A matheuristic for the driver scheduling problem with staff cars

Perumal, Shyam S.G.; Larsen, Jesper; Lusby, Richard M.; Riis, Morten; Sørensen, Kasper S.

Published in:
European Journal of Operational Research

Link to article, DOI:
10.1016/j.ejor.2018.11.011

Publication date:
2019

Document Version
Peer reviewed version

A matheuristic for the driver scheduling problem with staff cars

Shyam S. G. Perumal[1,2], Jesper Larsen[2], Richard M. Lusby[2], Morten Riis[1], and Kasper S. Sørensen[3]

[1]Qampo ApS, Aarhus Denmark
[2]Department of Management Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark
[3]Trapeze Group Europe A/S, Aarhus, Denmark

November 7, 2019

Abstract

In the public bus transport industry, it is estimated that the cost of a driver schedule accounts for approximately 60% of a transport company's operational expenses. Hence, it is important for transport companies to minimize the overall cost of driver schedules. A duty is defined as the work of a driver for a day and the driver scheduling problem (DSP) is concerned with finding an optimal set of driver duties to cover a set of timetabled bus trips. Numerous labor regulations and other practical conditions enforce drivers to travel within the city network to designated bus stops to start/end duty, to take a break or to takeover a bus from another driver. This paper focuses on the driver scheduling problem with staff cars (DSPSC), where staff cars can be utilized by the drivers to fulfill their travel activities. However, staff cars should always be returned to the depot and can perform multiple round trips during the day. The problem is restricted by the number of cars available at the depot. We present a matheuristic for solving the DSPSC and the proposed method is tested on instances from Danish and Swedish companies. A comparison with a state-of-the-art mixed integer programming (MIP) solver indicates that the matheuristic provides better solutions, with comparable computation times, for 6 out of 10 large instances. For instances that have more than 6 staff cars and 1200 bus trips, the improvement is 13-15% on average.
Keywords: Transportation, Driver scheduling problem, Heuristics

1 Introduction

Growing populations in cities worldwide demand well-organized public transport systems that prevent long travel times, traffic congestion, road accidents and pollution. Public transport systems are considered to be the backbone of sustainable urban development. The passengers expect a high level of service; i.e., the transport system should be accessible, comfortable, affordable and it should be possible to reach destinations quickly. The objective of transport companies is to provide high quality service to the passengers while minimizing their overall operational cost (Desaulniers and Hickman [2007]; Ibarra-Rojas et al. [2015]). Transport companies are constantly faced with the challenge of planning for cities with large scale transport systems. Government and EU policies, labor regulations and other practical conditions further challenge transport companies to efficiently utilize their resources. As a consequence, over the years, there has been an increase in the development of decision support tools based on mathematical programming approaches to aid transport companies in planning (Desrochers and Soumis [1989]; Wren et al. [2003]; Smith and Wren [1988]; Lourenço et al.

[2001]). Typically, the transport planning process involves solving several planning subproblems as it is too complex to solve the entire planning problem in one integrated step. The planning subproblems include Timetabling, the Vehicle Scheduling Problem (VSP), the Driver Scheduling Problem (DSP) and the Driver Rostering Problem (DRP). Public authorities often define the timetabled trips, where the arrival and departure times at all bus stops in a city network are determined. The timetabled trips are utilized by transport companies to schedule their buses and drivers. The VSP assigns buses to the timetabled trips such that every trip is covered by a bus and the objective is to minimize the operational cost based on bus usage. A bus typically covers a sequence of trips from the time it leaves the depot until it returns to the depot. A driver duty is defined as the work of a driver for a day and the DSP is concerned with finding an optimal set of duties that covers all bus trips. Given a set of generic duties over a certain time horizon, e.g. a month, the DRP assigns these duties to the available drivers.

In the DSP, transport companies have to plan driver schedules based on known bus schedules. In most cases, the bus schedules are different on weekdays, weekends and holidays. Hence, transport companies need to create driver schedule for each of these bus schedules. While determining the driver schedule, companies must consider two important aspects: minimizing the cost and ensuring feasibility of driver duties with respect to various labor regulations. Most commonly, the cost comprises of wages paid to the drivers, which is known to be more than the operational expenses of buses. It is estimated that the cost of a driver schedule accounts for approximately 60% of a transport company's operational expenses. Hence, a small improvement in the cost of driver schedule can lead to large savings. For a Danish transport company with over 600 drivers, it was shown that a reduction in cost of 1.2% represents 2-2.5 million DKK in savings in a year. Furthermore, the transport companies have to strictly abide by the labor rules and regulations. The three most common rules that apply for all transport companies are maximum working time for a driver, minimum/maximum number of breaks, and maximum time between breaks. These rules forbid the driver from driving a bus over a prolonged period and allow the driver to have sufficient breaks during the day. Hence, a driver may cover only a few consecutive bus trips before the driver takes a break or is relieved of duty. Another important condition for transport companies is the computation time required to build a driver schedule. Planning departments of transport companies with their experience in the bus transport industry and knowledge of the city network take three to four weeks to manually create a new driver schedule. A decision support tool based on mathematical programming approaches potentially eliminates the resources and the time required to plan. Therefore, a decision support tool that provides optimal or near-optimal solutions in quick computation time would assist transport companies to create and operate efficient transport systems.

Transport companies must also consider other practical limitations in the city network during the planning process. These limitations are concerned with the characteristics of the bus stops. Only at certain bus stops can the driver be allowed to sign on/off duty or to handover the bus to another driver. Furthermore, the driver is allowed to have a break only at certain bus stops, which is dependent on the availability of facilities such as restroom and canteen. Figure 1 illustrates an example of a duty where the driver covers a few trips. As it can be seen in the figure, the driver usually has to travel during the day of work within the city network to sign on/off duty, visit a bus stop where taking a break is allowed or takeover a bus from another driver. If the distance between the bus stops is short, then the driver can travel by foot. The driver can also travel as a passenger on a bus to designated bus stops. However, in some cases, a bus stop can only be reached feasibly with use of a car. Transport companies usually have a fleet of cars, which are defined as staff cars that the drivers could utilize. A driver, as part of his/her travel activity, usually takes a staff car from the depot to visit another bus stop and parks the staff car at the visited bus stop. Another driver may utilize the parked car from the same bus stop and drive it back to the depot as part of his/her travel activity during the duty. A round staff car trip is defined as the combination of a departure trip from the depot and an arrival trip to the depot. A staff car can perform multiple round trips during the day; however, a staff car should always be returned to the depot and only the limited cars that are available at the transport company's fleet can be used to fulfill the travel activities of

the drivers. Simultaneously, scheduling the drivers and the staff cars for the drivers gives rise to the driver scheduling problem with staff cars (DSPSC), which is an extension of the DSP. Such problems have not been reported in the Operations Research (OR) literature to the best of our knowledge. The DSP is a very complex problem, similar structured problems have been proven to be $\mathcal{NP}$-hard problems (Fischetti et al. [1987]), and the DSPSC adds further complexity to the DSP.

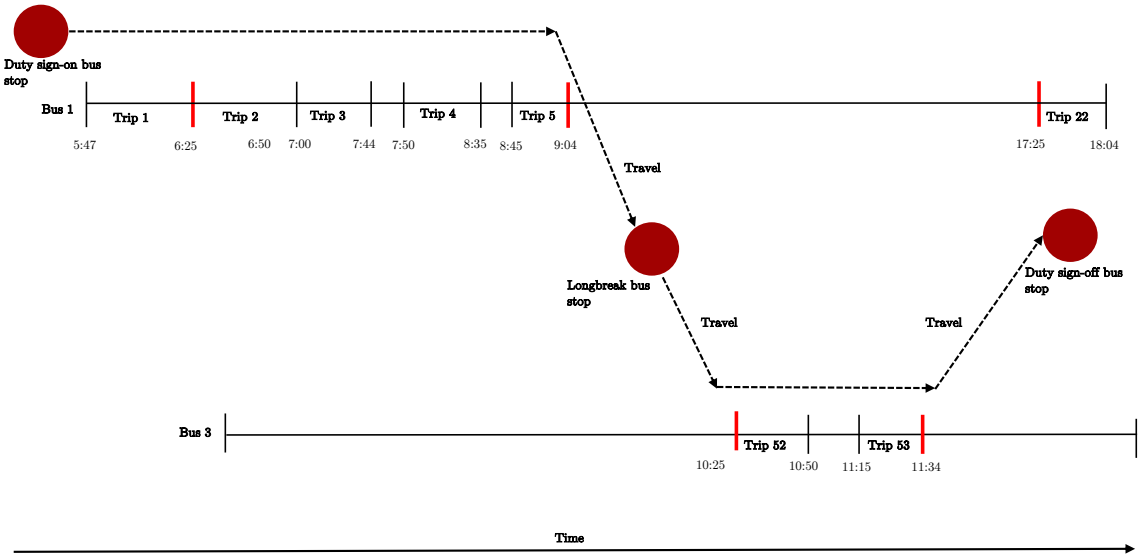


Figure 1: The figure illustrates an example of a duty where the driver covers a few trips of two buses. The red vertical lines represent the bus stops that the driver can handover/takeover the bus to/from another driver. The red circles indicate the bus stops that the driver visits during the duty to sign on/off or take a break.

Heuristic algorithms that are designed by combining metaheuristics and mathematical programming techniques are known as matheuristics (Boschetti et al. [2009]). In this paper, we propose a matheuristic for solving the DSPSC. A mixed integer programming (MIP) model, exact method, is embedded in an adaptive large neighborhood search (ALNS) heuristic. Large neighborhood search (LNS) was proposed by Shaw [1998] and the author applied the heuristic to vehicle routing problems. LNS is based on a local search framework such as simulated annealing (SA) or hill climber, where an initial solution is gradually improved by iteratively destroying and repairing the solution using a destroy and a repair method respectively. ALNS was proposed by Røpke and Pisinger [2006] where multiple destroy and repair methods are defined within the same search. Each destroy and repair method is assigned a weight that controls the selection of the particular method during the search. Problems, such as the DSP, where Dantzig-Wolfe Decomposition has been used with success are good candidates for LNS and ALNS heuristics (Pisinger and Røpke [2010]). Exact approaches are known to be highly effective for small to medium sized instances of hard problems but are inefficient for large instances; hence, heuristics are commonly used in practice (Jourdan et al. [2009]; Blum et al. [2011]). However, metaheuristics based on a local search framework are often ineffective for highly constrained problems where feasible areas of the solution space are disconnected (Dumitrescu and Stützle [2003]). Muller et al. [2012] designed a matheuristic based on ALNS for solving the lot sizing problem (LSP) with setup times. The authors' motivation for using a MIP model for the repair phase of the ALNS heuristic was that it could tackle the challenges of constructing and exploring a neighborhood of a given solution. MIP models can be very effective for exploring large neighborhoods within a local search procedure and guide the search to move between feasible regions of the solution space (Jourdan et al. [2009]). Since the DSPSC is considered to be a tightly constrained problem, where even finding a feasible solution could be challenging, the matheuristic approach is viewed as a

powerful optimization method for solving it. In 2009, Jourdan et al. [2009] stated that there has been an increase in number of works carried out on matheuristic approaches. The approach's ability to simultaneously exploit advantages of heuristics and exact methods has led to obtaining best solutions for most practical problems.

Trapeze Group Europe A/S (TGE) is an international provider of decision support tools within planning and operations for both public and private transport companies. Real-life instances of the DSPSC from a Danish and a Swedish transport company were acquired from TGE's system for this paper. Since DSPSC is a very realistic problem, this paper primarily contributes to research areas within traditional DSP and within applications of OR techniques for improving efficiency of transport systems.

This paper is organized as follows. Section 2 gives a description of the existing literature related to the DSP. In Section 3 we provide a formal description of the DSPSC with the help of a mathematical model. Section 4 introduces the proposed matheuristic framework for solving the DSPSC. The section also gives an outline of a greedy heuristic that provides an initial solution. Section 5 details the computational study based on experimental tests performed on instances from Danish and Swedish transport companies. Finally, Section 6 concludes the paper and addresses future directions of research. We also briefly discuss the challenges of integrating mathematical programming approaches such as the proposed matheuristic into decision support tools.

2 Related Literature

Common formulations of the DSP are based on set partitioning/covering problem (SCP), where the formulation is used as a duty selection module with the selected duties covering all bus trips at minimum cost (Ibarra-Rojas et al. [2015]). To find the optimal solution, all the feasible duties have to be considered in the SCP formulation. Due to potentially being a large number of possible duties, the formulation is intractable by exhaustive enumeration techniques. Some authors, e.g.-Smith and Wren [1988] and Wren et al. [2003], have considered reducing the number of duties being generated before solving the SCP. Smith and Wren [1988] heuristically generate a feasible subset of duties for the SCP. The SCP is solved by relaxing the integrality constraints and an integer solution is found using a branch-and-bound algorithm. The algorithm terminates when the current integer solution is within 0.5% of the integer optimum. The authors had developed the method as part of a commercial software system and reported some of the experiments run on the system. The largest instance included 309 constraints (bus trips), 4892 variables (duties) in the SCP formulation and the best integer solution was found in 238 seconds. Similar to Smith and Wren [1988], Wren et al. [2003] solve the SCP by considering only a subset of feasible duties. A large set of potential driver duties is generated and refined by heuristic procedures. The authors focused more on combining theory and practice to create a user-friendly and flexible system. Portugal et al. [2009] presented SCP based models that were developed in collaboration with planners and end-users of several transport companies in Portugal. The authors aimed at developing models to produce solutions that could be applied in real situations and, hence solution quality was not the only criteria for evaluating models. The authors tested instances with up to 347 bus trips and 23305 duties in the SCP formulation.

Exact approaches such as Branch & Price, where SCP implicitly considers all the possible duties, are commonly used in the literature for generating duties. Desrochers and Soumis [1989] devised a column generation method for solving real-life instance of a transport company operating in an American city that had a fleet of 25 buses. The resulting SCP formulation had 167 bus trips. However, solving large scale instances of the DSP by column generation approaches is notorious for being computationally expensive due to the need to solve resource constrained shortest path problem (RCSPP) at every iteration (Wren et al. [2003]; Ibarra-Rojas et al. [2015]). Yunes et al. [2005] devised a hybrid column generation approach that used constraint programming (CP) to generated feasible duties. The authors also reported that solving the RCSPP by dynamic programming techniques suggested by Desrochers and Soumis [1989] was computationally inefficient for large instances. The authors tested the two aforementioned methods on instances from a bus company in the city of Belo

Horizonte, Brazil. The sizes of the instances varied from 10 to 210 bus trips. The column generation algorithm based on dynamic programming could not solve instances more than 90 trips within a time limit of 24 hours. It was reported that 90% of the total computation time, on average, was spent on solving the RCSPP. However, the column generation based on CP was able to solve the largest instance of 210 trips in less than 15 hours. Mauri and Lorena [2007] applied a metaheuristic method known as the population training algorithm (PTA), a derivation of the genetic algorithm (GA), as part of the column generation framework to generate feasible duties. The authors tested the method on randomly generated instances that were based on real problems of a Brazilian transport company and the sizes of the instances varied from 25 to 500 bus trips. The authors compared the proposed method to a SA approach and the results for the largest instance indicated that the proposed method was faster than the SA approach by a factor 20 with an improvement of 0.15% in solution quality. Li et al. [2015] proposed a column generation approach that is based on a hyper-heuristic, which is similar to the idea of having multiple repair heuristics in a ALNS. The authors generate all feasible duties for a given instance and several heuristics (local search, swap heuristic and greedy based heuristic) are devised to select a subset of feasible duties at each iteration of the column generation framework. In the method proposed by Li et al. [2015], the hyper-heuristic evaluates the different heuristics based on their selection of duties that contribute to the improvement of the objective at each iteration of the column generation framework. The authors tested the method with instances provided by Mauri and Lorena [2007] and the largest instance with 500 bus trips had a total of 8.4 million feasible duties. The column generation based on hyper-heuristics yielded solutions that, on average, had a gap of 2.12% from the best known solutions, which were provided by the method proposed by Mauri and Lorena [2007]. However, for the largest instance, the hyper heuristic was faster by a factor 1.63.

Heuristic approaches, if designed well, are known to provide good solutions in reasonable computation time for large scale problems. Lourenço et al. [2001] proposed multiobjective metaheuristics for solving real-life instances of the DSP. In most formulations of the DSP, the objective is to minimize cost of driver duties. However, in practice, different transport companies take several objectives into account while planning. Hence, the authors solved the DSP involving multiple objectives such as minimizing total number of duties, minimizing number of bus changes and minimizing number of over-covered bus trips. A large number of duties that comply with the labor regulations and the transport companies' rules is heuristically generated for the SCP formulation. The SCP is solved by metaheuristic methods tabu search (TS) and GA. The authors also devised a greedy randomized adaptive search procedure (GRASP) for solving a subroutine of the TS and GA. The proposed methods were tested on instances with up to 348 bus trips and 74000 duties in the SCP formulation. The devised GA provided solutions that were comparable to that of linear programming (LP) based algorithm. For the largest instance, the computation time of the GA was approximately 5 times shorter than that of the LP based method. Similarly, Li and Kwan [2003] solved the DSP by GA. The authors tested the algorithm on instances with up to 1873 trips and 50000 duties in the SCP formulation. The proposed method was able to solve the largest instance in 1350 seconds, which had a gap of 3.89% from the best known solution provided by a MIP solver with a computation time of more than 10 hours. De Leone et al. [2011a] devised a GRASP for solving the DSP for instances with up to 161 bus trips from a Italian transport company. The proposed method was compared with an exact method that uses a MIP solver to solve the SCP model with up to 2 million feasible duties. The results showed that the exact method took more than 3 hours to find a feasible integer solution for the largest instance, whereas the GRASP was able to provide a solution in one minute that was almost 30% better than the first feasible solution provided by the exact method. The authors, De Leone et al. [2011b], also compared the devised GRASP heuristic to a hybrid of GRASP and variable neighborhood search (VNS). The results indicated that the hybrid version provided improved solutions with comparable computation times. For the largest instance, improvement was found to be 5.2%. Ma et al. [2016] proposed a VNS heuristic for solving the DSP with instances from a transport company in Beijing, China that had up to 501 bus trips. The authors compared solutions from the proposed method to that of the solutions manually created by the planners in the company. For the

largest instance, the VNS heuristic provided, on average, an improvement of 6% with a computation time of 22 minutes, whereas almost five hours was taken to create the manual plan.

Most of the works published in the literature have aimed at developing methods to solve large sized instances of the DSP, which include complexities that arise in practice. Since the DSP is a highly relevant problem in the public transport industry, some have considered implementing the methods as part of a commercial software system. However, none of the published works address the DSPSC. Several researchers in the 1980s ((Ball et al. [1983]; Darby-Dowman et al. [1988])) recognized the need to integrate the VSP and the DSP (IVDSP), where scheduling of buses and drivers are simultaneously carried out. The integration of the two scheduling problems could lead to further cost reductions and efficiency gains for transport systems (Freling et al. [2003]). However, the IVDSP has received very little attention in the literature due to its increased complexity of formulating and handling large real-life instances that require immense computation times to be solved (Borndörfer et al. [2008]; Huisman et al. [2005]). Combining lagrangian relaxation and column generation have commonly been used to integrate the problems, e.g. - Huisman et al. [2005], and the IVDSP is known to be a growing area of research. By introducing the DSPSC, we believe that there will be implications on the IVDSP, which has to be studied further.

3 Problem Description and Mathematical Modelling

This section presents the mathematical model which serves as the formal description of the problem. Let $\mathcal{T}$ be the set of bus trips that need to be covered and let $\mathcal{D}$ be the set of all valid duties that comply with the labor regulations and the company's operational rules. Each $d \in \mathcal{D}$ is checked to see if it requires one or more staff cars as part of its travel activities and all car travels are grouped into set $\mathcal{C}$. Let $\mathcal{N}$ denote the set of nodes that drivers could visit using a staff car and $r \in \mathcal{N}$ is denoted as the car depot. Each car travel, $i \in \mathcal{C}$ has a departure node k_i, an arrival node l_i, departure time u_i and arrival time v_i. A departure car travel is defined as a car travel that departs from depot r. Set of departure car travels is denoted as $\acute{C} \subset \mathcal{C}$, where $k_i = r$, $l_i = n \in \mathcal{N} \backslash \{r\}$ and $i \in \acute{C}$. Similarly, an arrival car travel is defined as a car travel that arrives at depot r. Set of arrival car travels is denoted as $\hat{C} \subset \mathcal{C}$, where $k_i = n \in \mathcal{N} \backslash \{r\}$, $l_i = r$ and $i \in \hat{C}$. The cost or paid time associated with duty $d \in \mathcal{D}$ is represented as c_d. Binary matrix A is defined, where a_{td} is 1 if duty $d \in \mathcal{D}$ is covering bus trip $t \in \mathcal{T}$ and 0 otherwise. Another binary matrix G is defined, where g_{id} is 1 if duty $d \in \mathcal{D}$ utilizes car travel $i \in \mathcal{C}$ and 0 otherwise. We define a car match as a combination of a departure car travel from the depot and an arrival car travel to the depot to form one round trip as depicted in Figure 2. Binary matrix H is defined, where h_{ij} indicates whether two car travels can be matched as one round trip, i.e. $l_i = k_j$, $v_i \leq u_j, i \in \acute{C}$ and $j \in \hat{C}$. The time a staff car is idle at a node other than the depot is defined as car idle time. To simplify the car matches notation, we define $\mathcal{H}$ as the set of all car matches, i.e. $\mathcal{H} = \{(i,j) \mid h_{ij} = 1\}$.

Four decision variables are defined in the mathematical model. Binary variable x_d indicates if duty $d \in \mathcal{D}$ is selected or not and binary variable y_t indicates if a bus trip $t \in \mathcal{T}$ remains uncovered or not. A penalty of β is incurred if a bus trip is uncovered. Binary variable z_i indicates if car travel $i \in \mathcal{C}$ is used or not and binary variable s_{ij} indicates if car travel $i \in \acute{C}$ is matched with car travel $j \in \hat{C}$ to form one round trip. The maximum number of staff cars that is available at the depot is denoted as Q. As depicted in Figure 3, a staff car can perform multiple round trips during the day. To estimate the number of staff cars that are being used at a particular time, we define $\mathcal{O}$ as the set of all departure times of a staff car from the depot, i.e. $\{u_i\}$ where $i \in \acute{C}$, and $\mathcal{P}_o$ as the set of all possible car matches that are active at time o,

$$\mathcal{P}_o = \{(i,j) \mid (i,j) \in \mathcal{H} \ \wedge \ u_i \leq o \ \wedge \ v_j \geq o\} \qquad \forall o \in \mathcal{O} \qquad (1)$$

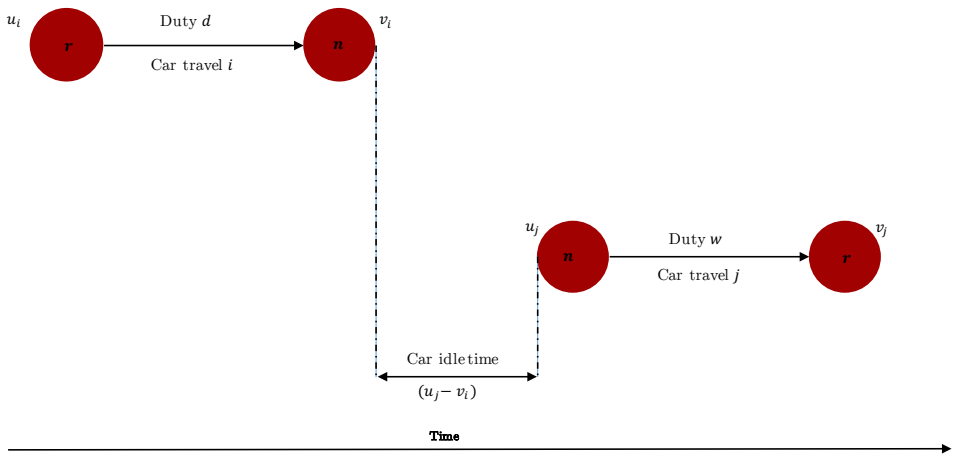


Figure 2: Staff Car Match. Car travel i of duty d departs from depot r to node n and car travel j of duty w arrives at depot r from node n. The idle time of the staff car at node n is calculated as $(u_j - v_i)$. The figure illustrates an example of a round trip.

The mathematical formulation of the DSPSC is as follows,

$$Minimize \sum_{d \in \mathcal{D}} c_d \cdot x_d + \beta \sum_{t \in \mathcal{T}} y_t \tag{2}$$

subject to:

$$\sum_{d \in \mathcal{D}} a_{td} \cdot x_d + y_t \geq 1 \qquad \forall t \in \mathcal{T} \tag{3}$$

$$\sum_{d \in \mathcal{D}} g_{id} \cdot x_d \leq z_i \cdot M \qquad \forall i \in \mathcal{C} \tag{4}$$

$$\sum_{d \in \mathcal{D}} g_{id} \cdot x_d \geq z_i \qquad \forall i \in \mathcal{C} \tag{5}$$

$$\sum_{j \in \acute{C}} h_{ij} \cdot s_{ij} = z_i \qquad \forall i \in \acute{C} \tag{6}$$

$$\sum_{i \in \acute{C}} h_{ij} \cdot s_{ij} = z_i \qquad \forall j \in \hat{C} \tag{7}$$

$$\sum_{(i,j) \in \mathcal{P}_o} s_{ij} \leq Q \qquad \forall o \in \mathcal{O} \tag{8}$$

$$x_d \in \{0,1\} \qquad \forall d \in \mathcal{D} \tag{9}$$

$$y_t \in \{0,1\} \qquad \forall t \in \mathcal{T} \tag{10}$$

$$z_i \in \{0,1\} \qquad \forall i \in \mathcal{C} \tag{11}$$

$$s_{ij} \in \{0,1\} \qquad \forall i \in \acute{C}, \quad \forall j \in \hat{C} \tag{12}$$

The objective (2) is to minimize the overall cost of driver duties and the penalty for leaving a bus trip uncovered. Constraints (3) ensure that a bus trip is covered by at least one duty or is left uncovered. Constraints (4) ensure that a car travel is selected if it is utilized by one or more duties in the final schedule. M is a large number and can be set as the seating capacity of the staff cars. Constraints (5) ensure that a car travel is not selected if none of the duties in the final schedule utilize it. Constraints (6) together with constraints (7) ensure that a selected departure car travel from the

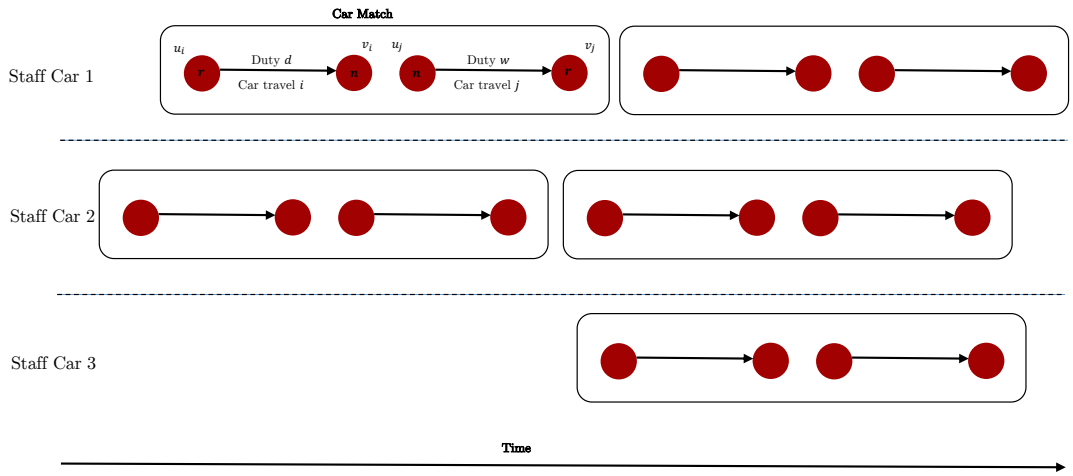


Figure 3: Staff Car Schedule. The final schedule of a staff car can consist of one or more car matches. Staff cars 1 and 2 in the figure illustrate an example of multiple round trips being performed.

depot is matched with an arrival car travel to the depot to form one round car trip. Constraints (8) ensure that at all times during the day the number of staff cars being utilized is not more than the maximum number of staff cars available at the depot.

4 Solution Method

4.1 Greedy heuristic

To construct an initial solution, a basic greedy heuristic is implemented. A duty is selected based on a evaluation function and added to the solution at each iteration of the greedy heuristic. For the DSPSC, two evaluation functions are applied; one for selecting duties and an another for selecting car matches. The duties are evaluated based on a function Δ and we determine $\Delta_d = c_d + \beta(I_d - J_d + K_d)$, where I_d is the number of trips covered by duty d that are already covered in the solution, J_d is the number of trips covered by duty d that are not covered in the solution and K_d is the number of cars travels used by duty d (i.e. $\sum_{i \in \mathcal{C}} g_{id}$). At each iteration, the values of I_d and J_d are adapted based on the trips being covered in the solution and duty with minimum Δ_d is the best candidate to be added to the solution. The newly inserted duty potentially involves car travels and they have to be matched. For instance, duty d might have a departure car travel from depot r to node n and hence would need a matching car travel that would return the car from node n to depot r to form one complete round trip. In the greedy heuristic, the matching car travel is selected based on minimum idle time of the staff car at node $n \in \mathcal{N} \backslash \{r\}$. By returning the staff car quickly back to the depot, it has the possibility of being used for multiple round trips. Hence, improving the utilization of the staff cars is the underlying motivation for minimizing car idle time.

The greedy heuristic procedure is shown in Algorithm 1. At each iteration of the greedy heuristic, a candidate list, q, of duties is created and added to the solution s. E in Line 5 denotes the set of unmatched car travels in the s and is updated (Line 20) when car travels are matched or new unmatched car travels are added to s. For each $i \in E$, all its matching car travels, i.e all the car travels that would form a round car trip with car travel i, are collected in set F. From set F, car travel j that forms a round car trip with minimum car idle time, carIdleTime(i, j), is selected (Line 10). Set G consists of all the duties with car travel j (Line 11) and duty $w = \arg\min_{w \in G}(\Delta_w)$ is selected (Line 12) and added to q (Line 13). In some cases during the advancement of the heuristic, it might select a duty with a car travel that cannot be matched. In such circumstances, the heuristic removes the selected duty from $\mathcal{D}$, empties q and is forced to select the next best duty in terms of Δ

(Line 15 - Line 17). The loop (Line 6 - Line 21) terminates when no more unmatched car travels exist in the partial solution, i.e. set E is empty.

The greedy heuristic terminates when all the bus trips have been covered and all the car travels in s have been matched. Even though the heuristic tries to improve the utilization of a staff car through function carIdleTime(), it cannot control the number of cars being used. Hence, the greedy heuristic often builds an initial solution which does not satisfy the maximum number of cars condition.

Algorithm 1: Greedy heuristic

```
1  Initialization: s ← ∅, q ← ∅;
2  while stop criterion not met do
3      d ← arg min_{d∈D}(Δ_d);
4      q ← q ∪ {d};
5      Set E = {i | g_id = 1, i ∈ C};
6      while E ≠ ∅ do
7          for i ∈ E do
8              Set F = {j | (i, j) ∈ H, j ∈ C};
9              if F ≠ ∅ then
10                 j ← arg min_{j∈F}(carIdleTime(i, j));
11                 Set G = {w | g_jw = 1, w ∈ D};
12                 w ← arg min_{w∈ G}(Δ_w);
13                 q ← q ∪ {w};
14             else
15                 D ← D − {d};
16                 q ← ∅;
17                 go to 3;
18             end
19         end
20         Update E;
21     end
22     s ← s ∪ q;
23     q ← ∅;
24 end
25 return s
```

4.2 Matheuristic

In our matheuristic setting, the mathematical model described in Section 3 is embedded in an ALNS framework to obtain high-quality solutions in reasonable computation time.

A neighborhood is defined as the set of neighboring solutions of a current solution and a local search procedure iteratively moves the current solution to a neighboring solution. In LNS, the neighboring solutions could be reached by applying a destroy method and then a repair method to the current solution. Hence, the neighborhood is implicitly defined by the destroy and repair methods. In ALNS, multiple destroy and repair methods are applied and hence different neighborhoods can be explored within the same search. Each of the destroy and repair methods is assigned a modifiable weight which is updated based on the performances of the methods during the course of the search. Røpke and Pisinger [2006] state that not all destroy and repair methods perform equally well and that, for example, one method might be very well-suited for one type of instance and another method might be well-suited for another instance. The diverse and robust nature of the ALNS heuristic has led to its gain in popularity in recent years and has been applied to a large selection of different optimization problems. Some applications include the capacitated arc-routing problem (Laporte

et al. [2010]), the vehicle routing problem (Pisinger and Røpke [2007]) and the patient admission scheduling problem (Lusby et al. [2016]).

Algorithm 2: Adaptive Large Neighborhood search

```
1  Initialization: s ← InitialSolution(), s* ← s;
2  ρ ← InitializeMethodWeights();
3  while stop criteria not met do
4      Select destroy and repair methods μ ∈ τ⁻ and γ ∈ τ⁺ using ρ;
5      s' ← Destroy(s, μ) ;
6      s' ← Repair(s', γ) ;
7      if Accept(s, s') then
8          s ← s';
9      end
10     if f(s') < f(s*) then
11         s* ← s';
12     end
13     ρ ← UpdateMethodWeights();
14 end
15 return s*
```

Algorithm 2 outlines the ALNS procedure where s denotes the current solution, s' is the neighboring solution and s^* is the best solution. The set of all destroy methods is denoted as τ^- and the set of repair methods is denoted as τ^+. As shown in Line 4, at each iteration of the heuristic a destroy method $\mu \in \tau^-$ and a repair method $\gamma \in \tau^+$ are selected to perform an operation on the current solution. The selection of the methods are dependent on the weights of the methods, ρ^μ and ρ^γ, which are dynamically updated during the execution of the heuristic. Well-performing methods have a high weight and thus would have a higher probability of being selected. The probability of a destroy method being selected is determined as,

$$\zeta_\mu = \frac{\rho^\mu}{\sum_{l \in \tau^-} \rho^l} \qquad \forall \mu \in \tau^- \tag{13}$$

Similarly, the probability of selecting a repair method is determined as,

$$\zeta_\gamma = \frac{\rho^\gamma}{\sum_{l \in \tau^+} \rho^l} \qquad \forall \gamma \in \tau^+ \tag{14}$$

The selection of the destroy and the repair method is made based on a *roulette wheel* principle using the probabilities calculated in Equations (13) and (14). The entire search is divided into n_{seg} segments and each segment is defined by n_{iter} iterations. At the end of each segment, the weights of the methods are updated, as shown in Line 13. For each destroy method $\mu \in \tau^-$ and repair method $\gamma \in \tau^+$, Ω^μ and Ω^γ define the accumulated score. The number of times a destroy method and a repair method have been selected during a segment is given by ν^μ and ν^γ respectively. At each iteration of the heuristic, a score of ψ is awarded to the chosen destroy and repair method and added to Ω^μ and Ω^γ. The score is given based on the quality of the solution obtained and could be one of ψ_1, ψ_2, ψ_3 or ψ_4. The description of the score parameters is shown in Table 1, where $\psi_1 > \psi_2 > \psi_3 > \psi_4$. The weights of destroy and repair methods are initialized to 1 and after each segment, the weights are updated as follows,

$$\rho^\mu = (1 - \lambda) \cdot \rho^\mu + \lambda \cdot \frac{\Omega^\mu}{\nu^\mu} \qquad \forall \mu \in \tau^- \tag{15}$$

$$\rho^{\gamma} = (1-\lambda)\cdot\rho^{\gamma} + \lambda\cdot\frac{\Omega^{\gamma}}{\nu^{\gamma}} \qquad \forall\gamma\in\tau^{+} \tag{16}$$

At the start of each segment, Ω^{μ}, Ω^{γ}, ν^{μ} and ν^{γ} are set to 0. $\lambda\in[0,1]$ is known as the reaction factor which controls the changes in weights. If $\lambda=1$ then the *roulette wheel* selection is only based on the scores of the most recent segment and if $\lambda=0$ then the weights are kept constant at the initial level. The past performances of the methods are taken into account when $0<\lambda<1$. The weight of a method remains unchanged if it was not selected in the segment.

Score(ψ)	**Description**
ψ_1	if the new solution is a new best solution
ψ_2	if the new solution is better than the current solution
ψ_3	if the new solution is accepted
ψ_4	if the new solution is rejected

Table 1: Score parameters for ALNS

Shaw [1998] proposed a hill climber accept criterion in a LNS framework where only improving solutions are accepted. This acceptance criterion, however, has the tendency to get trapped in a local optimum. To diversify the search, solutions that are worse than the current solution should be accepted occasionally. Hence, a score of ψ_3 is rewarded to methods that are able to visit unexplored solution spaces. One approach of introducing diversification to the search procedure is the simulated annealing (SA) acceptance criterion which has been successfully used in an ALNS framework by some authors(see eg. Røpke and Pisinger [2006] and Lusby et al. [2016]). Given a current solution s, a worse solution s' is accepted with a probability of $\exp-\dfrac{f(s')-f(s)}{\theta}$, where $\theta>0$ is the temperature. The heuristic starts with an initial temperature, $\theta=\theta_{start}$, and this is gradually decreased during the course of the heuristic with the aid of a cooling factor $\alpha\in(0,1)$. The temperature is decreased to $\theta=\theta\cdot\alpha$ at the end of each segment. During the last iterations of the ALNS, worse solutions are unlikely to be accepted and hence the framework behaves like a hill climber. Similar to authors Røpke and Pisinger [2006], we determine θ_{start} based on the problem instance at hand, i.e. θ_{start} is set such that a solution $\omega\%$ worse than the initial solution is accepted with probability 0.5. In our case, the initial solution is obtained from the greedy heuristic for the DSPSC described in Section 4.1. However, the greedy heuristic does not ensure feasibility with respect to the number of staff cars being used. In order to avoid setting θ_{start} too large, the penalty incurred for violating the maximum number of staff cars constraint (8) is disregarded. The heuristic terminates when it has performed n_{seg} segments.

Because the greedy heuristic often results in an infeasible solution, the ALNS heuristic is restarted when it reaches the first feasible solution. When restarting the heuristic, the weights of the methods are also reinitialized and θ_{start} is recalculated based on the first feasible solution. However during the execution of the heuristic, we allow it to visit infeasible regions of the solution space. It is believed that local search heuristics often have difficulties in moving from one promising area of the solution space to another in tightly constrained problems (Røpke and Pisinger [2006]). To tackle this, some authors, e.g.- Cordeau et al. [2001] and Lourenço et al. [2001], allow the search to visit infeasible solutions by relaxing some constraints. Similarly, in addition to the maximum number of staff cars constraint (8), we relax the car matching constraints (6) and (7) to allow for unmatched car travels in the solution; however, a penalty is added to the objective function when a car travel is unmatched.

4.2.1 Destroy Methods

Given a solution, let $\bar{\mathcal{D}}$, $\bar{\mathcal{C}}$ and $\bar{\mathcal{H}}$ denote the set of duties, car travels and car matches in the solution respectively. Sets $\mathcal{D}^{'}$, $\mathcal{C}^{'}$ and $\mathcal{H}^{'}$ denote the duties, car travels and car matches not in the solution. For the DSPSC, we propose three destroy methods and these are as follows,

1. Random removal of duties
   To diversify the search, random duties are removed from $\bar{\mathcal{D}}$ and the set of removed duties is denoted as $\mathcal{D}^R$. The number of duties to be removed, $|\mathcal{D}^R|$, is controlled by the degree of destruction parameter, ξ, and is determined as $1 \leq |\mathcal{D}^R| \leq \xi \cdot |\bar{\mathcal{D}}|$. The car travels to be removed from the solution are dependent on the duties removed from the solution, i.e. $\mathcal{C}^R = \{i \mid g_{id} = 1,\ i \in \bar{\mathcal{C}},\ d \in \mathcal{D}^R\}$, and the car matches to be removed are determined as $\mathcal{H}^R = \{(i,j) \mid (i,j) \in \bar{\mathcal{H}}\ \wedge\ (i \in \mathcal{C}^R\ \vee\ j \in \mathcal{C}^R)\}$. In most cases, when car travels and car matches are removed from the solution, the destroyed solution consists of unmatched car travels. For instance, a car travel i is in the solution but its matching car travel j was removed from the solution. The set of unmatched car travels in the destroyed solution is represented as $\mathcal{C}^U = \{i \mid (i,j) \in \mathcal{H}^R,\ i \in \bar{\mathcal{C}},\ j \in \mathcal{C}^R\}$.

2. Worst removal of duties
   The function Δ, as described in Section 4.1, prefers duties that cover many of the bus trips with minimum overcoverage and car travels. Hence, as part of intensification strategy, duty d given by $\arg\max_{d \in \bar{\mathcal{D}}}(\Delta_d)$ is likely to be removed from the solution. Duties in the solution, $\bar{\mathcal{D}}$, are sorted in descending order of Δ_d. The duties to be removed are determined as, $\mathcal{D}^R = \mathcal{D}^R \cup \bar{\mathcal{D}}[q^B \cdot |\bar{\mathcal{D}}|]$ where q is a random number in the interval [0,1) and $B \geq 1$ is a degree of randomization parameter that controls the randomness in the selection of the duties. A low value of B corresponds to random selection of duties and a high value corresponds to selecting duty with highest Δ_d value. The number of duties to be removed, $|\mathcal{D}^R|$ is determined in the similar manner as the random removal of duties using the ξ parameter.

3. Random removal of car travel matches
   The aforementioned destroy methods do not specifically target the car schedule in the solution where there are probably a larger number of staff cars being used than strictly required. The methods leave partial car matches in the solution and hence do no guarantee making significant changes in the car schedule. To address this issue, car matches in the solution, $\bar{\mathcal{H}}$, are randomly selected and removed from the solution with the objective of reducing the number of cars. The set of car matches to be removed is denoted as $\mathcal{H}^R = \{(i,j) \mid (i,j) \in \bar{\mathcal{H}}\}$ and is also controlled by the degree of destruction parameter such as $1 \leq |\mathcal{H}^R| \leq \xi \cdot |\bar{\mathcal{H}}|$.
   $\mathcal{C}^R$ is defined as the set of all car travels in $\mathcal{H}^R$ and $\mathcal{D}^R$ is defined as the set of duties in solution that contain removed car travels, i.e. $\mathcal{D}^R = \{d \mid g_{id} = 1,\ i \in \mathcal{C}^R,\ d \in \bar{\mathcal{D}}\}$.

The removed duties, car travels and car matches, $\mathcal{D}^R, \mathcal{C}^R$ and $\mathcal{H}^R$, are added to $\mathcal{D}^{'}, \mathcal{C}^{'}$ and $\mathcal{H}^{'}$ respectively.

4.2.2 Repair Methods

The repair methods of the ALNS are mainly intended to be fast heuristics; simple greedy insertion and regret heuristics have regularly been applied as repair methods (Pisinger and Røpke [2007]; Lusby et al. [2016]). The DSPSC is a very tightly constrained problem where exploring a neighborhood for a feasible solution could be a challenge. To tackle the challenge of finding feasible solutions for the lot sizing problem, Muller et al. [2012] used a MIP solver for repairing solution. Similarly, we use a MIP solver (ILOG CPLEX) as part of the repair phase of the heuristic. Muller et al. [2012] define two MIP based repair methods and differentiate them by either fixing or bounding the variables based on their values in the destroyed solution. However, in our approach, we differentiate the repair methods by the neighborhood defined for the MIP solver. The variables in the destroyed solution, given by $\bar{\mathcal{D}}$, $\bar{\mathcal{C}}$ and $\bar{\mathcal{H}}$, are used as a starting solution for the MIP solver but are set as "free" variables. The repair methods reduce the search space by defining the neighborhoods, $\mathcal{D}^N, \mathcal{C}^N$ and $\mathcal{H}^N$, for the MIP solver

which is capable of exploring numerous solutions within the defined neighborhood. Consequently, the repair methods are evaluated based on the quality of the constructed neighborhoods.

By solving a restricted subproblem, the MIP solver helps the local search to move from the current solution to a neighboring solution and the new improved solution determines the neighborhood that will be defined by the local search. Limiting the number of branch nodes to be explored or the time permitted for the MIP solver were suggested by Muller et al. [2012] in order to speed up the solution process. We keep a time limit, n_{time}, and the solution generated by the MIP solver is used to evaluate the repair methods.

After a solution has been destroyed, there are unmatched car travels, $\mathcal{C}^U$, and uncovered bus trips, $\mathcal{T}^U = \{t \mid \sum_{d \in \bar{\mathcal{D}}} a_{td} = 0,\ t \in \mathcal{T}\}$ in the solution. Hence, we define a repair method that constructs a neighborhood to focus primarily on covering the uncovered bus trips in the solution and another repair method that focuses on matching the unmatched car travels in the solution. The descriptions of the repair methods are as follows,

1. Neighborhood defined by duties that cover the uncovered bus trips
   Given a set of uncovered bus trips in the solution, $\mathcal{T}^U$, the neighborhood is formally defined as the set of duties that cover at least one of the trips as shown in Equation (17).

   $$\mathcal{D}^N = \{d \mid \sum_{t \in \mathcal{T}^U} a_{td} \geq 1,\ d \in \mathcal{D}^{'}\} \tag{17}$$

   Depending on the duties in the neighborhood, the set of car travels in the neighborhood is defined as $\mathcal{C}^N = \{i \mid g_{id} = 1,\ i \in \mathcal{C}^{'},\ d \in \mathcal{D}^N\}$ and subsequently the neighborhood car matches as $\mathcal{H}^N = \{(i,j) \mid (i,j) \in \mathcal{H}^{'},\ i \in \mathcal{C}^N,\ j \in \mathcal{C}^N\}$.

2. Neighborhood defined by duties that match with the unmatched car travels
   Given a set of unmatched car travels in the solution, $\mathcal{C}^U$, we denote $\mathcal{C}^M$ as the set of car travels from $\mathcal{C}^{'}$ that can match with the unmatched car travels. For instance, car travel $j \in \mathcal{C}^{'}$ can match with unmatched car travel $i \in \bar{\mathcal{C}}$, i.e $(i,j) \in \mathcal{H}^{'}$. Consequently, the set is defined as $\mathcal{C}^M = \{j \mid (i,j) \in \mathcal{H}^{'},\ i \in \bar{\mathcal{C}},\ j \in \mathcal{C}^{'}\}$. Therefore, the neighborhood is defined as the set of duties that contain one or more car travels that can match with the unmatched car travels in the solution as shown in Equation (18). The car travels and car matches in the neighborhood are defined based on $\mathcal{D}^N$ in similar manner as the previously described repair neighborhood.

   $$\mathcal{D}^N = \{d \mid g_{id} = 1,\ d \in \mathcal{D}^{'},\ i \in \mathcal{C}^M\} \tag{18}$$

The size of the neighborhood, $\mathcal{D}^N$, depends on the impact of the destroy methods and in most instances, destroying even a small fraction of the current solution creates a large neighborhood. For the DSP, Lourenço et al. [2001] considered a candidate list strategy where duties were evaluated based on a *penalized cost* and only duties with a *cost* less than or equal to the average *cost* were inserted in the candidate list. Similarly, we create a duty candidate list, where only the best η_{duty} duties in terms of Δ_d, where $d \in \mathcal{D}^N$, are considered. The candidate list makes the subproblem tractable for the MIP solver and provides solutions in quick computation time. The size of $\mathcal{H}^N$ could also potentially be quite large and hence the size of the neighborhood is controlled by carIdleTime() of car matches. For example, the staff car match candidate list only considers matches that are less than a maximum car idle time (η_{car}) of 120 minutes, i.e. carIdleTime$(i,j) \leq 120$ where $(i,j) \in \mathcal{H}^N$. The matheuristic procedure is shown in Algorithm 3.

5 Computational Study

5.1 Instances

Table 2 provides an overview of the test instances obtained from a Danish and a Swedish transport company. *SE1_OP* represents the instances from the Swedish company and the instances from the

Algorithm 3: Matheuristic

Initialization: $s \leftarrow$ `GreedyAlgorithm()`, $s^* \leftarrow s$;
$\theta \leftarrow$ `CalculateInitialTemperature(`s, ω`)`;
$\rho \leftarrow$ `InitializeMethodWeights()`;
$\Omega \leftarrow$ `InitializeMethodScores()`;
$\nu \leftarrow$ `InitializeMethodAttempts()`;
for $\kappa \leftarrow 1, n_{seg}$ **do**
  for $\eta \leftarrow 1, n_{iter}$ **do**
    Select destroy and repair methods $\mu \in \tau^-$ and $\gamma \in \tau^+$ using ρ;
    $s' \leftarrow$`Destroy(`s, μ, ξ, B`)`;
    $s' \leftarrow$`Repair(`s', γ, η_{duty}, η_{car}, n_{time}`)`; `// Solve using MIP solver`
    $\delta \leftarrow f(s') - f(s)$;
    if $\delta < 0$ *or* $\exp \frac{-\delta}{\theta} > random[0,1)$ **then**
      $s \leftarrow s'$;
    end
    if $f(s') < f(s*)$ **then**
      $s^* \leftarrow s'$;
    end
    $\Omega \leftarrow$`UpdateMethodScores(`ψ`)`;
    $\nu \leftarrow$ `UpdateMethodAttempts()`;
  end
  $\rho \leftarrow$ `UpdateMethodWeights(`Ω, ν, λ`)`;
  $\Omega \leftarrow$ `ResetMethodScores()`;
  $\nu \leftarrow$ `ResetMethodAttempts()`;
  $\theta \leftarrow \theta \cdot \alpha$;
end
return s^*

Danish company are represented by *DK1_OP* and *DK2_OP*. Instances *SE1_OP5*, *DK1_OP6* and *DK2_OP9* are known to be the complete instances that were used to extract other instances. The three complete instances are highlighted in light gray in Table 2. The instances were categorized into small, medium and large sized instances so that the matheuristic could be tested for a wide range of instances. Some of the instances are larger than what one would find currently in the literature. The instances are available at http://doi.org/10.5281/zenodo.1442661.

Table 2 shows that all instances involve the use of staff cars. In the DSP, travels by foot for short distances and bus travels are commonly allowed. A bus travel usually occurs when the driver is a passenger on another bus to reach a designated bus stop and the set covering constraints (3) allow drivers to use bus travels. One could argue that staff cars may not be needed if bus stops could be reached by bus or by foot. To analyze the importance of staff cars, the small and medium sized instances are tested with and without car travels. It was found that all bus trips could be covered when staff cars are put into use. However, when the instances do not involve car travels, it was found that, on an average, 41% of the bus trips could not be covered. The labor rules such as the minimum number of breaks and maximum time between breaks highly influence the feasibility of driver duties. Moreover, due to limitations in the city network where breaks are allowed only at a few bus stops, drivers have to travel between bus stops to have sufficient breaks during the day. Car travel is the most suitable option that allows drivers to reach bus stops in a timely manner and it is, hence, argued that staff cars are often necessary to generate feasible driver duties.

The mathematical formulation (2) - (12) solves the DSPSC using an integrated approach where the drivers and the staff cars are scheduled simultaneously. Another method of solving the DSPSC is by a sequential approach, where the DSP is solved first and independent of the staff car problem.

Category	Instance	$\|\mathcal{T}\|$	$\|\mathcal{D}\|$	$\|\mathcal{C}\|$	$\|\mathcal{H}\|$	$\mathcal{Q}$
Small	*SE1_OP1*	44	1239	91	345	4
	SE1_OP2	39	8880	100	391	1
	DK1_OP1	23	754	65	148	1
	DK2_OP1	73	1789	140	781	1
	DK1_OP2	84	7660	149	2621	2
	DK2_OP2	96	18370	280	4134	1
Medium	*SE1_OP3*	131	39683	294	3081	4
	DK1_OP3	152	41908	302	8176	2
	SE1_OP4	217	193652	501	8372	5
	DK1_OP4	279	195972	710	17744	4
	DK2_OP3	305	86703	753	20330	3
Large	*SE1_OP5*	293	621508	731	12293	6
	DK1_OP5	384	686499	1149	34558	5
	DK2_OP4	649	511803	1514	64238	4
	DK2_OP5	840	686370	1862	77778	5
	DK2_OP6	924	752705	2141	91371	6
	DK1_OP6	571	1205058	1746	50023	6
	DK2_OP7	1211	1015011	2852	150205	8
	DK2_OP8	1414	1187194	3512	189671	12
	DK2_OP9	1769	1738055	4560	267506	16
	DK2_OP10	1769	1738055	4560	267506	15

Table 2: Size of test instances. $|\mathcal{T}|$ represents the number of bus trips, $|\mathcal{D}|$ represents the number of duties generated, $|\mathcal{C}|$ represents the number of number of car travels, $|\mathcal{H}|$ represents the number of car matches and $\mathcal{Q}$ represents the number of staff cars at the depot.

After solving the DSP, car travels in the final set of duties are chosen as the input for the staff car problem, which is concerned with finding a feasible set of car matches that respect the number of staff cars available at the depot. The small and medium sized instances are solved by integrated and sequential approaches using a MIP solver. Table 3 compares the two approaches and the results show that the sequential approach is superior to the integrated approach in terms of total paid time for drivers, where the average improvement for small and medium sized instances are 5.31% and 2.13% respectively. However, the sequential approach often leads to infeasible solutions with the medium sized instances having, on an average, 10 unmatched car travels out of 47. The integrated approach provides feasible solutions; however, the computation time required to solve the DSPSC is significantly larger than that of the sequential approach. For example, the integrated approach did not find the optimal solution in 10 hours for the medium sized instance *DK2_OP3*, whereas only 18 seconds were required by the sequential approach. This computational study shows that simultaneously scheduling the drivers and the staff cars is a highly complex problem that often requires long computation times. Due to the computational advantage of the sequential approach, it can be considered for solving the DSPSC with alternative services. For example, taxis can be used by transport companies to fulfill the unmatched car travels in the solution. Transport companies would have to consider the commercial viability of using taxis to fulfill such car travels without losing much of the 2.13% savings made by the sequential approach for medium sized instances. However, the vehicle policies of companies we work with do not allow for any outsourced services and the drivers are required to use the staff cars. Hence, the DSPSC only considers solving the problem with a given number of staff cars at the depot and the work carried out in this paper does not consider alternative services.

Category	Instance	Integrated					Sequential				
		solution	gap(%)	$\lvert\bar{\mathcal{C}}\rvert$	$\lvert\mathcal{C}^U\rvert$	time (*sec*)	solution	gap(%)	$\lvert\bar{\mathcal{C}}\rvert$	$\lvert\mathcal{C}^U\rvert$	time (*sec*)
Small	*SE1_OP1*	4883	0.00	24	0	0.39	4473	0.00	26	2	0.08
	SE1_OP2	3144	0.00	4	0	0.66	2925	0.00	8	6	0.07
	DK1_OP1	1914	0.00	12	0	0.08	1914	0.00	12	0	0.02
	DK2_OP1	3120	0.00	22	0	0.72	2945	0.00	4	4	0.03
	DK1_OP2	5867	0.00	26	0	1.7	5867	0.00	25	1	0.24
	DK2_OP2	2795	0.00	20	0	105.75	2494	0.00	4	4	0.16
Medium	*SE1_OP3*	10925	0.00	30	0	265.06	10700	0.00	30	8	2.05
	DK1_OP3	10927	0.00	54	0	62.23	10890	0.00	51	7	1.66
	SE1_OP4	17861	1.36	46	0	36000.55	17424	0.00	44	12	14.16
	DK1_OP4	20253	0.00	88	0	4282.08	20226	0.00	80	6	18.69
	DK2_OP3	12641	2.32	40	0	36000.59	11925	0.00	28	16	17.58

Table 3: Comparison between integrated and sequential approaches. $|\bar{\mathcal{C}}|$ represents the number of car travels in the final solutions and $|\mathcal{C}^U|$ represents the number of unmatched car travels in the solution.

5.2 Parameter setup

The number of segments, n_{seg}, is set to 50 and the number of iterations to be performed in each segment, n_{iter}, is 15. For the SA accept criterion, θ_{start} is calculated such that a solution 5% (ω) worse than $f(s)$ is accepted with probability 0.5, i.e $\theta_{start} = \dfrac{-f(s) * 0.05}{\log 0.5}$, and α is set to 0.8. Table 4 shows the results for different values of α. The average number of solutions accepted in the SA framework decreases as α is decreased. The solution quality is determined by calculating the average gap between the solutions obtained from the matheuristic and the best known solution obtained from the MIP solver.

α	**0.99**	**0.9**	**0.8**	**0.7**
Avg. accepted solutions	107.8	67.4	40	23.8
Avg. gap(%)	-1.03	-2.23	-2.51	-2.02

Table 4: Test results for parameter α.

The score parameters of the matheuristic are $\psi_1 = 25$, $\psi_2 = 15$, $\psi_3 = 5$ and $\psi_4 = 0$, and the reaction factor λ is set to 0.1. For the destroy methods, the degree of destruction parameter, ξ, is set to 0.2, 0.1 and 0.025 for small, medium and large instances respectively. The degree of randomization B for the worst removal is set to 4. The time limit n_{time} of the MIP solver in the repair methods is set to 0.5, 2 and 3 seconds for small, medium and large instances respectively. For setting η_{duty} and η_{car}, tests were performed on a Danish (*DK2_OP9*) and a Swedish instance (*SE1_OP5*). Table 5 shows that the size of the neighborhood varies depending on the problem instance and on the applied repair method. Parameter η_{duty} was tested with different values as shown in Table 6, where it can be seen that the average size of $\mathcal{H}^N$ increases as η_{duty} is increased. The chosen value for parameter η_{duty} is 6000 and parameter η_{car} is adapted based on the size of $\mathcal{H}^N$. If $|\mathcal{H}^N| \geq 14000$, which is the approximate average from Table 6, then η_{Car} is set to 120 else it is set to 180.

Instance	**Repair method 1**	**Repair method 2**
DK2_OP9	117227.67	149249.63
SE1_OP5	40072.08	31851.04

Table 5: Average size of $\mathcal{D}^N$ defined by the repair methods.

Instance	η_{duty}		
	5000	**6000**	**7000**
DK2_OP9	20636.87	22320.3	24381.63
SE1_OP5	5538.86	5939.23	6226.83

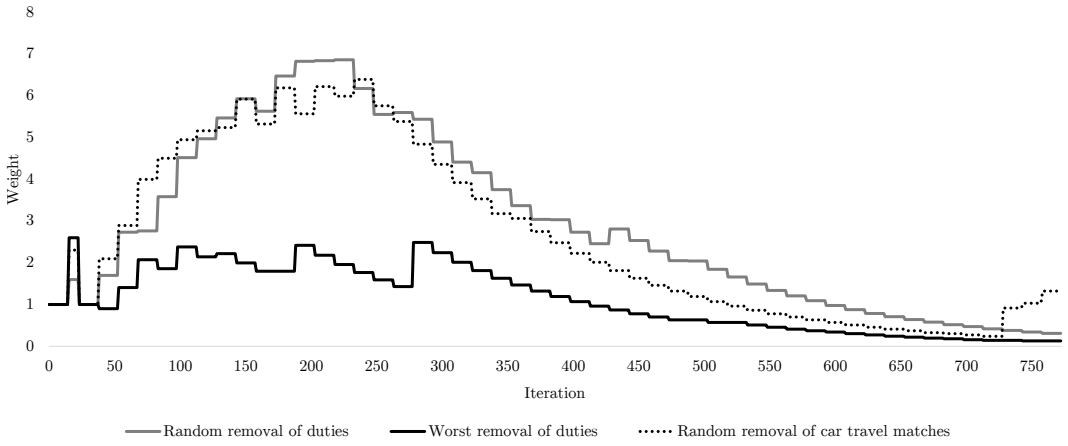
Table 6: Average size of $\mathcal{H}^N$ for different values of parameter η_{duty}.

5.3 Performance of destroy and repair methods

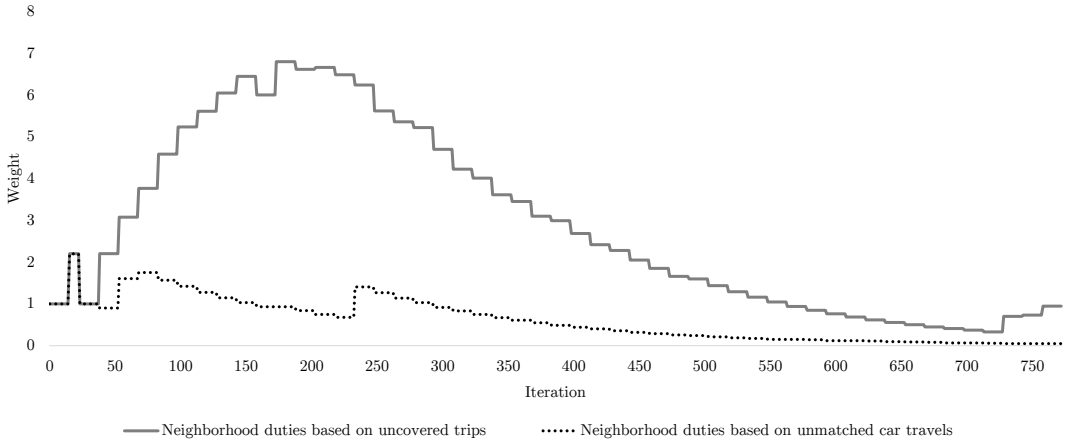
Instances *SE1_OP5* and *DK2_OP9* are tested with different combinations of destroy and repair methods of the matheuristic as shown in Table 7. Each combination or strategy is tested 5 times on the instances and Table 7 reports the average of the 5 runs. An observation made from the study is that strategies involving repair method 2 comparatively provide weaker solutions and, in some cases, do not yield feasible solutions within the iteration limit of the matheuristic. Repair method 1 consistently performs well when combined with the destroy methods. Figures 4 and 5 show an example of how the weights of the repair and destroy methods progressed during the execution of the matheuristic for the instances from the Swedish and Danish transport companies. The figures illustrate that the neighborhood based on the uncovered bus trips (repair method 1) outperforms the neighborhood based on unmatched car travels (repair method 2). For *DK2_OP9* instance, it was observed that strategies with repair method 2 often created a large neighborhood that increased the computation time required for defining the duty candidate list η_{duty}. Hence, the computation time for strategies with repair method 2 was, on average, 1.5 times longer than that of the strategies with repair method 1. Moreover, since repair method 2 does not consider the uncovered bus trips in the destroyed solution, it appears to define an ineffective neighborhood. Repair method 2 was initially developed for diversifying the search space; however, the results clearly indicate that the method does not aid the matheuristic much in improving the solution quality. Thus, it is decided to remove repair method 2 from the matheuristic setup.

Instance	Strategy	Avg. gap (%)	Avg. time (sec)
SE1_OP5	Destroy method 1, Repair method 1	5.35	2355.27
	Destroy method 2, Repair method 1	5.89	1728.54
	Destroy method 3, Repair method 1	4.64	3350.43
	All destroy methods, Repair method 1	3.69	2245.61
	Destroy method 1, Repair method 2	inf	2567.84
	Destroy method 2, Repair method 2	inf	2933.47
	Destroy method 3, Repair method 2	15.09	1452.27
	All destroy methods, Repair method 2	inf	2312.02
	All destroy methods, all repair methods	4.26	2192.77
DK2_OP9	Destroy method 1, Repair method 1	-1.21	4879.65
	Destroy method 2, Repair method 1	-1.8	3962.99
	Destroy method 3, Repair method 1	-1.78	4213.43
	All destroy methods, Repair method 1	-2.69	4435.53
	Destroy method 1, Repair method 2	6.41	7070.09
	Destroy method 2, Repair method 2	8.23	7933.52
	Destroy method 3, Repair method 2	4.08	6180.8
	All destroy methods, Repair method 2	3.56	6567.9
	All destroy methods, all repair methods	-1.67	5078.36

Table 7: Performance of destroy and repair methods. The results are based on an average of 5 runs and 'inf' indicates that a feasible solution could not be found within the iteration limit of the matheuristic in any one of the runs.



(a) Destroy methods



(b) Repair methods

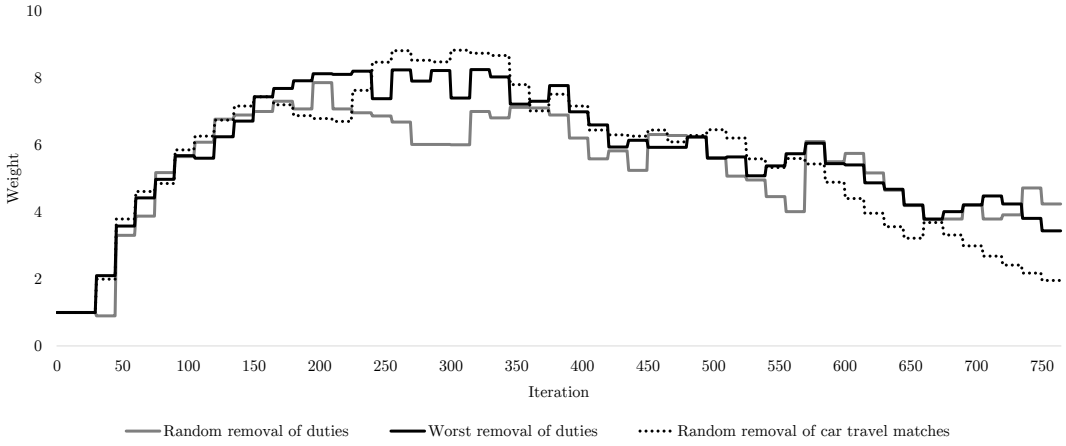
Figure 4: An example of performance of destroy and methods for an instance from Swedish transport company. (x-axis shows the iteration number and y-axis shows the weight of the methods.)

5.4 Results

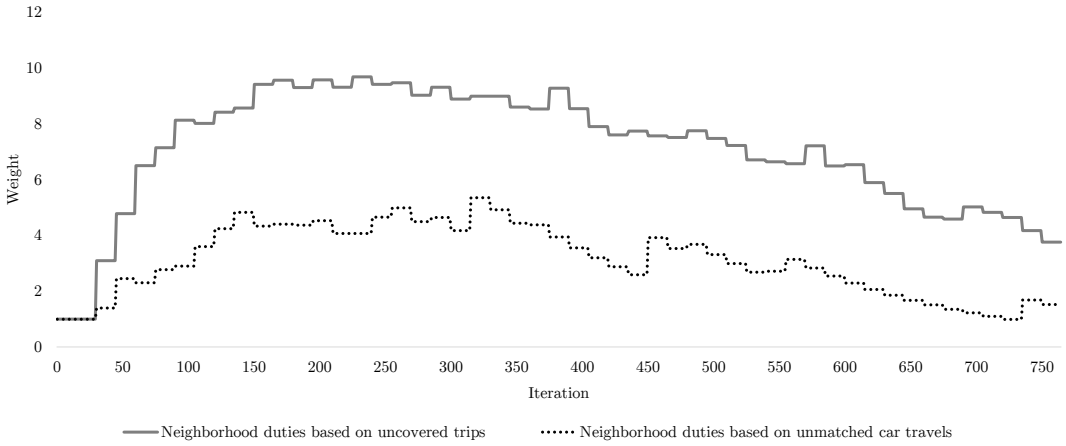
The solutions obtained from the MIP solver (ILOG CPLEX 12.8) are used as benchmarks to evaluate the performance of the matheuristic. The instances are solved by the MIP solver on an Intel Xeon E5-2680 v2 @ 2.80GHz with 128 GB memory and the results from using single thread and multi threads (10 threads) are reported in Table 8. The time limit of the MIP solver is 36,000 seconds (10 hours). Due to practical limitations of extracting data from TGE's integrated system, the matheuristic could not be tested on the machine that was used to provide the benchmark solutions. Table 9 shows the summary of the results from the matheuristic that was tested on an Intel core i5-5287U @ 2.9 GHz machine with 16 GB memory. The matheuristic is implemented in Java and has been run 10 times for each instance and the results are calculated as the average of the 10 runs.

For single thread applications, the processor used to test the matheuristic is comparable to the processor used to run the MIP solver in terms of speed[1]. However, the overall CPU benchmarks reveal that the processor used for the MIP solver is approximately 3.4 times faster than the processor

[1]According to CPUbenchmark, the processor used to run the MIP solver has a single thread rating of 1810, whereas the single thread rating of the processor used to test the matheuristic is 1847.



(a) Destroy methods



(b) Repair methods

Figure 5: An example of performance of destroy and methods for an instance from Danish transport company. (x-axis shows the iteration number and y-axis shows the weight of the methods.)

used to test the matheuristic[2]. The results in Table 8 and Table 9 are reported as observed. The best solution obtained from running the MIP solver single and multi threaded is used to evaluate the performance of the matheuristic. In most cases, the MIP solver provided best solutions in the multi thread environment. For two instances (*DK2_OP6* and *DK2_OP7*), the best solutions are obtained in the single thread environment.

The small instances are solved to optimality with ease by the MIP solver. The matheuristic achieves comparable results for all small instances except one (*DK2_OP2*), which has an average gap of 2.33% from the optimal solution. For medium sized instances, the MIP solver fails to prove optimality for two instances (*SE1_OP4* and *DK2_OP3*) within the time limit (10 hours); however, the integrality gap is very small ($< 1\%$). The matheuristic provides solutions less than 2.5% from optimality for instances with optimal solutions. For the two instances that could not be solved to optimality, the gaps are found to be 3.72% and 5.93% respectively. For large instances, the MIP solver could prove optimality for only one instance (*DK1_OP5*) and the integrality gap for large instances of *DK2_OP* (*DK2_OP7* to *DK2_OP10*) is quite large, an average gap of 14.89%. The matheuristic finds improved solutions for 4 out of the 10 large instances and the improvement is found to be 0.28 to

[2]The overall CPU rating of the processor used for the MIP solver is 15752 while that of the mathheuristic is 4681.

6.95% on average. Moreover, the time taken to obtain these solutions are less than 80 minutes. One of the major drawbacks of the heuristic is that it does not provide any lower bound (LB) information that could be used to evaluate the quality of the improved solutions. However, by considering the LB information provided by the MIP solver, it is estimated that the 4 improved instances are around 11 to 12.77% from the optimal solution. Table 9 also shows the average time taken by the matheuristic to find the first feasible solution. With the aid of the greedy heuristic (Algorithm 1), feasible solutions for large instances are found in the range of 37 seconds to 14 minutes.

Category	**Instance**	**MIP (single thread)**			**MIP (multi thread)**		
		solution	gap(%)	time (*sec*)	solution	gap(%)	time (*sec*)
Small	*SE1_OP1*	4883	0.00	0.38	4883	0.00	0.33
	SE1_OP2	3144	0.00	0.79	3144	0.00	0.44
	DK1_OP1	1914	0.00	0.09	1914	0.00	0.11
	DK2_OP1	3120	0.00	0.99	3120	0.00	0.49
	DK1_OP2	5867	0.00	1.28	5867	0.00	1.1
	DK2_OP2	2795	0.00	127.72	2795	0.00	40.35
Medium	*SE1_OP3*	10925	0.00	488.64	10925	0.00	130.31
	DK1_OP3	10927	0.00	178.42	10927	0.00	48.21
	SE1_OP4	17765	0.8	36000.2	17765	0.43	36005
	DK1_OP4	20253	0.00	5476.36	20253	0.00	1002.92
	DK2_OP3	12650	2.83	36000.3	12466	0.91	36003.9
Large	*SE1_OP5*	23833	2.01	36000.4	23506	0.25	36004
	DK1_OP5	27773	0.00	7604.25	27773	0.00	3687.24
	DK2_OP4	26606	14.71	36000.5	25347	10.63	36002
	DK2_OP5	36568	15.4	36002	35056	11.75	36002.9
	DK2_OP6	39081	10.72	36003.4	39743	12.2	36003.5
	DK1_OP6	42713	1.8	36000.2	42348	0.92	36005.7
	DK2_OP7	57507	11.27	36003.6	63053	19.03	36005.7
	DK2_OP8	76733	16.61	36010.5	74222	13.63	36009.3
	DK2_OP9	96603	21.14	36014.3	90094	15.46	36021.7
	DK2_OP10	97374	22.16	36017.6	94247	19.19	36013.2

Table 8: Results from the MIP solver.

Since the processors used for testing the two methods are different and their computation times vary, it is difficult to directly compare their performances. Hence, the MIP solver is tested with a time limit of 2 hours, which is comparable to the computation times of the matheuristic and Table 10 compares the results of the matheuristic to that of the MIP solver. The matheuristic outperforms the MIP solver for 6 out of the 10 large instances and the improvement is found to be 7 to 15% on average.

5.5 Sensitivity analysis

The matheuristic involves 14 parameters and finding the optimal values of parameters for each instance is a very tedious and time consuming process. In this paper, we chose a common set of parameter values for each category that was based on the sizes of the instances. However, it is believed that the structure and the characteristics of the instances also have to be considered when tuning the parameters. For the largest category, Swedish (*SE1_OP5*) and Danish (*DK2_OP9*) instances were taken as the training instances and parameter η_{car} was adapted such that if $|\mathcal{H}^N| \geq 14000$ then it was set to 120 otherwise it was set to 180. Hence, the matheuristic is over-fitted for the aforementioned instances and is potentially prone to a large deviation in performance for an unseen test instance, which may possess different characteristics. To analyze the sensitivity of the matheuristic, we tested *DK2_OP9* instance with different threshold values of $|\mathcal{H}^N|$ and values of η_{car} as shown in Table 11.

Category	Instance	Gap (%)			Time (sec)	
		best	worst	avg.	avg.	avg. to find first feasible solution
Small	*SE1_OP1*	0.00	0.1	0.02	60.08	0.18
	SE1_OP2	0.00	3.63	0.36	134.13	0.26
	DK1_OP1	0.00	0.00	0.00	22.89	0.14
	DK2_OP1	0.00	4.26	0.43	53.76	0.19
	DK1_OP2	0.00	0.55	0.05	310.61	1.13
	DK2_OP2	0.11	6.69	2.33	341.29	4.00
Medium	*SE1_OP3*	0.61	3.73	2.33	883.29	7.75
	DK1_OP3	0.66	3.39	1.93	1715.3	3.85
	SE1_OP4	2.86	4.85	3.72	2576.29	112.19
	DK1_OP4	0.78	1.68	1.25	1708.71	10.94
	DK2_OP3	3.87	9.04	5.93	2092.41	16.35
Large	*SE1_OP5*	2.72	4.51	3.91	2383.63	137.35
	DK1_OP5	3.91	5.53	4.48	1870.26	36.88
	DK2_OP4	1.08	5.23	3.47	2334.22	63.38
	DK2_OP5	-1.92	2.49	0.25	2704.05	120.55
	DK2_OP6	-0.09	2.11	1.28	3414.22	812.76
	DK1_OP6	3.53	5.86	4.32	2440.03	151.53
	DK2_OP7	-0.76	0.57	-0.28	3389.88	176.25
	DK2_OP8	-4.26	-3.2	-3.77	4197.76	658.68
	DK2_OP9	-3.48	-1.63	-2.69	4700.99	433.54
	DK2_OP10	-8.17	-5.68	-6.95	4726.6	475.82

Table 9: Results from the matheuristic.

The best known solution provided by the MIP solver is used to calculate the average gap (%) of solutions yielded by different settings. The results show that the performance of the matheuristic deteriorates with increase in threshold value of $|\mathcal{H}^N|$ and value of η_{car}, which indicate that the matheuristic is sensitive to parameter values.

6 Conclusion

In this paper, we have introduced the DSPSC and presented a matheuristic to solve the problem. Computational study with real-life instances from Denmark and Sweden revealed that small and medium sized instances were solved with ease by the MIP solver. However, for larger instances with more than 6 cars and 1200 bus trips, the integrality gap on average was around 14.89%. The matheuristic provided better solutions, with comparable computation times, for 6 out of the 10 large instances. On larger instances, the improvement is approximately 13-15% on average.

Therefore, in most cases, the proposed method is superior than an approach based on solving the problem as a MIP problem for large instances in terms of solution quality and computation time. However, integrating the matheuristic as part of a decision support tool could be a challenging task. For solving the DSP, other practical conditions may exist such as maximum number of duties, maximum/minimum average working time of the duties and occasionally the objective is to minimize the total number of duties rather than minimizing the cost. Hence, in addition to solution quality and computation time, the transport industry demands a flexible decision support tool that allows for analyzing various scenarios, which will be beneficial during the planning process. Consequently, the devised matheuristic should have the ability to adapt to the diverse requirements from the users of the decision support tool. Since the users of the tool generally have limited knowledge of OR, user-friendliness is considered to be another key factor for successful integration of heuristics into decision support tools. The work carried out in this paper aimed at testing the matheuristic for

Category	Instance	MIP (multi thread)			Matheuristic	
		solution	gap(%)	time (*sec*)	avg. gap(%)	avg. time(sec)
	SE1_OP3	10925	0.00	130.31	2.33	883.29
	DK1_OP3	10927	0.00	48.21	1.93	1715.3
Medium	*SE1_OP4*	17836	1.27	7200.4	3.31	2576.29
	DK1_OP4	20253	0.00	1002.92	1.25	1708.71
	DK2_OP3	12554	1.99	7200.34	5.19	2092.41
	SE1_OP5	23631	1.18	7201.98	3.36	2383.63
	DK1_OP5	27773	0.00	3687.24	4.48	1870.26
	DK2_OP4	25875	19.85	7201.69	1.36	2334.22
	DK2_OP5	37651	17.95	7203.34	-6.66	2704.05
Large	*DK2_OP6*	43492	12.57	7203.33	-8.99	3414.22
	DK1_OP6	42394	1.07	7204.23	4.21	2440.03
	DK2_OP7	65658	22.27	7205.68	-12.66	3389.88
	DK2_OP8	83117	23.81	7206.42	-14.07	4197.76
	DK2_OP9	102831	26.88	7215.21	-14.74	4700.99
	DK2_OP10	102844	26.92	7213.6	-14.72	4726.6

Table 10: Comparison of results from the MIP solver and results from the matheuristic.

		$\lvert\mathcal{H}^N\rvert \geq$					
		10000	**12000**	**14000**	**16000**	**18000**	**20000**
η_{car}	**60**	-2.34	-2.7	-1.83	-1.85	-1.51	-1.01
	90	-2.74	-2.77	-2.04	-1.45	-1.16	-1.02
	120	-2.73	-2.71	-2.52	-1.42	-1.4	0.43
	150	-2.36	-2.21	-1.36	-1.29	0.63	1.59
	180	-1.44	-1.61	-1.03	2.29	3.75	3.8

Table 11: Sensitivity analysis of the matheuristic for different threshold values of $|\mathcal{H}^N|$ and values of η_{car}.

a wide variety of problems from Danish and Swedish transport companies and creating a set of parameter values for each category. However, if a new set of problems with varying sizes is given, it may possess different characteristics. Parameter tuning is considered to be a time consuming and tedious process, and approaches such as F-RaceBirattari et al. [2010] have been addressed in the literature for automatic parameter configuration. Furthermore, problem-dependent knowledge may still be needed to make the heuristic effective and perform consistently, which requires highly skilled practitioners. In conclusion, the need to design flexible and user-friendly heuristics is considered as a primary challenge for real-life implementation and could, hence, be seen as future areas of research.

The DSPSC is a practical problem with many variations and we hope to inspire other researchers in the area of vehicle and driver scheduling. One interesting variation of the problem, which is of significant importance to the transport industry, is a car routing problem that could have cars visiting multiple nodes, rather than a single node, before returning to the depot.

Acknowledgement

This work was supported by the Innovation Fund Denmark [grant number 5189-00128B]. The authors would like to thank the three anonymous referees for their valuable suggestions.

References

References

M. Ball, L. Bodin, and R. Dial. A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Science*, 17(1):4–31, 1983. ISSN 15265447, 00411655. doi: 10.1287/trsc.17.1.4.

M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336, 2010.

C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11:4135–4151, 2011.

R. Borndörfer, A. Löbel, and S. Weider. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. *Lecture Notes in Economics and Mathematical Systems*, 600: 3–24, 2008. ISSN 21969957, 00758442.

M. A. Boschetti, V. Maniezzo, M. Roffilli, and A. Bolufe Roehler. Matheuristics: Optimization, simulation and control. *Lecture Notes in Computer Science*, 5818:171–177, 2009.

J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

K. Darby-Dowman, J. K. Jachnik, R. L. Lewis, and G. Mitra. Integrated decision support systems for urban transport scheduling: discussion of implementation and experience. *Computer-aided Transit Scheduling. Proceedings of the Fourth International Workshop on Computer-aided Scheduling of Public Transport*, pages 226–239, 1988.

R. De Leone, P. Festa, and E. Marchitto. A bus driver scheduling problem: a new mathematical model and a grasp approximate solution. *Journal of Heuristics*, 17:441–466, 2011a.

R. De Leone, P. Festa, and E. Marchitto. Solving a bus driver scheduling problem with randomized multistart heuristics. *International Transactions in Operational Research*, 18(6):707–727, 2011b. ISSN 14753995, 09696016. doi: 10.1111/j.1475-3995.2011.00827.x.

G. Desaulniers and M. D. Hickman. Public transit. In *Handbook in Operations Research and Management Science*, volume 14, pages 69–127. Elsevier, 2007.

M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13, 1989.

I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. *Lecture Notes in Computer Science*, 2611:211–223, 2003.

M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858, 1987.

R. Freling, D. Huisman, and A. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003. ISSN 10991425, 10946136. doi: 10.1023/A:1022287504028.

D. Huisman, R. Freling, and A. Wagelmans. Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502, 2005. ISSN 15265447, 00411655. doi: 10.1287/trsc.1040.0104.

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

L. Jourdan, M. Basseur, and E.-G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199:620–629, 2009.

G. Laporte, R. Musmanno, and F. Vocaturo. An adaptive large neighborhood search heuristic for the capacitated arc-routing problem with stochastic demands. *Transportation Science*, 44(1):125–135, 2010.

H. Li, Y. Wang, S. Li, and S. Li. A column generation based hyper-heuristic to the bus driver scheduling problem. *Discrete Dynamics in Nature and Society*, 2015:1–10, 2015. ISSN 1607887x, 10260226. doi: 10.1155/2015/638104.

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

H. R. Lourenço, J. P. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331–343, 2001.

R. M. Lusby, M. Schwierz, M. R. Troels, and J. Larsen. An adaptive large neighborhood search procedure applied to the dynamic patient admission scheduling problem. *Artificial Intelligence in Medicine*, 74:21–31, 2016.

J. Ma, A. A. Ceder, Y. Yang, T. Liu, and W. Guan. A case study of beijing bus crew scheduling: a variable neighborhood-based approach. *Journal of Advanced Transportation*, 50(4):434–445, 2016. ISSN 20423195, 01976729. doi: 10.1002/atr.1333.

G. Mauri and L. Lorena. A new hybrid heuristic for driver scheduling. *International Journal of Hybrid Intelligent Systems*, 4(1):39–47, 2007. ISSN 18758819, 14485869. doi: 10.3233/HIS-2007-4105.

L. F. Muller, S. Spoorendonk, and D. Pisinger. A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research*, 218:614–623, 2012.

D. Pisinger and S. Røpke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34:2403–2435, 2007.

D. Pisinger and S. Røpke. Large neighborhood search. In *M. Gendreau (Ed.), Handbook of Metaheuristics, 2nd ed.*, pages 399–420. Springer, 2010.

R. Portugal, H. R. Lourenço, and J. P. Paixão. Driver scheduling problem modelling. *Public Transport*, 1(2):103–120, 2009. ISSN 16137159, 1866749x. doi: 10.1007/s12469-008-0007-0.

S. Røpke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP '98 Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pages 417–431, 1998.

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

A. Wren, S. Fores, A. Kwan, R. Kwan, M. Parker, and L. Proll. A flexible system for scheduling drivers. *Journal of Scheduling*, 6:437–455, 2003.

T. Yunes, A. Moura, and C. de Souza. Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, 39(2):273–288, 2005. ISSN 15265447, 00411655. doi: 10.1287/trsc.1030.0078.