A heuristic for real-time crew rescheduling during small disruptions

Thijs Verhaegh[1] Dennis Huisman[2,3] Pieter-Jan Fioole[3]
Juan C. Vera[1]

[1] Department of Econometrics and Operations Research, Tilburg University,
P.O. Box 90153, 5000 LE Tilburg, The Netherlands

[2] Econometric Institute & ECOPT, Erasmus University Rotterdam,
P.O. Box 1738, 3000 DR Rotterdam, The Netherlands

[3] Process quality and Innovation, Netherlands Railways,
P.O. Box 2025, 3500 HA, Utrecht, The Netherlands

thijsverhaegh@gmail.com, huisman@ese.eur.nl,
pieterjan.fioole@ns.nl,j.c.veralizcano@tilburguniversity.edu

Econometric Institute Report EI2016-09

Abstract

Due to unforeseen problems, disruptions occur at railway passenger operators. Proper real-time crew management is needed to prevent disruptions to spread over space and time. Netherlands railways (NS) has algorithmic support from a solver to obtain good crew rescheduling solutions during big disruptions. However, small disruptions are still manually solved by human dispatchers who have limited solving capacity. In this paper the rescheduling for crews during small disruptions is modeled as an iterative-deepening depth-first search in a tree, which is combined with several OR techniques, obtaining a heuristic method. The heuristic focuses on real-life usability and uses the updated rolling-stock schedule as input. Testing the heuristic on about 5,000 test instances shows that the heuristic delivers good and desirable rescheduling solutions within fraction of seconds, outperforming other well-known methods from the literature.

1 Introduction

Every day, Netherlands' largest railway passenger operator, Netherlands railways (NS), faces disruptions due to unforeseen problems with the infrastructure, rolling stock, weather conditions, accidents, etcetera. These disruptions might lead to conflicts in the crew schedules. Human dispatchers try to solve these conflicts by adapting the schedules. However, they have limited solving capacity. During busy days, the workload and pressure on dispatchers is enormous and in extreme cases this could lead to canceled trains or even cancelation of all train traffic, leading to unsatisfied customers and reputational damage. One of NS highest priorities is to improve their worst case performance; therefore NS is building on more algorithmic support for human dispatchers. For big disruptions such as blocked tracks, NS has algorithmic support from a solver developed by Potthoff (2010). However, smaller disruptions such as delays, canceled trains and detours are still manually solved by dispatchers. In this paper the rescheduling for crews in case of small disruptions is modeled as an iterative-deepening depth-first search in a tree, which is combined with several OR techniques, obtaining a heuristic method that finds good rescheduling solutions within fraction of seconds. The heuristic focuses on real-life usability and uses the updated rolling stock schedule as input. Some ideas from an actor-agent approach, as discussed in the literature by Abbink et al. (2010), are combined with other OR techniques.

2 Literature review

Although many papers are written about crew scheduling, Potthoff (2010) discussed that operations research models are hardly applied in practice to solve disruptions for passenger railway operators. Crew rescheduling during disruptions in airline transport has been investigated as is reviewed by Clausen et al. (2010). A crew rescheduling problem for train driver duties, which are disrupted due to maintenance work on the tracks is discussed by Huisman (2007). Walker, Snowdon and Ryan (2005) solve the train timetabling and crew rescheduling during disruptions at the same time. However, their method is not applicable to NS since it is assumed that the railway system is of a relatively simple structure, which is not the case at NS. Jespersen-Groth et al. (2009) wrote a paper about disruption management in passenger railway transportation. They discuss the structure of disruption problems in railway transportation, but they do not provide an explicit solution method to solve the rescheduling problem.

An approach for the real-time rescheduling of train drivers in Denmark is

discussed by Rezanova and Ryan (2010). They model the problem as a set covering constraint model, just as Potthoff (2010). Potthoff uses the modified rolling stock schedule and modified timetable as input for his Operational Crew Rescheduling Problem, in which the crew schedule is adapted in such a way that the modified timetable can be carried out. Since there can be millions of feasible adaptions to the crew schedule, Potthoff considers a core problem first to solve the set covering constraint model. He solves the core problem using a column generation based heuristic, which combines column generation with dynamic duty selection and Lagrangian Relaxation. This heuristic "can be seen as a depth-first search in a branch-and-bound tree with column generation in every node" (Potthoff, 2010, p.38). There might be some canceled tasks in the solution for the initial core problem. Potthoff tries to cover these tasks by exploring a neighborhood for each uncovered task. For an exact definition of the core problem, the heuristic and the neighborhood selection, refer to Potthoff (2010). Poffhoff's algorithm performs well during big disruptions, but the quality of the neighborhood choice is questionable in case of small disruptions.

Another approach for the real-time rescheduling of train drivers is an actor-agent approach as discussed by Abbink et al. (2010), in which train drivers and dispatchers are represented in a computer model by agents. The main idea behind their approach is that agents interact with each other by sending messages, in order to solve the problems they face. As Abbink et al. (2010, p.253) state, "the main principle underlying the actor-agent based rescheduling process is that of task-exchange", meaning that drivers who help each other often exchange some tasks. This is similar to the approach human dispatchers use to solve real-time scenarios.

Both methods from Potthoff (2010) and Abbink et al. (2010) outperform human dispatchers. Testing by experts of NS show that Potthoff's algorithm provides better results during big disruptions and therefore NS uses his solver nowadays. For small disruptions, still no algorithmic support is used. In this paper, ideas from the literature are combined with operations research techniques in order to present a method to solve small disruptions. Testing shows that, in the case of small disruptions, our proposed method outperforms the methods from Potthoff and Abbink et al.

3 Problem Formulation

The schedule of a single driver for a day is called a driver duty and consists of several tasks that the driver has to perform. An example of a task might be driving an intercity train from station s_{dep} to s_{arr} from time t_{dep} to t_{arr}.

Sometimes, a passenger task is included for reallocation reasons, meaning the driver uses the train as passenger but is not actively involved in driving the train. There are several rules a duty must satisfy to be feasible, including labor rules regarding working times. Examples are minimum times for check in and check out, maximum duty length, rules regarding breaks and minimum transfer times between consecutive tasks. A crew schedule consist of all duties and is considered to be feasible if all driver duties are feasible and all tasks that have to be performed on a day are divided among the crews.

Disruptions lead to infeasibilities in the duties. During big disruptions such as blocked tracks, it is not straightforward how to repair infeasible duties; NS uses Potthoff's algorithm to find feasible solutions. During small disruptions as delays, canceled trains, detours and staff absence, dispatchers at NS are usually able to repair infeasible duties into feasible ones by dropping tasks from some driver duties. This creates unplanned tasks, that is, driving tasks that need to be performed by one of the drivers but which have not been planned in any driver duty yet. Dispatchers have to plan these unplanned tasks into driver duties in order to prevent canceled trains.

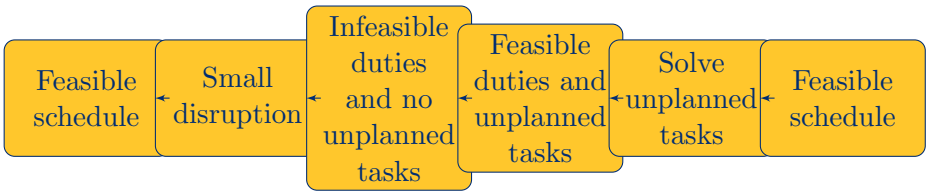

Figure 1: Flow chart of solving a small disruption

We assume that dispatchers are able to convert infeasible duties into feasible ones by creating unplanned tasks. Planning unplanned tasks in driver duties is the 'bottleneck' in the rescheduling process, since it is the most time-consuming part. Often, dispatchers are even not able to find feasible solutions without use of the so-called reserve drivers, who are standby drivers at railway stations. However, during small disruptions NS prefers not to use reserve drivers, as they are saved for larger disruptions which might occur later in time. Therefore in our solution method we **try to find solutions for unplanned tasks without the use of reserve drivers**. Since we are concerned with real-time rescheduling, it is important that rescheduling solutions are found in fractions of seconds.

To distinguish between different solutions an objective function is defined. The objective is chosen such that solutions are desired by both dispatchers as crews. Both parties desire to stick to the original schedule as much as possible and therefore a penalty is included for a changed duty. Drivers prefer

not having any overtime (especially during small disruptions), nor having a break at a different time or location than planned, and so penalties for used overtime and changed breaks are included. The weights of these penalties are carefully defined after consultations with crews, dispatchers and experts of NS.

4 Solution Methodology

When planning an unplanned task in a duty some new unplanned tasks might be created. These are iteratively solved in order to solve the original unplanned task. Inserting the unplanned task in a duty and iteratively solving new ones can be seen as a search in a tree. The root node of the tree is the original schedule, consisting of all feasible duties and a non-empty set of unplanned tasks M. An edge (s, s_i) indicates the insertion of the first unplanned task in M in duty i, where s_i denotes the new schedule after insertion. The root node has an outgoing edge for each driver in the population of train drivers, with exception of the reserve ones.

As we show in section 4.3, insertion of an unplanned task in a duty is sometimes **impossible**. In this case we consider the new schedule (node) **impossible**. This notation is somewhat imprecise, since there is not really a schedule at this node, but this notation is used to explain the search through the tree. In other situations insertion might be possible, either by creation of new unplanned tasks (insertion is **conditionally possible**), or without creation of new unplanned tasks (insertion is **unconditionally possible**). As long as $M \neq \emptyset$ at a certain node, and this node is also not impossible, we **branch further**. To avoid cycling, we do not allow the removal of a task from a duty once this task has been inserted in a duty.

In order to find the best solution we have to do a complete enumeration of the tree, which would take lots of time due to the size of the tree[1]. The emphasis of rescheduling is on obtaining the best possible solution within fraction of seconds of computation time. Therefore, only the most promising parts of the tree are searched and heuristics are used to improve the speed and quality of the search. The heuristic finds solutions which are not guaranteed to be optimal, but good enough for the given set of goals and outperforming human dispatchers. Several OR techniques are combined to find solutions in

[1] As we see later on, each node has over 1,000 children. On average 124 children survive the basic feasibility check. Inserting an unplanned task in a duty is conditionally possible in about 17% of the cases. This means we branch further at on average 21 children. If the depth of the tree is bounded to 7, this already leads to more than a billion leafs. However, the depth of the tree is unbounded.

the tree.

4.1 Basic feasibility check

Insertion of an unplanned task in a duty might not be possible at all. Therefore a **basic feasibility check** is developed to prune branches of the tree that would lead to impossibilities. By using the basic feasibility check computation time is saved. The check detects many (but not all) of the impossibilities in an early stage and consists of two parts: the license check and the time check. In the license check, we check whether the driver is allowed to take over the unplanned task regarding his route knowledge and rolling stock knowledge. If so, continue to the time check. If not, inserting the task in this duty is impossible.

The time check determines whether unplanned task possibly fit into the driver duty. Minimum travel times between each pair of locations and maximum overtime are used. For example, a driver duty from 8 a.m. to 4 p.m. is not able to take over a task from 10 p.m. to 11 p.m. Furthermore, a driver who starts his duty in Rotterdam at 8 a.m., is not able to take over a task from Groningen to Leeuwarden, starting at 8.15 a.m., since it takes at least two hours to reach Groningen from Rotterdam by the geographical location. These two examples do not pass the time check.

The output of the basic feasiblity check is a subset of driver duties who survive the basic feasibility check and remain possible candidates duties to take over the unplanned task. The basic feasibility check is performed on each driver duty each time an unplanned task is solved.

4.2 Priority system

After the basic feasibility check the unplanned task can still be inserted in many duties. Inserting a task in a duty is time consuming, so we try to insert the task in promising duties first. Therefore, we define a **priority system** which predicts how good the unplanned task would fit in each of these duties. The system uses prediction values based on linear regression, which are sorted using bucket sort. For example, drivers who are geographically close to the unplanned task or drivers who have lots of spare time in their duty obtain higher priority. The task is inserted in the most promising duty first, expecting to find a solution more quickly.

To obtain prediction values, we use exogenous variables indicating the occupation of the driver duty, whether the driver already had a break or not, whether the driver is (close) to the departure and/or arrival station or the

unplanned task, whether the driver will be able to take over the unplanned task without overtime and the end time of the duty.

The parameters for the prediction values are obtained by ordinary least squares regression on over half a million observations. Each observation passed the basic feasibility check and is an unique insertion of a task in a duty. Note that this not necessarily means that insertion is possible in each of these observations. Only insertions that are impossible for sure are detected by the basic feasibility check. If insertion is possible, the prediction value is equal to the corresponding value of the objective function plus a penalty for each new created unplanned task. When insertion is impossible, the prediction value is set to a big number. Hence, unconditionally possible insertions are better than conditionally possible insertions and have the best prediction values. Conditionally possible insertions with only one new unplanned task are better than conditionally possible insertions with 10 new unplanned tasks. Conditionally possible insertions are better than impossible insertions, which have the worst prediction values.

For each of the possible candidate duties the prediction value is calculated. The logical solution seems to sort the duties according to their prediction values. Sorting an array of n elements can be done in log-linear time $\mathcal{O}(n \log n)$ and bucket sorting of n elements can be done in linear time $\mathcal{O}(n)$ as showed by Shutler, Sim and Lim (2008). Since prediction values remain rough predictions about the solvability of the new schedule and the sort is done many times during the tree search, we decided to place the duties in buckets according to their prediction value and use the bucket sort to sort the possible candidates. Analysis turns out that it is indeed more efficient to put the prediction values into buckets.

We give duties who use the unplanned task as a passenger task a deviation by placing them in the highest bucket. These drivers are able to drive the unplanned task for sure, without the creation of any new unplanned tasks. Also, duties that are already changed on the path from the root node to the branch node are placed in a more promising bucket. These duties are preferred to be used, since we do not have to charge the costs of changing a duty in the objective function again. Driver duties in the bucket with the worst prediction values indicate that inserting the unplanned task in this duty would probably lead to impossibilities and are not considered as possible candidates duties anymore, meaning we **prune the unpromising parts of the tree**. In this way the solving time decreases significantly while hardly any feasible solution is lost.

Summarizing, the priority system calculates the prediction value for each of the candidate duties, puts the duties in buckets according to those prediction values and sorts them using bucket sort. Then, in the tree search the

unplanned task is inserted in the most promising duty first.

4.3 Method to insert a task in a duty

Since there are several possible outcomes when inserting a task in a duty, we develop a systematic **method to insert a task in a duty**. The outcome of this method is either: insertion is possible without creating any new unplanned tasks (unconditionally possible), insertion is possible by creating unplanned tasks (conditionally possible), or insertion is impossible.

The first step of the inserting method is to insert the unplanned task in the driver duty and make all tasks with overlap with the inserted task unplanned. If the minimum transfer time between two consecutive tasks and is violated, this is considered as overlap as well. As soon as one of the new created unplanned tasks starts before the time of rescheduling, report that insertion is impossible. Unplanned positioning tasks do not have to be rescheduled and are canceled. By inserting the original unplanned task and possibly creating new ones, some location conflicts might be created. A location conflict is generalized to the situation where the driver has to be at a location s_0 at a time t_0 and at a location s_1 at time $t_1 > t_0$, where $s_0 \neq s_1$, while no such trip is included in his duty.

The second step of the inserting method is solving possible location conflicts. We try to solve location conflicts by inserting positioning trips. These trips are not allowed to start before the time of rescheduling. First search for a positioning trip consisting of a single task. If this task does not exist, search for a trip consisting of two tasks. To limit computation time positioning trips are restricted to at most two tasks. Outcomes of positioning trip searches are saved in a register. Whenever such a computation is required again, results from the register are used, saving computation time. When looking for positioning trips two cases can happen: A positioning trip exists or it does not. If it exists, the task(s) of the positioning trip are included in the driver duty and the driver uses these task(s) as a passenger. If there are multiple positioning trips available, take the one starting first in time. If a positioning trip does not exist, we iteratively take an extra task out of the duty, make it unplanned and see whether we still have a location conflict. If so, we check whether we can find positioning trips now. If it turns out that a location conflict cannot be solved, the method reports insertion is impossible.

The last part of the inserting method checks whether the meal-break of the driver is affected. If so, try to plan a new meal-break in the duty, satisfying the labor rules. Furthermore, the break may not start before the time of rescheduling. If a new meal-break is needed but no break possibilities occur in the duty, report that inserting the unplanned task in the duty is

impossible. Improvements might be achieved by creating extra unplanned tasks in order to find a break possibility, but this is left for future work.

Summarizing, inserting an unplanned task in a duty can be impossible, conditionally possible or uncondionally possible. We illustrate the insertion method with the example in figure 2. We insert unplanned task from Ams-

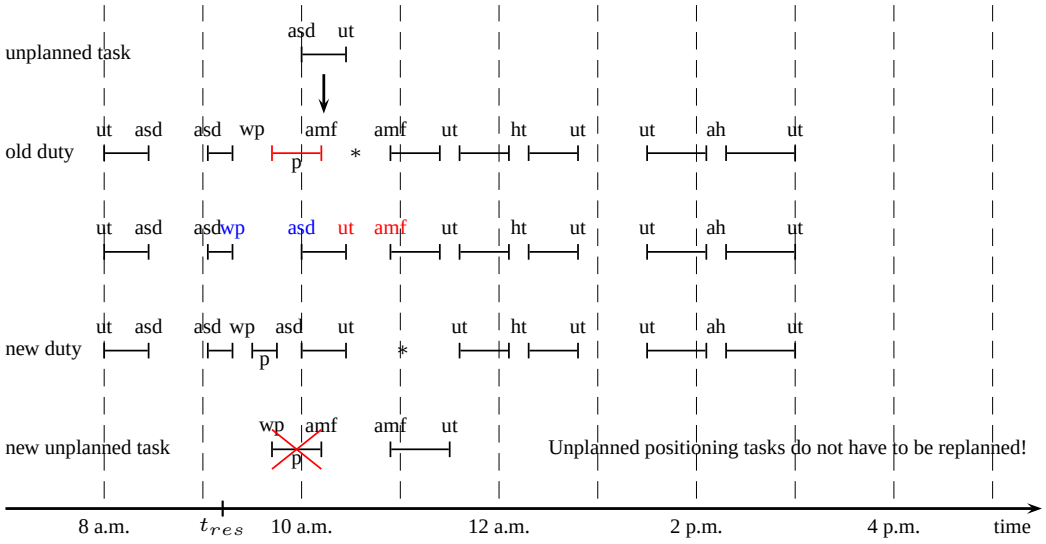

Figure 2: Conditionally possible insertion of unplanned task asd-ut

terdam (asd) to Utrecht (ut) in a duty. It is a duty starting in ut at 8 a.m., driving to asd, Weesp (wp), and then using a train as a passenger to Amersfoort (amf) where the driver has a break around 10.30 a.m. After the lunch the driver has tasks from amf to ut, back and forth to 's Hertogenbosch (ht) and back and forth to Arnhem (ah) after which the driver finishes his duty at ut at 3 p.m. Inserting the unplanned task creates overlap with the positioning task $wp - amf$, which is removed the duty and made unplanned. This task does not have to be rescheduled since it is a passenger task. Two location conflicts are created. For location conflict $wp - asd$ around 9 a.m. we find a positioning trip and the location conflict is solved. For location conflict $ut - amf$ no positioning trip exists, task $amf - ut$ is removed from the duty and hence considered as new unplanned task. The location conflict is solved. The new break is in ut and satisfies the labor rules. Insertion of $asd - ut$ is conditionally possible by creation of new unplanned task $amf - ut$.

4.4 Restricting the growth of the tree

As explained, rescheduling can be seen as a tree search. We defined the order in which we consider the children and how to insert a task in a duty. Much time is needed to search the complete tree, but large parts of the tree have undesired solutions. We want to prune these parts of the tree to save computation time.

As we search deeper in the tree the number of changed duties does not decrease. Solutions in some branches of the tree are of poor quality since they require too many changed duties. Therefore, the **number of allowed changed duties**, $maxChangedDuties$, is bounded. Iteratively this bound is increased. The maximum on this bound is set to 5, since solutions with more changed duties are not desired by NS. If more than 5 duties are affected to solve an unplanned task, other solutions are preferred, such as the use of a reserve driver. Changing duties is charged by the objective function. When $maxChangedDuties$ is equal to a low value large parts of the tree are pruned, but also solutions with low costs are found quickly. Therefore we start the tree search that we with $maxChangedDuties$ equal to 1 and then iteratively increase this number when the tree search is completed.

Moreover, inserting a task in a duty might be conditionally possible by creating one or more new unplanned tasks. It is more likely that we end up in a feasible schedule if we create only one new unplanned task, then by creating 10 new unplanned tasks. If we create 10 new unplanned tasks, the problem is much larger than before and we probably are not able to solve all these new unplanned tasks (especially not since $maxChangedDuties$ is at most 5). Furthermore, it takes computation time to solve the new unplanned tasks. So, insertions which lead to a small number of new unplanned tasks are expected to give better solutions in the subtree below. Therefore introduce the threshold $maxNewUnplannedTasks$, **bounding the number of new unplanned tasks** that is created by inserting a task in a duty. Stop branching at conditionally feasible nodes when this bound is exceeded. We start the tree search with $maxNewUnplannedTasks$ equal to 1 and then iteratively increase this threshold. In this way the most promising part of the tree is searched first.

4.5 Fathoming

The objective function is non-decreasing as we search deeper in the tree. Therefore, at each node lower bounds for the solutions are defined. Some parts of the tree only have solutions at very (undesired) high costs. Therefore, we prune parts of the tree where the lower bound is above a certain upper

bound. As soon as a solution is found, we set the cost of this solution as upper bound. We fathom parts of the tree with lower bounds higher than the upper bound (**fathoming** idea from branch and bound). This idea works well when a solution of good quality is found quickly, because large parts of the tree can be pruned. Furthermore, we define a threshold for the maximum value for the lower bound. Possible feasible solutions with costs higher than this value are undesired by NS and therefore parts of the tree with lower bounds above this threshold are pruned.

4.6 Depth-first iterative-deepening

We develop a **depth-first iterative-deepening** (DFID) search in a tree, in which the depth is bounded by the number of changed duties. DFID combines breadth-first searchs completeness and depth-first search space efficiency, which is optimal when the path cost is a non-decreasing function of the depth of the node (Korf 1985).

DFID heuristic to solve unplanned tasks

Set $maxTime = 2$ seconds;
for $maxNewMancos = 1, maxNewMancos \leq 10, maxNewMancos++$ **do**
  for $maxChangedDuties = 1, maxChangedDuties \leq 5, maxChangedDuties++$ **do**
    Depth First Search($maxNewMancos$, $maxChangedDuties$, $maxTime$);
  end
end
Report the best feasible schedule that is found, if any

To save time and memory, the tree is represented implicitly. In every moment a single node is maintained. When moving from a node to a child, the differences between the two nodes are saved (as edge information), since loading the data for a node is more expensive than constructing by performing the corresponding changes. The tree search stops as soon as no further explorations occur or two seconds of solving time have elapsed.

5 Test Instances

The daily schedule of Thursday June 14, 2012 is used as input data. This day was an average work day at NS with over 10,000 tasks planned in about

1,000 feasible duties. There were no unplanned tasks at the beginning of the day. A test instance is created by copying one of the 10,000 tasks and adding this task as unplanned task. We try to solve this unplanned task, meaning that we look for solutions with only feasible duties and no unplanned tasks. This procedure is repeated for each task to create 10,000 test instances. Half of them are used for fine-tuning the heuristic and obtaining the parameters for the prediction values, the other half are used to test the heuristic. For each instance, we assume it is known that the given task is unplanned only 45 minutes in advance, that is, changes are not allowed to any of the duties more than 45 minutes earlier than the starting of the unplanned task.

6 Results

The heuristic has been implemented in C++ on an Intel Core i7 processor with 2.96 GB RAM clocked at 2.80 GHz. An overview of the main results include the following items:

- Within two seconds, the heuristic finds solutions in over 78% of the instances. (See figure 3.)

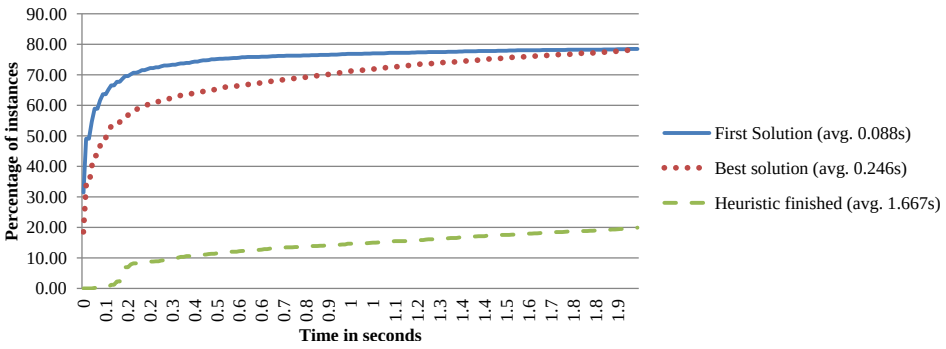

Figure 3: Even within one fifth of a second over 70% of the instances are solved

- The average time to find the best solution (among the ones found within 2 seconds) is only 0.246 seconds, while the first solution was found in on average 0.088 seconds, i.e. good solutions are found quickly.

- The best solution changes on average two duties, uses seven minutes of overtime and changes half a break.

- If we run the heuristic longer (up to a half a minute), the solution capacity tend to converge to 83%, while the average cost of the solutions increases with 3.5%.
- Initial analysis shows that the heuristic outperforms the methods from Potthoff (2010) and Abbink et al. (2010), in both calculation times as solution quality.
- Rescheduling is more difficult around 2 p.m. and during the nights. (See figure 4.) At NS, many duties start in the early morning and end around 2 p.m., and many new duties start around 2 p.m. Drivers have to return home around 2 p.m. so around this time there are less rescheduling possibilities. Without using overtime still 64% of the unplanned tasks can be solved. More solutions can be found when the allowed overtime increases, but also extra costs are charged for those solutions. In figure 4 the solvability over the different time slots of the day is plotted using allowed overtime of 0, 30 and 60 minutes. The purple dotted line indicates the total number of test instances in each time slot.

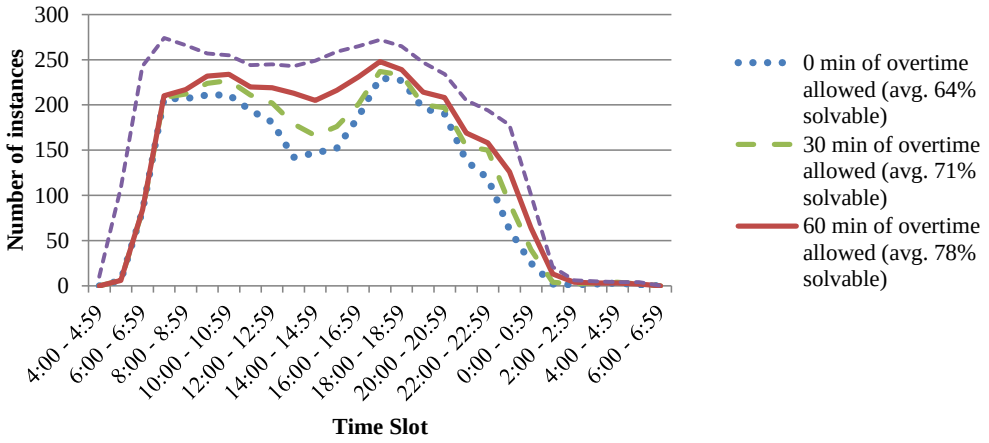

Figure 4: Without using overtime it is harder to solve unplanned tasks

- If we perform depth first search without iterative deepening, i.e. fixing $maxChangedDuties$ to 5 and $maxNewUnplannedTasks$ to 10, within two seconds we find solutions in 33.7% of the instances. The average time needed to find a solution increases with over 400% and the cost of the best solutions almost doubles.
- Unplanned tasks which last longer, are harder to reschedule. (See figure 5).

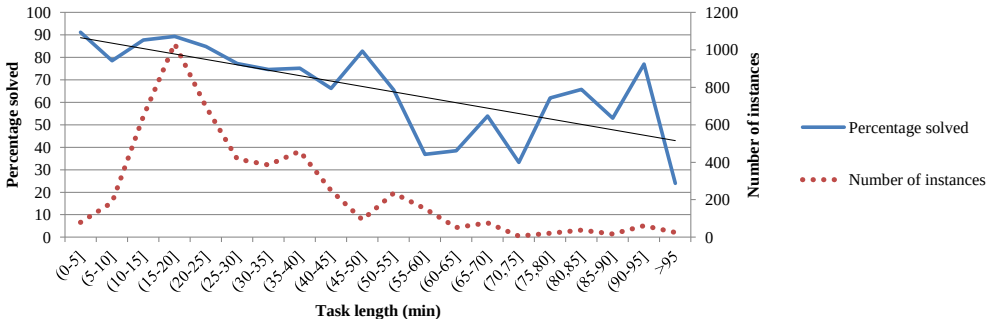

Figure 5: Rescheduling is more difficult for longer tasks

- Rescheduling of tasks in the Randstad[2], the most occupied part of the network, is easier. (See figure 6.)

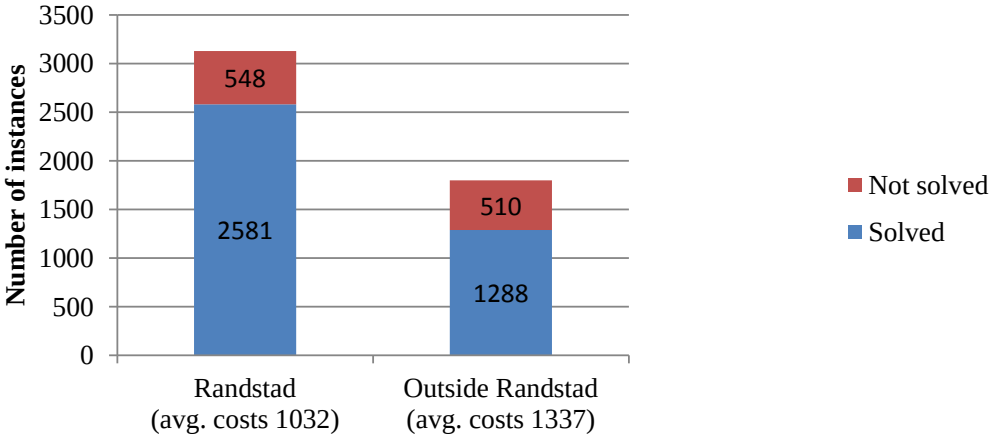

Figure 6: Rescheduling is more difficult outside the Randstad

- If the working rules regarding the minimum break length during rescheduling change, this influences the rescheduling. Increasing the minimum break length to 40 minutes, drops the solution quality and increases the time needed to find rescheduling solutions. (See figure 7.)

- So far, we supposed we know an 45 minutes in advance that a task is unplanned. In reality, the time in advance fluctuates a lot. Sickness

[2]Randstad is a conurbation in the Netherlands, consisting of the four largest cities and its surroundings. We include the following cities with all its train stations: Almere, Amersfoort, Amsterdam, Alphen aan den Rijn, Barendrecht, Capelle aan den IJssel, Delft, Dordrecht, Gouda, Haarlem, Hilversum, Hoofddorp, Hoek van Holland, Houten, Katwijk, Leiden, Maassluis, Purmerend, Rijswijk, Rotterdam, Schiedam, The Hague, Utrecht, Vlaardingen, Voorburg, Zeist, Zoetermeer.

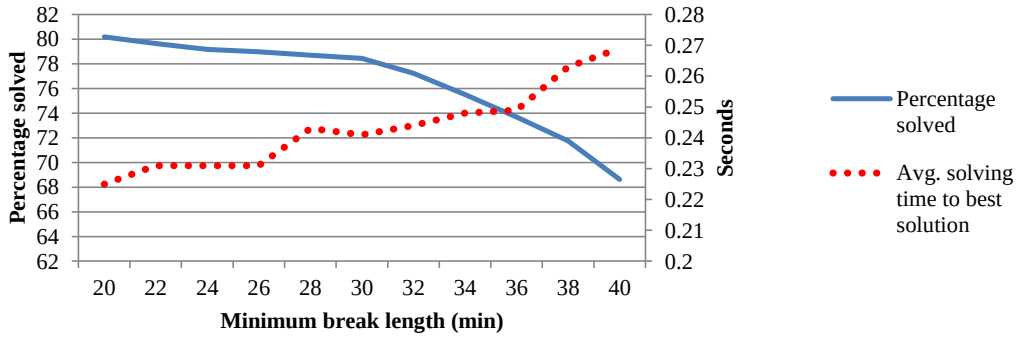

Figure 7: Changing the minimum break length to 20 minutes has minor effects on rescheduling

notices cause that hours before the task starts we might know the task is unplanned, while delays might occur at the latest moment and up to 0 minutes before the task start. Consider figure 8 in which we vary the time in advance from 0 to 95 minutes. Even if we need a driver instantaneously, we have solutions in over 45% of the instances.

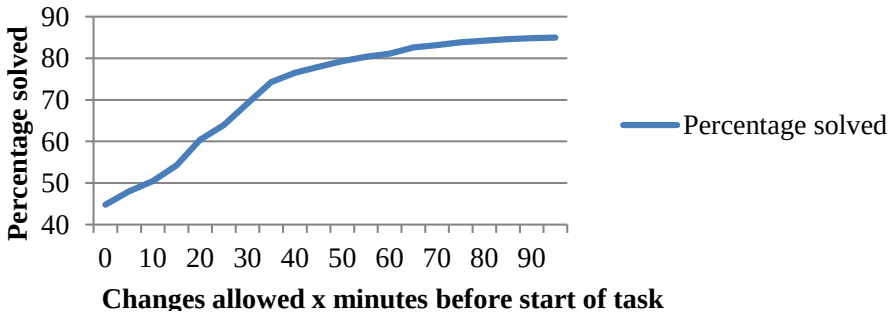

Figure 8: Even if a driver is needed instantaneously, 45% of the instances are solved

7 Conclusions and future work

Concluding, the heuristic works fast, delivers good and desirable solutions and outperforms other well-known methods from the literature. This paper focuses on train drivers only and a similar approach might be developed for train guards. The process from small disruption to unplanned tasks might be modelled and the solution method might be improved. Furthermore, this solution method might be useful to improve the solver for larger disruptions.

It might be interesting to test the solution method in case of larger disruptions. Last but not least it might be interesting to consider the rolling stock rescheduling and crew rescheduling at once.

References

[1] Abbink EJW, Mobach DGA, Fioole PJ, Kroon LG, Van der Heijden EHT, Wijngaards NJE (2010) Real-time train driver rescheduling by actor-agent techniques. Public Transp. doi 10.1007/s12469-010-0033-6

[2] Clausen J, Larsen A, Larsen J, Rezanova NJ (2010) Disruption management in the airline industry - concepts, models and methods. Computers & Operations Research 37: 809-821

[3] Huisman (2007) A column generation approach for the rail crew rescheduling problem. Eur. J. of Operations Research 180: 163-173

[4] Jespersen-Groth J, Potthoff D, Clausen J, Huisman D, Kroon L, Marti G, Nielsen MN (2009) Robust and Online Large-Scale Optimization. Springer Berlin Heidelberg

[5] Korf RE (1985) Depth-first iterative-deepening: An optimal admissible tree search. Artif. Intell. doi 10.1.1.91.288

[6] Potthoff D (2010) Railway Crew Rescheduling: Novel Approaches and Extensions. PhD thesis, Erasmus University Rotterdam

[7] Rezanova NJ, Ryan DM (2010) The train driver recovery problem - a set partitioning based model and solution method. Computers & Operations Research 37: 845-856

[8] Shutler PME, Sim SW, Lim WYS (2008) Analysis of linear time sorting algorithms. The Computer J. 51-4: 451-469

[9] Walker CG, Snowdon JN, Ryan DM (2005) Simultaneous disruption recovery of a train table and crew roster in real time. Computers & Operations Research 32: 2077-2094