Branch and Price for Bus Driver Scheduling with Complex Break Constraints

Lucas Kletzander,[1] **Nysret Musliu,**[1] **Pascal Van Hentenryck**[2]

[1]Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien, Karlsplatz 13, 1040 Vienna, Austria
[2]Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, USA
{lkletzan,musliu}@dbai.tuwien.ac.at, pascal.vanhentenryck@isye.gatech.edu

Abstract

This paper presents a Branch and Price approach for a real-life Bus Driver Scheduling problem with a complex set of break constraints. The column generation uses a set partitioning model as master problem and a resource constrained shortest path problem as subproblem. Due to the complex constraints, the branch and price algorithm adopts several novel ideas to improve the column generation in the presence of a high-dimensional subproblem, including exponential arc throttling and a dedicated two-stage dominance algorithm. Evaluation on a publicly available set of benchmark instances shows that the approach provides the first provably optimal solutions for small instances, improving best-known solutions or proving them optimal for 48 out of 50 instances, and yielding an optimality gap of less than 1% for more than half the instances.

Introduction

Scheduling employees for public transport requires to incorporate a variety of complex constraints, particularly for breaks, time schedules, spatial requirements, and conflicting objectives, giving rise to difficult optimization problems. Companies require cost-efficient schedules, while drivers and labor unions request employee-friendly schedules to reduce stress and increase compatibility of the required shift work with the private life of the drivers.

This paper presents a Branch and Price approach for a complex real-life Bus Driver Scheduling (BDS) problem based on an Austrian collective agreement. The column generation uses set partitioning as the master problem and the resource constrained shortest path problem (RCSPP) as the subproblem. However, due to the complex set of constraints, the RCSPP has to deal with a large number of high-dimensional labels. This paper proposes several novel contributions to improve the generation of new columns for high dimensional pricing problems, including the splitting of the subproblem, a throttling scheme, and the use of k-d trees and bounding boxes in a two stage dominance algorithm.

The new method is evaluated on a publicly available set of benchmark instances. It provides the first provably optimal solutions for small instances and improves previous best-known solutions or proves them optimal for 48 out of 50 instances, resulting in an optimality gap of less than 1% for more than half the instances. It also provides a lower bound for the quality of existing metaheuristic approaches which are mostly within 4% of the optimum.

Related Work

There are many versions of employee scheduling problems and surveys by Ernst et al. (2004); Van den Bergh et al. (2013) provide a good overview of this line of work. Driver scheduling, as a part of crew scheduling, resides between vehicle scheduling and driver rostering in the process of operating bus transport systems (Ibarra-Rojas et al. 2015).

Research on bus driver scheduling problems has a long history (Wren and Rousseau 1995) and uses a variety of solution methods. Exact methods mostly use column generation with a set covering or set partitioning master problem and a resource constrained shortest path subproblem (Smith and Wren 1988; Desrochers and Soumis 1989; Portugal, Lourenço, and Paixão 2009; Lin and Hsu 2016). Heuristic methods like greedy (Martello and Toth 1986; De Leone, Festa, and Marchitto 2011; Tóth and Krész 2013) or exhaustive (Chen et al. 2013) search, tabu search (Lourenço, Paixão, and Portugal 2001; Shen and Kwan 2001), genetic algorithms (Lourenço, Paixão, and Portugal 2001; Li and Kwan 2003), or iterated assignment problems (Constantino et al. 2017) are used in different variations.

However, most work so far focuses mainly on cost, rarely minimizing idle time and vehicle changes (Ibarra-Rojas et al. 2015; Constantino et al. 2017). Break constraints are mostly simple, often including just one meal break. Complex break scheduling within shifts has been considered by authors in different contexts (Beer et al. 2008, 2010; Widl and Musliu 2014). There is not much work on multi-objective bus driver scheduling (Lourenço, Paixão, and Portugal 2001), but multi-objective approaches are used in other bus operation problems (Respício, Moz, and Vaz Pato 2013).

This paper investigates a complex real-life Bus Driver Scheduling Problem that was recently introduced by Kletzander and Musliu (2020). Previous work explains the need for a combined objective function that goes beyond cost. The published instances were solved with simulated annealing and a hill climbing heuristic. Although these methods provide good results, they cannot guarantee optimality or provide lower bounds.

ℓ	$tour_\ell$	$start_\ell$	end_ℓ	$startPos_\ell$	$endPos_\ell$
1	1	360	395	0	1
2	1	410	455	1	2
3	1	460	502	2	1
4	1	508	540	1	0

Table 1: A Bus Tour Example

Branch and Price is a decomposition technique for large mixed integer programs (Barnhart et al. 1998). This work uses set partitioning (Balas and Padberg 1976) as the master problem and the RCSPP (Irnich and Desaulniers 2005) as the subproblem. Resources are modelled via resource extension functions (REF) (Irnich 2008). Different techniques for dealing with pareto-front calculation can mostly be found as maxima-finding algorithms in a geometrical context (Bentley 1980; Chen, Hwang, and Tsai 2012).

Problem Description

The investigated Bus Driver Scheduling Problem deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day of operation. The problem specification is taken from Kletzander and Musliu (2020).

Problem Input

The bus routes are given as a set $\mathbf{L}$ of individual bus legs, each leg $\ell \in \mathbf{L}$ is associated with a tour $tour_\ell$ (corresponding to a particular vehicle), a start time $start_\ell$, an end time end_ℓ, a starting position $startPos_\ell$, and an end position $endPos_\ell$. The actual driving time for the leg is denoted by $drive_\ell$. The given instances use $drive_\ell = length_\ell = end_\ell - start_\ell$.

Table 1 shows a short example of one particular bus tour. The vehicle starts at time 360 (6:00 am) at position 0, does multiple legs between positions 1 and 2 with waiting times inbetween and finally returns to position 0. A valid tour never has overlapping bus legs and consecutive bus legs satisfy $endPos_i = startPos_{i+1}$. A tour change occurs when a driver has an assignment of two consecutive bus legs i and j with $tour_i \neq tour_j$.

A distance matrix specifies, for each pair of positions p and q, the time $d_{p,q}$ a driver takes to get from p to q when not actively driving a bus. If no transfer is possible, then $d_{p,q} = \infty$. $d_{p,q}$ with $p \neq q$ is called the *passive ride time*. $d_{p,p}$ represents the time it takes to switch tour at the same position, but is not considered passive ride time.

Finally, each position p is associated with an amount of working time for starting a shift ($startWork_p$) and ending a shift ($endWork_p$) at that position. The instances in this paper use $startWork_p = 15$ and $endWork_p = 10$ at the depot ($p = 0$). These values are 0 for other positions.

Solution

A solution to the problem is an assignment of exactly one driver to each bus leg. Criteria for feasibility are:

- No overlapping bus legs are assigned to any driver.

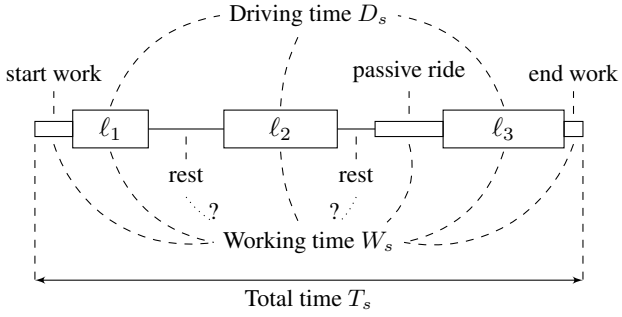

Figure 1: Example shift

- Changing tour or position between consecutive assignments i and j requires $start_j \geq end_i + d_{endPos_i, startPos_j}$.
- Each shift respects all hard constraints regarding work regulations as specified in the next section.

Work and Break Regulations

Valid shifts for drivers are constrained by work regulations and require frequent breaks. First, different measures of time related to a shift s containing the set of bus legs $\mathbf{L}_s$ need to be distinguished, as visualized in Figure 1:

- The total amount of driving time: $D_s = \sum_{i \in \mathbf{L}_s} drive_i$.
- The span from the start of work until the end of work T_s with a maximum of $T_{max} = 14$ hours.
- The working time $W_s = T_s - unpaid_s$, which does not include certain unpaid breaks.

Driving Time Regulations. The maximum driving time is restricted to $D_{max} = 9$ hours. The whole distance $start_j - end_i$ between consecutive bus legs i and j qualifies as a driving break, including passive ride time. Breaks from driving need to be taken repeatedly after at most 4 hours of driving time. In case a break is split in several parts, all parts must occur before a driving block exceeds the 4 hour limit. Once the required amount of break time is reached, a new driving block starts. The following options are possible:

- One break of at least 30 minutes;
- Two breaks of at least 20 minutes each;
- Three breaks of at least 15 minutes each.

Working Time Regulations. The working time W_s has a hard maximum of $W_{max} = 10$ hours and a soft minimum of $W_{min} = 6.5$ hours. If the employee is working for a shorter period of time, the difference has to be paid anyway. The actual paid working time is $W'_s = \max\{W_s; 390\}$.

A minimum rest break is required according to the following options:

- $W_s < 6$ hours: no rest break;
- 6 hours $\leq W_s \leq 9$ hours: at least 30 minutes;
- $W_s > 9$ hours: at least 45 minutes.

The rest break may be split into one part of at least 30 minutes and one or more parts of at least 15 minutes. The first part has to occur after at most 6 hours of work.

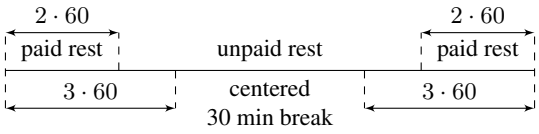

Figure 2: Rest break positioning

Whether rest breaks are paid or unpaid depends on break positions according to Figure 2. Every period of at least 15 minutes of consecutive rest break is unpaid as long as it does not intersect the first 2 or the last 2 hours of the shift (a longer rest break might be partially paid and partially unpaid). The maximum amount of unpaid rest is limited:

- If 30 consecutive minutes of rest break are located such that they do not intersect the first 3 hours of the shift or the last 3 hours of the shift, at most 1.5 hours of unpaid rest are allowed;
- Otherwise, at most one hour of unpaid rest is allowed.

Rest breaks beyond this limit are paid.

Split Shifts. If a rest break would be at least 3 hours long, it is instead considered a shift split, which is unpaid and does not count towards W_s. However, such splits are typically regarded badly by the drivers. A shift split counts as a driving break, but does not contribute to rest breaks.

Objectives

As argued in Kletzander and Musliu (2020), practical schedules must not consider only operation costs. The objective

$$cost_s = 2 \cdot W'_s + T_s + ride_s + 30 \cdot ch_s + 180 \cdot split_s \quad (1)$$

represents a linear combination of several criteria for shift s. The paid working time W'_s is the main objective and it is combined with the total time T_s to reduce long unpaid periods for employees. The next sub-objectives reduce the passive ride time $ride_s$ and the number of tour changes ch_s, which is beneficial for both employees and efficient schedules. The last objective aims to reduce the number of shift splits $split_s$ as they are very unpopular.

Branch and Price

The problem is solved by Branch and Price (Barnhart et al. 1998). Set partitioning is used as the master problem and the resource constrained shortest path problem (RCSPP) as the subproblem. Column generation is applied on each node of the branching tree. The duals of the relaxed master problem are used to find new shifts that have the potential to improve the solution of the master problem. Branching occurs when the resulting solution is not integer. The process continues until all branches either result in integer solutions or are cut off by the current objective bounds.

Master Problem

The goal of the master problem is to select a subset of the shift set $\mathbf{S}$, such that each bus leg is covered by exactly one shift while minimizing total cost. This corresponds to the set-partitioning problem:

$$\text{minimize} \sum_{s \in \mathbf{S}} cost_s \cdot x_s \quad (2)$$

$$\text{subject to} \sum_{s \in \mathbf{S}} cover_{s\ell} \cdot x_s = 1 \qquad \forall \ell \quad (3)$$

$$x_s \in \{0, 1\} \qquad \forall s \quad (4)$$

Here x_s is the variable for the selection of shift s. The objective (2) minimizes the total cost, Equation (3) states that each bus leg needs to be covered exactly once (using $cover_{s\ell} \in \{0, 1\}$ to indicate whether shift s covers leg ℓ), and Equation (4) states the integrality constraint. This constraint is relaxed to $0 \leq x_s \leq 1$ for the relaxed master problem which is repeatedly solved at each node of the branching tree. Once no more new shifts can be found by the subproblem, the result of the relaxed master problem provides a local lower bound for the solution of the integer problem.

In some cases, the resulting solution might already be integer. Usually, however, the result will be fractional. Then, the integer version of the master problem is solved with the current set of columns. While there are no guarantees regarding the quality of the integer solution in this case, in practice, solutions are often very close to the lower bound already. In any case, the result from the integer master problem is a feasible solution that provides a global upper bound for the problem. If the solution is not integer, branching will be done, resuming calculation on one of the open branches.

Subproblem

The goal of the subproblem is to find the column (shift) with the lowest reduced cost. For BDS this corresponds to the resource constrained shortest path problem (RCSPP) on an acyclic graph. In this problem, a graph $G = (\mathbf{N}, \mathbf{A})$ is given where $\mathbf{N}$ is the set of n nodes, corresponding to all bus legs, a source node s, and a target node t, and $\mathbf{A}$ is the set of arcs, corresponding to connections between bus legs. As the bus legs are naturally ordered by time, the RCSPP is acyclic.

Each node and arc is associated with a cost and an r-dimensional resource vector representing the resource consumption when using the node or arc. A shift is defined as a path from s to t such that the path satisfies all feasibility criteria associated with the resources. Each node corresponding to a bus leg is associated with a dual from the master problem. The reduced cost of a path is therefore the cost of a path minus the sum of the duals along the path. The RCSPP aims at finding the least-cost feasible path from s to t.

There is an arc from node s to each node i except $i = t$, and there is an arc from each node i except $i = s$ to node t. Nodes corresponding to bus legs i and j are only connected by an arc ij if chaining the bus legs is feasible according to their times and the distance matrix d. Building the graph is done once per instance in $O(n^2)$. The rest of this section goes into the details of this graph, its costs, and its constraints. The description is rather complex because the regulations themselves are complex.

Costs. The costs for nodes and arcs are based on objective (1). The cost c_i for each node i corresponding to a bus leg i

is defined as $c_i = 3 \cdot length_i$ as each bus leg contributes its length to both W_s (weight 2) and T_s (weight 1). By definition, $c_s = c_t = 0$.

Several helpful properties of arcs from node i to node j are defined and used for defining the arc costs:

$$length_{ij} = \begin{cases} start_j - end_i & \text{if } i \neq s \wedge j \neq t \\ startWork_{startPos_j} & \text{if } i = s \\ endWork_{endPos_i} & \text{if } j = t \end{cases} \quad (5)$$

$$ride_{ij} = \begin{cases} d_{endPos_i, startPos_j} & \text{if } i \neq s \wedge j \neq t \wedge \\ & endPos_i \neq startPos_j \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$change_{ij} = \begin{cases} 1 & \text{if } i \neq s \wedge j \neq t \wedge tour_i \neq tour_j \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$split_{ij} = \begin{cases} 1 & \text{if } i \neq s \wedge j \neq t \wedge \\ & length_{ij} - ride_{ij} \geq 3 \cdot 60 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

$$remain_{ij} = \begin{cases} length_{ij} - ride_{ij} & \text{if } split_{ij} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

$$rest_{ij} = \begin{cases} remain_{ij} & \text{if } i \neq s \wedge j \neq t \wedge \\ & remain_{ij} \geq 15 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

Equation (5) defines the length of an arc, taking into account start and end arcs. Equation (6) states the passive ride time, Equation (7) whether a tour change occurs, Equation (8) whether the arc corresponds to a shift split, Equation (9) captures the remaining arc length after removing the passive ride time and the shift split time, and Equation (10) captures a potential rest break.

The cost c_{ij} for arc ij (i.e., bus leg i to j) is defined as

$$c_{ij} = 2 \cdot remain_{ij} + length_{ij} + 3 \cdot ride_{ij} + 30 \cdot change_{ij} + 180 \cdot split_{ij} \quad (11)$$

Note that unpaid rest cannot be determined at this point, therefore all rest is treated as paid for this computation. Unpaid rest is separately treated when solving the subproblem. $ride_{ij}$ contributes to both working time W_s and the additional passive ride penalty.

Resources. The solution method is a label-setting algorithm (Irnich and Desaulniers 2005). Due to the acyclic nature of the graph, each node can be processed in temporal order, starting with an initial label representing an empty path at s where each resource usage is 0. For each node i and each label x on that node, all arcs ij are processed and a label y is placed at the end node j of the arc unless some constraints are violated. Therefore, each label x represents a path from s to its node i, capturing the nodes in the path, the cost, and the resource usage. In total, eleven resources need to be tracked in order to satisfy the BDS constraints. The first resources are classical additive resources with a maximum allowed usage.

$$d_y = d_x + drive_j \quad (12)$$

$$s_y = s_x + length_{ij} + length_j \quad (13)$$

Equation (12) tracks driving time, Equation (13) the span.

$$rd_y = \begin{cases} true & \text{if } length_{ij} \geq 30 \vee \\ & (length_{ij} \geq 20 \wedge b20_x \geq 1) \vee \\ & (length_{ij} \geq 15 \wedge b15_x \geq 2) \\ false & \text{otherwise} \end{cases} \quad (14)$$

$$dc_y = \begin{cases} 0 & \text{if } rd_y \\ dc_x + drive_j & \text{otherwise} \end{cases} \quad (15)$$

$$b15_y = \begin{cases} 0 & \text{if } rd_y \\ b15_x + 1 & \text{if } \neg rd_y \wedge length_{ij} \geq 15 \\ b15_x & \text{otherwise} \end{cases} \quad (16)$$

$$b20_y = \begin{cases} 0 & \text{if } rd_y \\ b20_x + 1 & \text{if } \neg rd_y \wedge length_{ij} \geq 20 \\ b20_x & \text{otherwise} \end{cases} \quad (17)$$

Resources for monitoring drive breaks need to be reset at each full drive break. Equation (14) defines a helping flag to indicate whether the current drive block is finished. Equation (15) uses this flag to reset or increase the current driving time. Equation (16) tracks the number of 15-minute breaks in the current driving block and Equation (17) the number of 20-minute breaks. 30-minute breaks do not need to be tracked as each of them resets the driving block.

Constraints regarding rest breaks need to consider different sums of rest breaks and their positioning.

$$w_y = w_x + u_x - u_y + remain_{ij} + ride_{ij} + length_j \quad (18)$$

$$r_y = \min(r_x + rest_{ij},\ 45) \quad (19)$$

$$b30_y = b30_x \vee rest_{ij} \geq 30 \quad (20)$$

$$rest'_{ij} = rest_{ij} - \max(2 \cdot 60 - s_x,\ 0) - \max(end_i + rest_{ij} - (end_y - 2 \cdot 60),\ 0) \quad (21)$$

$$u_y = \min\left(u_x + \begin{cases} rest'_{ij} & \text{if } rest'_{ij} \geq 15 \\ 0 & \text{otherwise} \end{cases},\ 90\right) \quad (22)$$

$$bc30_y = bc30_x \vee rest''_{ij} \geq 30 \quad (23)$$

Equation (18) tracks working time, assuming that all of u_y is unpaid so far, which constitutes a lower bound for the actual value. Equation (19) tracks the amount of required rest time, capping it at 45 as higher values do not matter. Equation (20) deals with the occurrence of a 30-minute rest break.

Equation (21) captures the part of the rest break that can be unpaid depending on the position. However, this requires knowing the end time of the shift end_y which violates the general assumption of the algorithm that nodes can be processed in temporary order. Therefore, for each node j, with known end time end_j if j is the last bus leg in the shift, new nodes are added to the graph: $\mathbf{N}_j$ is the set of nodes reachable within 3 hours from end_j when traversing the network backwards from j (including j itself). For each node $i \in \mathbf{N}_j$ a new node i_j is created. For each pair of nodes i and k both in $\mathbf{N}_j$, an arc $i_j k_j$ is added if ik is in the original graph. For $i \notin \mathbf{N}_j$ and $k \in \mathbf{N}_j$, an arc ik_j is added if ik is in the original graph. The arc $j_j t$ replaces the arc jt. Each new node is associated with the end time end_j. All original nodes have end time ∞. This solves the problem, but increases the graph size by a factor of 6 for small instances up to more than 30 for large instances.

The potential total amount of unpaid rest is captured in Equation (22). Equation (23) tracks the existence of a centered 30-minute break using $rest''_{ij}$ defined like in (21) except with $3 \cdot 60$ instead of $2 \cdot 60$.

Finally, the cost needs to be computed.

$$cost'_y = cost'_x + 2 \cdot (u_x - u_y) + c_{ij} + c_j - dual_j \quad (24)$$

$$cost_y = cost'_y + 2 \cdot \max(W_{min} - w_i,\ 0) \quad (25)$$

Equation (24) takes into account the unpaid rest and the duals for the bus legs. Equation (25) further considers the minimum working time.

Constraints. A shift is a path from s to t that does not violate any resource constraints. The following resource constraints need to be satisfied for every label x: $d_x \leq D_{max}$, $s_x \leq T_{max}$, $dc_x \leq 4 \cdot 60$, $w_x \leq W_{max}$, and $r_x \geq 15$ if $w_x \geq 6 \cdot 60$. Additionally, at node t, all labels x need to satisfy: $r_x \geq 45$ if $w_x > 9 \cdot 60$ and $b30_x$ if $w_x \geq 6 \cdot 60$.

Dominance. In order to track only paths with the potential to become minimum-cost paths, only pareto-optimal labels with respect to both cost and resource usage are kept at each node, i.e., labels which are not dominated. Regarding more complex constraints, it is important to observe that, if label x dominates label y at a node i, extensions of x to future nodes must still dominate extensions of y. Finally, the least-cost label at node t represents the minimum-cost path.

Therefore, a label x dominates label y if all following conditions are true: $d_x \leq d_y$, $s_x \leq s_y$, $dc_x \leq dc_y$, $b20_x \geq b20_y$, $b15_x \geq b15_y$, $w_x + u_x \leq w_y$, $r_x \geq r_y$, $b30_x \vee \neg b30_y$, $bc30_x \vee \neg bc30_y$, $c_x + 2 \cdot u_x \leq c_y$.

In general, lower values dominate for resources with upper bounds and higher values dominate for resources with lower bounds. The most complex case arises from the fact that the maximum value of unpaid break might be 0, 60, or 90 depending on the existence and position of a 30-minute rest break. This results in neither higher nor lower values of u_i being dominant. Rather, the uncertain amount of unpaid rest weakens the domination power of cost and working time, as those might vary in the amount of unpaid rest until the end of the path. However, in the next section a more useful way to deal with this problem is described.

Branching

Branching is performed on the connections between bus legs in a shift that correspond to arcs in the subproblem. The branching considers a connection between bus legs i and j that appears in a fractional shift in the master and selects the most fractional to maximize impact. In the left branch, all columns containing i or j, but not both consecutively, are removed. In the RCSPP graph all outgoing arcs from i except for the one connecting to j are removed. In the right branch, all columns with i and j assigned consecutively are removed, as well as the arc from i to j in the RCSPP graph.

Lagrangean Bound

In the case when the column generation terminates at the root node, small optimality gaps are typically achieved. However, when there is not enough time to complete the column generation at the root node, a Lagrangean lower bound is computed as $O(RMP) + \kappa \cdot O(PP)$, where $O(RMP)$ is the optimum of the reduced master problem, κ is an upper bound on the number of shifts and $O(PP)$ is the optimum of the pricing problem. The implementation uses $\kappa = \lfloor O(RMP)/minCost \rfloor$, where $minCost = 2 \cdot W_{min} + \min_{\ell \in \mathbf{L}} length_\ell$. The minimum reduced cost is only known when no more throttling (see next section) is in play. Otherwise a lower bound for the minimum reduced cost is computed where the constraints regarding break positions for unpaid breaks are relaxed.

Handling the Subproblem Dimensionality

As described in the previous section, due to the complex constraints for valid shifts, the subproblem complexity is very high. In particular, eleven resources are tracked in the labels. However, the efficiency of the label-setting algorithm depends on its ability to keep a small number of non-dominated labels. The more dimensions the labels have, the easier it is for labels to be non-dominated, drastically increasing the number of labels for processing. Therefore, the increase in the number of processed labels turns out to be the bottleneck of scaling the solution method to larger instances. Several novel improvements to reduce this bottleneck are introduced. These are generally applicable to other problems with similar characteristics.

Subproblem Partitioning

Instead of generating all possible shifts from one graph, the subproblem is split into three similar, but disjoint RCSPP problems, removing the uncertainty for unpaid breaks depending on whether the maximum amount of rest break is 0, 60, or 90. This depends on the existence and position of a 30-minute rest break:

- *No 30-minute rest break*: All arcs corresponding to rest breaks of at least 30 minutes can be removed, making the graph very spare and therefore fast to process. Unpaid rest is guaranteed to be 0, all rest break resources are ignored.
- *Uncentered 30-minute rest break*: $b30_y$ must be true at t. The maximum of u_y is changed to 60, but the current amount of u_y is guaranteed to be unpaid. Therefore, $w_x \leq w_y$ and $w_x + u_x \leq w_y + u_y$ can be used instead of $w_x + u_x \leq w_y$ as the domination criterion (same change for c_x and c_y), reducing the number of undominated labels.
- *Centered 30-minute rest break:* $b30_y$ and $b30c_y$ must be true at t. The maximum of u_y is set to 90, but the improved dominance criteria from the previous graph still apply as again all of u_y is guaranteed to be unpaid.

The pricing subproblem is used to add multiple columns (up to 1000 per graph) at once. Indeed, solving the relaxed master problem even with thousands of columns is much faster than solving the pricing subproblem. Therefore, accepting some unnecessary columns while reducing the number of times the subproblem is solved pays off. The different graphs are ordered by their complexity, measured by the number labels they expand. The pricing subproblem avoids

searching more complex graphs, as long as those with less complexity still produce enough shifts with negative reduced costs (usually 10, 100 if the objective did not change, 1000 if the objective did not change repeatedly).

Cost Bound

An upper bound for the reduced cost for each node can be computed by processing the RCSPP graph backwards with the current duals. The upper bound describes the maximum reduced cost at each node, such that there is still a path to arrive at t with reduced cost < 0, disregarding resource constraints. All labels above this bound can be discarded during the solution process.

Exponential Arc Throttling

While it might be hard to find the best column, at least in the beginning of the solving process, there are many columns with negative reduced cost. Two options were considered to reduce the size of the subproblem in early iterations. The first option is to reduce the number of labels at each node. This can be done either by rejecting new labels after a certain threshold or only retaining labels according to certain criteria, e.g., the best 100 by cost.

The better option is to exploit the fact that good columns are unlikely to include very costly arcs. It proposes a new throttling mechanism that imposes a maximum cost per arc, starting at 100 (just enough for a 30 min break). This greatly reduces the size of the graph and therefore the number of labels. Once the number of new columns is small, this factor is multiplied by 2. This is repeated until all arcs are included. While a lot of arcs are not present in the early stage, the focus on arcs that are likely to appear in good columns leads to very fast convergence to good solutions, even if there is not enough time to finish the column-generation process.

Improved Elimination of Non-Dominated Labels

Even with the previously mentioned improvements, the majority of runtime is spent figuring out which labels are non-dominated. The naive approach is to compare each new label on a node with each label in the current set of non-dominated labels. However, the runtime complexity of this method is quadratic in the number of non-dominated labels. Therefore, two different methods are explored to speed up the dominance checks. The first method is the multi-dimensional divide-and-conquer of Bentley (1980): It is based on recursively solving the k-dimensional problem with n labels by two k-dimensional problems of size $n/2$ and one $k-1$-dimensional problem of size n.

The second method is a two-stage approach based on k-d trees and bounding boxes (Chen, Hwang, and Tsai 2012). The two-stage approach is more effective for several reasons. First, the algorithm is based on two stages. Instead of comparing each new label with all previously non-dominated labels both ways and deleting dominated labels in the process, two passes are performed. Labels are added as long as they are not dominated, but no removal operations are executed. The second pass is performed in opposite order before expanding the labels at a node, ensuring that only non-dominated labels are expanded. Second, instead of storing labels in a list, a k-dimensional tree is used for storage. For each node r on level ℓ, the left child is better with respect to dimension ℓ mod k and the right child is worse (or equal). Finally, each node r is associated with a bound u_r where each dimension contains the best value among all nodes in the subtree rooted at r. Therefore, if a label is not dominated by u_r, it is not dominated by any label in the subtree rooted at r, saving several comparisons.

Evaluation

The implementation uses Java (OpenJDK 14.0.1), the master problem is solved by CPLEX 12.10. The evaluation was performed on Windows 10 Professional using an Intel Core i7-6700K with 4x4.0 GHz and 32 GB RAM. The application is deterministic: Repeated runs produce the same result.

The method was evaluated on a publicly available set of benchmark instances (Kletzander and Musliu 2020). There are 50 instances in 10 size categories ranging from 10 tours (about 70 legs) to 100 tours (almost 1000 legs). The instances are based on real-life demand distributions. The small to medium size instances represent clusters of bus lines that often need to be optimized when competing for new contracts. Large instances correspond to the scale of medium-size Austrian cities.

Results

Table 2 shows the results of using the best configuration on the benchmark instances. Results are presented as average over the respective size category. The runtime for the Branch and Price was limited to one hour and up to one more hour to finish the last ongoing computation. This is needed to find a feasible solution even if the column generation at the root node does not terminate.

Results show that small instances can be optimally solved in very short time. For size 20, 4 out of 5 instances can be solved optimally within an hour, with a gap of only 0.07 % for the remaining instance. For all instances up including size 60, the root node terminates within one hour and the optimality gap of the solution is always less than 1%; A similar result also occurs for 2 instances of size 70 and one instance of size 80. For larger instances, the lower bound is provided by the traditional Lagrangean bound, whose quality depends on the column generation upon termination. However, due to the exponential arc throttling, the results obtained in this time are still of high quality, improving or proving the optimality of previous best known solutions on 48 out of 50 instances. It is expected that the new results are very close to the optimum as the lower bound shows slower convergence compared to the upper bound. For the larger instances (starting with some of size 70), solving the integer set-partitioning problem with around 50000 generated columns also times out with a gap of a few percent. Those are the instances with sightly above 2 hours of runtime.

There are two main contributions to this research. On the one hand, the approach provides lower bounds to evaluate the quality of the previous solutions obtained by metaheuristics. In particular, they show that, for 4 out of 5 instances of

Instances	Time	BP Lower bound	BP Best	BP Gap	Prev time	Prev best	Gap to BP LB	Gap to BP best
10	7.2	14709.2	14709.2	0 %	22.8	14717.4	0.06 %	0.06 %
20	1201.4	30290.3	30294.8	0.01 %	62.2	30790	1.65 %	1.63 %
30	3610.6	49674.4	49846.4	0.35 %	108.8	50947.4	2.56 %	2.21 %
40	3605.8	66780.3	67000.4	0.33 %	267.0	69072.2	3.43 %	3.09 %
50	3674.4	84042.1	84341.0	0.36 %	295.4	86539.6	2.97 %	2.61 %
60	4373.2	99334.0	99727.0	0.40 %	432.8	103445.2	4.15 %	3.74 %
70	6460.4	108083.1	118524.2	9.66 %	751.4	122531.4	13.4 %	3.38 %
80	5912.4	106499.2	134513.8	26.3 %	959.4	139518.2	31.0 %	3.72 %
90	7390.4	104848.1	150370.8	43.4 %	1516.6	156046.0	48.8 %	3.77 %
100	7395.8	102858.6	172582.2	67.8 %	1483.2	172269.6	67.5 %	-0.18 %

Table 2: Results for the benchmark dataset (new method left, comparison to previous results right)

size 10, the optimum was previously found (but not proven optimal). On the other hand, except for size 100, the Branch and Price algorithm improves existing results by 1-4%, with results for sizes 60–90 being close to 4%. This is very significant for such logistics and transportation problems.

Note that the previous solution method has the advantage of lower runtimes (except for size 10) and it is useful for even larger instances. In practical scenarios like providing cost calculations when competing for new bus lines, or investigating the potential improvement for the current bus network, spending some extra time for better and bounded solutions is very important. Therefore, the use of Branch and Price for such scenarios is very useful.

Effects of Subproblem Improvements

While all the design choices and parameter settings have been carefully tested, this section focuses on the effects of our crucial subproblem improvements.

We highlight the benefits for splitting the subproblem on the example of instance 40_16. At the root node the subproblem is solved 197 times. Only 76 times the uncentered 30-min break graph is solved and only 14 times the centered 30-min break graph. However, the runtimes for the different graphs are 4 ms, 1.5 s and 1.9 s respectively at the beginning (arc throttling) and 16 ms, 7.7 s and 51.1 s at the end (all arcs). Even though easier subproblems only provide parts of the new shifts, their runtime advantage makes it worth only going to the more complex subproblems when necessary.

Figure 3 shows the comparison of cost-based throttling (100 best regarding cost per node, factor 10 increase), no throttling, node-based throttling (100 first per node, factor 10 increase), and arc-based throttling (arc cost limited to 100, factor 2 increase) on the objective of the reduced master problem over time. The results show a significant overhead of cost-based throttling, and weak improvements early but more efficiency later for node-based throttling. Arc-based throttling, however, clearly shows superior performance starting to drop significantly in objective value compared to other approaches in less than 10 seconds. Its overall time is 727 seconds compared to 1354 seconds for node-based throttling, the second best approach. Overall, using node-based throttling would provide improvement on 16 fewer instances than arc-based throttling and terminate the root node for 5 fewer instances.

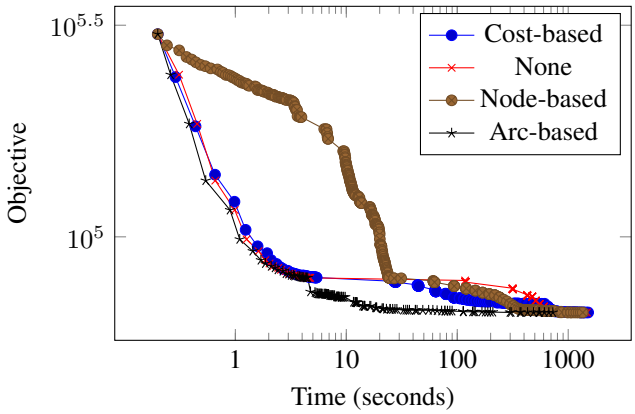

Figure 3: Throttling approaches for 40_16

Regarding the dominance algorithm, one run of the largest subproblem on instance 40_16 takes 290 seconds with the default dominance algorithm, 293 seconds with the multi-dimensional divide-and-conquer algorithm, and 51 seconds with the two-stage algorithm. Overall, for default dominance the root node terminates in only 25 compared to 33 instances for the two-stage algorithm within the time limit.

Conclusion

This paper presented a Branch and Price approach for a complex Bus Driver Scheduling Problem. The Branch and Price includes a range of novel improvements for solving the high-dimensional subproblem that can be applied to other problems with similar characteristics. Experimental results show that the approach provides strong lower bounds for existing metaheuristics and improves or proves optimality for 48 out of 50 instances. The novel results have optimality gaps below 1% for small to medium size instances and represent high-quality solutions for larger instance sizes in reasonable time. Therefore the Branch and Price approach may be considered the new state-of-the-art exact method for this complex scheduling problem. Future work might might include applying the method to different BDS problems, or incorporating robustness features like break buffers.

Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

Ethics Statement

The resulting schedules can have tremendous impact on the lives of the employees that are ultimately required to work these shifts. Badly designed shifts can have a negative impact on both the social life and health of workers. Therefore, we consider an objective function that does not only include cost, but was developed to incorporate several criteria that are important for the affected employees. Even though this complicates the optimization process, we see social compatibility as a mandatory aspect in employee scheduling.

References

Balas, E.; and Padberg, M. W. 1976. Set partitioning: A survey. *SIAM review* 18(4): 710–760.

Barnhart, C.; Johnson, E. L.; Nemhauser, G. L.; Savelsbergh, M. W.; and Vance, P. H. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations research* 46(3): 316–329.

Beer, A.; Gaertner, J.; Musliu, N.; Schafhauser, W.; and Slany, W. 2008. Scheduling Breaks in Shift Plans for Call Centers. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, 1–17.

Beer, A.; Gärtner, J.; Musliu, N.; Schafhauser, W.; and Slany, W. 2010. An AI-Based Break-Scheduling System for Supervisory Personnel. *IEEE Intelligent Systems* 25(2): 60–73.

Bentley, J. L. 1980. Multidimensional divide-and-conquer. *Communications of the ACM* 23(4): 214–229.

Chen, S.; Shen, Y.; Su, X.; and Chen, H. 2013. A Crew Scheduling with Chinese Meal Break Rules. *Journal of Transportation Systems Engineering and Information Technology* 13(2): 90–95.

Chen, W.-M.; Hwang, H.-K.; and Tsai, T.-H. 2012. Maxima-finding algorithms for multidimensional samples: A two-phase approach. *Computational Geometry* 45(1-2): 33–53.

Constantino, A. A.; de Mendonça Neto, C. F. X.; de Araujo, S. A.; Landa-Silva, D.; Calvi, R.; and dos Santos, A. F. 2017. Solving a Large Real-world Bus Driver Scheduling Problem with a Multi-assignment based Heuristic Algorithm. *Journal of Universal Computer Science* 23(5): 479–504.

De Leone, R.; Festa, P.; and Marchitto, E. 2011. A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution. *Journal of Heuristics* 17(4): 441–466.

Desrochers, M.; and Soumis, F. 1989. A Column Generation Approach to the Urban Transit Crew Scheduling Problem. *Transportation Science* 23(1): 1–13.

Ernst, A.; Jiang, H.; Krishnamoorthy, M.; and Sier, D. 2004. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153(1): 3–27.

Ibarra-Rojas, O.; Delgado, F.; Giesen, R.; and Muñoz, J. 2015. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological* 77: 38–75.

Irnich, S. 2008. Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum* 30(1): 113–148.

Irnich, S.; and Desaulniers, G. 2005. Shortest path problems with resource constraints. In *Column generation*, 33–65. Springer.

Kletzander, L.; and Musliu, N. 2020. Solving Large Real-Life Bus Driver Scheduling Problems with Complex Break Constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 421–429.

Li, J.; and Kwan, R. S. 2003. A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research* 147(2): 334–344.

Lin, D.-Y.; and Hsu, C.-L. 2016. A column generation algorithm for the bus driver scheduling problem: Bus Driver Scheduling Problem. *Journal of Advanced Transportation* 50(8): 1598–1615.

Lourenço, H. R.; Paixão, J. P.; and Portugal, R. 2001. Multiobjective Metaheuristics for the Bus Driver Scheduling Problem. *Transportation Science* 35(3): 331–343.

Martello, S.; and Toth, P. 1986. A heuristic approach to the bus driver scheduling problem. *European Journal of Operational Research* 24(1): 106–117.

Portugal, R.; Lourenço, H. R.; and Paixão, J. P. 2009. Driver scheduling problem modelling. *Public Transport* 1(2): 103–120.

Respício, A.; Moz, M.; and Vaz Pato, M. 2013. Enhanced genetic algorithms for a bi-objective bus driver rostering problem: a computational study. *International Transactions in Operational Research* 20(4): 443–470.

Shen, Y.; and Kwan, R. S. K. 2001. Tabu Search for Driver Scheduling. In Fandel, G.; Trockel, W.; Aliprantis, C. D.; Kovenock, D.; Voß, S.; and Daduna, J. R., eds., *Computer-Aided Scheduling of Public Transport*, volume 505, 121–135. Berlin, Heidelberg: Springer Berlin Heidelberg.

Smith, B. M.; and Wren, A. 1988. A bus crew scheduling system using a set covering formulation. *Transportation Research Part A: General* 22(2): 97–108.

Tóth, A.; and Krész, M. 2013. An efficient solution approach for real-world driver scheduling problems in urban bus transportation. *Central European Journal of Operations Research* 21(S1): 75–94.

Van den Bergh, J.; Beliën, J.; De Bruecker, P.; Demeulemeester, E.; and De Boeck, L. 2013. Personnel scheduling: A literature review. *European Journal of Operational Research* 226(3): 367–385.

Widl, M.; and Musliu, N. 2014. The break scheduling problem: complexity results and practical algorithms. *Memetic Computing* 6(2): 97–112.

Wren, A.; and Rousseau, J.-M. 1995. Bus Driver Scheduling — An Overview. In Fandel, G.; Trockel, W.; Daduna, J. R.; Branco, I.; and Paixão, J. M. P., eds., *Computer-Aided Transit Scheduling*, volume 430, 173–187. Berlin, Heidelberg: Springer Berlin Heidelberg.