# Solving Large Scale Crew Scheduling Problems in Practice

E.J.W. Abbink[1], L. Albino[3], T.A.B. Dollevoet[1,2], D. Huisman[1,2], J. Roussado[3] and R.L. Saldanha[3]

[1] Department of Logistics, Netherlands Railways (NS),
P.O. Box 2025, NL-3500 HA Utrecht, The Netherlands

[2] Erasmus Center for Optimization in Public Transport(ECOPT) &
Econometric Institute, Erasmus University Rotterdam,
P.O. Box 1738 NL-3000 DR Rotterdam, The Netherlands

[3] SISCOG - Sistemas Cognitivos, SA
Campo Grande 378 - 3, 1700-097 Lisboa, Portugal

E-mail: erwin.abbink@ns.nl, lalbino@siscog.pt, dollevoet@ese.eur.nl, huisman@ese.eur.nl, jroussado@siscog.pt, rsaldanha@siscog.pt

Econometric Institute Report EI2010-63

**Abstract**

This paper deals with large-scale crew scheduling problems arising at the Dutch railway operator, Netherlands Railways (NS). NS operates about 30,000 trains a week. All these trains need a driver and a certain number of guards. Some labor rules restrict the duties of a certain crew base over the complete week. Therefore splitting the problem in several subproblems per day leads to suboptimal solutions.

In this paper, we present an algorithm, called LUCIA, which can solve such huge instances without splitting. This algorithm combines Lagrangian heuristics, column generation and fixing techniques. We compare the results with existing practice. The results show that the new method significantly improves the solution.

*Keywords*: crew scheduling, large-scale optimization, column generation

# 1 Introduction

Netherlands Railways (NS) is the main Dutch railway operator of passenger trains. It operates about 4,700 trains on a working day. All these trains need a driver and a certain number of guards (depending on the length of the train). At the end of 2009, NS employed about 2,700 drivers and 3,000 guards.

Because many rules deal with a whole week instead of a single day, it is more efficient to solve the crew scheduling problem for a single week in one run (see Abbink *et al.* (2008)). In addition, night trains connect the individual days to each other. However, it should be mentioned that the number of night trains on the Dutch railway network is limited.

Because we will consider instances for a whole week, the instances that we solve are a magnitude larger than one typically finds in the Operations Research (OR) literature. Kohl (2003) claims to solve the largest Crew Scheduling Problem (CSP) in the world, namely the instance for one day of the German long-distance traffic. Those instances are comparable to the single-day instances of NS, and thus about 7 times smaller than the instances for a whole week, which are solved in this paper.

Since 1998, NS uses the manual crew scheduling system CREWS, developed by the company SISCOG (Morgado & Martins (1998)). A few years later, since 2002, NS uses an Operations Research based algorithm to create the crew schedules (see Kroon & Fischetti (2001), Abbink *et al.* (2005)). These schedules were loaded and manually adjusted in the CREWS system. Because the crew scheduling algorithm could only solve single-day instances, Abbink *et al.* (2008) described several adjustments to improve the use of this algorithm.

To obtain further improvements and to get a full connection between the manual system and the algorithm support, NS and SISCOG decided to join forces and started a research project in 2007. This resulted in the development of the algorithm, called LUCIA. In a first phase, an algorithm was developed capable of providing better solutions than the already highly optimized solutions generated by the previous algorithm. This first version was capable of solving the individual day instances efficiently. The first production plan was created with the stand-alone version of LUCIA for the schedules for 2009. Overall an estimated efficiency gain of 2% was achieved. As SISCOG is also supplying the decision support system CREWS to NS, the next step was to fully integrate LUCIA into this system. This resulted in a very powerful system that enables the planners of NS to perform what-if scenario analyses and make very efficient schedules. During the summer of 2009, NS needed to change the crew schedules due to an agreement with

the unions to change the minimum transfer time. With the new system, NS was able to create these new plans in a few weeks. This paper describes the extension of the algorithm LUCIA, in order to address the instances for a whole week, with the aim to gain additional efficiency improvement.

The remainder of this paper is organized as follows. The crew scheduling process at NS, including a detailed description of the labor rules, is explained in Section 2. Section 3 provides a mathematical formulation of the problem. In Section 4, we discuss the outline, and some important details, of the LUCIA algorithm. The results of several computational experiments with this algorithm, comparing solving 7 individual days and a whole week in one single run, are reported in Section 5. Finally, we finish this paper with some concluding remarks on the business impact.

# 2 Crew planning at NS

## 2.1 Planning process

In Figure 1, we give a schematic overview of the crew planning process for drivers and guards at NS. Other crew members (at ticketing offices, the call center, mechanics, etc.) fall outside the scope of this paper. The crew scheduling problem (CSP) is the problem of assigning tasks to anonymous duties. These tasks are given by the timetable and by the rolling stock schedule (see Huisman *et al.* (2005) for a discussion on all planning problems at NS). More formally, a *task* is the smallest amount of work that has to be assigned to one crew member. A *duty* is the work for one crew member from a specific crew base on a certain day.

At NS, the crew scheduling process has been split in two stages. First, the crew schedules for the annual plan are constructed. Secondly, the crew rosters are created, where the crew members are assigned to operate the duties (see Hartog *et al.* (2009)). The annual plan deals with a generic Monday, Tuesday and so on. It is constructed once a year from scratch, and it is modified about 6 times a year as a result of changes in timetable and rolling stock schedules. In addition, there are exceptions on every individual day resulting in modified daily crew schedules (see Huisman (2007)). This paper deals with constructing the crew schedules from scratch.

## 2.2 Labor rules

The rules which a crew schedule needs to satisfy, can be divided into two categories: (1) rules that each duty should satisfy individually, and (2) rules

that involve more duties at the same time.

Each duty should satisfy the following restrictions:

- The minimal duration of a duty is 4 hours. The maximal duration of a duty can be determined from the starting time of the duty and varies between 8 and 9.5 hours.

- It is not allowed to work more than 5.5 hours without a break. If the duration of a duty is less than 5.5 hours, the duty does not need a meal break. The minimal break time depends on the location of the canteen. It varies between 32 and 40 minutes.

- For tasks on the same rolling stock, the minimal transfer time is zero. For normal driving tasks, the minimal transfer time is 20 minutes. Sometimes, there are e.g. two drivers assigned to the same task. In fact, only one of the drivers can actually drive the train and the other driver is actually a passenger. The minimal transfer time for the passenger is only 10 minutes. Also, for driving empty trains a minimal transfer time of 10 minutes is used. There is one exception to these minimal transfer times of 10 minutes. It occurs when a passenger task or a task on an empty train is followed by a normal task *on the same rolling stock*. If the transfer time before the normal task is smaller than 20 minutes, the minimal transfer time before the empty or passenger tasks is set to 20 minutes.

- Drivers can only drive trains on routes that they are familiar with, as they should know the maximum speed on that track, the location of the signals and so on. The same holds for rolling stock: A driver can only drive certain kinds of rolling stock. It is assumed that drivers from the same crew base have roughly the same route and rolling stock knowledge. For guards there are no restrictions related to route and rolling stock knowledge.

- The transfer times for tasks on the same rolling stock can be arbitrarily small. It is only allowed to stay 3.5 hours on the same rolling stock without a small break of 15 minutes.

- Since it is boring to work all day on the same train line, it is only allowed to go up and down twice on the same train line.

To assure that feasible rosters can be created afterwards, the set of duties on each crew base has to satisfy the following constraints:

- The average duration of the duties for a crew base should be less than 8 hours.
- A maximum number of duties can be assigned to a crew base.
- The percentage of short duties should be less than 5%, where a short duty is defined as a duty with a duration of less than 5 hours.
- A duty with a duration of more than 9 hours counts as a long duty. The percentage of long duties is at most 5%.
- A duty is a night duty when it ends after one o'clock in the night or starts before five o'clock in the morning. The percentage of night duties should be lower than 53.6 %.
- The percentage of duties in the weekend, with respect to the total amount of duties, should be not more than 12% on a Saturday and 11% on a Sunday.

Moreover, it is important that to obtain a fair division of the work over the week for the different crew members, the work should be fairly spread over the different bases. The latter constraints are typical for the Dutch situation and are known as "Sharing Sweet & Sour" rules. They aim at allocating the popular and the unpopular work as fairly as possible among the different crew bases. For the following properties, there is a bound both on the average for each crew base and on the standard deviation over the crew bases.

- Preferred trains
- Aggression work
- Nasty stock

For example, we give a detailed description of the rule related to the preferred trains (A-trains). The work in every task consists of working on an A-train, a B-train or of different work. Generally, stoptrains are B-trains, while all other normal (non-empty, national) trains are considered A-trains. Define now for every crew base $s \in S$, the total amount of work on an A-train, $d_s^A$ and the total amount of work on a B-train, $d_s^B$. The following should be satisfied.

1. The percentage of A-trains for each crew base $s \in S$ should be at least 25 %.
$$p_s^A := \frac{d_s^A}{d_s^A + d_s^B} \times 100\% \geq 25\%.$$

2. The standard deviation of the percentages of A-trains should be below 14 percent. We use the number of duties for the crew bases as the weights, and denote $D_s$ for the number of duties for crew base $s \in S$. The weighted average percentage of A-trains is then defined as

$$p^A := \frac{\sum_{s \in S} D_s p_s^A}{\sum_{s \in S} D_s} \times 100\%.$$

The standard deviation is now defined with respect to this average. We should have

$$\sigma^A := \sqrt{\frac{\sum_{s \in S} D_s (p_s^A - p^A)^2}{\sum_{s \in S} D_s}} \leq 14\%.$$

For a more detailed description of other "Sweet-and-Sour" rules, we refer to Abbink *et al.* (2005).

# 3 Problem formulation

We will now formulate the crew scheduling problem as an integer programming problem. Denote $S$ for the set of crew bases. For every crew base $s \in S$, we consider the set $J_s$ containing the possible duties that can be performed by crew base $s$. To every duty $j$ we associate costs $c_{sj}$. These costs are (affinely) dependent on the length of the duty. Finally, let $T$ be the set of tasks that need to be covered. We define a 0/1 parameter $a_{sjt}$ that indicates whether the duty performs task $t$ or not. Furthermore, we introduce the binary decision variables $x_{sj}$ that indicate whether a duty is selected or not. The crew scheduling problem is now given by

$$\min \sum_{s \in S} \sum_{j \in J_s} c_{sj} x_{sj} \tag{1}$$

$$\sum_{s \in S} \sum_{j \in J_s} a_{sjt} x_{sj} \quad \geq 1 \quad \forall t \in T, \tag{2}$$

$$L_i \leq \sum_{s \in S} \sum_{j \in J_s} w_{sji} x_{sj} \quad \leq U_i \quad \forall i \in I, \tag{3}$$

$$x_{sj} \quad \in \{0, 1\} \quad \forall s \in S, j \in J_s. \tag{4}$$

The objective is to minimize the total cost of the duties. Constraints (2) ensure that every task is being performed at least once. Additional constraints, such as crew base capacity and average duty duration are taken into account via (3). The set $I$ is an index set for the additional constraints.

The capacity constraint for crew base $s \in S$ can be formulated as

$$\sum_{j \in J_s} x_{sj} \leq B_s,$$

where $B_s$ is the capacity of crew base $s$. Note that this is of the form (3) if we choose $w_{s'ji} = 1$ for $s = s'$ and 0 otherwise. The constraint that the average duration of the duties should be less than $\Pi_s$ hours for every crew base $s \in S$ can be expressed as

$$\sum_{j \in J_s} (\pi_{sj} - \Pi_s) x_{sj} \leq 0.$$

Here $\pi_{sj}$ is the duration of the duty. The average amount of work on an A-train can be incorporated in the model by replacing it by

$$\sum_{j \in J_s} \left( \frac{3}{4} d^A_{sj} - \frac{1}{4} d^B_{sj} \right) x_{sj} \geq 0,$$

where $d^A_{sj}$ and $d^B_{sj}$ correspond to the amount of A and B work in duty $j$, respectively.

Most of the other constraints can also be formulated as (3). For the constraints that deal with a standard deviation, this does not hold. In a first run, we often discard these constraints. When a solution is found, the average for every crew base and the standard deviation over the crew bases can be computed. If the standard deviation is too large, we usually notice that a small number of crew bases deviates too much from the average. For these crew bases, the bounds for the constraints on the average are adjusted. By solving the problem with the new bounds, one usually finds solution for which the standard deviation value is closer to the required value. By repeating the procedure, solutions that satisfy the constraints on standard deviation are found.

# 4 LUCIA

In this section, we describe the LUCIA algorithm that was developed to solve the CSP. LUCIA has been fully integrated in the CREWS package developed by SISCOG. It uses dedicated networks where nodes corresponds to tasks, and arcs to connections between tasks. A path in such a network corresponds then to a duty.

The CSP is solved by a Lagrangian-based heuristic combined with column generation. We refer the reader for details about the theory of column

generation to a few surveys on this topic: Barnhart *et al.* (1998); Lübbecke & Desrosiers (2005); Desaulniers *et al.* (2005).

As we will see later in more detail, the algorithm is designed to take advantage of the modern CPU architectures with multiple cores. The pricing problems (generation of the duties) and parts of the master problem can be processed in parallel.

## 4.1 Network of connections

First, we explain how we construct the networks, where paths represent all possible duties. We construct such a network for each crew base such that we can take the route and rolling stock knowledge of a crew base directly into account. Since we are planning duties for a complete week, a task in this context refers to an activity that the crew member has to perform on a particular weekday. This means that in this network: (i) a train that runs from Monday to Friday corresponds to 5 nodes, one per weekday; (ii) the starting time of a task is the time elapsed between 0:00 of the first day of the week and 0:00 of the day where the task starts plus the clock time where the task starts. This also means that the network contains all tasks operated in a standard week. Arcs in this network connect either: (i) nodes on the same weekday, (ii) nodes at the end of a weekday to nodes at the beginning of the next weekday; and (iii) nodes at the end of the week to nodes at the beginning of the week. The length of an arc is limited, as will be explained later. So, this means that the network is typically continuous, because the subnetworks of consecutive weekdays may be connected to each other; and cyclic, because it can contain cycles that take one or more weeks to complete the turn.

The length of an arc in this network is limited. More precisely, two tasks $i$ and $j$ can only be connected by an arc if the time between them satisfies the following upper bound:

$$[(taskStartTime_j - taskEndTime_i) \bmod weekTime] \leq maxArcLength, \quad (5)$$

where $taskStartTime_j$ is the starting time of task $j$, $taskEndTime_i$ is the ending time of task $i$, mod is the modulo operator, $weekTime$ is a constant representing the time elapsed in one entire week, and $maxArcLength$ is a parameter representing the maximum length allowed for an arc, which must always be smaller than the maximum duty length.

To check whether an arc between two nodes exists and is useful, we solve a shortest path in another network, which contains much more details. This is necessary because the minimal transfer time between two trains differ

when a crew member is a passenger or working on a certain train. In this second network, the nodes corresponds to *trips*. Here, a trip is defined as the movement of a train between a pair of stations. There are arcs between two nodes if two trips can be assigned to the same crew member and when there is enough transfer time between two trips. In addition, we introduce a copy of each node which represents the same trip but it is used a positioning trip (with the shorter transfer time). We illustrate this with an example (see Figure 2).

In this example six nodes are depicted. There is not enough transfer time between node 4 and node 2, so there is no connecting arc. Node $2'$ is added as an optional trip and there is an arc between node 4 and $2'$. By adding the optional trip, the solution can be made with two duties (1, 2, 3) and (4, $2'$, 5). Without the optional trip it would be required to generate an additional duty. A drawback of this formulation is that the size of the problem is increased significantly when the number of additional optional trips is large.

A simple, yet very efficient solution is depicted in Figure 3. We add an arc between node 4 and node 5 (and between node 4 and 3) because we know that there is a possibility to get from the end location of node 4 to the start location of node 5 by using a positioning trip. Please note that we could also add an arc between node 1 and node 3 but, due to the mechanism of allowing node 2 to be over-covered, this arc would be redundant.

This idea can also be found in the work of Rezanova & Ryan (2010). A special preprocessing procedure determines when such additional arcs are to be added. Adding arcs instead of nodes is computationally efficient and solves our problem for formulating differentiated transfer times.

## 4.2 Lagrangian heuristic combined with column generation

The main characteristics of the algorithm are a dynamic pricing scheme for the variables, coupled with subgradient optimization and greedy algorithms, and the systematic use of column fixing to obtain improved solutions. To tackle the large number of potential duties, LUCIA uses column generation to generate feasible duties ("columns"). These duties are generated by solving a resource constrained shortest path problem in a network of connections (see Section 4.1).

At the top level, as shown in Table 1, the algorithm obtains solutions based on a given network of connections by executing a loop (line 4) that alternates between column generation and optimisation that is fed by the

```
1  algorithm Solve (network) returns sols
2    begin
3      (cols, Λ) ← GenerateInitialSetOfColumns(network)
4      return GenerateAndOptimise(network, cols, Λ)
5    end
```

Table 1: The main algorithm

```
1  algorithm GenerateInitialSetOfColumns (network) returns(cols, Λ)
2    begin
3      cols ← GenerateSlackVariableColumns(tasks(network))
4      Λ ← (0, 0, ..., 0)
5      while  Number of iterations without improving lBound(Λ, cols)
                    below certain limit and
6                Total number of iterations below certain limit do
7        begin
8          cols ← cols \ ChooseColumnsToDelete(cols, Λ)
9          Λ ← SolveLagrangeanRelaxation(Λ, cols)
10         aggregLevel ← 100% − (percentage of iterations done so far)
11         cols ← cols ∪ GenerateColumns(network, ∅, Λ, aggregLevel)
12       end
13     return(cols, Λ)
14   end
```

Table 2: Algorithm that generates the initial set of columns

columns stored in the pool *cols* and the set $\Lambda$ of Lagrange multipliers (line 3) related with the constraints (2) and (3).

The generation of the initial set of columns is shown in Table 2. The algorithm starts by generating for each task in the network a column representing an infeasible duty that covers that task (line 3). These duties have a very high cost so that they are included in the solution only if there is no feasible way of covering the task. Next to that, the algorithm enters in an iterative process where several steps are executed. First (line 8), uninteresting columns are removed from the column pool *cols*. Then (line 9), the Lagrangian multipliers $\Lambda$ are updated by solving the Lagrangian relaxation problem in a very similar way as described in Caprara *et al.* (1999). Finally (line 11), the columns are generated (i.e. the pricing problem is solved) for the updated multipliers according to what will be explained later in this

```
1  algorithm GenerateAndOptimise (network, cols, Λ) returns sols
2    begin
3      sols ← ∅
4      fixedCols ← ∅
5      while Gap between lBound(Λ, cols) and uBound(sols)
6                    above certain limit  and
6              Total number of iterations below certain limit
7        begin
8          Λ ← SolveLagrangeanRelaxation(Λ, cols)
9          (Λ, sols) ← SolveOptimisationProblem(cols, Λ, fixedCols, sols)
10         fixedCols ← ChooseColumnsToFix(cols, Λ)
11         cols ← cols \ ChooseColumnsToDelete(cols, fixedCols, Λ)
12         while Number of sub-iterations below certain limit do
13           begin
14             cols ← cols ∪ GenerateColumns(network, fixedCols, Λ, 0%)
15             Λ ← SolveLagrangeanRelaxation(Λ, cols)
16           end
17       end
18     return sols
19   end
```

Table 3: Algorithm that solves the problem by combining generation and optimisation phases

document. In order to speedup the column generation process, tasks that share the same rolling stock are aggregated in a decreasing degree of strength during the iterative process. Aggregating a set of tasks means considering them (temporarliy) as just one task, so that they all must be assigned to the same duty. By doing so, the number of nodes in the network reduces as well as the number of arcs (in a more dramatic way), and therefore the pricing problems become easier and faster to solve. This procedure is known as partial pricing.

The loop that alternates between column generation and optimisation is the most important part of the algorithm and is shown in Table 3. In the main loop, that terminates when the gap between the lower and upper bound becomes small enough or when a certain number of iterations is reached (line 5), after updating the Lagrangian multipliers in line 8, the optimisation problem is solved for the current set of columns and multipliers (line 9) to obtain solutions. The optimisation problem is solved with a Lagrangian-based heuristic inspired by Caprara *et al.* (1999).

Before switching to the column generation phase, the most interesting columns (with smallest reduced cost) are fixed (line 10) to reduce the complexity of the problem, and uninteresting columns are removed from the column pool *cols* (line 11). The column generation (line 14) is executed several times in a secondary loop where the multipliers are updated (line 15) to reflect the new columns that were added to the container.

**Column generation** The generation of columns (i.e. the resolution of the pricing problem) is done by an algorithm that uses as a starting point the solution of the shortest path problem over the network of connections. As in Huisman (2007), the algorithm looks for variations over the shortest path (including the shortest path itself) and it collects the ones that have negative reduced cost and that are feasible. A path is feasible if it satisfies some additional constraints such as the meal break constraint.

The fact that the network is cyclic poses a problem to the shortest path algorithm. This problem is easily solved by solving the pricing problem over different parts of the network (subnetworks), instead of doing it over the entire network. Splitting the pricing problem in parts, not only solves the cyclic network problem, but also opens a door for running the algorithm in parallel. A natural way of dividing the network in parts is by weekday. This means that each of the 7 subnetworks is associated to a particular weekday and should be such that the columns generated inside it correspond to all duties that start on that weekday (and therefore belong to that weekday). So, following these guidelines, we defined the subnetwork of weekday $j$ as the part of the overall network containing all sign on and sign off nodes and the tasks (nodes) that fit inside the following time intervals:

$$\left[dayStart_j,\ dayEnd_j + maxDutyLength\right], \qquad \text{if } j = 1, ..., 6 \quad (6)$$

$$[dayStart_1,\ maxDutyLength] \cup [dayStart_7,\ dayEnd_7], \qquad \text{if } j = 7, \quad (7)$$

where: $dayStart_j$ is the time elapsed between the start of the week and the time where weekday $j$ starts; $dayEnd_j$ is the time elapsed between the start of the week and the time where weekday $j$ ends; and $maxDutyLength$ is the maximum time length allowed for a duty.

A conclusion that we can take out of this definition is that the subnetwork of a weekday overlaps with the subnetwork of the next weekday in a time length that is equal to $maxDutyLength$. These overlapping areas contain tasks that belong to both weekdays (can be covered by duties of both weekdays). By looking to the definition it is also possible to confirm that the subnetworks are acyclic as required beforehand.

Due to the size of the network, solving the pricing problem is usually a very time consuming process. In order to obtain very significant performance improvements we run the algorithm in parallel. The number of threads depend on the number of cores available. If there are enough cores, there can be a thread for each weekday and crew base. In this case, each thread would run the pricing algorithm over a subnetwork corresponding to a given weekday with a sign in and sign off node corresponding to a given crew base.

# 5 Results

In this section we present the improvement we reach by solving the CSP for a complete week. We will not compare the performance of the algorithm for a single weekday instance with existing approaches in detail. Internal benchmarks show that the optimizer is capable of generating solutions which are about a few percent better than the results presented in Abbink *et al.* (2005) and Abbink *et al.* (2008). These results had already a high quality and were used in practice for many years. Abbink *et al.* (2008) also addressed the issue of the global (week) constraints. However, the authors did not solve the problem as a whole, like the presented approach does.

To compare the results of optimizing the instances per weekday with the results of the optimizing the week as a single instance, we have created two cases: (i) all tasks for the drivers and (ii) all tasks for the guards. With these two cases we varied the number of applied constraints. All experiments were carried out on the same hardware[1].

First, we made an experiment without global constraints. In the second experiment we added the constraints on the duration (D) of the duties. To be specific, the constraint of 8 hours on the maximum average duration of the duties and the constraint saying that no more than 5% of the duties can have a length of more than 9 hours. In a third experiment we have added the constraint on having a minimum of 40% and a maximum of 70% of work on preferred trains (PT) per crew base. We have created these three experiment instances for both drivers and guards.

Table 4 shows the result for each of the instances. The first two columns indicate the type of the problem. The second and third column show the value of the cost function for the individual day cases, respectively for the week instance. The first experiment shows that solving the complete week as a single instance gives a worse result. In theory, the solution can be better because the solver can decide to plan a task in a duty of a different day than the day it was assigned to in the runs for the individual days. However, the

[1]Intel Octo Core (2x4 Cores), 2.6GHz, 8Gb RAM

| Case | Constraints | Indiv. Days | Week | Delta | Relative |
|---|---|---|---|---|---|
| Drivers 1 | none | 22907166 | 22962526 | 55360 | 0.2% |
| Drivers 2 | D | 23079190 | 22882933 | –196257 | –0.9% |
| Drivers 3 | D & PT | 23403709 | 23097736 | –305973 | –1.3% |
| Guards 1 | none | 27608478 | 27738766 | 130288 | 0.5% |
| Guards 2 | D | 27863687 | 27790539 | –73148 | –0.3% |
| Guards 3 | D & PT | 28229258 | 28094205 | –135053 | –0.5% |
| Guards 1 + Drivers 1 | none | 50515644 | 50701292 | 185648 | 0.4% |
| Guards 2 + Drivers 2 | D | 50942877 | 50673472 | –269405 | –0.5% |
| Guards 3 + Drivers 3 | D & PT | 51632967 | 51191941 | –441026 | –0.9% |

Table 4: Results Week optimization

experiments show that the increased size of the problem makes the problem harder to solve which results in a worse solution. The effect is larger for the guards, which can be explained by the fact that the instances are larger than the drivers instances.

The other experiments show that solving the complete week as a single instance gives a better result than solving the single weekday instances. This can be explained by the fact that in the week instances, the algorithm can for instance decide to have a larger average duration for the duties in the weekend as long as it compensates this with shorter duties during the weekdays.

It is not shown in the table, but when we inspect the solution in more detail, this is exactly what happens in some cases. In other cases the system decides to have longer duties during the weekdays and to have shorter duties during the weekend. For the total set of bases we see that the average duration gets longer in the weekend and also the related percentage of long duties (with a duration of more than 9 hours), is higher than during the weekdays.

Having short duties during the weekdays is efficient because in railway operations there is generally a peak–hour pattern for the commuters. During these peak-hours there is a higher need for crew than during the rest of the day and this need can be efficiently covered with short duties. In the weekend there is no peak-hour pattern, and it is efficient to create long duties. The positive effect of this is the fact that the amount of duties during the weekend is reduced. Crew generally like to have the weekend off, so they regard having to operate relatively less duties during the weekend as being attractive.

The results of the third set of experiments, with the additional constraint on the preferred trains, show that the effect of solving the week instance as a whole even gives a larger improvement. The effect for the guards in-

stances is somewhat smaller than for the drivers, but still it is a significant improvement. The results show that the gain in solving the week as a single instance gets larger when there are more global constraints that apply for the complete week. Even though the problem has more global constraints, in some cases, the algorithm is still able to find better solutions. With the presented approach we are able to solve the complete week instance with global constraints that are to be applied in the NS case. The overall effect is that we gain an efficiency increase of about 1%.

# 6 Conclusions and Business Benefits

As explained in the introduction, initially we developed an algorithm LUCIA capable of solving the individual day instances efficiently. Practical results show that the use of this algorithm improved the quality of the crew schedules at NS. Then we extended this algorithm such that it was able to solve instances for the complete week.. The presented work shows that solving the complete week can be done with the current state-of-the-art techniques and that it leads to a significant improvement of about 1% efficiency gain, which reflects a cost reduction of 3 million euros.

The work described in this paper is the result of joined research between a commercial software vendor and a large railway operator. It is interesting to see that combination of the common Operations Research knowledge, the specific IT knowledge of SISCOG and the specific domain knowledge of NS lead to benefits for both parties. For SISCOG, the research resulted in a new optimizer that was integrated in their software suite. Their competitive strength was enlarged and SISCOG is now able to support the most complex crew scheduling problems. The basic algorithm for single days is currently used in production at NS. The results of the solving the complete week instances have not been used in practice of NS yet. As the results of the experiments are very promising, we foresee that the new approach will be applied in practice in the near future so the indicated cost reduction of about 3 million can be achieved.

# Acknowledgements

We want to thank Tiago Dias and Rudi Araújo for their important contribution on the implementation of the software code related with preparation of data, in particular the generation of connections.

# References


Abbink, E., Van 't Wout, J., Huisman, D. 2008. Solving large scale crew scheduling problems by using iterative partitioning. http://drops.dagstuhl.de/opus/volltexte/2007/1168/pdf/07001.AbbinkErwin.Paper.1168.pdf.

Abbink, E., Fischetti, M., Kroon, L., Timmer, G., & Vromans, M. 2005. Reinventing crew scheduling at Netherlands Railways. *Interfaces*, **35**, 393–401.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., & Vance, P.H. 1998. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, **46**, 316–329.

Caprara, A., Fischetti, M., & Toth, P. 1999. A Heuristic Algorithm for the Set Covering Problem. *Operations Research*, **47**, 730–743.

Desaulniers, G., Desrosiers, J., & Solomon, M.M. (eds). 2005. *Column Generation*. Springer, New York.

Hartog, A., Huisman, D., Abbink, E.J.W., & Kroon, L.G. 2009. Decision Support for Crew Rostering at NS. *Public Transport*, **1**, 121–133.

Huisman, D. 2007. A Column Generation Approach to solve the Crew Re-Scheduling Problem. *European Journal of Operational Research*, **180**, 163–173.

Huisman, D., Kroon, L.G., Lentink, R.M., & Vromans, M.J.C.M. 2005. Operations Research in passenger railway transportation. *Statistica Neerlandica*, **59**, 467–497.

Kohl, N. 2003. Solving the World's Largest Crew Scheduling Problem. *ORbit*, 8–12.

Kroon, L.G., & Fischetti, M. 2001. Crew Scheduling for Netherlands Railways "Destination: Customer". *Pages 181–201 of:* Voß, S., & Daduna, J.R. (eds), *Computer-Aided Scheduling of Public Transport*. Springer, Berlin.

Lübbecke, M.E., & Desrosiers, J. 2005. Selected Topics in Column Generation. *Operations Research*, **53**, 1007–1023.

Morgado E. M., Martins, J. P. 1998. CREWS_NS: Scheduling train crews in The Netherlands. *AI Magazine*, **19**, 25-38.

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

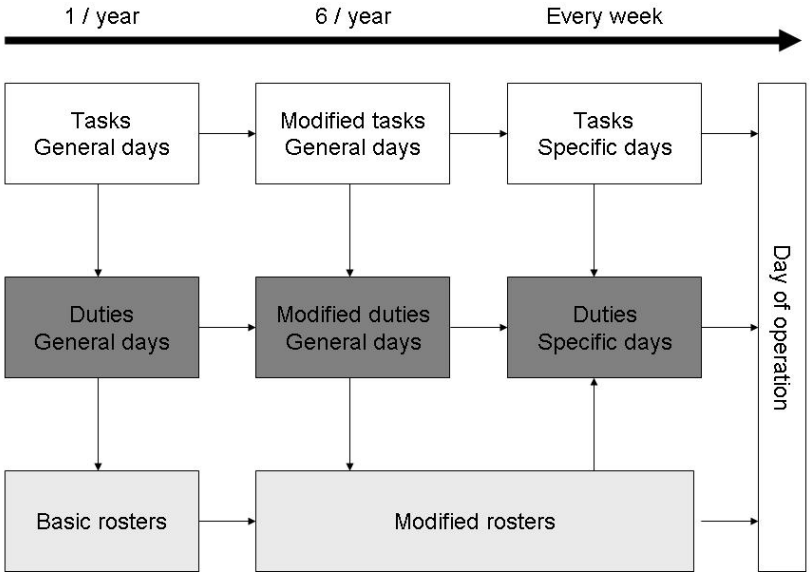


Figure 1: Crew planning process

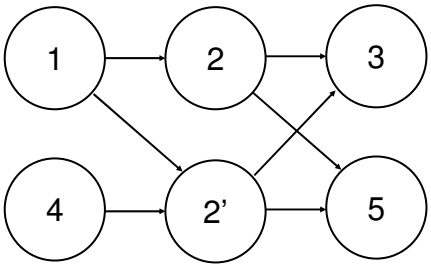


Figure 2: Graph with optional node

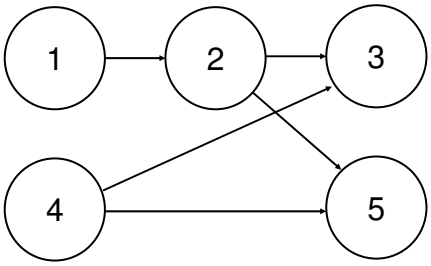


Figure 3: Graph with additional arc