of task-exchanges is determined in which all conflicts have been resolved, or any remaining conflicts have been shifted forward in time sufficiently to be resolved at a later point in time (re-introduced as new conflicts later). At this point, the recursive team formation process is reversed: Each layer within a team selects the task-exchange associated with the lowest cost, starting at the lowest layer. Once all team leaders have determined a final team configuration, the entire solution is presented to the dispatcher.

In the remainder of this paper we use the term *conflict* for the set of tasks that cannot be operated due to a delay or disruption or due to the take-over of a set of tasks from another driver-agent.

During the team formation process, a number of heuristics are used, aimed at limiting the team extension to promising additional team members. These heuristics currently included are described below.

3.3 Team formation heuristics

Due to the large number of driver-agents engaged in various roles in the team formation process and the fact that a short calculation time is an important success factor in the disruption management process, heuristics are used to constrain the extension of teams with additional team members. The main goal behind the applied heuristics is to 'only let driver-agents participate that have a high probability of improving the solution found'. In this section, the applied heuristics are described: *interest determination*, *cost calculation and scoreboard mechanism*. These will be discussed in more detail below.

3.3.1 Interest determination

Based upon the schedule impact determined earlier, a driver-agent decides whether it is *interested* in joining a team. Several domain-specific criteria are evaluated to determine a takeover desirability:

- If the impact of taking over a conflict results in a new conflict which is *larger* than the original, the agent is not interested. Here, larger is defined as the duration of the set of tasks contained in the conflict. The idea is in general that it is easier to find a solution for a small conflict than for a large conflict.
- Taking over tasks from a driver-agent may only result in introducing a new conflict if the tasks in the new conflict are situated *later* in time. The idea is that, if a conflict is later in time there are more possibilities for drivers to get in time to the starting location of the conflict.
- If the new conflict introduced by taking over tasks from a driver-agent consists of the *same set* (or superset) of tasks introduced by this agent elsewhere in the team, the agent will not be interested. This prevents repeating evaluation of sequences of task-exchanges that are being evaluated already.

3.3.2 Cost calculation

A driver-agent interested in taking over tasks must determine the costs associated with this takeover. The cost function is the sum of the (positive) costs assigned to the following elements:

- The amount of overtime introduced by extending a schedule past the original end time of the driver-agent's duty;
- An affected meal break;
- The use of stand-by drivers. Stand-by drivers are preferred in case of larger disruptions, and should not be used for resolving small/simple conflicts;
- The size of the team (e.g. preferring recurring team members);
- Individual train driver preferences (i.e., modifiers to the above elements).

The relative weights of the cost elements can be varied to prioritize different aspects of the solutions that are found (e.g. prevent the use of stand-by drivers by increasing their relative cost). The costs are subsequently compared to costs of other exchanges using a scoreboard mechanism, which is described below.

3.3.3 Scoreboard mechanism

Upon starting a task exchange process, each team leader publishes a *scoreboard*, which can be used by team members to inform other driver-agents of the progress within a task-exchange team. The scoreboard is based on the principle that every driver-agent in a configuration of a team has knowledge of the 'cost' of the exchange configuration (the task exchanges leading up to the task-exchange this driver-agent is currently examining).

The moment a driver-agent has determined that a conflict can be resolved without generating a new conflict, it publishes the value of the current solution (i.e., the cost of one complete possible configuration) on the scoreboard. All driver-agents can access the scoreboard and examine the costs of the solutions which are already found. Driver-agents can use the values on the scoreboard to decide whether to continue a task-exchange or not. If the cost of the current part of the solution is already the same or higher than the scoreboard value, the driver-agent will terminate its involvement in this branch of the task-exchange.

Figure 3 shows an example: First, driver-agent B determines that it can solve the conflict of A (TL = Team Leader) at cost 25. It finds that the scoreboard is still empty, and updates it with the calculated costs (arrow 1). Subsequently, driver-agent C finds that in order to solve the team leader's conflict, it has to introduce a new conflict. Before continuing, it checks the scoreboard to determine if the current costs are an improvement. In this case C continues. Driver-agent D determines that it can solve the conflict of driver-agent C without introducing a new conflict. It also determines that the cost of the solution (15) improves the current score on the scoreboard. D updates the scoreboard (arrow 2). Furthermore, A (DA = Driver Agent) determines it can solve the conflict of driver-agent C with the cost of 25, checks the scoreboard and aborts, as it has determined that its (partial) solution can never improve the team's current score. The scoreboard mechanism ensures that only solution alternatives are

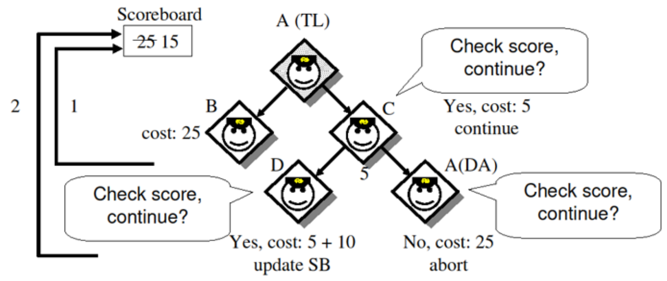


Fig. 3 Scoreboard example

evaluated which improve the currently found solution(s). Note that this mechanism depends on the cost function to be strictly increasing and to accurately discriminate between 'better', 'similar' and 'worse' solutions.

3.4 Analyzing routes

In this section, the agents involved in determining the feasibility and impact of schedule changes on driver schedules are discussed. This computationally expensive functionality is deliberately separated from the driver-agents task-exchange capabilities to enhance performance and ensure a stricter division of responsibilities. The route-analyzer-agent (RAA) is the central point of contact for the driver-agents; distributed network agents (NAs) support the RAA.

3.4.1 Route-analyzer-agent

Requests for route calculations from the driver-agents are handled by a RAA. A request consists of a duty and a conflict. The duty is the current duty of the requesting driver-agent. The conflict consists of one or more tasks to take over from another driver-agent. The answer returned by the RAA can either be *feasible*, *feasible conditional* or *not feasible*. Feasible indicates that it is possible to add the conflict to the duty of the driver-agent without introducing a new conflict. In addition to the conflict one or more passenger tasks (trips to/from the conflict) may be added to the duty. If it is possible to add the conflict to the duty of the driver-agent, but only if a new conflict is introduced in the process, the answer is *feasible conditional*. In this case, the new conflict may not contain any tasks contained in the old conflict. When it is not possible to add the conflict to the duty of the driver-agent at all, the answer will be *not feasible*.

The RAA attempts to determine the correct answer to a request without having to take into account the detailed current state of the rail network. To this end, the RAA performs three steps. First, a check is performed to determine if a negative answer can be given quickly. After this, the request is compared to a history of previously received requests. If no answer is found the request is distributed over the available NAs for detailed examination. These steps are described in more detail below.

Step 1: *Sanity Check*: In a large number of cases (∼50%) it can be easily determined that a request is not feasible. Consider for example a request where a driver-agent's

Table 1 Example duty for driver-agent

Departure	Destination	Departure times	Arrival times
Rotterdam	Utrecht	7:45	8:24
Utrecht	Gouda	9:08	9:22
Gouda	Rotterdam	9:32	10:08
Rotterdam	Maassluis	10:58	11:21
Maassluis	Rotterdam	11:29	11:53
Rotterdam	Eindhoven	11:47	13:00
Eindhoven	Rotterdam	13:32	14:42

current location is Rotterdam and tasks to take over are in the vicinity of Groningen, and start within 15 minutes. The distance between Rotterdam and Groningen is more than 200 kilometers: this is clearly an impossible takeover. The RAA uses an origin-destination matrix with lower bounds on the travel time between all stations. These lower bounds are static and calculated beforehand by taking into account only geographical distances in the network and maximum driving speeds of the available train-units. If the lower bound is larger than the available time it is not possible to find any route.

Step 2: *Request History*: The RAA retains all calculated routes in memory. If the same request is received more than once (possibly by different driver-agents), the answer is retrieved from this history (∼4–5% of the remaining requests). Furthermore, if a previous request with a wider time-interval for the same route resulted in *not feasible*, the RAA can conclude that the current request with a smaller time-interval also results in a *not feasible* answer. This history is reset when changes to the rail network take place.

Step 3: *Send to NA*: If no relevant requests are found in the history, the RAA sends the request to one of the NAs and returns the thus obtained answer to the driver-agent.

As an illustration, consider a driver-agent with a duty as shown in Table 1. This driver-agent wants to know if he can take over a trip from another agent. This trip departs at 10:51 from The Hague and arrives at 11:10 in Gouda (not shown in Table 1; this driver-agent is not a team leader, i.e. not directly affected). The driver-agent sends this request to the RAA when the actual time is 9:00.

The RAA determines that the driver-agent's current location is Utrecht. The minimal time to get from Utrecht to The Hague is 35 minutes. In this case the driver-agent has 111 minutes available (i.e., from 9:00 tot 10:51) so a route might exist. Similarly, the RAA concludes that a route from Gouda back to Rotterdam could also exist. The RAA concludes that further examination of the request is necessary.

When all NAs are unavailable, the RAA maintains a priority queue of requests to send to a NA. For each request the RAA assigns a *prediction-value* and keeps the queue sorted according to this value. When a NA becomes available, the RAA sends the request with the best prediction-value to this NA. The prediction-value represents an expectation by the RAA of how well the conflict can be fit into the duty. This is the weighted sum of:

Table 2 New duty for driver-agent

Departure	Destination	Departure times	Arrival times
Rotterdam	Utrecht	7:45	8:24
Utrecht	Gouda	9:08	9:22
Gouda	Rotterdam	9:32	10:08
Rotterdam	**The Hague**	**10:14**	**10:40**
The Hague	Gouda	10:51	11:10
Gouda	**Rotterdam**	**11:16**	**11:38**
Rotterdam	Eindhoven	11:47	13:00
Eindhoven	Rotterdam	13:32	14:42

- Train driver duty task lengths in the conflict time interval.
- Lower bounds on the travel time from the current location to the start of the conflict, and
- Lower bounds on the travel time from the end of the conflict to the base.

These items are determined by an initial analysis of the outcome of a couple of thousand requests, where the weights were determined by regression analysis. For example, it is straightforward to understand that if a driver-agent has a lot of work within the time-interval of the conflict he will surely have to send out a new conflict if he takes over the current conflict in his request. Similarly, agents that are located closer to a conflict are more likely to take over the conflict in an efficient way.

An important factor contributing to the success of the scoreboard mechanism (see Sect. 3.3.3) is a *first value* being published as soon as possible. Sorting the requests this way helps to find good solutions more quickly. Once a good solution has been found in a team, the scoreboard is seeded with this first score, activating the scoreboard mechanism.

3.4.2 Network-agents

The NA maintains knowledge of the current timetable, including all disruptions and delays. In the actor-agent community at least one NA has to exist, but possibly more. NAs can be distributed over multiple platforms (no cooperation is necessary and the data can be replicated). Based on this timetable the NA can determine, by using a shortest-path algorithm, if a route between two stations exists, and if so, which route. The objectives in this process are (i) to maintain as much of the original duty of the driver-agent as possible, and (ii) to arrive at the destination (i.e. the conflict's location) as soon as possible. To illustrate this, consider again the example of Table 1, continued in Table 2.

The NA attempts to find a route from Utrecht to The Hague between 9:00 and 10:51. The fastest way to get to The Hague is a direct connection which departs from Utrecht at 9:03 and arrives in The Hague at 9:51. This means a route actually does exist. But there also exists a better alternative for this driver-agent: The NA also finds a direct connection between Rotterdam and The Hague departing at 10:14 and arriving at 10:40. This way the train driver can still perform the tasks of its own duty until 10:08 and be in time for the task-exchange trip.

Finally the NA is able to find a trip from Gouda to Rotterdam, so the driver-agent can also perform the last two tasks from its own duty. This means, if the driver-agent wants to take over the requested trip, its new duty will be as shown in Table 2. The underlined trip is taken over by this driver-agent, while for the **bold** trips the driver rides the train as a passenger. In this case the driver-actor can no longer perform the task from Rotterdam to Maassluis (10:38–11:21) and from Maassluis to Rotterdam (11:29–11:53). This is the *feasible conditional* answer the NA will return to the RAA, which forwards this to the driver-agent.

Note that this change to the driver-agent's duty also means that the 50 minute break in Rotterdam from 10:08 until 10:58 no longer exists. The driver-agent will take this into account when examining the schedule changes; the NA only assesses if a route is *possible*, not whether the route is *desirable*.

In case the conflict can be fit into the duty (as specified in the request received by the NA), the returned answer consists of the necessary duty adjustments and a set of tasks which can no longer be performed. This set of tasks can be empty; in that case the conflict can be fit into the duty without introducing a new conflict. When the conflict cannot be fit into the duty ('failure'), the NA returns no duty adjustments nor a set of tasks.

4 Implementation

The research system is constructed in the real-world domain of disruptions on the Dutch railroad network as a reliable and scalable operational system, intended to operate together with the current systems in place. The research system is not constructed as a full production system but as a prototype. An important requirement for the system is that of *performance*: Solution times in the order of minutes are required (i.e. better than human dispatcher performance). The following implementation decisions were made:

- The actor-agent based solution process is defined first, after which an accompanying architecture is designed, on the basis of which suitable technologies are selected.
- To support the explorative nature of the project, it is necessary to (always) have a running research system for testing, debugging and assessment purposes.
- The research system is to be as decentralized as possible.

The research system focuses on rescheduling train driver duties in real-time over the course of a single day. It is assumed that any necessary timetable modifications to cope with the disruption have been implemented, and a new rolling stock plan is in place, to which the driver schedules are to be adapted. Whenever a solution to the disruption has been accepted, this 'new' plan is the basis for the rescheduling process in response to new disruptions (or a change in duration/consequences of an 'old' disruption).

For the implementation of the research system, the Cougaar agent framework is used, see Helsinger et al. (2004). Cougaar has initially been developed for supporting logistics operations and provides a useful Java-based agent platform with emphasis on stability. Cougaar provides an agent model based on *plug-ins*, allowing for clean separation of functionality within agents. Furthermore, Cougaar allows for implementing *services*, which can be made available to agents running on a Cougaar *node*. Cougaar agents use a distributed *blackboard* for storing information and communication purposes.

To start the system, a bootstrap-agent dynamically instantiates the driver-agent population and other main agents. This allows for a flexible start-up process, enabling for example dynamic instantiation of the agents across a multiple-node configuration.

In order to run realistic scenarios, a dataset containing train activities and driver tasks for a full day has been provided by NS.

4.1 Approach

The implementation of the rescheduling system is realized using a cyclic (re)design, implementation and evaluation process, comparable to the well-known rapid prototyping approach. Three main cycles have been performed during the implementation phase:

Cycle 1: *Demonstrator*: A small-scale version of the research system (see Abbink et al. 2008) was implemented first, consisting of less than 10 driver-agents and with coarse grained route calculation capabilities. This version focuses mainly on the implementation of the driver-agents and the task-exchange protocol: All driver-agents are instantiated with a driver schedule and are able to communicate with each other using the aforementioned blackboard in order to form teams and resolve conflicts due to disruptions.

Cycle 2: *Scaling up*: Evaluation of the demonstrator showed that incorporating a model of the real-time rail network in every driver-agent introduces computational overhead. This resulted in the separation of this functionality into the earlier mentioned *route-analyzer-agent* and *network-agents*.
In addition to this, efforts were made to apply heuristics to make the rescheduling process more efficient by regulating the size of the task-exchange teams. To this end, the earlier mentioned *scoreboard* mechanism was introduced. The scoreboard mechanism ensures that only solution alternatives are evaluated which may improve the currently found solution(s), due to the strictly increasing property of the cost-function.
At the end of cycle 2, a version of the research system was realized that was capable of finding solutions for relatively large disruptions with a driver-agent population consisting of up to 200 agents within reasonable time. The results of this cycle are described in Abbink et al. (2009) and Mobach et al. (2009).

Cycle 3: *Full-scale prototype*: In the third cycle the full-scale prototype was developed, capable of supporting the complete driver-agent population. Other important issues confronted in this cycle were the dispatcher-interface to the rescheduling system and visualization and debugging tools.

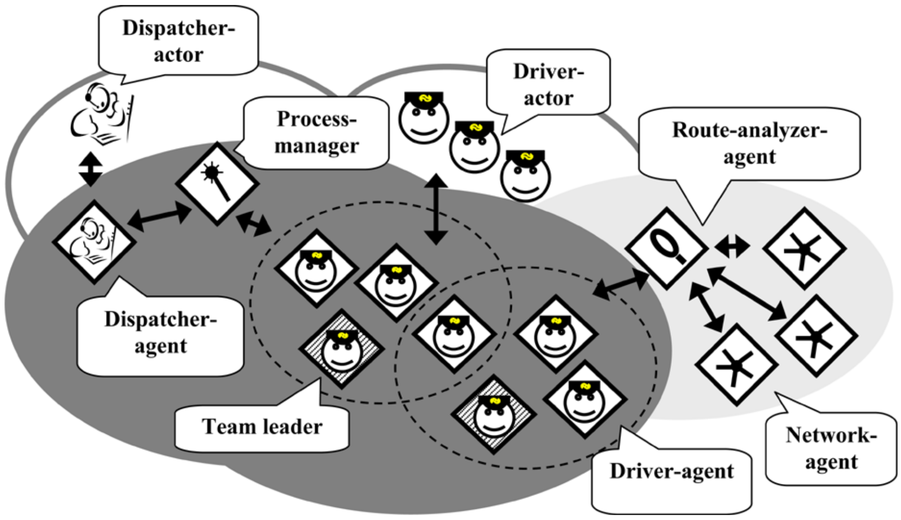


Fig. 4 Current research system architecture

4.2 (De)centralisation

Our cyclic development approach allowed for specific reflection moments to assess our current progress, including the (emergent) behaviour of the overall system. Figure 4 shows the agents and their relations as implemented in the final research system:

The above described architecture shows how an initial, fully decentralised approach was deliberately changed into an improved architecture in which two distributed subsystems can be distinguished:

- The driver-agent rescheduling subsystem, including dispatcher-agents, a process-manager-agent (PMA), and the driver-agents.
- The route-analyzer subsystem, including the RAA and NA.

In this figure two new agents are introduced: (1) The Dispatcher-Agent responsible for the communication to the Dispatcher via the graphical interface, and (2) the Process-manager-agent who is responsible for coordinating the overall solution process. The Process-manager-agent determines the order in which the team-leaders solve their problem. In the current implementation the problems are solved sequentially, based on the time of the problem.

5 Experimental results

This section provides an overview of our current[1] findings. The current version (October 2009) of the system is able to find solutions for relatively large disruptions, using the full driver-agent populations.

[1]The findings presented in this section are based on the Q4 2009 version of the research system.

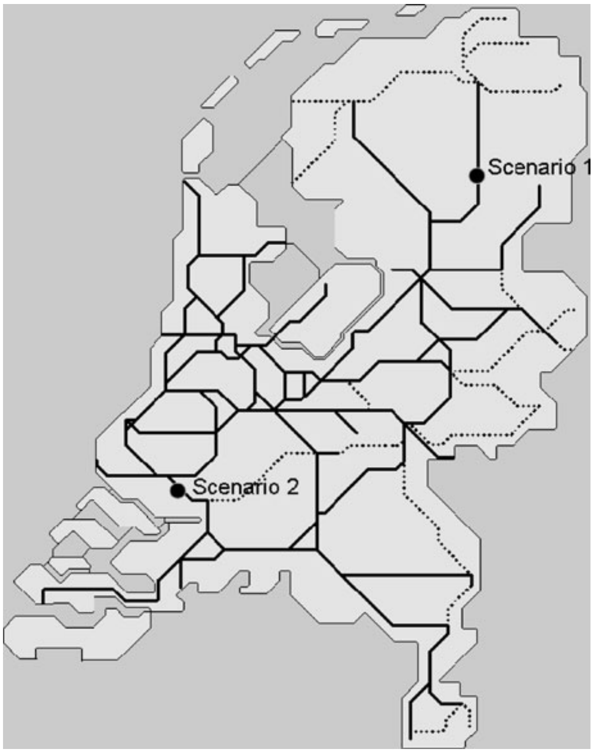


Fig. 5 The Dutch railway network and the locations of the two disruption scenarios

To illustrate this, results of two relatively complex example scenarios are presented. The first scenario consists of a complete blockage of the railway network between stations Groningen and Zwolle from 16:00 to 17:00. The second scenario consists of a partial blockage at station Lombardijen. Figure 5 provides a simplified overview of the Dutch railway network, in which the locations of the disruption scenarios are indicated. Solid lines denote the network responsibilities of NS, dotted lines are handled by other operators.

The results described below are based on pre-defined scenarios solved with the research system. It is difficult to compare the quality of the solutions with the real world dispatchers' responses. This is the case because the dispatchers solve the problems at hand step-by-step, when meanwhile the world changes. This changing environment directly alters the consecutive problems to be solved. The presented algorithm solves the problems at hand in a single step. This means that the solutions can be compared only for instances where the real world does not change during the manual solution process. Unfortunately, this is not realistic.

To still get an indication of the quality of the solutions, we discussed these with several dispatchers. Initially this lead to comments that were used to improve the algorithm, to solve several problems in the data preparation, and to adjust the weights of the cost function. After these adjustments, for all solutions presented in this paper, the planners had no further remarks and indicated that the solutions were better than the solutions generally generated by themselves. The solutions generated by our system are based on a higher number of task take-overs (as described in Sect. 3.2) than a planner can mentally produce with the (registration) tools currently available. As a result, solutions are found with a smaller usage of stand-by drivers, less overtime and, in the end, less additionally cancelled train services.

The results discussed in this section are evaluated by the number of driver-agents involved in the final solution, as well as by the number of stand-by drivers and the additional overtime introduced. The aim of the result analysis in the two scenarios is to assess the scalability and the further behaviour of the research system. Interestingly enough, not only the number of driver-agents is of importance to assess scalability, but also whether stand-by driver-agents are involved. The cost function adds extra cost for the use of stand-by driver-agents; the intent is to avoid making stand-by driver capacity too 'cheap'. In addition, the cost function heavily depends on overtime by the involved driver-agents, which is shown for the final solutions found.

In addition, calculation times are provided: these times are indicative only (although they do indicate that the research system clearly outperforms a single human dispatcher), as the research system has not been (re-)engineered as a real-time operational system yet.

5.1 Scenario 1: blockage Groningen-Zwolle

For scenario 1 the number of cancelled train services due to the blockage is 11, which leads to 11 driver-agents to act as team-leaders. Table 3 shows the results for the first scenario of various runs with different driver-agent populations.

In Run 1 we selected 80 driver-agents based on the fact that they are located near the disruption at the time of the disruption: in this case a solution can be found quickly. This solution could be improved by including a stand-by driver, as is shown in Run 2. In Run 3 there are 42 additional driver-agents included (compared to run 1) that are located near the disruption, but after the actual disruption itself. This also improves the solution found, as driver-agents now have the possibility to exchange tasks forward in time until later in the evening, when more free capacity is available.

Table 3 Results scenario 1

Run	# driver-agents (stand-by drivers)	Total # final team members	Overtime (minutes)	# stand-by drivers in final solution	Calculation time[a]
1	80 no stand-by	24	456	0	1:17
2	80 +1 stand-by	22	328	1	0:55
3	122 no stand-by	26	425	0	2:56
4	122 +1 stand-by	25	243	1	2:45
5	122 +4 stand-by	20	159	3	1:39
6	602 no stand-by	24	150	0	11:44
7	602 +49 stand-by	18	150	2	2:12
8	961 +77 stand-by	18	150	2	6:25

[a]Calculation times indicative only; System configuration: 2 × Intel Xeon X5472 3.0 GHz. 8.0 GB RAM

In the rest of the scenarios we add both regular driver-agents as well as stand-by driver agents. The table shows that both have a positive effect on the solution found. What is not shown in the table but what can be deduced from the logging of the solution process is that for the teams where no stand-by drivers are used in the solution, the solutions are found more quickly due to the scoreboard mechanism: Even though the stand-by drivers are not chosen in the final configuration, these driver-agents publish a value on the scoreboard early on in the process, aiding in the elimination of many more costly solutions at an early stage.

The results of Run 8 show that using all driver-agents (full population) does not improve the solution This can be explained by the fact that many of the train drivers have finished their duties already at the moment the disruption occurs, so that they cannot help anymore. The calculation times show that the system is capable of running with a full population. In section 1 we explained that the average time needed to manually reschedule a single affected duty by a dispatcher is 5 minutes. The results show that the system is capable of rescheduling 11 duties within a few minutes, which is a big improvement.

5.2 Scenario 2: partial blockage at Lombardijen

For scenario 2 the number of cancelled train services due to the blockage is 38, which lead to 28 affected driver duties. In this scenario, the route is not completely blocked and 2 of the 4 tracks remain available for operating trains. Table 4 shows the results for the second scenario of various runs with different cost values. These experiments show the sensitivity of the system to the applied cost function. Our analyses focus on the most important cost elements; the amount of overtime, the number of team-members, and the usage of stand-by drivers. For each run we used the same driver-agent population, 602 regular drivers and 49 stand-by drivers.

The results show that a solution for the scenario can be found in about one minute, a much shorter time than needed for manually solving the problem, which would take about 38 × 5 minutes.

Table 4 Results for scenario 2 (see text for explanation)

Run	Cost per per minute of overtime	Cost per team members	Cost per used stand-by driver	Overtime (minutes)	# team members	# stand-by # drivers in final solution	Calculation time
1	1	25	50	554	31	3	0:54
2	2	25	50	476	33	3	0:53
3	10	25	50	449	35	3	0:48
4	2	5	50	343	34	3	0:57
5	2	50	50	460	33	4	0:49
6	2	25	10	476	33	3	0:40
7	2	25	5	460	34	4	0:36
8	2	25	0	460	34	5	0:27

The first three runs show that when the penalty on overtime is increased, the amount of overtime in the solution is reduced and, as a trade-off, the number of changed duties (#team members) increases.

Run 3 and Run 4 show that when both the overtime costs and the costs per team member are decreased with the same ratio, we get a better result on both values. At first sight this might seems awkward, but the result can be explained by the relatively high costs of the other elements. The cost of affecting a meal break is 30. In Run 3, the system chooses to add a team member (costs 25) and affect its meal-break (30) with costs 55. This solution would cost 35 in Run 4. In Run 4 the system prefers to change an already affected duty so that it gets some overtime (16 minutes) with costs 32 which is cheaper than selecting the additional team member with costs 35. Choosing this solution in Run 3 would have costs 160, which is more expensive than the chosen solution with costs 55.

In Run 2 and Run 4, we show that, when the cost of an extra team member is increased, the number of team members decreases and, as a trade-off, the overtime increases. In Run 5 we increase the cost of an extra team member even further. This leads to the use of an extra stand-by driver and a decrease of the overtime. Analyses of the solution show that while solving the problem of the 20th team leader, the system chooses not to include an additional team member (as in Run 4) but to extend the duty of an already affected driver. This team member could help another team leader in Run 2, where in Run 4 an additional stand-by driver is needed because this driver is already helping the other team leader. This example shows the heuristic nature of the system.

Runs 2, 6, 7, and 8 show the effect of decreasing the cost per used stand-by driver. In Run 6 the same solution is found as in Run 2, only in less time. In this case, the score-board mechanism helps to find good solutions per team quickly. In Run 7, an additional stand-by driver is used and this results in a reduction of the total overtime. Setting the cost to zero in Run 8, leads to a solution with a second additional stand-by driver without any other improvement of the solution.

The results show that for this scenario, the settings of the cost function have a significant influence on the results. Further discussions with the dispatchers need to be held in order to find a default setting which leads to good solutions in the real-life instances.

6 Discussion

The train-driver rescheduling research system is a real-world application of an actor-agent system. Since summer 2007, about ten man years of research and development have been successfully invested. The performance of the current system includes the full set (1000+) of driver-agents. The current calculation times indicate that the research system easily outperforms a human dispatcher in processing a number of affected duties. A potential drawback of the system is that the solution is generated sequentially per team leader without a mechanism to get the overall best solution. The algorithm presented by Potthoff et al. (2010), has such an overall mechanism, but is limited by the selection of the drivers to be included in the problem instance.

The research system does not have this limitation and can potentially get a better solution per team leader. Current research focuses on a comparison of both algorithms.

One possible use of the research system is to study the combination of rescheduling properties of initial daily crew schedules given specific disruptions and rescheduling parameters. This may lead to new insights in the role and use of stand-by train-drivers as well as the amount of 'free time' required in the train driver duties.

As stated, validation of the prototype is conducted by means of a number of sessions in which the rescheduling solutions as well as the usability of the system are examined by experts of the dispatching centres. This resulted in calibration of the cost function and an improvement of the associated heuristics. In the end, the conclusion is that the research system provided better solutions than the ones generally created by hand.

A number of lessons learned can be distilled from our experiences in this project:

- *Co-development is crucial.* The close cooperation between NS and D-CIS Lab enabled the transfer of domain knowledge as well as actor-agent knowledge to the mutual benefit of both parties and the overall success of the research system.
- *Real-world data is difficult.* More effort than expected was invested in understanding the problem domain and distilling useful parts of real-world data (including incompleteness, inaccuracy and sometimes unavailability).
- *Re-design is not evil.* During the development of the research system progressive insight led to several re-designs of important parts of the driver-agents and the RAA. Although this effort seemed distracting, the overall research system's quality improved substantially.
- *Decentralization is not a silver bullet.* A careful balance has to be made between decentralizing task-exchange functionality from centralizing route analysis and calculation to boost performance without violating the actor-agent principles.
- *Multi-agent debugging tools are a necessity.* Message analyzers are not enough to debug a full fledged multi-agent system; an additional visualization tool was created to debug the (emergent) team-formation process.

The results of this project encourage us to continue our work on agent-based rescheduling in real-world domains. Future work entails using the research system as a platform for additional performance analysis and testing of techniques, cost-functions as well as agent platforms.

Acknowledgements The authors express their gratitude to NS for starting this project and to Cor Baars (DNV/Cibit) for his useful suggestions. The following D-CIS Lab colleagues provided valuable contributions: Hilbrandt van Boven, Pascal Hoetmer, Michel Oey, Reinier Timmer, Louis Oudhuis, Martijn Broos, Sorin Iacob, Thomas Hood, and Kees Nieuwenhuis. The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project (www.icis.decis.nl), supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024. The ICIS project is hosted by the D-CIS Lab (www.d-cis.nl), the open research partnership of Thales Nederland, the Delft University of Technology, the University of Amsterdam and the Netherlands Organisation for Applied Scientific Research (TNO).

References

Abbink EJW, Fischetti M, Kroon LG, Timmer G, Vromans MJCM (2005) Re-inventing crew scheduling at Netherlands railways. Interfaces 35(5):393–401

Abbink EJW, Mobach DGA, Fioole PJ, Kroon LG, Wijngaards NJE, van der Heijden EHT (2008) Actor-agent based approach to train driver rescheduling. In: Proceedings of the 20th Belgian-Dutch conference on artificial intelligence (BNAIC 2008), Bad Boekelo, pp 1–8

Abbink EJW, Mobach DGA, Fioole PJ, Kroon LG, van der Heijden EHT, Wijngaards NJE (2009) Actor-agent application for train driver rescheduling. In: Proceedings of the eighth international conference on autonomous agents and multiagent systems, Budapest, pp 513–520

Helsinger A, Thome M, Wright T (2004) Cougaar: A Scalable, Distributed multi-agent architecture. In: Proceedings of the international conference on systems, man and cybernetics, The Netherlands

Iacob SM, Nieuwenhuis CHM, Wijngaards NJE, Pavlin G, Van Veelen JB (2009) Actor-agent communities: design approaches. In: Papadopoulos, G.A, Badica, C. (eds) Intelligent distributed computing III, SCI, vol 237, pp 237–242

Jiang Z, Xie C (2009) Multi-agent delay simulation model in mass rail transit system. In: International conference on measuring technology and mechatronics automation, Icmtma, vol 3, pp 717–720

Kroon LG, Huisman D, Abbink EJW, Fioole PJ, Fischetti M, Maróti G, Schrijver A, Steenbeek A, Ybema R (2008) The new dutch timetable: the OR revolution. Interfaces 39(1):6–17

Mao X, ter Mors A, Roos N, Witteveen C (2007) Coordinating competitive agents in dynamic airport resource scheduling. In: Petta P, Mueller JP, Klusch M, Georgeff M (eds) Proceedings of the 5th German conference on multiagent system technologies, LNAI, vol 4687, pp 133–144. Springer, Berlin

Mobach DGA, Abbink EJW, Fioole PJ, Lentink RM, Kroon LG, van der Heijden EHT, Wijngaards NJE (2009) Train driver rescheduling using task-exchange teams. In: Proceedings of the second international workshop on: optimisation in multi-agent systems (OPTMAS) at AAMAS 2009

Potthoff D, Huisman D, Desaulniers G (2010) Column generation with dynamic duty selection for railway crew rescheduling. Transp Sci. doi:10.1287/trsc.1100.0322

Rezanova NJ, Ryan DM (2010) The train driver recovery problem—a set partitioning based model and solution method. Comput Oper Res 37(5), 845–856

Shibghatullah AS, Eldabi T, Rzevski G (2006) A framework for crew scheduling management system using multi-agents system. In: 28th international conference on information technology interfaces (ITI 2006), Cavtat, Croatia

Tranvouez E, Ferrarini A (2006) Multiagent modelling of cooperative disruption management in supply chains. In: International conference on service systems and service management, Troyes, pp 853–858

Veelenturf LP, Potthoff D, Huisman D, Kroon LG (2009) Railway crew rescheduling with retiming. Report EI2009-24, Econometric Institute, Erasmus University Rotterdam, 24 p (Forthcoming in: Transp Res Part C)

Walker CG, Snowdon JN, Ryan DM (2005) Simultaneous disruption recovery of a train timetable and crew roster in real-time. Comput Oper Res 32(8):2077–2094