View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Erasmus University Digital Repository

A Quasi-Robust Optimization Approach for Resource Rescheduling

Lucas P. Veelenturf[1], Daniel Potthoff[3], Dennis Huisman[2,4], Leo G. Kroon[1,4], Gábor Maróti[4] and Albert P.M. Wagelmans[2]

[1] Rotterdam School of Management and ECOPT,
[2] Econometric Institute and ECOPT,
Erasmus University Rotterdam
P.O.Box 1738, 3000 DR, Rotterdam, The Netherlands

[3] Ab Ovo Germany
Prinzenallee 1, 40549, Düsseldorf, Germany

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

E-mail: {lveelenturf,lkroon}@rsm.nl, {huisman, wagelmans}@ese.eur.nl
daniel.potthoff@ab-ovo.com, gabor.maroti@ns.nl,

Econometric Institute Report EI2013-28

Abstract

If a disruption takes place in a complex task-based system, where tasks are carried out by a number of resource units or servers, real-time disruption management usually has to deal with an uncertain duration of the disruption. In this paper we present a novel approach for rescheduling such systems, thereby taking into account the uncertain duration of the disruption. We assume that several possibilities for the duration of the disruption are given.

We solve the rescheduling problem as a two-stage optimization problem. In the first stage, at the start of the disruption, we reschedule the plan based on the optimistic scenario for the duration of the disruption, while taking into account the possibility that another scenario will be realized. In fact, we require a prescribed number of the rescheduled resource duties to be *recoverable*. This means that they can be easily recovered if it turns out that another scenario than the optimistic one is realized.

We demonstrate the effectiveness of our approach by an application in real-time railway crew rescheduling. This is an important subproblem in the disruption management process of a railway company with a lot of uncertainty about the duration of a disruption. We test our approach on a number of instances of Netherlands Railways

(NS), the main operator of passenger trains in the Netherlands. The numerical experiments show that the approach indeed finds schedules which are easier to adjust if it turns out that another scenario than the optimistic one is realized.

1 Introduction

Transportation and production systems sometimes have to deal with disruptions. Due to a disruption, the underlying plans for the allocation of tasks to resource units or servers have to be rescheduled. Indeed, some tasks may be cancelled by the disruption, which makes the original plans infeasible. Examples are the rescheduling of vehicles and crews in various transportation systems (bus, rail and air), or the rescheduling of machines in a production system. This paper focuses on dealing with large-scale disruptions, often caused by machine failures, weather conditions or the temporary unavailability of the underlying infrastructure.

In such rescheduling problems, uncertainty in the duration of the disruption usually plays a major role. For example, recovery works on a broken switch in a railway network may take two hours in the optimistic scenario. However, they may stretch up to four hours in the pessimistic scenario. During the recovery works, the railway traffic is usually interrupted, leading to cancelled tasks for rolling stock and crews. It is unclear at the start of the disruption how many tasks will be cancelled.

A common approach to tackle this uncertainty is to modify the schedule at the start of the disruption, given an estimated duration of the disruption. Usually the shortest possible or optimistic duration is taken as an initial estimate for the duration of the disruption. Later, when it turns out that the disruption lasts longer than estimated initially, the schedule is modified again (and possibly again). This approach is also called a wait-and-see approach.

In this paper we present a new approach to deal with this uncertainty about the duration of the disruption. In particular, we study real-time disruption management for task-based scheduling systems *under uncertainty*. In a task-based scheduling system, a number of timetabled *tasks* have to be carried out by a given number of resource units or servers. Each task has a fixed start and end time and a given start and end location. A *duty* is an ordered sequence of tasks that have been assigned to a single server.

Existing algorithmic frameworks for dealing with uncertainty include the classical approaches of *robust optimization* and *stochastic programming*:

- *Robust optimization* tries to find the best solution that, without any modifications or recovery actions, remains feasible under all specified scenarios. For more information about robust optimization we refer to Bertsimas and Sim (2003) and Ben-Tal and Nemirovski (2002).

- Two-stage *stochastic programming* with *recourse* minimizes the sum of the costs of the first stage solution plus the expected costs of the recovery in the second stage. An important assumption is that the probability for the occurrence of each of the considered scenarios is known a priori. For more information about stochastic programming we refer to Birge and Louveaux (1997) and Kall and Wallace (1994).

Both robust optimization and stochastic programming lead to significantly more complex optimization problems than the underlying deterministic problems. Usually, realistic instances cannot be solved in (near) real-time. In addition, robust optimization is a very conservative approach, while stochastic programming needs information about a probability distribution for the occurrence of the different scenarios, which is usually not available in practice.

In this paper we propose a *quasi-robust rescheduling approach*, which is built upon the recently introduced concept of *recoverable robustness* by Liebchen et al. (2009). The main idea is to compute a good schedule for the optimistic scenario in such a way that it can easily be turned into a feasible schedule when another scenario than the optimistic one is realized. This is achieved by requiring that a given number of the duties that are rescheduled in the first stage have an alternative for the tasks for which it is not certain at the start of the disruption that they must be carried out. Thus, also if such tasks turn out to be cancelled in the second stage, such a duty can be made feasible again. Duties with this property are called *recoverable*.

Recoverability of a duty is a local property of a duty that depends on the duty itself, and not on the entire solution. It is therefore rather easy to incorporate it in column generation based algorithms without substantially raising their running time. Furthermore, the approach admits to balance the robustness and the operational costs by requiring a *given* number of the rescheduled duties to be recoverable.

Our method consists of a two-stage approach. In the first stage we assume that the optimistic scenario takes place, and we compute the modified schedule subject to a constraint that a given number of rescheduled duties must be recoverable. Then, in the second stage, when it turns out that another scenario than the optimistic scenario is realized, we compute the rescheduled duties from scratch. That is, we do not limit the recovery action in the second stage to simply falling back to the recovery alternatives of the recoverable duties. Computing the rescheduled duties from scratch in the second stage will be necessary when not all duties are recoverable after the first stage. Moreover, rescheduling from scratch in the second stage also helps to reduce the second stage costs.

The primary criterion for assessing the quality of a schedule is the number of additionally cancelled tasks (i.e. the ones that are cancelled in addition to the ones that are cancelled

due to the disruption), both in the first stage and in the second stage. The two-stage evaluation framework allows us to analyze how the robustness requirements in the first stage influence the actual rescheduling performance in the pessimistic scenario or in any other scenario. We also compare our approach with a typical rolling horizon approach where initially only the optimistic duration of the disruption is taken into account, and where the duties are rescheduled whenever the information about the duration of the disruption is updated.

We demonstrate the effectiveness of our approach by the results of the computational tests that were carried out on a number of railway crew rescheduling instances of Netherlands Railways (NS), the main operator of passenger trains in the Netherlands. Summarizing, the contributions of this paper are as follows.

- We consider disruption management of task-based scheduling systems under uncertainty.
- We develop a framework to deal with the uncertainty about the duration of a disruption.
- We evaluate our approach on a number of railway crew rescheduling instances of Netherlands Railways.

We want to emphasize that our focus lies both on developing new methods and on practical applications. We consider real-time disruption management of substantially complex scheduling systems. In fact, our computational tests are based on railway crew rescheduling instances that are quite challenging, even without taking into account the uncertainty in the duration of the disruption.

This paper is organized as follows. In Section 2, we give a description of our quasi-robust rescheduling approach and of the uncertainty that has to be dealt with. We also give a formal definition of the concept of quasi-robustness. In Section 3, we explain the application of our approach on the rescheduling of railway crews during a disruption. Section 4 presents our computational results based on instances of NS. This paper is concluded in Section 5 with suggestions for further research.

2 Quasi-robust optimization approach

2.1 Rescheduling problems

Many scheduling problems can be seen as a set of timetabled *tasks* which must be carried out by a number of servers. In the scheduling problems that we consider, each task has a

fixed start and end time and a given start and end location. A sequence of tasks to be carried out by a single server is called a *duty*. If a task is carried out by a certain server, we say that the task is *covered* by that server.

If a disruption occurs, usually a number of tasks must be cancelled. As a consequence, some of the original duties of the servers become infeasible and must be rescheduled. In such a rescheduling problem (RSP), the duties must be modified such that as many of the remaining tasks as possible are covered by a server and such that the modifications in the duties are minimal.

In this section we assume that the disruption starts at time τ_1 and that the duration of the disruption is known. Thus the set of remaining tasks that still have to be carried out is known at time τ_1. At time τ_1 every server that has not yet finished its duty must get a new feasible sequence of tasks to end its duty. Moreover, every server that has not yet started its duty must get a new feasible sequence of tasks to replace its duty, preferably in the same time interval as the original duty. In both cases, this is the *completion* of a duty. Furthermore, we use the following notation:

- T: The set of tasks that have not started yet at the time of rescheduling, and that, given the duration of the disruption, are still to be carried out.
- Δ: The set of unfinished servers.
- K^δ: The set of all feasible completions for server $\delta \in \Delta$. For every feasible completion $k \in K^\delta$ we have:
  - c_k^δ: The cost of completion k for server δ. The cost of a completion is zero if the original duty of the server is not modified.
  - a_{ik}^δ: A binary parameter indicating if task i is covered by completion k for server δ.
- f_i: The cost for not covering a task i.

Given these definitions, we can formulate such an RSP at time τ_1 with a given duration of the disruption as a Mixed Integer Program (MIP). This MIP is an adapted version of a Set Covering model. The model uses binary variables x_k^δ to represent whether or not completion k is selected for server δ, and binary variables z_i to indicate if task i is covered or not. The model can be formulated as follows:

$$(RSP): \min \quad \sum_{\delta \in \Delta} \sum_{k \in K^\delta} c_k^\delta x_k^\delta + \sum_{i \in T} f_i z_i \tag{1}$$

$$\text{s.t.} \quad \sum_{\delta \in \Delta} \sum_{k \in K^\delta} a_{ik}^\delta x_k^\delta + z_i \geq 1 \qquad \forall i \in T \tag{2}$$

$$\sum_{k \in K^\delta} x_k^\delta = 1 \qquad \forall \delta \in \Delta \tag{3}$$

$$x_k^\delta, z_i \in \{0,1\} \qquad \forall \delta \in \Delta, \forall k \in K^\delta, \forall i \in T \tag{4}$$

Here the objective function (1) describes the aim of minimizing the sum of the costs of the completions and the costs of not covering certain tasks. Constraints (2) make sure that every task is either covered by a completion or is not covered. Furthermore, constraints (3) ensure that every server is assigned to exactly one completion. Constraints (4) describe the binary character of the decision variables.

2.2 Rescheduling under uncertainty

In contrast with the assumption in Section 2.1, the duration of the disruption is usually not known at time τ_1, the start time of the disruption. In this paper we deal with this uncertainty in the duration of the disruption by considering the rescheduling problem as a two-stage optimization problem.

The first stage rescheduling is carried out at time τ_1. At time τ_1 an optimistic estimate $\underline{\tau}$ and a pessimistic estimate $\bar{\tau}$ of the end time of the disruption are assumed to be known. Later, at time τ_2, with $\tau_1 < \tau_2 < \underline{\tau}$, the actual end time of the disruption τ becomes known with $\underline{\tau} \leq \tau \leq \bar{\tau}$. Time τ_2 is the time at which the second stage rescheduling is carried out.

Since at time τ_1 we do not know the actual duration of the disruption, we consider a finite set of scenarios S with index s. The optimistic scenario $\underline{s}$ and the pessimistic scenario $\bar{s}$ correspond with the optimistic and the pessimistic end time of the disruption, respectively. In the optimistic scenario the set of tasks $T_{\underline{s}}$ must be carried out, and in the pessimistic scenario the smaller set of tasks $T_{\bar{s}}$ must be carried out. The set of tasks $T_{\underline{s}} \setminus T_{\bar{s}}$ is called the set of *critical* tasks C.

All other scenarios are obtained by cancelling a number of critical tasks from the set C. In general, the set of tasks that must be carried out in scenario s is denoted by T_s. We assume that the scenarios have been ordered in such a way that $T_{s_2} \subset T_{s_1}$ if in scenario s_1 the disruption ends earlier than in scenario s_2.

Then the rescheduling problem under uncertainty can be stated as follows. Given the

set of possible scenarios S, find at time τ_1 a new schedule valid for the optimistic scenario $\underline{s}$ such that the sum of the costs of this schedule and the worst case costs for the additional rescheduling in the second stage at time τ_2 is minimized. Note that, since we do not assume knowledge of a probability distribution of the scenarios, we cannot minimize the expected costs. The objective function that we use in our model will be explained in more detail in Section 2.3.

2.3 Definitions

In this section we give a definition of the concept of q-quasi-robustness, but first we give an informal description. The idea behind q-quasi-robust optimization is to generate for q servers completions that are, in some sense, robust against all possible scenarios, i.e. they can be carried out whatever scenario is actually realized. In this way we minimize the costs (in particular the number of cancelled tasks) for rescheduling in the second stage at time τ_2 if a scenario other than the optimistic scenario $\underline{s}$ is realized. Note that we do not require *all* completions to be robust. That is why we call our approach *quasi-robust* rather than robust.

Now the formal definition of q-quasi-robustness follows in three steps in Definitions 1, 2 and 3.

Definition 1. *If k is a completion for server δ that is feasible in the first stage RSP, then a completion γ_s for server δ that is feasible in the second stage RSP when scenario s is realized is said to be a* ***recovery alternative*** *for completion k in scenario s if $a^{\delta}_{i\gamma_s}$ = 1 for each task $i \in T_s$ with $a^{\delta}_{ik} = 1$.*

In other words, the completion γ_s of server δ is a recovery alternative for completion k of server δ in scenario s, if each task $i \in T_s$ that is covered by completion k is also covered by completion γ_s. Informally speaking, this means that the critical tasks that are covered by completion k and that are cancelled in scenario s can be circumvented in some way if scenario s is realized, so that all non-critical tasks in k are still carried out in γ_s. Usually, this requires the completion k to contain a certain amount of idle time around (i.e. before and/or after) a critical task.

Now a recoverable completion is defined as follows:

Definition 2. *A completion k of server δ that is feasible in the first stage RSP for the optimistic scenario $\underline{s}$ is called* ***recoverable*** *if*

- *k does not contain two critical tasks directly after each other, and*
- *there exists a recovery alternative γ_s for completion k in all scenarios $s \in S$.*

Note that by Definition 2 every completion that does not contain any critical task is recoverable. Note further that a recoverable completion may contain more than one critical task, but only if there is at least one non-critical task between each pair of critical tasks.

Definition 3. *A schedule that is obtained in the first-stage rescheduling phase is called* ***q-quasi robust*** *if $q > 0$ servers have a recoverable completion.*

Based on the above Definitions 1, 2 and 3, we next describe the q-quasi-robust rescheduling problem (q-QRSP) that is to be solved in the first-stage rescheduling phase. We denote the set of recoverable completions for server δ by $R^\delta \subset K^\delta$. Furthermore, for each server δ and each feasible completion k of δ, the binary decision variable x_k^δ describes whether or not completion $k \in K^\delta$ is selected in the solution. For each task $i \in T_{\underline{s}}$, the binary variable z_i describes whether or not task i is covered. Now we can state q-QRSP in the first stage as follows:

$$\min \quad \sum_{\delta\in\Delta}\sum_{k\in K^\delta} c_k^\delta x_k^\delta + \sum_{i\in T_{\underline{s}}} f_i z_i \tag{5}$$

$$\text{s.t.} \quad \sum_{\delta\in\Delta}\sum_{k\in K^\delta} a_{ik}^\delta x_k^\delta + z_i \geq 1 \qquad \forall i \in T_{\underline{s}} \tag{6}$$

$$\sum_{k\in K^\delta} x_k^\delta = 1 \qquad \forall \delta \in \Delta \tag{7}$$

$$\sum_{\delta\in\Delta}\sum_{k\in R^\delta} x_k^\delta \geq q \tag{8}$$

$$x_k^\delta, z_i \in \{0,1\} \qquad \forall \delta \in \Delta, \forall k \in R^\delta, \forall i \in T_{\underline{s}} \tag{9}$$

The objective function (5) describes that the aim is to minimize the sum of the costs of the selected completions and the costs of leaving certain tasks uncovered. Constraints (6) specify that each task $i \in T_{\underline{s}}$ must be covered by a completion or remains uncovered. Constraints (7) describe that each server must get a completion. Constraints (8) determine that at least q servers must get a recoverable completion. Finally, constraints (9) require the decision variables to be binary valued.

Note that the model (5)–(9) for q-QRSP in the first stage is almost the same as the model (1)–(4) for RSP. The difference is that in (5)–(9) we require at least q servers to have a recoverable completion. Note that if $q = |\Delta|$, then all servers must have a recoverable completion. That implies that in each scenario a feasible solution can be obtained by using for each server the corresponding recovery alternative.

If another scenario is realized than the optimistic one, then in the second stage RSP, where we assume the remaining duration of the disruption to be known, we solve the RSP,

given the solution that was obtained for q-QRSP in the first stage. We aim at minimizing the total costs of the first stage and the second stage by varying q, the number of servers required to get a recoverable completion in the first stage.

2.4 Solution approach

The solution approach for q-QRSP in the first stage consists of a combination of Lagrangian relaxation and column generation, which is based on Caprara et al. (1999), Huisman et al. (2005) and Potthoff et al. (2010). If the problem contains many tasks, then servers can have a huge number of feasible completions. Therefore we use a column generation approach where only promising completions are considered.

2.4.1 Lagrangian relaxation

In the master problem for q-QRSP in the first stage, Constraints (6) are relaxed which results in the following Lagrangian subproblem:

$$\Theta(\lambda) = \min \quad \sum_{\delta\in\Delta}\sum_{k\in K^\delta} c_k^\delta x_k^\delta + \sum_{i\in T_{\underline{s}}} f_i z_i \quad + \sum_{i\in T_{\underline{s}}} \lambda_i (1 - \sum_{\delta\in\Delta}\sum_{k\in K^\delta} a_{ik}^\delta x_k^\delta - z_i) \tag{10}$$

$$\text{s.t.} \quad \sum_{k\in K^\delta} x_k^\delta = 1 \qquad \forall \delta \in \Delta \tag{11}$$

$$\sum_{\delta\in\Delta}\sum_{k\in R^\delta} x_k^\delta \geq q \tag{12}$$

$$x_k^\delta, z_i \in \{0,1\} \qquad \forall \delta \in \Delta, \forall k \in R^\delta, \forall i \in T_{\underline{s}}, \tag{13}$$

The latter can be rewritten as:

$$\Theta(\lambda) = \min \quad \sum_{i\in T_{\underline{s}}} \lambda_i + \sum_{\delta\in\Delta}\sum_{k\in K^\delta} (c_k^\delta - \sum_{i\in T_{\underline{s}}} \lambda_i a_{ik}^\delta) x_k^\delta \quad + \sum_{i\in T_{\underline{s}}} (f_i - \lambda_i) z_i$$

$$\text{s.t.} \quad (11) - (13)$$

For given Lagrange multipliers λ_i, the Lagrangian subproblem can be solved in the following way. First, $z_i = 1$ if $f_i - \lambda_i < 0$, and $z_i = 0$ otherwise. To determine the x_k^δ values, we have to choose at least q servers which must have a recoverable completion. This can be accomplished in the following way.

First, for notational reasons, we replace $c_k^\delta - \sum_{i\in T_{\bar{s}}} \lambda_i a_{ik}^\delta$ by $\bar{c}_k^\delta$. Here for every server δ

we have $k^\delta = \arg\min\{\bar{c}_k^\delta | k \in K^\delta \backslash R^\delta\}$ and $r^\delta = \arg\min\{\bar{c}_k^\delta | k \in R^\delta\}$. Thus k^δ represents the completion with the lowest reduced costs of all non recoverable completions for server δ, and r^δ represents the completion with the lowest reduced costs of all recoverable completions for server δ. For every server either of these completions must be selected. Based on this information we can determine the optimal values for the x_k^δ variables as follows:

1. Set all x_k^δ variables equal to 0.
2. Compute for every server δ the difference between the reduced costs of the two completions $c^{\delta,*} = \bar{c}_{r^\delta}^\delta - \bar{c}_{k^\delta}^\delta$.
3. For the q servers δ with the lowest values of $c^{\delta,*}$, set $x_{r^\delta}^\delta = 1$.
4. For all remaining servers, set $x_{r^\delta}^\delta = 1$ if $c^{\delta,*} \leq 0$. Otherwise set $x_{k^\delta}^\delta = 1$.

Now the Lagrangian dual problem is to find:

$$\Theta^* = \max_{\lambda \geq 0} \Theta(\lambda) \tag{14}$$

2.4.2 Restricted master problem

Since we use column generation, we consider a restricted master problem (RMP) of (10)-(13) containing only a subset of the x_k^δ variables. Note that we implemented this such that this subset always contains at least one recoverable completion for every server (and not necessarily a non-recoverable completion). For example, a recoverable completion for a server that is feasible in each scenario is to end its duty without carrying out any further tasks.

In the n^{th} column generation iteration the x_k^δ variables in the RMP are given by $\cup_{\delta \in \Delta}\{x_k^\delta : k \in K_n^\delta\}$, where $K_n^\delta \subseteq K^\delta$ is a subset of completions for server δ, $R_n^\delta \subseteq K_n^\delta$ is a subset of recoverable completions for server δ, and $R_n^\delta \neq \emptyset$.

Let Θ_n^* be the optimal value of the Lagrangian subproblem of the n^{th} column generation iteration. For every RMP we use subgradient optimization to approximate Θ_n^*. We call this approximation A_n^*. This leads to the following relation: $A_n^* \leq \Theta_n^*$. Let λ_n^* be the corresponding multiplier vector. We solve a pricing problem for every server $\delta \in \Delta$ to check if A_n^* is a good approximation of Θ^*. Otherwise we need to add more completions to the RMP in order to improve the solution. For the details of this procedure we refer to Potthoff et al. (2010).

The pricing problems are modeled as shortest path problems with resource constraints (SPPRC) in dedicated graphs. These graphs are introduced in Section 2.4.3. Let $u_n^\delta = \min\{\bar{c}_k^\delta(\lambda_n^*) : k \in R_n^\delta\}$ and $v_n^\delta = \min\{\bar{c}_k^\delta(\lambda_n^*) : k \in K_n^\delta\}$ be the smallest Lagrangian

reduced costs of the already generated recoverable and general completions, respectively. Furthermore, let $s_n^\delta = \min\{\bar{c}_k^\delta(\lambda_n^*) : k \in R^\delta\}$ and $t_n^\delta = \min\{\bar{c}_k^\delta(\lambda_n^*) : k \in K^\delta\}$ be the optimal values of the recoverable and the general pricing problem for server δ, respectively. Then the completions corresponding to u_n^δ and v_n^δ should be added to the RMP if $s_n^\delta - u_n^\delta < 0$ and $t_n^\delta - v_n^\delta < 0$.

2.4.3 The pricing problem

We deal with rescheduling problems where a set of tasks that are fixed in time must be performed by a number of servers. In the same way as described by Potthoff et al. (2010), the pricing problem for every server δ can be modeled as a shortest path problem with resource constraints (SPPRC) in the pricing problem graph. The resources are required to handle certain additional properties of the completions. For example, in a crew scheduling application the time before and after a meal break may not be too long. This kind of constraints can be handled by considering the problem as SPPRC.

In a pricing problem graph, a node represents the start or the end of a task. Arcs are used to represent the tasks and to indicate which tasks can follow each other. The latter depends on the scheduled time but also on other characteristics such as the start and end locations of a task.

Example 4. *Figure 1 shows an example of a pricing problem graph involving tasks g, h, i, j, l, m, and n. The bold arcs correspond to the tasks. The thin arcs indicate transfers from one task to another.*

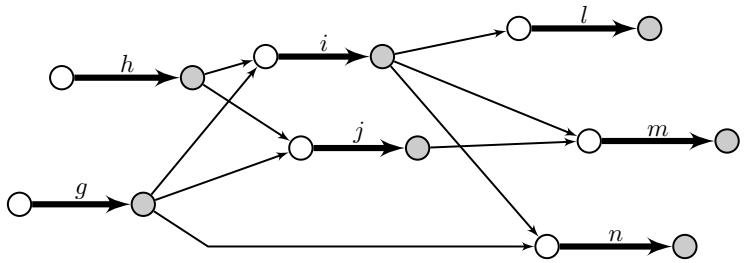

Figure 1: A pricing problem graph involving tasks g, h, i, j, l, m, and n.

By construction of the pricing problem graph, any feasible completion corresponds to a path in a pricing problem graph. Note that the reverse is not true in general since sometimes complex rules, e.g. about meal breaks, have to be taken into account. These rules are handled by the resource constraints.

A task i is said to be covered directly after task h in a completion if task h is followed directly by task i in the corresponding path. In this case, task i is a successor of task h, and task h is a predecessor of task i. We denote all predecessors of task i by *pred*(i), and all successors of task h by *suc*(h).

Finding recoverable completions The regular completions for a server can be generated based on regular pricing problem graphs as shown in Figure 1.

However, for generating *recoverable* completions, we have to modify the pricing problem graphs in order to guarantee the existence of a recovery alternative for completions containing critical tasks. In other words, when constructing a completion, we have to guarantee that for each critical task in the completion also an alternative path is available that can be used in case the critical task is cancelled. Here we use the following lemma.

Lemma 5. *If a feasible completion k for server δ is recoverable, then for every critical task i in completion k there exists a path, consisting of non-critical tasks only, from the end node of the non-critical predecessor of task i to the start node of the non-critical successor of task i.*

Proof. If completion k is recoverable, then it does not contain two critical tasks directly after each other. Thus each critical task i in k has a non-critical predecessor and a non-critical successor. Now the implication follows from the fact that completion k should have a recovery alternative in the pessimistic scenario $\bar{s}$, in which all critical tasks have been cancelled. In this pessimistic scenario the non-critical predecessor of task i and the non-critical successor of task i still have to be carried out, but task i is cancelled. Thus there exists a path as indicated. □

In Lemma 5, if a critical task is the first task of completion k, then the end node of its predecessor is meant to be a dummy node representing the start of the duty. Similarly, if a critical task is the last task of completion k, then the start node of its successor is meant to be a dummy node representing the end of the duty.

Note that the relation in Lemma 5 is not an equivalence, since the existence of a path does not imply the existence of a feasible completion, as was noted earlier.

Lemma 5 suggests that the generation of the recoverable completions can be accomplished by (i) modifying the pricing problem graph, and (ii) considering additional resources in the SPPRC. As was indicated before, these additional resources are required to handle certain rules that must be satisfied by the completions, such as the time before or after the meal break in a crew duty.

For this purpose we introduce additional arc properties next to the costs. These arc properties are used to define so-called Resource Extension Functions. For each scenario, a separate Resource Extension Function is needed to check whether a generated completion is feasible in the corresponding scenario. For the details of the Resource Extension Functions, we refer to Potthoff (2010).

Finding alternative paths We have to find out for each critical task i whether there exists an alternative path in the pricing problem graph consisting of non-critical tasks only that can be used to replace critical task i if this task is cancelled.

To that end, we remove all arcs corresponding to the critical tasks from the graph. In the reduced graph we can use a shortest path algorithm to determine for every *non-critical* predecessor task h of critical task i all *non-critical* successor tasks of critical task i that can be reached from task h via a path consisting of non-critical arcs only. In case of a SPPRC we are not interested in reachability alone, but also in information about *how* a successor node can be reached. To be more precise, we would like to find the path from the non-critical predecessor task h to the non-critical successor task j which uses the fewest resources.

Modifying the pricing problem graph Next we go back to the original pricing problem graph, and we modify it as follows. Let task i be a critical task. Then the arc corresponding to this task is removed from the graph. For each *non-critical* predecessor task of critical task i and each *non-critical* successor task of critical task i that can be reached from the predecessor task via a path consisting of non-critical arcs only (as described in the previous paragraph), we replace the removed arc corresponding to task i by a copy of the removed arc corresponding to task i. This procedure is illustrated in Example 6.

Note that we consider here only non-critical predecessors and non-critical successors in order to avoid the occurrence of two critical tasks directly after each other in a completion. We also copy the original transfer arcs to and from the copy of the critical task i, and we set the resource consumptions accordingly. These steps are carried out for each critical task.

Example 6. *Figure 2 shows the modified pricing problem graph that is obtained when the preprocessing steps are applied to the pricing problem graph shown in Figure 1. Here task i is a critical task that is carried out in scenario $\underline{s}$ and cancelled in scenario $\overline{s}$. Task i has two predecessors,* $\text{pred}(i) = \{h, g\}$*, and three successors,* $\text{suc}(i) = \{l, m, n\}$*. From task h, only task m can be reached via task j in the auxiliary problem when critical task i has been removed. For this relation we introduce a new task i' and the necessary arcs. From task g, tasks m and n can be reached in the auxiliary problem, which is represented by the copies*

i'' and i''' of critical task i. Note that task l cannot be reached from any predecessor of task i. Therefore, task l cannot be covered by any recoverable completion.

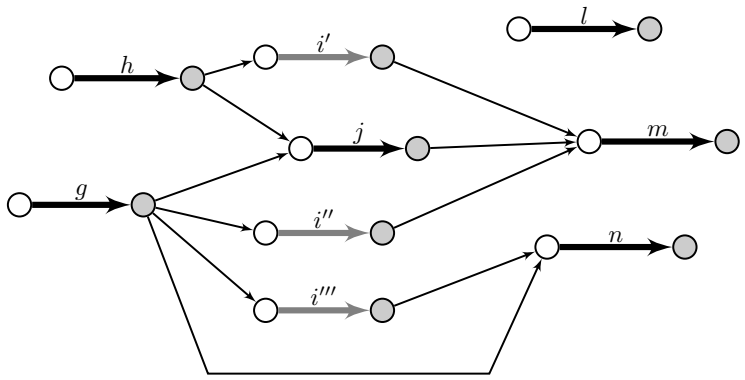

Figure 2: Part of a pricing problem graph *after* the preprocessing for critical task i that is cancelled in scenario $\overline{s}$ has been carried out.

Lemma 7. *A feasible completion k that is obtained as a resource constrained shortest path in the graph constructed according to the above described procedure is recoverable.*

Proof. If the feasible completion k does not contain any critical task, then it is recoverable by definition.

If completion k contains exactly one critical task i, then task i is not preceded nor succeeded in k directly by another critical task. Thus the alternative path for critical task i that was determined in the described procedure fits between the non-critical predecessor of task i in k to the non-critical successor of task i in k.

If completion k contains more than one critical task, then, due to the construction of the graph, two critical tasks in k do not follow each other directly in k. As a consequence, their alternative paths do not interact with each other: they are separated from each other in time by at least one non-critical task.

The foregoing cases imply that, if task i is a critical task, then in any scenario not containing task i, this task can be replaced by its alternative path. And, obviously, in any scenario containing task i, this task can be carried out as planned.

Finally, the Resource Extension Functions per scenario are used to handle the resource consumptions in each scenario. As a consequence, completion k is a feasible completion in the first stage under the optimistic scenario, and if any other scenario than the optimistic one is realized, then the corresponding recovery alternative for completion k is a feasible completion in the second stage. □

It is clear from the above example that the number of nodes and arcs in the pricing problem graphs increase significantly if many successors of the arrival node of a critical task can be reached from many predecessors of the departure node of the critical task. This has consequences for applying the concept of recoverability on instances of practical relevance.

Note that the number of copies of critical tasks can be reduced slightly by merging copies of tasks that have the same set of successors or the same set of predecessors. For example, if copies i'' and i''' in Figure 2 are merged since these tasks have the same predecessor, then the distinguishing characteristics of tasks i'' and i''' can be moved to the two arcs between the newly merged task i'' and tasks m and n. Alternatively, if copies i' and i'' are merged since these have the same successor, then the distinguishing characteristics of tasks i' and i'' can be moved to the two arcs between tasks h and g and the newly merged task i'. However, this usually does not lead to a significant reduction of the size of the pricing problem graph.

3 Application: Railway crew rescheduling

In this section we describe the application of the concept of q-QRSP to the real-time rescheduling of railway train drivers due to a disruption with an uncertain duration.

The operation of a railway system is usually based on an extensive planning process, resulting in a timetable and schedules for the rolling stock and crews. The interested reader can find further details about the underlying planning problems in Abbink et al. (2005) and Kroon et al. (2009).

In the regular timetable a number of tasks have to be carried out by the train drivers. Each task corresponds to driving a train or to deadheading on a train from one station to another. Thus each task has a departure time and station, and an arrival time and station. A duty is a set of consecutive tasks to be carried out by a single driver on a single day. Each driver belongs to a crew base, where all his or her duties should start and end.

In an ideal situation, the timetable and the schedules for rolling stock and crews are executed exactly as planned. However, in a railway system disturbances and disruptions happen frequently. For example, malfunctioning infrastructure or rolling stock or an accident may block the railway traffic at a certain location and during a certain time period. As a consequence, the timetable and the schedules for rolling stock and crews cannot be executed as planned: they have to be rescheduled.

Therefore, effective disruption management is key to a good operational performance of a train operating company. We refer to Jespersen-Groth et al. (2009) for a detailed description of the disruption management process. Within the disruption management process,

the ability to reschedule the crews is particularly crucial, since the crew duties are subject to complex rules and regulations. In the following, the problem of rescheduling the crew duties in a disrupted situation is called the Operational Crew Rescheduling Problem (OCRSP).

Potthoff et al. (2010) proposed an approach for solving OCRSP. However, this approach assumes that an accurate estimate of the duration of the disruption is available at the time the rescheduling is carried out. The same holds for the approach of Rezanova and Ryan (2010) and for the models developed for crew rescheduling in the airline industry. We refer to Clausen et al. (2010) for an overview of crew rescheduling models in the airline industry.

3.1 Examples

Example 8. *This example deals with a case in the north of the Netherlands that is represented in Figure 3. Due to broken catenary, no railway traffic is possible between Hoogeveen (Hgv) and Beilen (Bl) from 7:10 on. It is estimated that the repair works will last between 3 and 4 hours. The timetable is updated according to a pattern described by the contingency plan that is applicable in this situation.*

In this case, the trains of the train lines 500, 700, and 9100, that are operated between Zwolle (Zl) and Groningen (Gn) (and vice versa) in an hourly periodic timetable, are turned in four intermediate stations. In particular, the intercity trains of the 500 and 700 lines are turned in Hoogeveen and Assen (Asn). The regional trains of the 9100 train line are turned in Meppel (Mp) and Beilen. The corresponding trips between Hoogeveen and Assen (and vice versa) and between Meppel and Beilen (and vice versa) are cancelled.

At the intermediate stations where the trains are turned, the crew is supposed to stay with the turning trains. This means effectively that tasks from Groningen to Zwolle are changed into tasks from Groningen to Groningen, and that tasks from Zwolle to Groningen are changed into tasks from Zwolle to Zwolle. Such combined tasks are also called rerouted *tasks. The rerouted tasks are indicated later with "/r" after their corresponding train number. Note that the rerouted tasks fit perfectly well within the framework that has been developed so far.*

Figure 3 shows how the timetable between Zwolle and Groningen is updated. Since the repair works take at least 3 hours, the turning pattern is applied for sure for three southbound and three northbound trains of each of the three involved train lines. For the trains in the fourth hour after the start of the disruption, it is uncertain whether the trains will take their normal routes (dashed lines in Figure 3) or whether they will be turned as well (dotted arcs in Figure 3).

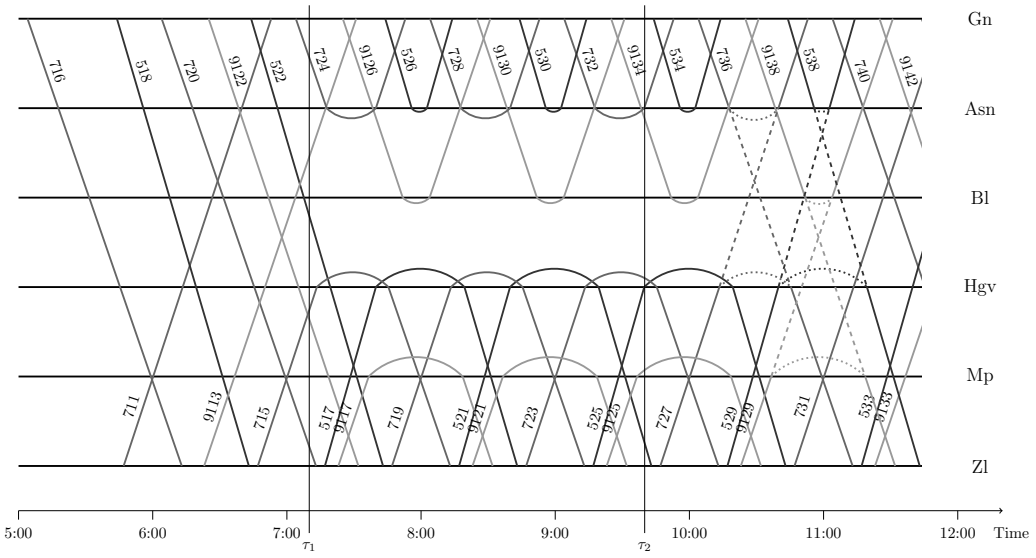

Figure 3: Time space diagram showing how the timetable between Groningen (Gn) and Zwolle (Zl) is updated, if the route between Beilen (Bl) and Hoogeveen (Hgv) is blocked temporarily.

Current crew rescheduling approaches deal with this situation as follows. At time τ_1 the optimistic duration of the disruption is estimated, and the modified timetable corresponding to this estimate is used as input for OCRSP.

In Example 8 this means that it is estimated that the blockage will be over by 10:10. Therefore, the modified timetable that is given as input to OCRSP assumes that the trains 727, 736, 529, 538, 9129 and 9138 can run between Beilen and Hoogeveen as planned. Therefore the corresponding tasks that are indicated with dashed lines in Figure 3 are considered in the instance of OCRSP.

However, it may happen that at time τ_2, which is 9:40 in the example, new information becomes available saying that the route will be blocked until 11:10. This means that the timetable has to be updated again and that the trains 727, 736, 529, 538, 9129 and 9138 must also be turned at the intermediate stations. Thus at time τ_2 the rolling stock and crew schedules must be rescheduled as well, given this new information and the rescheduled timetable. The rerouted tasks 727/r, 736/r, 529/r, 538/r, 9129/r, and 9138/r are used as input for a second instance of OCRSP. Here, for example, task 727/r consists of a task from Zwolle to Hoogeveen in the time slot of the original task 727, followed by a return task from Hoogeveen to Zwolle in the time slot of the original task 736, see Figure 3. Since the two consecutive parts of task 727/r must be carried out by the same crew, the two parts together are considered as one single rerouted task.

Given the route blockage between Hoogeveen and Beilen and the scenarios that were presented in Example 8, Example 9 gives a number of examples of feasible completions to illustrate the concept of recovery alternatives and recoverability.

Example 9. *Figure 4a shows a planned duty from crew base Groningen (Gn). Due to the route blockage between Hoogeveen and Beilen, the task 724 from Groningen to Zwolle (Zl) is rerouted and returns to Groningen. Therefore, the driver cannot follow his planned duty. A feasible completion of the duty under the optimistic scenario $\underline{s}$ is shown in Figure 4b. The optimistic scenario $\underline{s}$ assumes that the route blockage lasts until 10:10. Since this completion does not cover any critical task, it is a recoverable completion. The completion in Figure 4c is not recoverable. It covers critical task 736 from Groningen to Zwolle. If the pessimistic scenario $\overline{s}$ is realized, which means that the route is blocked until 11:10, then this task is rerouted (736/r) and ends in Groningen. Then the driver is not able to get to Zwolle in time to deadhead on task 538 from Zl to Amersfoort (Amf). Figure 4d shows a recoverable feasible completion covering this critical task. Its recovery alternative that is valid in the pessimistic scenario $\overline{s}$ is shown in Figure 4e. In this recovery alternative, task 9142 is in fact a deadheading task, since, according to the definition of a recoverable duty, this task is also covered by another feasible completion.*

3.2 Solution approach for crew rescheduling

Potthoff et al. (2010) describe a heuristic approach for solving OCRSP, assuming that the duration of the disruption is known in advance. Their solution approach uses dynamic column generation, Lagrangian lower bounds, and a greedy heuristic for constructing feasible solutions. Since the running times should be low, preferably within a few minutes, they do not aim at rescheduling *all* duties, but only a subset of them. This subset is updated as long as there are still tasks uncovered. The initial subset consists of the duties that are directly affected by the disruption (these duties *must* be rescheduled), but also of a set of heuristically chosen additional duties that may help to solve the instance of OCRSP in a better way.

Our approach selects the subset of duties in the same manner as Potthoff et al. (2010). However, we do not update this subset in the same way as was described above. We adapt the master and pricing problem to handle the recoverability. Especially in the pricing problems we have to be aware that we are dealing with an SPPRC in which we have to ensure that meal break rules are not violated.

In the first stage, our approach ensures for crews that need a recoverable completion that there is a recovery alternative if another scenario than the optimistic scenario is realized. From all feasible recovery alternatives, this approach picks the one which consumes

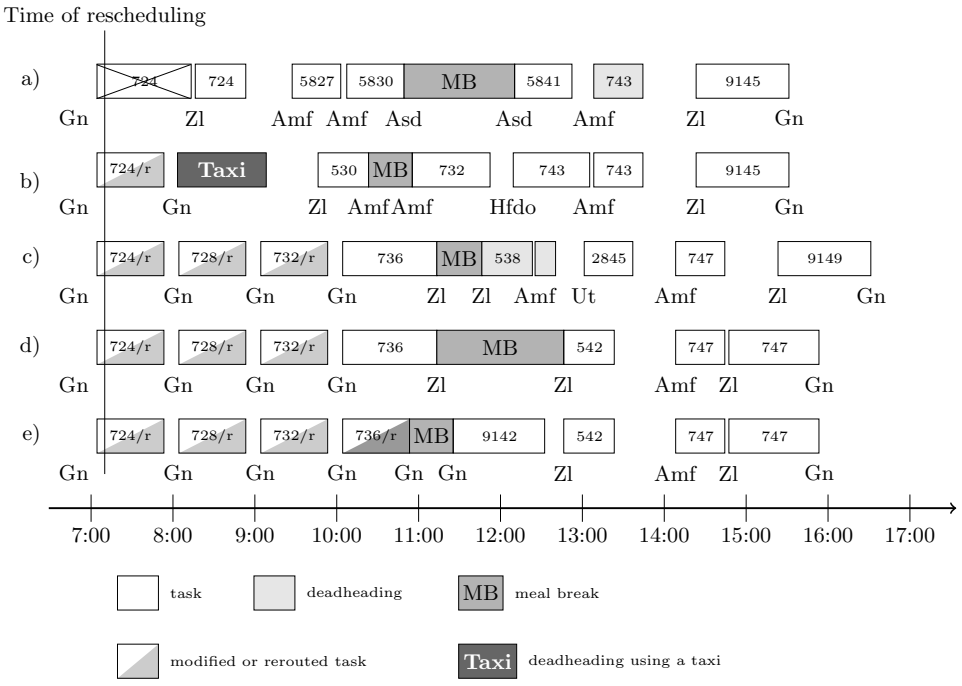

Figure 4: Examples of feasible completions for an affected original duty from crew base Groningen (Gn).

the fewest resources. Such an alternative does not have to be the cheapest alternative. Therefore we reschedule again in the second stage to search for the cheapest alternative. For rescheduling in the second stage we use the method of Potthoff et al. (2010) to solve OCRSP, since we assume that the duration of the disruption is known at that time. Note that we can only skip the rescheduling in the second stage if in the first stage *all* crews have a recoverable completion, but even then rescheduling can reduce the second stage costs.

4 Computational results

In this section we report our computational results for the q-quasi-robust optimization approach for crew rescheduling under uncertainty. We test the method on realistic crew scheduling cases of NS.

In order to explore the balance between robustness and nominal costs, every case is solved multiple times. We start with the instance where no completion is required to be recoverable, which corresponds with the wait-and-see approach. Then we gradually increase the number of completions that must be recoverable, until we finally reach the point where

all completions are required to be recoverable.

Since we are using a heuristic to solve the problem it may happen that we do not find an optimal solution for an instance. Due to this phenomenon, it may happen that a solution of the first stage where q_1 crews need a recoverable completion has lower costs than a solution where q_2 crews need a recoverable completion, even though $q_1 > q_2$. In such a case we use in our results the solution of q_1 for all values of q with $q_2 \leq q \leq q_1$.

It is worthwhile to compare the two extreme cases. If no crew needs a recoverable completion, we just solve the underlying OCRSP for the optimistic scenario without any robustness requirements. This may lead to high rescheduling costs in the second stage. On the other hand, if the entire schedule is required to be quasi-robust and if we find a solution that covers all tasks, then in principle no further rescheduling steps are necessary no matter which scenario is realized.

The q-quasi-robust optimization approaches have been implemented in C++ and compiled with the Visual C++ 10.0 compiler.

4.1 Cases

For our computational study, we used five large-scale disruptions that actually took place in the past in the Netherlands. On a regular working day about 10,000 tasks are carried out by about 1,000 duties, about 90 of which are reserve duties. In all five cases a route becomes suddenly unavailable for 2 to 3 hours due to a disruption.

As a preparation to the crew rescheduling step, we modify the timetable according to the procedures currently used by NS. In particular, the timetable services on the disrupted line are immediately cancelled. Rolling stock rescheduling is a challenging problem in itself, see Nielsen (2011). Therefore we consider here a simplified rolling stock schedule. The original (undisrupted) duties of the crews are taken from the operational schedules of NS on a workday in September 2007.

Location	ID	Time	Type	# crews	
				affected	in core
Abcoude	Ac_A	16:30-18:30	Two-sided blockage	67	142
Abcoude	Ac_B	16:30-18:30	Two-sided blockage	59	116
Beilen	Bl_A	07:00-10:00	Two-sided blockage	15	42
Beilen	Bl_B	16:00-19:00	Two-sided blockage	15	39
's-Hertogenbosch	Ht	08:00-11:00	Two-sided blockage	55	98

Table 1: Summary of the different cases.

A brief description of the five cases is given in Table 1. This table considers the opti-

mistic scenario that the disruption ends after the minimum possible duration.

The cases around Abcoude involve a disruption of the centrally located and heavily utilized route between Utrecht and Amsterdam. The network does allow rerouting possibilities for passengers and crews, although these are time-consuming. Around 60 drivers are directly affected by the disruption in these cases. The two cases around Beilen show a disruption on a less heavily used route, but the blockage cuts off the northern part of the network from the rest. The cases around Beilen have a direct effect on 15 drivers. The case with a disruption around 's Hertogenbosch has a big impact, since this also involves a heavily utilized route. In total 55 drivers are directly affected by this disruption.

For each of the 5 cases we consider two scenarios: we define the optimistic scenario $\underline{s}$ and the pessimistic scenario $\overline{s}$ as the scenarios corresponding to the shortest and the longest duration of the disruption, respectively. For the q-quasi-robust optimization approach, especially the number of critical tasks is important. Table 2 shows the main characteristics of the five cases. For every case we present the optimistic duration of the disruption, and the time the disruption lasts longer in the pessimistic scenario $\overline{s}$. We also show the number of critical tasks and the number of rerouted tasks. The rerouted tasks were explained in Examples 8 and 9.

	Optimistic	Considered	Critical tasks		
Case	Duration	Extension	Canceled in $\overline{s}$	Rerouted in $\overline{s}$	Total
Ac_A	2:00	0:30	4	4	8
Ac_B	2:00	0:30	4	4	8
Bl_A	3:00	1:00	0	6	6
Bl_B	3:00	1:00	1	4	5
Ht	3:00	0:30	8	2	10

Table 2: Information about the disruptions and the considered uncertainty.

4.2 Objective function

The quality of a solution is measured by a combination of the operational costs and the rescheduling costs. The most important goal is that all remaining tasks are covered by the modified duties. Therefore, we account cost 20,000 for the additional cancellation of a task due to a missing driver.

The cost of each completion is zero if the duty is unchanged. Otherwise the cost is defined as the sum of the individual penalties depending on the way the duty is changed. We used the following values for the penalties. We account a cost of 400 for each duty that is changed anyhow. Every task that is not assigned to its original duty has a cost of 50. A

cost of 1 is accounted for every transfer between two tasks that was not used in the original plan by some crew member. Finally, if a crew member has to be repositioned by using a taxi ride, the accounted costs equals 1,000. These values for the cost parameters performed best in a preliminary study.

4.3 Numerical results

The first stage problem of q-QRSP amounts to computing the completions for the optimistic scenario subject to the additional requirements about the number of recoverable completions. In the first stage problem of q-QRSP we consider initial core problems as described in Section 3.2. In order to account for the uncertainty in the duration of the disruption, we construct the initial core problems based on the optimistic duration of the disruption plus the possible extension.

In the second stage problem we assume that the pessimistic scenario is realized, and we solve the RSP using the results of the first stage problem as input. All second stage problems are solved by the algorithm presented in Potthoff et al. (2010) with the same subsets of duties as in the first stage. Note that we do *not* restrict the recovery action to the mere use of the recovery alternatives of the completions. That approach would lead to a feasible solution only if *all* completions were required to be recoverable in the first stage.

Table 3 shows the objective values obtained for the first and second stage problems. For every fixed value of q, denoting the number of duties that are required to have a recoverable completion, we report the results of the first stage, the second stage, and the sum of them. The outcome of the first stage indicates what happens under the optimistic scenario, while the sum of the first and second stage represents what happens under the pessimistic scenario. For every solved instance we give the lower bound (*LB*), the cost of the best found solution (*UB*), and the number of cancelled tasks (*#CANC*).

First, we notice the intuitive result that more robustness requirements (i.e., a higher value of q) leads to higher first stage costs. The number of additionally cancelled tasks shows the same pattern, which can be expected since these tasks constitute the main part of the first stage costs. So we see that in the first stage more tasks are cancelled to have, most of the time, a better solution in the second stage. Canceling tasks in the first stage also means that there will be more slack in the completions, which can be used in the second stage.

The total costs of the two stages indicate the tradeoff between costs and robustness. Especially for the cases Bl_A en Bl_B, the requirement of more and more robustness initially decreases the total costs. From a certain value of q on, however, the total costs start increasing again. That is, the robustness requirements help to decrease the second stage

Case Ac_A	Stage 1			Stage 2			Stage 1 + 2	
q	LB	UB	#CANC	LB	UB	#CANC	UB	#CANC
0-135	471,430	530,162	22	55,296	56,302	2	586,464	24
136-137	479,646	530,513	22	103,037	112,425	5	642,938	27
138	494,392	550,767	23	115,631	116,349	5	667,116	28
139-140	*506,108*	*551,377*	*23*	*43,206*	*44,604*	*2*	*595,981*	*25*
141	534,303	592,182	25	9,979	10,332	0	602,514	25
142	575,931	612,734	26	297	1,807	0	614,541	26
Case Ac_B	Stage 1			Stage 2			Stage 1 + 2	
q	LB	UB	#CANC	LB	UB	#CANC	UB	#CANC
0-109	46,182	52,486	0	55,434	58,031	2	110,517	2
110-111	43,690	54,148	0	43,852	48,484	2	102,632	2
112	38,478	55,244	0	49,085	66,529	3	121,773	3
113	**42,863**	**57,667**	**0**	**26,491**	**26,582**	**1**	**84,249**	**1**
114	42,326	63,252	0	26,969	28,840	1	92,092	1
115-116	48,309	86,567	1	3,918	3,918	0	90,485	1
Case Bl_A	Stage 1			Stage 2			Stage 1 + 2	
q	LB	UB	#CANC	LB	UB	#CANC	UB	#CANC
0-36	32,093	32,350	1	36,046	45,168	2	77,518	3
37	32,786	32,896	1	34,516	43,661	2	76,557	3
38	**31,222**	**34,653**	**1**	**22,747**	**24,110**	**1**	**58,763**	**2**
39	36,645	39,021	1	5,623	23,314	1	62,335	2
40	56,550	77,666	3	2,157	2,157	0	79,823	3
41	78,384	80,925	3	1,833	1,909	0	82,834	3
42	108,055	118,568	5	1,240	1,305	0	119,873	5
Case Bl_B	Stage 1			Stage 2			Stage 1 + 2	
q	LB	UB	#CANC	LB	UB	#CANC	UB	#CANC
0-34	46,626	58,406	2	65,453	66,511	3	124,917	5
35	**47,955**	**59,765**	**2**	**29,868**	**42,360**	**2**	**83,923**	**4**
36	56,549	76,803	3	24,114	24,114	1	83,923	4
37	73,236	78,164	3	21,876	21,908	1	103,612	4
38	93,673	99,165	4	1,405	1,405	0	102,972	4
39	117,441	119,727	5	2,609	2,609	0	121,832	5
Case Ht	Stage 1			Stage 2			Stage 1 + 2	
q	LB	UB	#CANC	LB	UB	#CANC	UB	#CANC
0-86	151,455	157,135	5	50,620	55,138	2	212,273	7
87-88	150,683	158,198	5	55,903	61,674	2	219,872	7
89	149,497	158,599	5	60,226	61,818	2	220,417	7
90	148,476	158,903	5	71,199	85,320	3	244,223	8
91	147,851	161,222	5	84,797	99,855	4	261,077	9
92	145,239	162,071	5	51,011	60,712	2	222,783	7
93	145,634	164,484	5	55,768	56,091	2	220,575	7
94	141,731	164,578	5	74,104	97,043	4	261,621	9
95	**144,067**	**164,581**	**5**	**14,698**	**15,735**	**0**	**180,316**	**5**
96	145,375	188,234	6	94,815	98,510	4	286,744	10
97	164,382	191,141	6	7,126	7,974	0	199,115	6
98	183,565	209,077	7	4,436	6,721	0	215,798	7

Table 3: Results of the different cases for different values of q

rescheduling costs, but too much robustness turns out to be expensive in the first stage without any further added value in the second stage.

In the other cases we have somewhat irregular behavior: a more robust schedule in the first stage may lead to higher costs in the second stage. This can happen because not all duties have a recoverable completion and then rescheduling the duties which did not yet have a recovery alternative could lead to bad luck and additionally cancelled tasks in the second stage. Anyway, in all cases the second stage costs are negligible if all crews have a recoverable completion in the first stage.

For all cases, except for Ac_A, we have solutions that have the same number of cancelled tasks in the first stage as the solution where no uncertainty is taken into account ($q = 0$), but with less cancelled tasks in the second stage. Thus in these cases, at the price of some slightly higher total costs for the first stage but without additional cancellations if the optimistic scenario is realized, we can reduce the total number of cancellations if the pessimistic scenario is realized. For these cases, the instance with minimum total costs, a minimum number of cancellations if the optimistic scenario is realized, and a minimum total number of cancellations if the pessimistic scenario is realized is indicated with **bold figures** in Table 3.

For the case Ac_A, the instance with minimum total costs and a minimum total number of cancellations if the pessimistic scenario is realized has one more cancellation if the optimistic scenario is realized than the instance with $q = 0$. This instance is indicated with *italic figures* in Table 3.

To illustrate the foregoing, Figure 5 gives a graphical representation of the results for the case Bl_A. The blue line indicates the costs of the first stage for varying levels of the number of recoverable duties q, and the dashed blue line gives the corresponding lower bounds for the costs of the first stage. The green line indicates the costs of the second stage for varying levels of q, and the dashed green line gives the corresponding lower bounds. The red line represents the total costs for the first and second stage for varying levels of q. The figure clearly indicates that the solution corresponding to $q = 38$ has only slightly higher costs for the first stage than the solution corresponding to a value of q between 0 and 36. However, for this solution the total costs for the first and second stage are significantly lower. Note that the differences between the upper and the lower bounds are small.

Summarizing, the computational results confirm the intuition that a higher degree of first stage q-quasi-robustness in general leads to higher first stage costs as well as to lower second stage costs, and that an optimal level of q-quasi-robustness can be obtained by varying the number of recoverable duties q. Most of our cases reach the lowest total costs at an intermediate robustness level: no robustness and full robustness are both inferior. Our algorithm can explore the consequences of several robustness levels, and thereby help the decision makers to find the best balance between total costs and robustness.

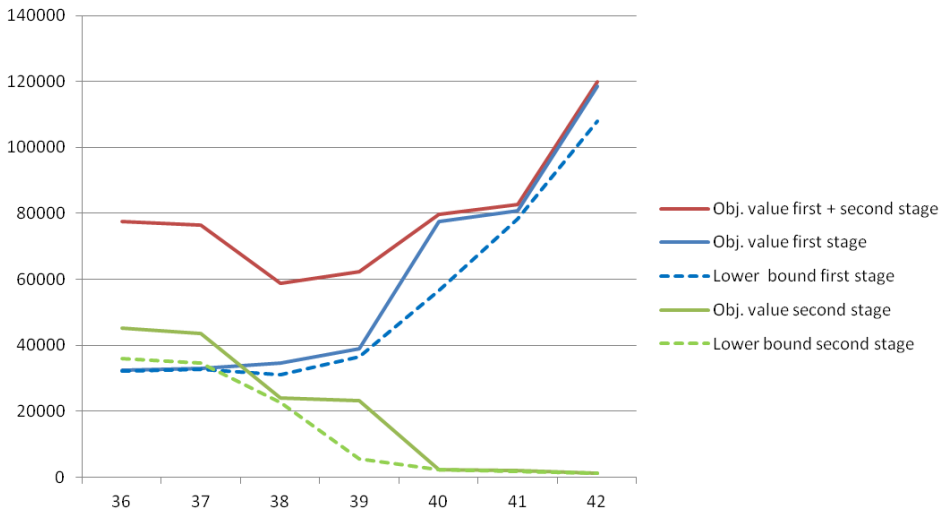

Figure 5: Graphical representation of the results for the case Bl_A.

4.4 Computation times

Computation times are very important for our application, since we are dealing with an application of real-time rescheduling. The cases around Beilen can be solved very quickly. For any given value of q for the number of crews that need to have a recoverable completion, the first stage is solved within 10 seconds and the second stage is solved within 3 seconds. The other cases need longer computation times. For a given value of q, the second stage is solved within 1.5 minutes, but the first stage can take up to 3.5 minutes. These running times are promising and fast enough for a real-time application.

It is beyond the scope of this paper to fully address the running time issues. We are currently working on improving the computational times, though. One may exploit the fact that the key ingredients of the underlying column generation algorithm are suitable for parallel computations. We expect a major improvement from a multi-threaded reimplementation of our programs. Another idea is that the computation runs of the same case for different values of q share a large amount of information, in particular the graph representation of the tasks. Our prototype algorithm handles each value of q as an independent run. We expect to save quite some CPU time by a more careful reuse of the earlier built data structures.

5 Concluding remarks and future work

In this paper we study real-time resource rescheduling problems in case of large-scale disruptions. We propose a novel rescheduling approach that explicitly deals with the uncertain

duration of the disruption. We introduce the concept of q-quasi-robustness, and argue why classical models (such as robust optimization and stochastic programming) are unsatisfactory for the problems we consider.

Our method is widely applicable to real-life vehicle, crew and machine scheduling problems. Furthermore, the robustness requirements can easily and tractably be integrated into existing column generation models, a commonly used optimization framework for resource scheduling and rescheduling.

We demonstrate the power of our approach on real-life crew rescheduling problems of NS. Our method is able to find solutions of reasonably good quality (proven by lower bounds) in a matter of minutes. A detailed analysis shows that q-quasi-robustness reflects the intuitive notion of robustness quite well.

Besides its methodological contributions, the method has good prospects to be valuable in practice. First, computations on challenging real-life cases reliably lead to good solutions. Second, the computation times of a few minutes are close to what is needed in real-life decision making. And third, our approach is able to balance robustness requirements against operational and recovery costs. This allows decision makers to explore several variants and with different robustness levels.

References

Abbink, E.J.W., M. Fischetti, L.G. Kroon, G. Timmer, and M.J.C.M. Vromans (2005). Reinventing Crew Scheduling at Netherlands Railways. *Interfaces 35*, 393–401.

Ben-Tal, A. and A. Nemirovski (2002). Robust optimization - methodology and applications. *Mathematical Programming 92*, 453–480.

Bertsimas, D. and M. Sim (2003). Robust discrete optimization and network flows. *Mathematical Programming 98*, 49–71.

Birge, J.R. and F. Louveaux (1997). *Introduction to Stochastic Programming*. New York: Springer.

Caprara, A., M. Fischetti, and P. Toth (1999). A Heuristic Method for the Set Covering Problem. *Operations Research 47*, 730–743.

Clausen, J., A. Larsen, J. Larsen, and N.J. Rezanova (2010). Disruption management in the airline industry - Concepts, models and methods. *Computers & Operations Research 37*, 809–821.

Huisman, D., R. Jans, M. Peters, and A.P.M. Wagelmans (2005). Combining Column Generation and Lagrangian Relaxation. In G. Desaulniers, J. Desrosiers, and M.M. Solomon (Eds.), *Column Generation*, GERAD 25th Anniversary Series, pp. 247–270. New York: Springer.

Jespersen-Groth, J., D. Potthoff, J. Clausen, D. Huisman, L.G. Kroon, G. Maróti, and M. Nyhave Nielsen (2009). Disruption Management in Passenger Railway Transportation. In R.K. Ahuja, R.H. Möhring, and C.D. Zaroliagis (Eds.), *Robust and Online Large-Scale Optimization*, Volume 5868 of *LNCS*, pp. 399–421. New York: Springer.

Kall, P. and S.W. Wallace (1994). *Stochastic Programming*. Chichester: John Wiley & Sons.

Kroon, L.G., D. Huisman, E. Abbink, P.-J. Fioole, M. Fischetti, G. Maróti, A. Schrijver, A. Steenbeek, and R. Ybema (2009). The New Dutch Timetable: The OR Revolution. *Interfaces 39*, 6–17.

Liebchen, C., M.E. Lübbecke, R.H. Möhring, and S. Stiller (2009). The concept of recoverable robustness, linear programming, and railway applications. In R.K. Ahuja, R.H. Möhring, and C.D. Zaroliagis (Eds.), *Robust and Online Large-Scale Optimization*, Volume 5868 of *LNCS*, pp. 1–27. New York: Springer.

Nielsen, L.K. (2011). *Rolling Stock Rescheduling in Passenger Railways: Applications in short-term planning and in disruption management*. Ph. D. thesis, Erasmus University Rotterdam, The Netherlands.

Potthoff, D. (2010). *Railway Crew Rescheduling: Novel Approaches and Extensions*. Ph. D. thesis, Erasmus University, Rotterdam, The Netherlands.

Potthoff, D., D. Huisman, and G. Desaulniers (2010). Column Generation with Dynamic Duty Selection for Railway Crew Rescheduling. *Transportation Science 44*, 493–505.

Rezanova, N.J. and D.M. Ryan (2010). The train driver recovery problem - A set partitioning based model and solution method. *Computers & Operations Research 37*, 845–856.