The Train Driver Recovery Problem – a Set Partitioning Based Model and Solution Method

Natalia J. Rezanova
Informatics and Mathematical Modelling, Technical University of Denmark
Richard Petersens Plads 1, Building 305, DK-2800 Kgs. Lyngby, Denmark
e-mail: njr@imm.dtu.dk

David M. Ryan
The Department of Engineering Science, Faculty of Engineering
The University of Auckland
Private Bag 92019, Auckland Mail Centre, Auckland 1142, New Zealand
e-mail: d.ryan@auckland.ac.nz

IMM-TECHNICAL REPORT-2006-24

Abstract

The need to recover a train driver schedule occurs during major disruptions in the daily railway operations. Using data from the train driver schedule of the Danish passenger railway operator DSB S-tog A/S, a solution method to the Train Driver Recovery Problem (TDRP) is developed. The TDRP is formulated as a set partitioning problem. The LP relaxation of the set partitioning formulation of the TDRP possesses strong integer properties. The proposed model is therefore solved via the LP relaxation and Branch & Price. Starting with a small set of drivers and train tasks assigned to the drivers within a certain time period, the LP relaxation of the set partitioning model is solved with column generation. If a feasible solution is not found, further drivers are gradually added to the problem or the optimization time period is increased. Fractions are resolved with a constraint branching strategy using the depth-first search of the Branch & Bound tree. Preliminarily results are encouraging, showing that nearly all tested real-life instances produce integer solutions to the LP relaxation and solutions are found within a few seconds.

1 Background for the Project

While the disruption management applications within the airline industry have been addressed by many Operations Research practitioners during the last decade, the subject of recovery from daily disruptions of the railway crew and rolling stock is not as well-studied. A limited competition among railway operators, governmental subsidies to the railway industry and lower costs associated with railway disruptions compared to the costs of airline disruptions are some of the main reasons for the fact that the need for optimization within the railway industry has not been as vital as within the airline industry.

Every railway operator experiences disruptions during the daily operation due to external influences and internal failures. The disrupted operation causes crew and passenger dissatisfaction, ending up in extra expenses and revenue losses. The Danish railway operator DSB S-tog A/S (hereinafter referred to as S-tog) is no exception. Disruptions in the daily train schedule disturb the train driver schedule, forcing the dispatchers to manually re-schedule the driver duties. A Decision Support System (DSS), which is able to find train driver recovery solutions automatically, may help train driver dispatchers in their work. Building a prototype for the Train Driver Recovery DSS is a part of a Ph.D. study at the Department of Informatics and Mathematical Modelling of the Technical University of Denmark in a cooperation with S-tog.

The rest of the paper is organized as follows. An introduction to the train driver schedule and recovery methods at S-tog and a literature review are presented in Section 2. The set partitioning formulation of the problem, integer properties of the problem and the chosen solution method are outlined in Section 3. Data structure and a dynamic programming algorithm for recovery duty generation are presented in Section 4. The Branch & Price approach for solving the TDRP is described in Section 5. Test results are presented in Section 6. The research presented in this report is summarized in Section 7.

2 Introduction

2.1 The S-train Network

DSB S-tog A/S operates on the S-train network, which covers the Greater Copenhagen area. S-tog is a part of the Danish State Railways (DSB), the largest train operator in Denmark. The S-train network is shown on Figure 1. It consists of 170 km double tracks and 85 stations. Each segment of the network is covered by at least two train lines with a cyclic schedule and a frequency of 3 trains per hour in each direction. A line is either a *main line*, running from approximately

5 am to 1 am next morning, or an *extra line*, operating during the daytime hours, from about 6 am to 7 pm. A main line is indicated by a colour and a capital letter, e.g., a line between Hundige and Hillerød is the light-blue **A** line. An extra line is indicated by a colour and a capital letter with a "**+**" or an "**x**" after the letter, e.g., the green **B+** runs between Høje Taastrup and Holte, while the purple **Ex** runs between Køge and Østerport. A combination of main lines and extra lines at each segment of the network allows a higher departure frequency at almost every station of the network during the daytime hours.

Figure 1: The S-train Network 2006

Train lines **F** and **F+** are the only two lines, which do not pass København H (Copenhagen Central Station). This part of the network is called a *circular rail* segment. The other parts of the network are called *the fingers*. The bottleneck of the network is between København H and Svanemøllen station north of Copenhagen. This segment is the busiest part of the network. Since all trains, excluding

the trains on the circular rail segment, run through this part of the network, it defines the minimum headways for the trains. Currently, the minimum headways is set to 2 minutes, giving a possibility for 10 train lines to traverse the central segment in each direction.

2.2 Introduction to S-tog's Train Driver Schedule

A daily S-tog schedule is covered by approximately 270 drivers on a workday and less than 200 drivers on a weekend day, excluding reserve drivers. Each driver is able to operate all types of the rolling stock owned by S-tog and only one driver is required to operate a train. A driver's *duty* starts with a check-in and ends with a check-out at one of the three *check-in depots*, located at København H, Hillerød station in the north and Køge station in the south. Each duty consists of a sequence of tasks, i.e. train drives, meal breaks, preparing a train to the first daily drive, riding as a passenger on a train (a *passengering task*), driving empty trains to the maintenance depot, etc. The length of a daily duty is between 6 and 8 hours. Longer duties are allowed on weekends (approx. 8 hours and 40 minutes on Saturdays and 8 hours and 20 minutes on Sundays).

A duty contains either one *full break* or two *half-breaks* between train tasks. The length of a full break is 30 minutes. The length of a half-break is either 20 or 25 minutes. The total duration of the two half-breaks in a duty must be at least 45 minutes, so two short half-breaks are not allowed. A break is held at one of the two *crew depots*, placed at København H and Hellerup stations. The crew depot at Hellerup station serves the drivers assigned to the circular rail, while the rest of the drivers hold their meal breaks at the main crew depot at København H. A driver is entitled to a break after at most 3 hours of work, with the 3 hours and 30 minutes exception for drivers assigned to lines **H** and **H+**, where the return drive between end-stations Farum and Frederikssund takes 3 hours and 20 minutes.

A *train task* in a schedule is a train drive between two terminal stations. All end-stations of S-tog train lines (e.g. Farum station for lines **H** and **H+**) and the central station København H are *terminal* stations. A *subsequence* of a train task v is another train task w, which is the next (subsequent) task following v in a train driver duty. Check-in depots and crew depots are *relief* terminal stations, i.e. stations, where a driver can hand over a train to another driver and go on a break or to a check-out. In an undisturbed schedule, the driver arriving on a train at a *non-relief* terminal station is the one assigned to the first departure of the line back from that station. Lines **B** and **B+** constitute an exception from this rule: the driver arriving on a train of the line **B** to a terminal station is the one assigned to the first departure of the line **B+** back from the station and vice versa. Therefore the subsequence of a train task arriving at a non-relief terminal station is unique. Earliest trains of the lines departing from non-relief stations are assigned

to drivers, who arrive at the station as passengers on a train of another line (if there is one) or in a taxi. Likewise, at the end of the day, the driver assigned to the last train of a line takes a passengering task or a taxi back from the non-relief station.

The regular duties are constructed such that each driver has a high variation in tasks during the day. For instance, a driver is not allowed to drive back and forth between two terminal stations during the whole duty period. Rather, different lines must be assigned to each driver during the day. Variation in duty tasks implies that many drivers are assigned to each line during the day. An example of a train driver duty is shown on Figure 2. The driver is assigned to three different train lines during the duty: **E** (Hillerød - København H - Køge - København H), **H+** (København H - Frederikssund - København H) and **A** (København H - Hillerød).

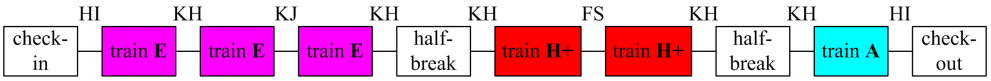

Figure 2: S-tog's Train Driver Duty Example

Up to 19 drivers are scheduled to be in reserve on a workday and up to 12 are scheduled for reserve on a weekend day. Most of the reserve drivers are located at København H, but there are a few who are located at check-in depots in Hillerød and Køge. Most reserve drivers are in reserve for the whole duty, while a few are in reserve only for a shorter period during the day, being assigned to train tasks during the remaining duty time.

2.3 Recovery from Daily Disruptions

The daily operation of S-tog suffers from disruptions caused by speed limitations due to track conditions, signalling problems, delays due to passenger boarding, accidents etc. Train delays caused by minor disruptions are recovered by re-establishing the original plan using the slack time build into the timetable, delaying other trains, making arriving trains wait if a platform at the station is occupied by a delayed train or using different platforms for delayed trains. Major disruptions in the train schedule are recovered by re-routing or cancelling trains. A train is re-routed if it is for instance turned back before reaching the end terminal station in the finger segment or if it is driven through some stations without stopping. A cancellation is applied either to a single train task, i.e. when only one train between two terminal station is cancelled, or to a whole train line, resulting in cancellations of all train tasks of a particular line for a certain period of time. Sometimes several lines are cancelled for a few hours or even for the rest of the day. Most often, extra lines are cancelled in order to maintain operations by

covering the segments affected by cancellations with main lines. Train schedule recovery decisions are taken by the network traffic control center of the Danish railway infrastructure manager Banedanmark.

When a train is delayed, re-routed or cancelled, a driver might be late for the scheduled break or the scheduled train task. Since the time of the break cannot be shortened, the driver is also late for the assigned train task after the break. If the driver delay is severe, the train task is re-assigned to another driver. If there is no available driver to take the train task, the train is significantly delayed or cancelled, causing further disruption in the schedule. A combination of train line cancellations and delays is damaging with respect to the train driver schedule. When a line is cancelled, the excess rolling stock is shunted to depot tracks available along the network segments of the line. For example, if line **H+** is cancelled, the rolling stock assigned to the line is shunted to depot tracks at Frederikssund, Ballerup, København H and Farum stations. The drivers assigned to the cancelled train tasks then end up at terminal stations, which do not coincide with the departure stations of their next scheduled tasks.

In a regular schedule, the drivers are usually assigned to change trains only at terminal stations of the line. In a recovery schedule, a driver can be assign to change a train at an *intermediate* station, where several lines meet, e.g., Lyngby or Buddinge stations. For instance, a driver arriving to Lyngby station with the line **A**, which goes from København H to Hillerød, can be asked to swap trains with a driver, arriving to Lyngby station with the line **B**, which goes from Holte to København, if trains arrive at about the same time at adjacent platforms. Task variation rules are not very strict in recovery situations. A driver can be assigned to drive the same line for the rest of the day, if it is found appropriate for a recovery solution. A driver can also be asked to postpone his break or to take one long break instead of two scheduled short breaks, or vice versa. In extreme situations, a driver can be asked to work overtime or to skip a break. However, recovery schedules involving the latter solutions must be avoided if possible.

Train driver dispatchers responsible for re-scheduling driver duties often work under a tremendous pressure. Every decision has to be communicated to each train driver involved in the recovery solution. The size of the schedule and the unpredictability factor makes it hard for operators to find a good recovery solution fast. Decisions are often based on previous experience, and the quality of each recovery schedule depends on the particular dispatcher. In the worst case drivers are assigned to tasks on the "first in first out" basis without considering their original duties.

2.4 Literature Review of Train Driver Re-Scheduling Solutions

Crew disruption management within the railway industry has not been given much attention by operations researchers, unlike the research within the airline crew recovery (for a recent review refer to [3]). Only two applications within railway industry related to the present research have appeared in the literature.

An integer programming approach to a simultaneous train timetable and crew roster recovery problem is presented in [11]. Starting with an integrated train and driver scheduling model, the authors develop an approach for disruption recovery in realtime. The objective of the model is to minimize the deviation from the existing schedule while minimizing the cost of the adjusted crew shifts. Limiting the time period for which the schedule is resolved, the problem size is kept small enough to produce optimal solutions fast. The problem is modelled as a set partitioning problem with additional constraints ensuring the time consistency of the pieces of work within driver shifts and the maximum duration of shifts. The problem is solved with a Branch & Bound algorithm with column and constraint generation. The linear programming relaxation of the problem at each node of the Branch & Bound tree is solved using the primal revised simplex method. Hot-starting from a solution, which was feasible prior to the disruption, the optimization starts with pricing out variables, which represent pieces of work and idle time for train tasks. When the train variables are priced out, the driver shift variables are considered, constructing columns from initially constructed potential driver shifts, inserting a personal needs break to each shift. Fractions are resolved using constraint branching. Illegal train crossings and overtakes of trains are resolved by adding constraints as these are needed. The model is tested on a one day timetable for the Wellington Metro line in New Zealand, covered by 36 trains. The train services were split into 564 pieces of work at possible relief points. Delays of different duration were introduced on 3 trains in order to disrupt the schedule. Optimal solutions were produced in reasonable time, ranging from about 26 seconds to under 2 minutes.

Other work related to the present research is reported by [5] on the Crew Re-Scheduling Problem for train driver duties disrupted due to the maintenance work on train tracks. The data from NS Reizigers, the largest passenger railway operators in the Netherlands, is used to test the model. The model is formulated as a set covering problem, allowing each train task to be covered more than once, representing the deadheading of the crew. For each original duty, a large set of "look-alike" duties is generated, replacing parts of the original duties with different pieces of work. The Lagrangian relaxation of the model is solved with column generation, pricing out "look-alike" duties with negative reduced cost. If the set of "look-alike" duties is not sufficient to find an optimal (or near optimal) solution, other duties with negative reduced cost are generated as simple shortest paths on

the direct graph, where each vertex represents a piece of work, i.e. the sequence of train tasks on the same rolling stock. The problem is solved to integrality using the heuristic approach for the set covering problem of the crew scheduling package TURNI. Test results from two real-life cases are presented. The first case with 586 duties and 5 683 train tasks generates 8 767 "look-alike" and 184 420 other duties. The problem is solved within 9 hours. The second case with 773 duties and 7 740 train tasks generated 169 974 "look-alike" and 203 961 other duties. The problem is solved within 16 hours.

3 Problem Formulation

The objective of a *recovery* is to "get back" to the original train driver schedule after a disruption as soon as possible by means of *re-scheduling* the original train driver duties. The *Train Driver Recovery Problem* (TDRP) aims at finding the optimal set of feasible train driver recovery duties in a disturbed schedule, such that all train tasks within a certain recovery period are covered and the driver duties outside the recovery period are unchanged. Even though the TDRP can be viewed as a *feasibility* problem, it is important to reflect the stability (or robustness) of the recovery solution by minimizing the number of modification from the original schedule, involving as few drivers as possible in the recovery solution and eventually introducing extra slack time between tasks in recovery duties.

We formulate the Train Driver Recovery Problem as a set partitioning problem. Let K be the set of train drivers involved in the recovery and N be the set of train tasks belonging to drivers in K within the chosen recovery period. Let P^k be the set of feasible recovery duties for a driver $k \in K$. Each recovery duty $p \in P^k$ contains either a subset of train tasks in N or does not contain any tasks, corresponding to the driver spending the time as stand-by.

The cost c_p^k reflects the unattractiveness of the recovery duty $p \in P^k$ for the driver $k \in K$ in the recovery schedule according to the defined objective for the recovery. A binary decision variable x_p^k equals 1 if the duty $p \in P^k$ for the driver $k \in K$ is included in the recovery solution and equals 0 otherwise. A binary parameter a_{ip}^k is used to define whether or not the task $i \in N$ is covered by the duty $p \in P^k$.

$$\text{(TDRP)} \qquad \min \sum_{k \in K} \sum_{p \in P^k} c_p^k x_p^k \tag{1}$$

subject to

$$\sum_{p \in P^k} x_p^k = 1 \qquad \forall k \in K \tag{2}$$

$$\sum_{k \in K} \sum_{p \in P^k} a_{ip}^k x_p^k = 1 \qquad \forall i \in N \tag{3}$$

$$x_p^k \in \{1, 0\} \quad \forall p \in P^k, \ \forall k \in K \tag{4}$$

The objective function (1) of the model minimizes the total cost of the recovery solution. The *train driver constraints* (2) ensure that each train driver is assigned to exactly one recovery duty in the schedule. The train driver constraints have a generalized upper-bounded (GUB) structure, since the constraints are disjoint and each column contributes to exactly one driver constraint. The *train task constraints* (3) have a set partitioning structure and ensure that each train task in the recovery schedule is covered exactly once. Constraints (4) are the integer constraints of the model.

The TDRP model (1) - (4) has the pure set partitioning problem structure of a crew rostering problem. The generalized set partitioning formulation of the crew rostering problem is used in real-life applications within the airline industry for e.g. Air New Zealand [2] and Air France [4]. It is observed in [9] that the linear programming relaxation of the set partitioning formulation of the crew rostering problem possesses strong integer properties due to the existence of the generalized upper-bound crew constraints, which contribute to the perfect structure of the submatrix, corresponding to each crew member. This observation implies that the linear programming relaxation of the Train Driver Recovery Problem (TDRP-LP) also possesses strong integer properties addressed in the next section.

3.1 Integer Properties of the Problem

Let A be a zero-one matrix corresponding to the constraints (2) and (3) of the problem. Let A^k be a submatrix of A corresponding to columns belonging to the driver k. A is an $m \times n$ matrix, where $m = |K| + |N|$ is the number of rows in the problem and $n = \sum_k |P^k|$ is the number of columns. A^k is an $m \times n^k$ matrix, where $n^k = |P^k|$ is the number of columns in the problem belonging to the same driver k and $m = |K| + |N|$ is the number of rows in the problem. An example of a matrix A for a TDRP involving $|K| = 4$ drivers who are to be assigned to $|N| = 3$ train tasks is shown on Figure 3.

According to Theorem 3.16 in [7], an $m \times n$ matrix A is perfect if and only if it does not contain any $m \times l$ submatrices A_l, where $3 \leq l \leq n$, with the following property $\Pi_{\beta,l}$: A_l contains an $l \times l$ nonsingular submatrix B_l with row and column sums all equal to $\beta \geq 2$, while each row of A_l, which is not in B_l, is either componentwise equal to a row in B_l or has a row sum strictly less than β.

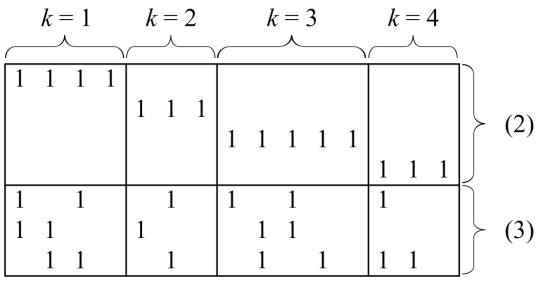

Figure 3: Structure of an A-Matrix of the TDRP

In order to demonstrate the theorem, let us examine a 4×4 matrix A^1 on Figure 4. A^1 is a submatrix of A from Figure 3, which belongs to the driver $k = 1$. Rows with only zero entries are neglected. All $m \times l$ submatrices of A^1 for $3 \leq l \leq 4$ are shown on Figure 5. According to the theorem, the five matrices shown on Figure 5 must not contain any $l \times l$ nonsingular submatrices with the $\Pi_{\beta,l}$ property, where $\beta \geq 2$ and $3 \leq l \leq 4$, in order for A^1 to be perfect. As an example, let us induce all nonsingular $l \times l$ submatrices B_l for $l = 3$ from the 4×3 submatrix A^1_3. The four induced 3×3 submatrices are shown on Figure 6. All matrices are nonsingular.

The only one matrix among the four B_3 matrices with equal sums of rows and columns is B^3_3 with $\beta = 2$. The only row of A^1_3, which is not in B^3_3, is row 1, which is a crew constraint represented by a vector $(1, 1, 1)$. The row sum of row 1 is $3 > \beta$. Hence, A^1_3 does not contain any submatrices with the $\Pi_{\beta,l}$ property. The procedure of nonsingular matrix induction of $l \times l$ matrices with $l = 3, 4$ and equal sums of rows and columns is applied to A^2_3, A^3_3 and A^1_4. None of the matrices contains any submatrices with the $\Pi_{\beta,l}$ property. Hence, the matrix A^1 is perfect.

	1	2	3	4
1	1	1	1	1
2	1		1	
3	1	1		
4		1	1	

A^1

Figure 4: Driver $k = 1$ Submatrix A^1 of A.

Every submatrix A^k of A has a row corresponding to the train driver constraint (2). The sum of the row is either larger than the sum of any row in A^k or componentwise equal to a row in A^k. As observed in [9], the row corresponding to the

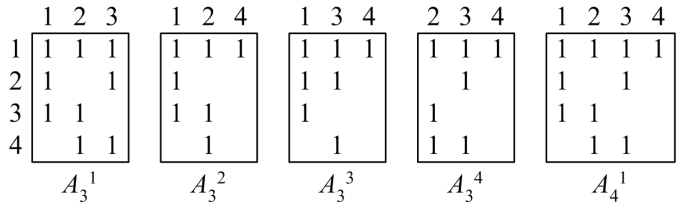

Figure 5: All $m \times l$ Submatrices of A^1 with $l = 3, 4$.

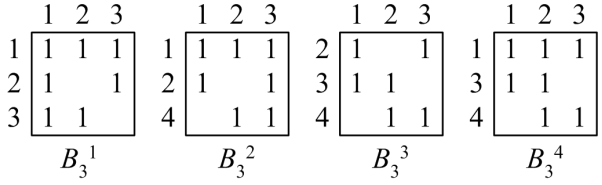

Figure 6: All $l \times l$ Submatrices of A_3^1 with $l = 3$.

crew constraint in the SPP formulation of the crew rostering problem is a "dominant" row in the submatrix of the crew member k, which prevents the existence of the $\Pi_{\beta,l}$ property and ensures that each A^k is perfect. Indeed, none of the $m \times l$ submatrices A_l, where $3 \leq l \leq n$ can induce a nonsingular submatrix B_l with the row sum strictly larger than the dominant row corresponding to (2). Therefore, due to the existence of the dominant train driver constraint, every driver submatrix A^k of A is perfect.

A zero-one matrix is called perfect if the polytope of the associated set partitioning problem has integral vertices only. Hence, fractional solutions will never appear within one driver's block of recovery duties. Any fractions in the TDRP-LP can therefore only occur between blocks of columns, belonging to different drivers. In other words, if a fractional solution occurs, it means that two or more drivers compete for one or more train tasks in their recovery duties. Two drivers can only compete for the same train task, if both drivers are available at the departure station prior the time of the train departure. As mentioned in Section 2.2, in the regular S-tog's train driver schedule, the train tasks arriving at non-relief terminal stations have unique subsequences due to the geographical position of the stations and the train line pattern in the S-train network. Hence the number of train tasks each driver competes for with other drivers in the S-tog schedule is very limited. There is a high probability that solutions to many TDRP-LP are naturally integer.

3.2 Solution Approach

The strength of the LP relaxation of the TDRP is the reason for choosing the solution approach based on solving the set partitioning formulation of the the Train Driver Recovery Problem with an LP-based Branch & Price algorithm. In the best case the solution to the TDRP-LP is integer. Under any circumstances lower bounds in the Branch & Price algorithm are very tight. There is therefore a good chance that the integer problem is solved with only a few iterations of the Branch & Price algorithm.

The size of the TDRP grows exponentially with the number of drivers and the number of train tasks, which are used to generate recovery duties for the set partitioning problem. In order to be able to solve the TDRP in realtime, the size of the problem must be kept reasonably small. When a disruption occurs, only a small number of drivers is directly affected. These drivers comprise the initial set of drivers K. There is no reason to involve *all* train drivers in the optimization problem, since a large part of the original train driver schedule will not be modified in the recovery solution. The number of train tasks used in the recovery duty generation is limited by the length of the *recovery period*, i.e. the time window, which bounds the part of the train driver schedule to be recovered. The recovery period must be sufficiently long to be able to achieve a feasible solution, but still as short as possible in order to limit the time period of re-scheduling and get back to original schedule as soon as possible. Train tasks assigned to the initial set of drivers within the recovery period define the initial set of train tasks N involved in the recovery. If the initial number of drivers is not enough to cover all train tasks in N, the TDRP is expanded by gradually involving other drivers to the problem. If no duties can be generated for one or more drivers, the recovery period is extended.

Since disruptions might occur one after another during the day, the prototype for the Train Driver Recovery DSS is based on solving TDRP instances sequentially. The changes caused by the first disruption are applied to the undisturbed schedule. The first TDRP is resolved over a defined set drivers and trains. As the solution is achieved, the driver schedule is modified according to that solution. When another disruption occurs, the changes caused by the second disruption are applied to the "new" schedule, which contains the solution to the previous TDRP. The new instance of the TDRP is resolved over a new set of trains and drivers. The schedule is again modified accordingly. The continuous recovery process is illustrated in Figure 7.

Since the instances of the TDRP are solved sequentially whenever a new disruption occurs, the term *original schedule* refers to the train driver schedule obtained by solving the previous instance of the TDRP or, in case of the first daily disruption, to the regular schedule employed at the beginning of the day. Likewise,

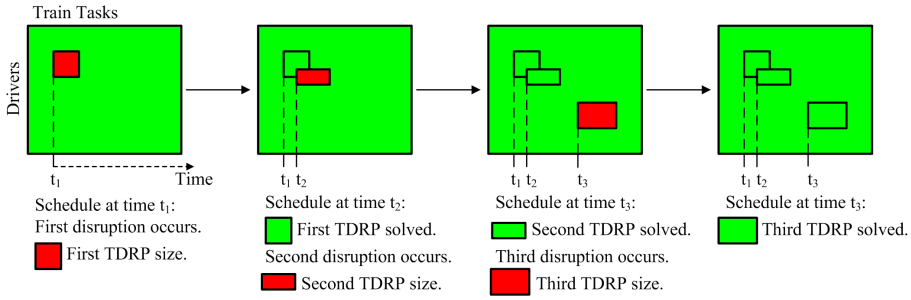

Figure 7: The Continuous Train Driver Recovery Process.

the *original duty* of a driver is the duty generated for the driver in the previous solution of the TDRP or the unchanged duty from the regular schedule.

4 Recovery Duty Generation

4.1 Duty Graph Construction

Recovery duties for each driver $k \in K$ are generated by assigning train tasks from the set N, which contains non-cancelled train tasks originally scheduled for drivers in K within the recovery period. A recovery duty must start and end only at a relief station, because the subsequences of train tasks arriving to the non-relief stations are unique and hence not interesting from the optimization point of view. Therefore, the recovery period determines both the time and the station, where the driver k must be available at the end of his recovery duty. The recovery period is very likely to cover one or two originally scheduled breaks for drivers in K. A recovery duty is *feasible* only if all originally scheduled driver's breaks are held, each break starts no later than originally scheduled and the duration of the break corresponds to the originally scheduled duration. The break can be held at either of the two crew depots, no matter which crew depot the driver is originally assigned to.

In order to generate recovery duties, a directed acyclic *duty graph* $G^k = (V^k, A^k)$ is constructed for every driver $k \in K$. The set of vertices V^k contains the source vertex o^k, the sink vertex d^k and a set of train task vertices N^k. The source vertex $o^k \in V^k$ represents the last task in the original duty of the driver k before the start time of the recovery period or the check-in task of k, if the driver's duty starts within the recovery period. The sink vertex $d^k \in V^k$ represents the check-out task of the driver k if the driver's duty ends within the recovery period or the first train task in the driver's original duty after the end of the recovery pe-

riod. Train tasks in the vertex set $N^k \subset V^k$ are collected from the train task set N. A train task $i \in N$ is included in N^k, if the departure time of i is larger than or equal to the arrival time of the driver's source vertex o^k and if the arrival time of i is less than or equal to the departure time of the sink vertex d^k of driver k.

The set of arcs A^k represents connections to feasible train task subsequences. The task w is a feasible subsequence of v if the departure time of w is later than or equal to the ready time of v and if the departure station of w coincides either with the arrival station of v or the arrival station of the passengering task taken by the driver from the arrival station of v to the departure station of w. The *ready time* $t_{\text{ready}}(v, w)$ is subsequence-dependent and is calculated as the arrival time of v plus a certain time span, which is necessary for the driver to conduct intermediate actions after finishing the task v in order to begin the task w. As an example shown on Figure 8, if the driver holds a full break between the train task v and the train task w and the arrival station of v is the same as the departure station of w, a time for handing the train over to another driver (abbreviated PIF), walking from the platform to the crew depot (BEV), holding the full break (PAU), walking from the crew depot back to the platform (BEV) and taking over the train from another driver (PIT) is added to the arrival time of the train task v in order to calculate $t_{\text{ready}}(v, w)$.

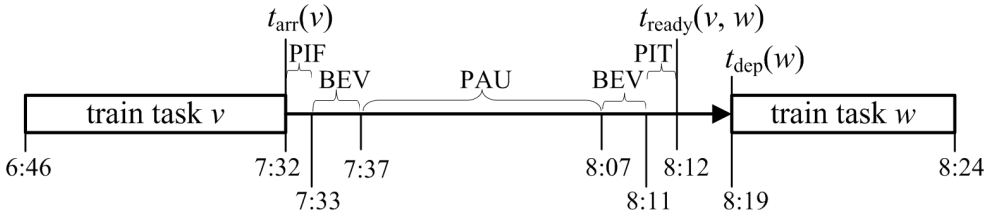

Figure 8: Feasible Subsequence Without Deadheading

An example on Figure 9 shows the ready time $t_{\text{ready}}(v, w)$ of the check-in task v, where the subsequent train task w departs from a different station than the check-in depot station of v and w is the earliest morning task of the particular train unit. The driver needs to walk from the depot to the platform of the passengering train task or to a taxi stand (BEV), ride as a passenger on a train or in a taxi to the departure station of w (PAS), walk to the departure platform (BEV) and make the train ready for driving (KLG). Since platforms on some stations are situated further away from the depot and each other than platforms on other stations, minimum required durations of BEV and PIT are station-dependent.

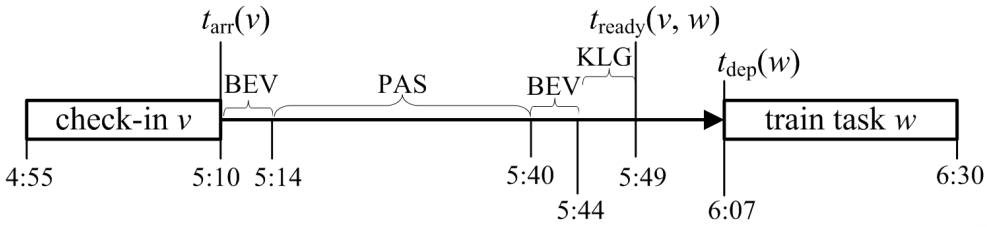

Figure 9: Feasible Subsequence With Deadheading

The difference between the departure time $t_{\text{dep}}(w)$ and the ready time $t_{\text{ready}}(v, w)$ is called the *idle time*. A feasible subsequence without any idle time is very tight and non-robust with respect to even small delays. On the other hand, a very long idle time is not very attractive either, particularly on terminal stations without crew facilities. In order to control the tightness of the schedule and the unwanted idle time, a minimum required idle time $\tau_{\text{idle}}^{\text{min}}$ and a maximum allowed idle time $\tau_{\text{idle}}^{\text{max}}$ are defined. For example, by setting $\tau_{\text{idle}}^{\text{max}} = 15$ min, situations where the driver waits for his next assignment for more than 15 minutes are avoided and many unnecessary arcs are not constructed. An arc (v, w) is only feasible if the departure time of w satisfies the inequality (5).

$$t_{\text{ready}}(v, w) + \tau_{\text{idle}}^{\text{min}} \;\leq\; t_{\text{dep}}(w) \;\leq\; t_{\text{ready}}(v, w) + \tau_{\text{idle}}^{\text{max}} \tag{5}$$

The arcs in A^k reflect the majority of allowed task in the S-tog train driver schedule. The overview of arcs which do not require a deadheading is presented in Table 1.

Arc Type	Ready Time $t_{\text{ready}}(v, w)$	Special Requirements
Immediate Subsequence	$t_{\text{arr}}(v)$	w is first train of same line after v, in opposite direction from end stations.
Train Change	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{PIT}}(v.\text{ArrSt})$	
Break Opportunity	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{break}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{PIT}}(v.\text{ArrSt})$	v.ArrSt is a crew depot. Short half-break: $\tau_{\text{break}} = 20$ min, long half-break: $\tau_{\text{break}} = 25$ min, full break: $\tau_{\text{break}} = 30$ min.
Check-In	$t_{\text{arr}}(o^k) + \tau_{\text{BEV}}(o^k.\text{ArrSt}) + \tau_{\text{PIT}}(o^k.\text{ArrSt})$	o^k is a check-in task.
Check-Out	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(d^k.\text{DepSt})$	d^k is a check-out task.
Reserve	$t_{\text{arr}}(v)$	$\tau_{\text{idle}}^{\text{min}} = 20$ min, $\tau_{\text{idle}}^{\text{max}} = \infty$

Table 1: Arcs Without Deadheadings.

If the arrival station of v is different from the departure station of w, a deadheading is required. A passengering arc exists if there is a non-cancelled train u in the schedule, which can transport the driver from one station to another. Overviews of passengering and taxi arcs and their ready times are presented in Table 2 and Table 3 respectively. Special requirements outlined for arcs without deadheadings in Table 1 are also applied to passengering and taxi arcs. In the current version of the prototype for the Train Driver Recovery DSS it is however refrained from adding taxi arcs in the graph construction.

Arc Type	Ready Time $t_{\text{ready}}(v,u)$	Ready Time $t_{\text{ready}}(u,w)$
Passengering	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt})$	$t_{\text{arr}}(u) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Passengering To Break	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt})$	$t_{\text{arr}}(u) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{break}} + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Break To Passengering	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{break}} + \tau_{\text{BEV}}(v.\text{ArrSt})$	$t_{\text{arr}}(u) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Check-In Passengering	$t_{\text{arr}}(o^k) + \tau_{\text{BEV}}(o^k.\text{ArrSt})$	$t_{\text{arr}}(u) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Passengering Check-Out	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt})$	$t_{\text{arr}}(u) + \tau_{\text{BEV}}(d^k.\text{DepSt})$
Passengering Reserve	$t_{\text{arr}}(v) + \tau_{\text{BEV}}(v.\text{ArrSt})$	$t_{\text{arr}}(u) + \tau_{\text{BEV}}(w.\text{DepSt})$

Table 2: Passengering Arcs.

Arc Type	Ready Time $t_{\text{ready}}(v,w)$
Taxi	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{taxi}}(v.\text{ArrSt}, w.\text{DepSt}) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Taxi To Break	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{taxi}}(v.\text{ArrSt}, w.\text{DepSt}) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{break}} + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Break To Taxi	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{break}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{taxi}}(v.\text{ArrSt}, w.\text{DepSt}) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Check-In Taxi	$t_{\text{arr}}(o^k) + \tau_{\text{BEV}}(o^k.\text{ArrSt}) + \tau_{\text{taxi}}(o^k.\text{ArrSt}, w.\text{DepSt}) + \tau_{\text{BEV}}(w.\text{DepSt}) + \tau_{\text{PIT}}(w.\text{DepSt})$
Taxi Check-Out	$t_{\text{arr}}(v) + \tau_{\text{PIF}} + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{taxi}}(v.\text{Arr}, d^k.\text{DepSt}) + \tau_{\text{BEV}}(d^k.\text{DepSt})$
Taxi Reserve	$t_{\text{arr}}(v) + \tau_{\text{BEV}}(v.\text{ArrSt}) + \tau_{\text{taxi}}(v.\text{ArrSt}, w.\text{DepSt}) + \tau_{\text{BEV}}(w.\text{DepSt})$

Table 3: Taxi Arcs.

4.2 Resource Constrained Path Generation

A directed path p from the source vertex o^k to the sink vertex d^k of the duty graph G^k is called *resource constrained*, if for each meal break originally scheduled to be held within the recovery time in the original driver duty there exists at least one break opportunity arc or one deadheading-to-break arc or one break-to-deadheading arc (v, w), such that the arrival time of v is smaller than the originally scheduled start time of the break, and the break arc (v, w) is suitable for the originally scheduled break type (a short half-break, a long half-break or a full break). A resource constrained path $p \in P^k$ represents a feasible recovery duty for the driver $k \in K$.

Resource constrained paths are generated with a *duty generation algorithm* based on dynamic programming. The algorithm runs through the list of labels L^k. Every label $l(v) \in L^k$ stores sufficient information for backtracing a feasible subpath $p_l(v)$ from the vertex v to the source vertex o^k. A subpath is a directed path from o^k to the vertex v. A subpath is feasible if it is not in a conflict with the break constraint within the time interval $[t_{\text{arr}}(o^k); t_{\text{arr}}(v)]$. A label $l(v)$ contains the cost of the subpath $p_l(v)$ represented by the sum of vertex and arc costs, the pointer to the predecessor label $l(u)$ of $l(v)$ for backtracing and a set of outstanding resources, representing the number, types and originally scheduled start times of breaks that are still due after $t_{\text{arr}}(v)$.

Label in L^k are treated in a sequential order. During the label treatment of $l(v)$ feasibility checks are applied on outgoing arcs of the vertex v. The feasibility check procedure determines if the subpath $p_l(v)$ can be extended along the arc (v, w) to the subsequent vertex w. The feasibility check starts with determining if a full break is still due after $t_{\text{arr}}(v)$ in the duty. If it is, the algorithm checks if the arc (v, w) is a break opportunity arc corresponding to a full break. If the arc provides a break opportunity for a full break, the subpath $p_l(w) = p_l(v) + (v, w)$ is feasible. If the arc does not provide a break opportunity for a full break and the earliest scheduled break start time is due prior the arrival time of the train task w, the subpath $p_l(w) = p_l(v) + (v, w)$ is infeasible. However, if the earliest scheduled break start time is not due before the arrival time of the train task w, the subpath $p_l(w)$ is considered to be feasible. The feasibility of the arc (v, w) with respect to the outstanding half-breaks is checked likewise. If no breaks are due after $t_{\text{arrival}}(v)$ in the original duty, the subpath $p_l(w)$ is feasible.

If the subpath $p_l(v)$ can be extended along the arc (v, w) to the vertex w, a new label $l(w)$ is created, representing the subpath $p_l(w) = p_l(v) + (v, w)$. The label $l(v)$ is the predecessor label of $l(w)$. The set of outstanding resources in $l(w)$ is updated according to the resource use along the arc (v, w). The cost of the subpath $p_l(w)$ is the cost of the predecessor subpath $p_l(v)$ plus the cost of the arc (v, w) plus the cost of the vertex w. The label $l(w)$ is added to the label list L^k

and is treated in turn.

Every label belonging to the sink vertex d^k stores a subpath $p_l(d^k)$, which corresponds to a resource constrained path p from o^k to d^k and can be added to the set of feasible recovery duties P^k. The path p is backtraced from d^k using predecessor labels.Figure 10 shows an example of a small graph with an application of the duty generation algorithm. All generated labels are shown next to vertices. The index i of a label $l(v)^i_j$ indicates the label number in the label list, while the index j points at the predecessor label of $l(v)^i_j$.

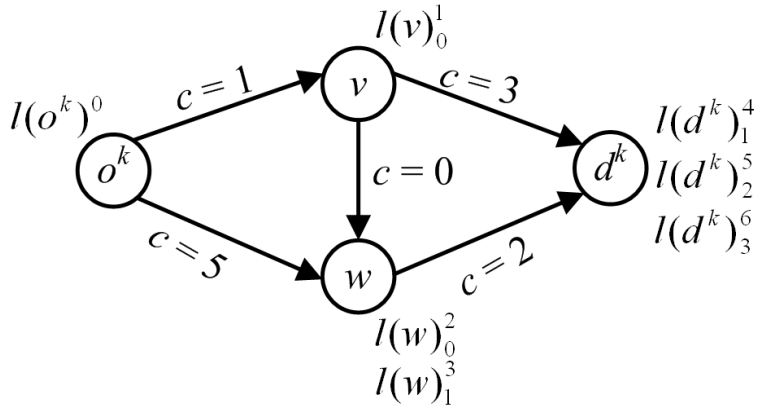

Figure 10: Labelled Vertices in the Duty Generation.

There are three labels, which belong to the sink vertex d^k. It means that there are three directed paths from o^k to d^k, which represent feasible recovery duties. The path $o^k \rightarrow v \rightarrow d^k$ is backtraced from the label $l(d^k)^4_1$ to the label $l(v)^1_0$, which is in turn backtraced to the source label $l(o^k)^0$. The path $o^k \rightarrow w \rightarrow d^k$ is backtraced from the label $l(d^k)^5_2$ to the label $l(w)^2_0$, which is in turn backtraced to the source label $l(o^k)^0$. The path $o^k \rightarrow v \rightarrow w \rightarrow d^k$ is backtraced from the label $l(d^k)^6_3$ to the label $l(w)^3_1$, which is in turn backtraced to the label $l(v)^1_0$, backtracing to the source label $l(o^k)^0$.

The duty generation algorithm terminates when some termination criteria is satisfied. If a *total enumeration* of feasible recovery duties is required, the algorithm terminates when all labels in L^k are treated and hence all resource constrained paths from o^k to d^k are collected. If only a certain number of feasible recovery duties is required, the algorithm terminates when the number of collected paths reaches the required number.

The *path cost* $c(p)$ is a sum of vertex and arc costs as expressed in equation (6). The sum of arc costs of the path p represents *recovery duty cost* c^k_p. The cost

of an arc (v, w) in G^k reflects the unattractiveness of the subsequence w after a task v in the recovery duty of the driver k. The objectives set for the Train Driver Recovery Problem are expressed through the sizes of costs assigned to different arcs in the duty graph.

$$c(p) = [c(o^k) + \sum_{i=1}^{h} c(v_i) + c(d^k)] + [c(o^k, v_1) + \sum_{i=1}^{h-1} c(v_i, v_{i+1}) + c(v_h, d^k)] \quad (6)$$

In order to minimize the number of modifications from the original schedule, an arc (v, w) is assigned a zero cost, if it appears in the original duty of the driver k, i.e., if both train tasks v and w belong to the original driver duty. A slightly less preferable is a semi-original arc (v, w), where the train task v is *not* in the original duty of the driver, but the task w is originally scheduled in the duty of the driver. Semi-original arcs are attractive since there is a chance that the driver continues with his original duty after completion of the task w. Immediate subsequence arcs, which do not require changing the rolling stock, are attractive from the recovery robustness point of view. In order to increase the robustness of the recovery duties, the minimum required idle time $\tau_{\text{idle}}^{\text{min}}$ is either set high, for example to 10 minutes, or all arcs with a short idle time are given a high cost. In order to avoid driver's waiting time at terminal stations without crew facilities, either the maximum allowed idle time $\tau_{\text{idle}}^{\text{max}}$ is set low, for example to 15 minutes, or arcs with a long idle time are given a high cost. In general, deadheading arcs are not very attractive, since the time spent on deadheading is the time spent without working. Taxi arcs are more costly than passengering arcs, since an extra expense for a taxi ride is required.

5 Solving TDRP with Branch & Price

Branch & Price is a method for solving large integer programming problems, where the LP-relaxation of the IP problem is solved with column generation at each node of the Branch & Bound tree. A general Branch & Price methodology is described in [1]. Solving the TDRP-LP involves dynamic column and constraint generation. The solution process is illustrated on Figure 11. The column generation part of the algorithm consists of solving two problems in turn. The *restricted master problem* (RMP) is the TDRP-LP with a restricted set of variables and the subproblem is the *pricing problem*, which generates columns with negative reduced cost. For a review of applications of dynamic column generation see [6].

The initial set of columns for the RMP at the root node of the Branch & Bound tree is generated as resource constrained paths on duty graphs G^k for all drivers

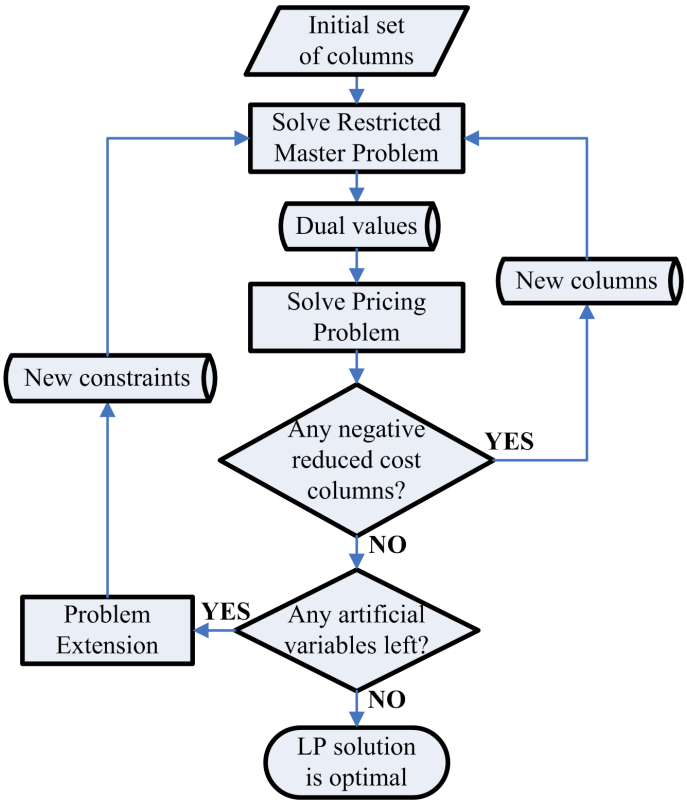

Figure 11: Solving the TDRP-LP.

in K using a *limited subsequences strategy*. The strategy is similar to the one described in [8]. Feasible recovery duties for drivers are constructed under the restriction that the number of choices for performing different tasks after finishing a task in the duty is limited, even though many possibilities (i.e. many subsequences) might exist. The limited subsequences strategy is represented by an additional restriction in the duty generation algorithm, described in Section 4.2. At every label treatment a feasibility check is applied only to a limited number of outgoing arcs of the vertex v. The number of outgoing arcs is limited to η, which is a small number compared to the number of outgoing arcs from v.

The outgoing arcs for a feasibility check can be chosen at random, but we prefer to force the attractive recovery duties to be included to the optimization problem at an early stage of the column generation. Outgoing arcs of every vertex $v \in V^k$ are therefore sorted in an ascending order by their cost. Hence, at most η cheapest arcs are investigated. The limited subsequences strategy allows to gather a small set of attractive recovery duties fast.

A limited subsequences strategy is illustrated on Figure 12. The duty gener-

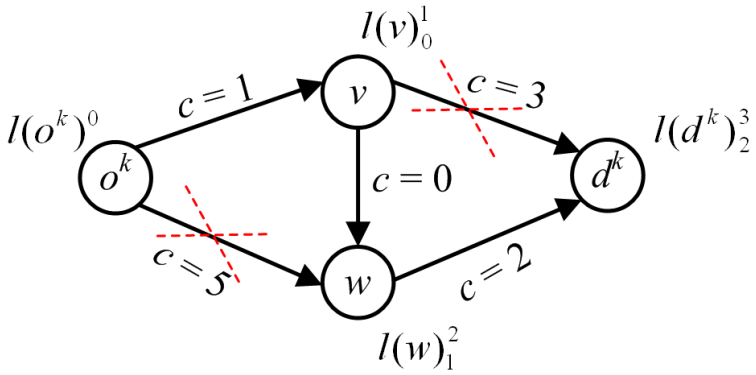

Figure 12: Vertices Labelled with the Limited Subsequences Strategy $\eta = 1$.

ation algorithm with a limited subsequences strategy, where $\eta = 1$, is applied on the graph example from Figure 10, described in Section 4.2. The number of labels is decreased from 7 to 4 compared to the total enumeration shown on Figure 10 and the number of paths is decreased from 3 to 1, so the only generated path is $o^k \rightarrow v \rightarrow w \rightarrow d^k$ with the cost of 3. The cost c_p^k of a column, represented by a variable x_p^k, is a linear sum of arc and vertex costs of the path $p \in P^k$. The costs of vertices are set to zero when the initial set of columns is generated.

The RMP is solved using the primal revised simplex solver of MOSEK Optimization Software, version 4.0. In order to ensure feasibility of the linear problem, artificial variables are added to the RMP. One artificial variable f_i is added for each train task $i \in N$ and one variable e^k for each driver $k \in K$. The TDRP-LP is formulated in (7) - (12). Due to the very large cost M the artificial variables are only included in the optimal linear programming solution if their presence is necessary for the problem feasibility.

(TDRP-LP)
$$\min \sum_{k \in K} \sum_{p \in P^k} c_p^k x_p^k + \sum_{k \in K} M e^k + \sum_{i \in N} M f_i \tag{7}$$

subject to

$$\sum_{p \in P^k} x_p^k + e^k = 1 \qquad \forall k \in K \tag{8}$$

$$\sum_{k \in K} \sum_{p \in P^k} a_{ip}^k x_p^k + f_i = 1 \qquad \forall i \in N \tag{9}$$

$$x_p^k \geq 0 \qquad \forall p \in P^k,\ \forall k \in K \tag{10}$$

$$e^k \geq 0 \qquad \forall k \in K \tag{11}$$

$$f_i \geq 0 \qquad \forall i \in N \tag{12}$$

The values of the dual variables, corresponding to train driver constraints (8) and train task constraints (9) are used in the pricing step of the column generation algorithm. At every pricing step, described in details in Section 5.1, a set of columns with negative reduced costs is constructed by a column generator written in C#.NET. When the pricing problem cannot generate any negative reduced cost columns, the optimal solution to the RMP is an optimal solution of the master problem TDRP-LP.

5.1 Pricing Problem

Let λ^k be the dual variable corresponding to the k'th train driver constraint (8) and let π_i be the dual variable corresponding to the i'th train task constraint (9) in the restricted master problem of the TDRP-LP. The *reduced cost* $\bar{c}_p^k$ of the variable x_p^k, which represents a recovery duty $p \in P^k$ for the driver $k \in K$ is:

$$\bar{c}_p^k = c_p^k - \lambda^k - \sum_{i \in N} a_{ip}^k \pi_i \tag{13}$$

The value of the dual variable λ^k with an opposite sign is set as the cost of the source vertex o^k of the graph G^k: $c(o^k) = -\lambda^k$. Values of dual variables π_i for $i \in N^k$ with opposite signs are set as costs on train task vertices in G^k: $c(v_i) = -\pi_i$ for each vertex $v_i \in N_p^k$, where N_p^k is the set of train vertices in the path p. The cost of the sink vertex d^k remains zero: $c(d^k) = 0$. According to (6), the cost of the resource constrained path $p \in P^k$ for the driver $k \in K$ is:

$$\begin{aligned} c(p) &= [c(o^k) + \sum_{i \in N_p^k} c(v_i) + c(d^k)] + c_p^k \\ &= [-\lambda^k + \sum_{i \in N_p^k} (-\pi_i) + 0] + c_p^k \\ &= c_p^k - \lambda^k - \sum_{i \in N_p^k} \pi_i \end{aligned} \tag{14}$$

Using (14), the reduced cost $\bar{c}_p^k$ is expressed as:

$$\begin{aligned} \bar{c}_p^k &= c_p^k - \lambda^k - (\sum_{i \in N_p^k} 1 \cdot \pi_i + \sum_{i \in N \setminus N_p^k} 0 \cdot \pi_i) \\ &= c_p^k - \lambda^k - \sum_{i \in N_p^k} \pi_i - 0 \\ &= c(p) \end{aligned} \tag{15}$$

Hence, the reduced cost of the variable representing a recovery duty for a driver k is the cost of the resource constrained path p generated on the duty graph G^k with vertex costs represented by dual values with opposite signs. The pricing problem identifies variables in the TDRP-LP with $\bar{c}_p^k < 0$ by collecting resource constrained paths in duty graphs with $c(p) < 0$.

Following pricing strategies are implemented:

- PS1 At each pricing iteration, scan all duty graphs until at most θ columns with negative reduced cost are collected.
- PS2 *Partial pricing*: Collect at most θ columns with negative reduced cost from one duty graph at each pricing iteration. Each duty graph is scanned in turn. If no negative reduced columns exist for a particular driver at a particular pricing step, the duty graph for the next driver on the list is scanned.
- PS3 Collect at most one column from each duty graph. Every collected column represents the most negative reduced cost column in the duty graph.
- PS4 Collect the most negative reduced cost column among all columns available.
- PS5 Collect all negative reduced cost columns from all duty graphs.

PS3 and PS4 employ a *resource constrained shortest path algorithm* on a direct duty graph G^k, which is implemented as a label setting algorithm, where only a set of the dominating labels for each vertex v are kept, the other labels are removed from the label list. A label $l(v)$ dominates label $l(w)$ if following conditions are satisfied: the cost of $l(v)$ is less than or equal to the cost of $l(w)$, the number of outstanding short half-breaks in $l(v)$ equals to the number of outstanding short half-breaks in $l(w)$, the number of outstanding long half-breaks in $l(v)$ equals to the number of outstanding long half-breaks in $l(w)$ and the number of outstanding full breaks in $l(v)$ equals to the number of outstanding full breaks in $l(w)$. This dominance criteria is week, since the fulfilment of a one type of break constraint cannot substitute the other type. It means that not many labels

are removed from the label list during a run of the algorithm and the label setting algorithm in the worst case corresponds to the total enumeration of recovery duties. Hence, the two pricing strategies are not very efficient for this problem.

PS3, PS4 and PS5 correspond to *full pricing* strategies, where all are scanned at each pricing iteration. PS5 is the worst of all implemented strategies. In the beginning of the column generation algorithm many non-basic columns have negative reduced cost. However, they do not necessarily end in the optimal solution to the TDRP-LP. Adding all negative reduced cost columns results in a huge size of the RMP and the concept of dynamic column generation looses its essence. All three full pricing strategies are also computationally expensive and are therefore dismissed at an early stage of the project research.

PS1 and PS2 are almost similar performance-wise. However, as the sizes of TDRP's grow, the partial pricing strategy PS2 demonstrates a better performance, because it uses less time per pricing iteration by scanning one graph at a time unless no negative cost columns are collected from a particular graph. Even though PS1 can settle with fewer pricing iterations, each pricing step takes longer time and, as a general rule, more columns are added. The partial pricing strategy PS2 is therefore selected for further work.

5.2 Problem Extension

If an artificial variable f_i remains present in the optimal solution to the TDRP-LP, it implies that the train task i is not included in any generated feasible recovery duty. Hence none of the present drivers in the set K can cover the train task i and train task has to be assigned to a driver outside the set K. The obvious choice of a new driver to be added to the problem is the one in reserve during the execution of the train task i. If no reserve drivers are available, active drivers who happen to be in the geographical "neighborhood" of the i's departure station are added. More than one driver may be added at the same time.

Introducing another driver k' to the problem corresponds to adding a train driver constraint (16) to the TDRP-LP and constructing a duty graph $G^{k'}$. If the driver k' is active during the whole or part of the recovery period, the set of train tasks N', which are assigned to the driver in his original duty within the recovery period, is included in the set of train tasks N. A set of train task constraints (17) are added to the TDRP-LP and all duty graphs are updated with vertices corresponding to train tasks from the set N' as described in Section 4.1.

$$\sum_{p \in P^k} x_p^k + e^k = 1 \quad \text{for } k = k' \tag{16}$$

$$\sum_{k \in K} \sum_{p \in P^k} a_{ip}^k x_p^k + f_i = 1 \quad \forall i \in N' \tag{17}$$

When the new constraints are added to the problem, the column generation process continues until the problem is solved to optimality. If a driver k' is not able to cover the train task i either, the expansion problem is solved again by adding another driver or drivers to the set K. Since the costs of original duties are zero, all train tasks in the set N' will be covered by the driver k'. Hence, the expansion problem does not contribute to more modifications of the driver schedule than necessary.

If an artificial variable e^k is present in the optimal solution the TDRP-LP, it implies that no feasible recovery duty could be generated for the driver k within the recovery period. If the sink vertex d^k of the driver k corresponds to the check-out activity, the infeasibility can only be eliminated by delaying the check-out start time, which is acceptable if no other choice exists in a recovery situation. In this case constraints corresponding to the driver k and train tasks, which belong to the driver in the original schedule, are removed from the optimization problem. If the sink vertex of the driver k is represented by a train task, the recovery period is extended with a certain time period instead of delaying d^k. The train task, which corresponds to the sink vertex d^k before the extension, becomes an ordinary train vertex. The recovery period extension corresponds to adding train tasks from original duties of drivers in K to the set N, which results in adding train task constraints to the TDRP-LP. All duty graphs are updated for further pricing iterations. In order to avoid recovery period expansion, the length of the recovery period must be chosen carefully from the beginning.

5.3 Finding Integer Solutions

Any fractions which occur in the optimal solution of the TDRP-LP are resolved by applying a *constraint branching strategy* originally proposed by [10], which is very useful in the context of achieving integer solutions for set partitioning problems. As shown in Section 3.1, the fractions occur in the TDRP-LP only across train drivers' blocks of columns. It is therefore sensible to force one driver k to cover a train task i, which also appears in another driver's optimal recovery duty while forbidding other drivers to include i in their recovery duties.

We define $J(r, s)$ as a set of variables from the optimal fractional solution to the TDRP-LP, which cover the train driver constraint r and the train task constraint s simultaneously and $\Pi_{(r,s)} = \sum_{j \in J(r,s)} x_j$ as the sum of solution values of the variables in the set $J(r, s)$. The program identifies all constraint pairs $\{r, s\}$, for which $\Pi_{(r,s)}$ lies strictly between zero and one:

$$0 < \sum_{j \in J(r,s)} x_j < 1 \tag{18}$$

The 1-branch forces both constraints r and s to be covered together simultaneously, which is expressed through (19). The 0-branch expressed through (20) implies that the driver constraint r must not be covered by the same variable as the train task constraint s.

$$\sum_{j \in J(r,s)} x_j = 1 \tag{19}$$

$$\sum_{j \in J(r,s)} x_j = 0 \tag{20}$$

In the contest of the TDRP, the constraint r is a train driver constraint, while the constraint s is a train task constraint. Hence, the 1-branch forces while the 0-branch forbids the driver corresponding to r to cover the train task corresponding to s in recovery duties. As suggested in [8], the pair of constraints with the largest (i.e., closest to 1) sum $\Pi_{(r,s)}$ is to be chosen for branching. The larger the sum, the stronger the inclination of the optimal solution to the TDRP-LP about assigning the driver corresponding to the constraint r to the train task corresponding to the constraint s. Therefore, if a depth-first search of the Branch & Bound tree is implemented, there is a high probability for many 0-branches to be fathomed because 1-branches are preferred by optimal LP solutions of parent nodes.

Branching constraints (19) and (20) are not explicitly added to the TDRP-LP. Instead, in a 1-branch, the columns covering either the driver constraint r or the train constraint s, but not both, are removed from the optimization problem by setting the upper bounds of the violating variables in the RMP to zero. In a 0-branch, all columns covering both constraints r and s simultaneously are eliminated.

The column generation continues in every node of the Branch & Bound tree. The branching restrictions are applied to the column generator used in the pricing problem. Since both branching decisions involve forcing and forbidding some train tasks to be covered by some drivers, a set N^k_{force} of *forced* train tasks and a set N^k_{forbid} of *forbidden* train tasks are generated for every driver $k \in K$. Both sets are updated at each node of the Branch & Bound tree. On a 1-branch, a train task i is added to the set N^k_{force} of a driver k, if i and k correspond to the set partitioning constraint pair $\{r, s\}$. At the same time, a train task i is added to the set $N^{k'}_{\text{forbid}}$ of another driver k', if i corresponds to the constraint s, but the driver k' does not correspond to the constraint r. On a 0-branch, a train task i is added to the set N^k_{forbid} of a driver k, if i and k correspond to the constraint pair $\{r, s\}$.

During the duty generation algorithm on the graph G^k, an arc (v, w) is not examined at the label treatment of $l(v)$ if the train task represented by the vertex w belongs to the set of forbidden train tasks N^k_{forbid} of the driver k. Hence, none of the paths containing the train task w are generated in the duties of the driver k. For example, none of the generated paths contains the vertex w from the graph example on Figure 13. At the same time, if any train task represented by a vertex

u belongs to the set N^k_{force} of the driver, only paths containing u are returned to the restricted master problem. On the graph example on Figure 13 the only generated path is the one containing vertex u.

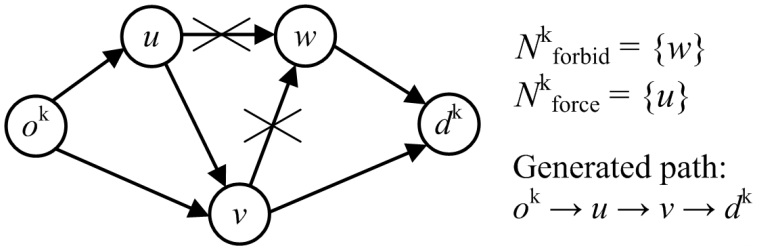

Figure 13: Graph Example with Forbidden and Forced Train Tasks.

6 Results

The prototype for the Train Driver Recovery DSS is implemented in C#.NET, calling the .NET interface of MOSEK 4.0. All tests are run on a Pentium 4 PC with 3,40 GHz and 1 GB RAM. All running times are measured in seconds. The running time represents the *real time* elapsed from the beginning to the end of the measured part of the program, i.e., CPU time consumed by other processes is included in the measurement.

ID	Recov. Period	#Duties	#N.-Canc. Trains	#Cancel. Trains	$\|K\|$	$\|N\|$	Graph Time (sec)
09:00_3A+	3h	141	326	36	14	9	00,09
09:00_4A+	4h	168	434	48	19	19	00,22
09:00_5A+	5h	196	544	60	24	29	00,41
09:00_6A+	6h	204	652	72	27	41	00,75
09:00_7A+	7h	206	760	84	30	71	02,42
09:00_8A+	8h	214	868	96	34	88	04,59
12:00_3A+	3h	123	326	36	14	16	00,16
12:00_4A+	4h	125	434	48	17	40	00,81
12:00_5A+	5h	133	542	60	21	54	01,89
12:00_6A+	6h	141	647	72	25	89	06,41
12:00_7A+	7h	164	743	84	27	112	11,61
12:00_8A+	8h	183	825	86	28	123	14,26

Table 4: Test Scenarios.

Twelve test scenarios of different sizes are presented in Table 4. Line cancellations are used to disrupt the train driver schedule. The period of cancellation in all test scenarios is equal to the length of the recovery period. Recovery periods of 3 to 8 hours are tested. Identification (ID) names of test instances illustrate the recovery start time, the length of the recovery period and the cancelled train line. Even though the S-tog train schedule is periodical, the number of train drivers at work vary during different time periods of the day. Scenarios with the same lengths of recovery periods starting at two different times of the day are therefore tested. The table presents the number of original duties and train tasks (cancelled and non-cancelled) within the recovery period and the number of drivers and train tasks in the initial sets K and N respectively. The last column of the table presents the time used in the graph construction phase, where the disruption is applied to the schedule, the train tasks and information about the duties within the recovery period are collected, the initial sets of drivers and train tasks are identified and all duty graphs are constructed. The running times are acceptable for all test instances.

Table 5 shows results from the optimization phase of the program. The number of nodes in the Branch & Bound tree, the total number of calls to the column generator (i.e., the total number of pricing iterations), the number of variables and constraints in the optimal integer solution and the running time of the Branch & Price algorithm are presented in the table.

ID	Integer LP	#Nodes in B&B	#ColGen Calls	#Var	#Const	B & P Time (sec)
09:00_3A+	TRUE	1	8	90	23	00,11
09:00_4A+	TRUE	1	7	145	38	00,11
09:00_5A+	TRUE	1	11	243	53	00,14
09:00_6A+	TRUE	1	17	311	68	00,19
09:00_7A+	TRUE	1	46	3 417	101	01,56
09:00_8A+	FALSE	13	135	6 848	122	36,08
12:00_3A+	TRUE	1	5	119	30	00,11
12:00_4A+	TRUE	1	19	589	57	00,23
12:00_5A+	FALSE	21	126	1 735	75	04,72
12:00_6A+	FALSE	5	86	20 584	114	37,84
12:00_7A+	TRUE	1	74	18 342	139	62,72
12:00_8A+	TRUE	1	86	30 354	151	101,02

Table 5: Branch & Price Results.

The test results of the solution approach to the set partitioning formulation of the TDRP are very encouraging. Due to small initial problem sizes and the dy-

namic column generation approach the running times of the linear programming optimization at every node of the Branch & Bound tree are very small. As expected, due to strong integer properties of the problem formulation, the majority of test scenarios (9 out of 12) produced integer solutions to the TDRP-LP. The constraint branching provides an integer solution only in a few iterations. The structure of the Branch & Bound tree, shown on Figure 14, exposes the usefulness of the constraint branching strategy described in Section 5.3, when the depth-first search of the Branch & Bound tree is applied. All 0-branches in the tree of 09:00_8A+ are fathomed by bound. The numbering of nodes in the tree on Figure 14 gives the order in which the nodes are examined. The tree on the left shows branching decisions. For instance, the root node is branched with the constraint pair $\{r, s\} = \{222, 2312\}$. The tree on the right shows upper and lower bounds.

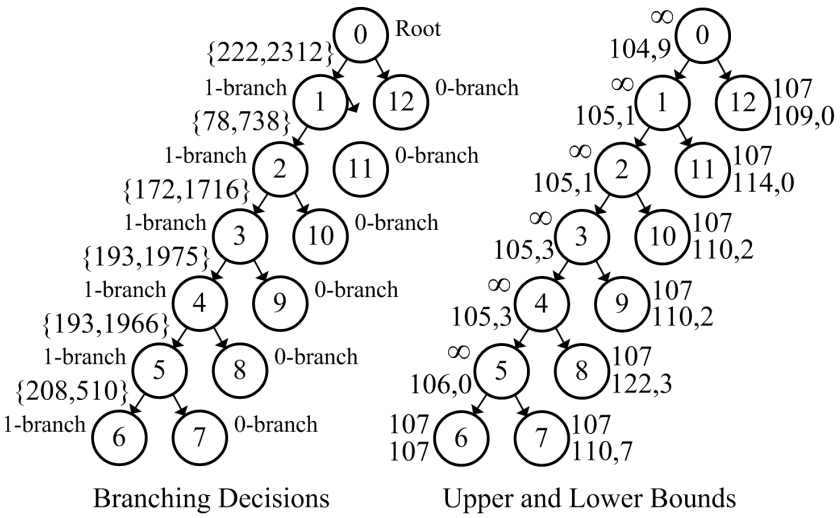

Figure 14: Branch & Bound Tree of 09:00_8A+.

The TDRP is often a feasibility problem. It is therefore not crucial to prove the optimality of the solution. It might be chosen to terminate the Branch & Price algorithm either as soon as the first integer solution is achieved or when the gap between the global lower bound and the achieved upper bound of the integer solution is less than a certain percentage, for instance 1-3%. An integer solution is then achieved faster, since all 0-branch nodes are not examined.

Test scenarios which cover 6-8 hours of the train driver schedule are presented in order to demonstrate how the problem size grows with the length of the recovery period and in order to test the solution method on larger instances. In real-life disruption situations, it is impossible to predict the disruption pattern that far into the future. Therefore, it does not make sense to recover the schedule 6-8 hours

ahead. Instead, the recovery shall cover the near future (for instance, 3-4 hours) in order to find an immediate re-scheduling of the drivers affected by the disruption. TDRP for test scenarios which cover 3-5 hours of the schedule are resolved within a few seconds. These running times are acceptable for employment of the Branch & Price solution method to the TDRP in S-tog's daily operations.

7 Conclusion

This report presents a set partitioning formulation of a Train Driver Recovery Problem and a solution method based on solving the problem with a Branch & Price algorithm. The problem is formulated over a small set of drivers and a certain recovery period, which bounds a part of the drivers' original duties. If the initial problem size is not sufficient for reaching a feasible recovery solution, other drivers are added to the problem or the recovery time is extended. The solution approach is tested on data from the Danish passenger railway operator DSB S-tog A/S. In test scenarios, a train line is cancelled for the whole recovery period, resulting in cancellation of all train tasks belonging to the line. Optimal integer solutions to test instances with the recovery period of 3-5 hours are achieved within 5 seconds.

The computational results show that optimization techniques can be advantageously applied to recovery problems. Exploiting the strong integer properties of the set partitioning formulation of the problem, integer solutions to the LP relaxation of the problem are achieved in the majority of test instances. The constraint branching strategy combined with a depth-first search of the Branch & Bound tree resolves the fractional solutions within a few iterations. The future research is focused on testing the prototype to the Train Driver Recovery Decision Support System on historical disruption data from the S-train network and on continuous improvement of data structures and algorithms of the system.

References

[1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research Management Science*, 46(3):316–329, 1998.

[2] E. R. Butchers, P. R. Day, A. P. Goldie, S. Miller, J. A. Meyer, D. M. Ryan, A. C. Scott, and Ch. A. Wallace. Optimized crew scheduling at Air New Zealand. *Interfaces*, 31(1):30–56, 2001.

[3] J. Clausen, A. Larsen, and J. Larsen. Disruption management in the airline industry - concepts, models and methods. Technical Report 01, Informatics & Mathematical Modelling, Technical University of Denmark, 2005.

[4] M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A column generation approach for large-scale aircrew rostering problems. *Operations Research Management Science*, 47(2):247–263, 1999.

[5] D. Huisman. A column generation approach for the rail crew re-scheduling problem. *European Journal of Operational Research*, 180(1):163–173, 2007.

[6] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research Management Science*, 53(6):1007–1023, 2005.

[7] M. W. Padberg. Perfect zero-one matrices. *Mathematical Programming*, 6(2):180–196, 1974.

[8] D. M. Ryan. The solution of massive generalized set partitioning problems in aircrew rostering. *The Journal of the Operational Research Society*, 43(5):459–467, 1992.

[9] D. M. Ryan and J. C. Falkner. On the integer properties of scheduling set partitioning models. *European Journal of Operational Research*, 35(3):442–456, 1988.

[10] D M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport*, pages 269–280. North-Holland Publishing Company, 1981.

[11] C. G. Walker, J. N. Snowdon, and D. M. Ryan. Simultaneous disruption recovery of a trian timetable and crew roster in real time. *Computers & Operations Research*, 32(8):2077–2094, 2005.