A metaheuristic for the time-dependent pollution-routing problem

Anna Franceschetti[a,*], Emrah Demir[b], Dorothée Honhon[c], Tom Van Woensel[b], Gilbert Laporte[a], Mark Stobbe[b]

[a] *Canada Research Chair in Distribution Management, HEC Montréal, Montréal, Canada H3T 2A7*

[b] *School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands 5600MB*

[c] *Naveen Jindal School of Management, University of Texas at Dallas, Richardson, Texas 75080-3021, USA*

Abstract

We propose a metaheuristic for the Time-Dependent Pollution-Routing Problem, which consists of routing a number of vehicles to serve a set of customers and determining their speed on each route segment with the objective of minimizing the cost of driver's wage and greenhouse gases emissions. The vehicles face traffic congestion which, at peak periods, significantly restricts vehicle speeds and leads to increased emissions. Our algorithm is based on an adaptive large neighborhood search heuristic and uses new removal and insertion operators which significantly improve the quality of the solution. A previously developed departure time and speed optimization procedure is used as a subroutine to optimize departure times and vehicle speeds. Results from extensive computational experiments demonstrate the effectiveness of our algorithm.

Keywords: Routing, Freight transportation, Green vehicle routing, Greenhouse gases emissions, Metaheuristic algorithm, Departure time and speed optimization.

1. Introduction

In the past, the planning of freight transportation activities mostly focused on cutting costs and increasing profitability by considering internal transportation costs only, that is, mainly fuel cost and drivers' wages (see, e.g., Forkenbrock 1999, 2001). Nowadays, freight companies also need to consider the impact of their vehicle fleet on the environment and particularly the amount of greenhouse gases (GHG) they emit, which is typically measured using the carbon dioxide equivalent (CO2e) measure. This is because many cities have enacted new environmental legislations which restrict heavy freight vehicle traffic in certain areas - and at certain times of the day (see, e.g., Demir et al. 2014, 2015). Congestion, which is a major problem in many cities, increases GHG emissions and therefore should be taken into account when planning vehicle routes.

In this paper, we consider a vehicle routing problem, called the *Time-Dependent Pollution-Routing Problem* (TDPRP), which considers GHG emissions as well as traffic congestion. The TDPRP is an extension of classical the *Vehicle Routing Problem* (VRP), which consists of determining the optimal set of routes for a fleet of vehicles in order to satisfy the demands of a set of customers. The traditional objective in the VRP is the minimization of the total distance traveled by the vehicles. Considering

* *Corresponding author*
Email address: anna.franceschetti@cirrelt.ca (Anna Franceschetti)

the negative externalities of freight transportation, Bektaş and Laporte (2011) introduced the *Pollution-Routing Problem* (PRP) which aims at minimizing a total cost function comprising drivers' wages and vehicle fuel costs (given that GHG emissions are proportional to fuel consumption). The TDPRP was first studied by Franceschetti et al. (2013); it extends the PRP by considering traffic congestion at peak traffic congestion periods, which constrains the vehicle travel speeds and increases GHG emissions. The authors provided a mixed integer linear programming formulation for the TDPRP and derived a complete characterization of the optimal solution for a single-arc version of the problem. Finally, they proposed the *Departure time and Speed Optimization Procedure* (DSOP) to optimize the travel speeds and the departure times of a single vehicle visiting a given sequence of customer locations in the presence of traffic congestion, which builds on the analytical results they obtain for the single-arc version of the problem.

In this paper we develop a metaheuristic algorithm to solve the TDPRP, which uses the DSOP from Franceschetti et al. (2013) as a subroutine. Other recent studies have developed metaheuristic algorithms for the PRP and its variants: Demir et al. (2012) proposed a metaheuristic that iterates between the solution of the Vehicle Routing Problem with Time Windows (VRPTW) and a speed optimization problem. The VRPTW is solved by an Adaptive Large Neighborhood Search (ALNS) and the speed optimization problem is solved by means of a procedure that runs in polynomial time. In a related study, Demir et al. (2013) investigated the trade-offs between fuel consumption and driving time. The authors showed that in order to achieve a considerable reduction in fuel consumption and GHG emissions, trucking companies do not have to compromise significantly in terms of driving time. Koç et al. (2014) introduced the *Fleet Size and mix PRP*, which considers a heterogenous vehicle fleet and developed a hybrid evolutionary algorithm. Dabia et al. (2016) obtained exact solutions based on a branch-and-price algorithm by formulating the master problem as a set partitioning problem, and the pricing problem as a speed and start time elementary shortest path problem with resource constraints. They solved the master problem by means of column generation, and the pricing problem by a tailored labeling algorithm. Kramer et al. (2015b) proposed a method that combines a local search-based metaheuristic embedding an integer programming algorithm to solve a set covering formulation and a recursive speed optimization algorithm for the PRP. In a related study, Kramer et al. (2015a) presented an exact and efficient algorithm to optimize the travel speeds and the departure times of a single vehicle visiting a sequence of customer locations in the absence of traffic congestion. We note that none of these approaches can be directly used to solve the TDPRP, because they do not consider traffic congestion and ignoring the drop in vehicle speed at peak hours may lead to infeasible solutions in a congested network in the presence of hard time windows at the customers locations.

The main contributions of our paper are as follows. First we consider an extension of the PRP to a time-dependent setting with traffic congestion, namely the TDPRP, for which, to our knowledge, no tailored solution method has ever been proposed. Second, our algorithm is based on an adaptive large neighborhood search (ALNS) heuristic for which we develop new removal and insertion operators. Our new removal operators are motivated by the impact of congestion on vehicle speed and as such, are specifically tailored to the TDPRP problem. All newly developed operators are shown to significantly improve the quality of the solution. Third, we show numerically that our algorithm performs well and even yields very good results even for the PRP, which is a special case of the TDPRP with no traffic congestion.

The remainder of this paper is organized as follows. In §2 we describe the main features of the TDPRP. In §3 we describe the proposed metaheuristic algorithm. In §4, we present our numerical study. Conclusions are stated in §5.

2. The time-dependent polution-routing problem

In this section we present the main features of the TDPRP (see Franceschetti et al. (2013) for a more in-depth description) and discuss the feasibility conditions of the problem.

2.1. Problem description

The TDPRP consists of routing vehicles to make deliveries from a depot to a set of customers. It is defined on a complete graph $G = (N, A)$, where N is the set of nodes, and A is the set of arcs between every pair of nodes. Let 0 denote the depot, and $N_0 = N \setminus \{0\}$ denote the set of customer nodes. The distance between two nodes ($i \neq j \in N$) is denoted by $d_{i,j}$. We consider a homogeneous fleet with an unlimited number of vehicles of capacity of Q, initially located at the depot. Let h_i denote the service time at customer node $i \in N_0$. We set the service time at the depot equal to 0, i.e. $h_0 = 0$. Also let $[l_i, u_i]$ denote the hard time window at customer node $i \in N$ during which service must start: if a vehicle arrives at node $i \in N$ before the lower time window limit l_i, the driver must wait until time l_i to start serving the customer; we refer to this as the (mandatory) *pre-service waiting time*. After the service has been completed, the vehicle is allowed to wait idly at the customer node before leaving for the next customer node; we refer to this time as the *post-service waiting time*. Without loss of generality, we assume that any voluntary waiting time at customer nodes takes place after the completion of service. As shown by Franceschetti et al. (2013), in some cases waiting idly at the customer is an effective strategy to avoid traveling in congestion and may lead to a reduction in fuel consumption and GHG emissions. Note that the time window limits l_0 and u_0 at the depot are defined for the vehicles return trips. Without loss of generality we set $l_0 = 0$ and we can interpret u_0 as the end of the planning horizon. Finally, let q_i denote the delivery quantity at customer node $i \in N_0$.

In line with Franceschetti et al. (2013) we consider two methods to calculate the drivers' wages, referred to as *drivers' wage policies*: the drivers are paid from the beginning of the planning horizon, or the drivers are paid from their departure time from the depot. The objective of the TDPRP is to determine: (*i*) the set of vehicle routes, each starting and ending at the depot, (*ii*) the vehicle speed on each arc, and (*iii*) the departure times from each node, so as to minimize GHG emissions and drivers' wages.

As in Jabali et al. (2012), traffic congestion in the TDPRP is modeled through a two-level speed function with an initial period of congestion, lasting a units of time, during which the vehicle is forced to travel at a congestion speed v_c, followed by a *free-flow* period, during which the vehicle is allowed to drive at any speed up to a maximum value of $v^{max} > v_c$. Let v_f denote the speed chosen by the vehicle during the free-flow period. The left panel of Figure 1 shows the vehicle speed as a function of time.

Let $T(d, w, v_f)$ denote the total travel time needed to traverse an arc of length d assuming a departure time from its origin node of w and a free-flow speed of v_f. The right panel of Figure 1 shows how T varies with w (for fixed d and v_f). We have

$$T(d, w, v_f) = \begin{cases} \frac{d}{v_c} & \text{if } w \leq \left(a - \frac{d}{v_c}\right)^+ \\ \frac{v_f - v_c}{v_f}(a - w) + \frac{d}{v_f} & \text{if } \left(a - \frac{d}{v_c}\right)^+ < w < a \\ \frac{d}{v_f} & \text{if } w \geq a. \end{cases} \tag{1}$$

The first row in (1) corresponds to the case where the vehicle drives the entire distance of the arc in

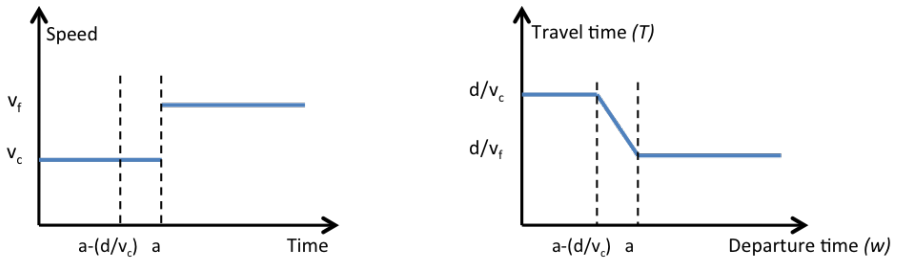

FIGURE 1: Time-dependent speed and travel time profiles

congestion. The second row corresponds to the case where she drives partially during the congestion period and partially during the free-flow period. Finally the third row corresponds to the case where she drives entirely during the free-flow period.

Alternatively we can write the travel time as $T(d, w, v_f) = T^c(d, w, v_f) + T^f(d, w, v_f)$, where $T^c(d, w, v_f) = \min\{(a - w)^+, d/v_c\}$ is the time spent by the vehicle traveling in congestion, and $T^f(d, w, v_f) = [d - (a - w)^+ v_c]^+ / v_f$ is the time spent by the vehicle traveling in free flow.

To calculate the amount of vehicle GHG emissions we use the *comprehensive modal emissions model* (CMEM) by Scora and Barth (2006) and Barth and Boriboonsomsin (2009). According to this model, the amount of GHG produced by a vehicle is directly proportional to the amount of fuel consumed. This amount is dependent on the type of vehicle, environment and traffic-related parameters, such as vehicle load, travel speed, acceleration, etc. (see, e.g., Demir et al. (2011)). Specifically, let F denote the amount of fuel consumed by a vehicle when traversing a distance d at a constant speed of v carrying a load of f, which is given by

$$F(d, v, f) = \tilde{A}(\mu + f)d + \tilde{B}\frac{d}{v} + \tilde{C}dv^2, \tag{2}$$

where μ is the vehicle curb weight and $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$ are non-negative constants (see Franceschetti et al. (2013) for how to calculate these values). In this expression, the first term is independent of the vehicle speed and it is called *weight module*, the second one is linear in the travel time (which, in this case, is equal to the distance divided by the (constant) vehicle speed) and it is called *engine module*, finally the last one is quadratic in the speed and it is called *speed module*. Figure 2 illustrates how the three modules vary with the speed. The values of the parameters used in Figure 2 are reported in Table 1.

As shown in Figure 2, the amount of fuel consumed (and therefore the quantity of GHG generated by a vehicle) increases significantly as the speed decreases below to a certain threshold (which for most applications is approximately equal to 15 km/h). This suggests that traffic congestion has a very strong impact both on the drivers' wage and GHG emissions costs.

Let $A = f_c\tilde{A}$, $B = f_c\tilde{B}$ and $C = f_c\tilde{C}$, where f_c is the fuel cost per liter. Also let D denote the drivers' wage per unit of time. Furthermore, let $TC(d, w, v_f, f)$ denote the cost of a vehicle traversing an arc of length d given a departure time from its origin node of w, a free-flow speed of v_f and a transported load of f; this cost is measured from time w until the completion of service at the arrival node. If the arrival

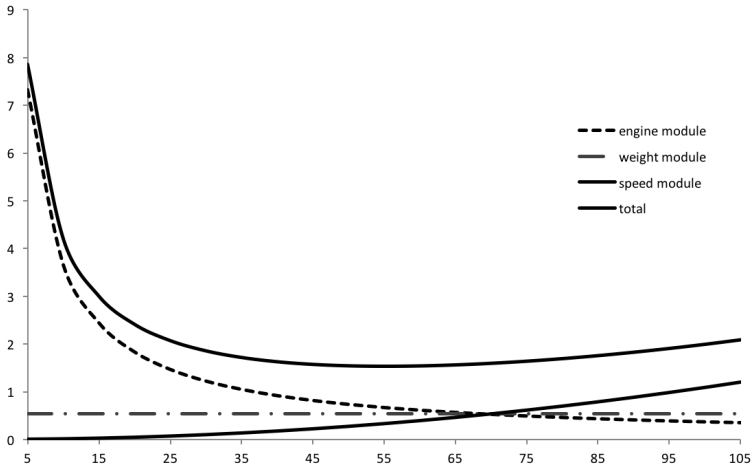

FIGURE 2: Fuel consumption rate F as a function of speed v

time at the node is before its upper time window, then

$$TC(d,w,v_f,f) = A(\mu+f)d + BT(d,w,v) + C[T^c(d,w,v_f)v_c^3 + T^f(d,w,v_f)v_f^3] + T^f(d,w,v_f)v_f^3] + D\left(\max\{l-w, T(d,w,v)\}+h\right), \quad (3)$$

where l and h are the lower time window limit and service time at the arrival node respectively. If the origin node of the arc is the depot, i.e., if this is the first arc of the vehicle route, and the driver is paid from her departure time from the depot, then a quantity of Dw must be subtracted from TC. For the linear mixed-integer programming formulation of the TDPRP we refer to Franceschetti et al. (2013).

An example of a feasible route is described below.

Example 1. *Figure 3 depicts a feasible route with five arcs and four customer nodes. We report under each arc, the corresponding length in km; we report under each node i the lower time window limit, i.e., l_i and the upper time window limit. Without loss of generality we assume that the service time and the demand at each node are zero. The departure time from each node is reported in brackets above the corresponding node. The travel speeds are reported in bold above each arc. We assume a congestion period of 9000 seconds and a congestion speed of* 10 *km/h. In this example the vehicle waits at the depot for*

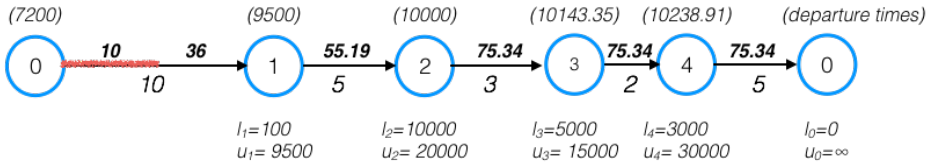

FIGURE 3: Representation of a feasible route. The congestion period is represented by the thick line.

7200 *seconds, then travels on the first half of arc* $(0,1)$ *in congestion at* 10 *km/h and on the second half in free-flow at* 36 *km/h, to arrive at node* 1 *exactly at the upper time window limit* $u_1 = 9500$. *After serving node* 1 *the vehicle travels on arc* $(1,2)$ *at* 55.19 *km/h reaching node* 2 *before the lower time window limit. The vehicle waits pre-service at node* 2 *and then leaves the node exactly at time* l_2. *The vehicle travels on the rest of the route at 75.34 km/h without pre- or post- service waiting at any intermediate node. Given the cost parameter reported in Table 1, the total cost of this solution is £ 30.55.*

2.2. Feasibility conditions

The TDPRP is feasible if it is possible to serve each customer with a separate vehicle, driving at the maximum speed outside of the congestion period without exceeding the vehicle capacity constraint nor violating their upper time window at the customer node as well as at the depot for the return trip, that is, if $q_i \leq Q$, $\min\{a, d_{0,i}/v_c\} + ((d_{0,i} - av_c)^+)/v^{max} \leq u_i$ and $\mu_0^i \leq u_0$ for all $i \in N_0$, where μ_0^i denotes the earliest possible arrival time back the depot from a trip to visit only customer node i, which is given by

$$\mu_0^i = \begin{cases} \max\left\{a + \frac{d_{0,i} - av_c}{v^{max}}, l_i\right\} + h_i + \frac{d_{i,0}}{v^{max}} & \text{if } a < \frac{d_{0,i}}{v_c}, \\ \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i + \frac{d_{i,0}}{v^{max}} & \text{if } \frac{d_{0,i}}{v_c} < a < \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i, \\ a + \frac{d_{i,0} - \left[a - \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} - h_i\right] v_c}{v^{max}} & \text{if } \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i < a < \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i + \frac{d_{i,0}}{v_c}, \\ \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i + \frac{d_{i,0}}{v_c} & \text{if } a > \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i + \frac{d_{i,0}}{v_c}. \end{cases}$$

The first row corresponds to the case where the outbound trip is driven in transient zone and the return trip in free flow. In the second row, the outbound trip is driven in congestion and the return trip is driven entirely in free flow. In the third row the outbound trip is driven in congestion and the return trip in the transient zone. Finally in the last row both trips are driven entirely in congestion. For use in our numerical experiments, we define a^{max} to be the maximum value for the length of the congestion period so that the problem is still feasible. We show how to calculate this value in Appendix A.

Next we discuss the feasibility conditions for a given vehicle route. Let $(0, 1, ..., 0)$ denote a fixed route where 0 is the depot and $i \in \{1, ..., n\}$ are customer nodes which are visited on this route. Let $\underline{w}_i$ denote the earliest possible service completion time at node i, which is obtained by assuming that the vehicle drives at the maximum speed v^{max} on every arc of the route after the end of the congestion period and never waits at any node following the completion of service. The values of $\underline{w}_i$ can be computed recursively as

$$\begin{aligned} \underline{w}_0 &= 0 \\ \underline{w}_i &= \begin{cases} \max\left\{\underline{w}_{i-1} + \frac{d_{i-1,i}}{v_c}, l_i\right\} + h_i & \text{if } \underline{w}_{i-1} \leq (a - d_{i-1,i}/v_c)^+, \\ \max\left\{a + \frac{d_{i-1,i} - (a - \underline{w}_{i-1})v_c}{v^{max}}, l_i\right\} + h_i & \text{if } (a - d_{i-1,i}/v_c)^+ \leq \underline{w}_{i-1} \leq a \\ \max\left\{\underline{w}_{i-1} + \frac{d_{i-1,i}}{v^{max}}, l_i\right\} + h_i & \text{if } \underline{w}_{i-1} \geq a. \end{cases} \quad \text{for } i = 1, ..., n \end{aligned} \tag{4}$$

The first term in (4) corresponds to the case in which the vehicle travels from node $i-1$ to node i during the congestion period. The second term corresponds to the case in which the vehicle departs from node $i-1$ before the end of the congestion period and arrives at node i past the congestion period, that is, during the free-flow period. Finally the last term corresponds to the case in which the vehicle travels from node $i-1$ to node i entirely during the free-flow period.

The route is feasible if (i) the sum of delivery quantities does not exceed the vehicle capacity, i.e., $\sum_{i=1}^{n} q_i \leq Q$, and (ii) the vehicle arrives at each customer node before its upper time window limit when driving at the maximum speed without any post-service waiting time or, alternatively, if the earliest possible service starting time at each customer node is smaller or equal than its upper time window limit, i.e. $\underline{w}_i - h_i \leq u_i$ for $i = 1, ..., n$.

3. An Adaptive Large Neighborhood Search Heuristic for the TDPRP

This section presents an Adaptive Large Neighborhood Search (ALNS) heuristic for the TDPRP. Pioneered by Pisinger and Ropke (2007) and Ropke and Pisinger (2006a), the ALNS heuristic is an extension of Large Neighborhood Search (LNS) introduced by Shaw (1998). Both methods are metaheuristics aimed at computing near-optimal solutions by repeatedly looking for a better solution in a large neighborhood around the *current* solution. Specifically, the neighborhood of the current solution is explored using a *removal* operator and an *insertion* operator in order to create an *incumbent* solution. The removal operator partially deconstructs the current solution and the insertion operator rebuilds it in a different way. Whether the incumbent solution is accepted as the new current solution is determined using a *simulated annealing* acceptance rule: the incumbent solution is always accepted if it has a better objective value than that of the current solution and is accepted with a certain probability otherwise. This probability is calculated using a *temperature* variable which decreases at the end of each iteration such that the probability of accepting the incumbent solution goes down over time. The ALNS heuristic extends the LNS heuristic by allowing the use of multiple removal and insertion operators. At each iteration, the ALNS heuristic selects one removal and one insertion operator using a *roulette wheel mechanism*, where the probability of choosing a certain operator is adjusted dynamically and depends on the past and current performance of all operators. The ALNS heuristic has been proved to be very efficient on a wide variety of transportation problems, (see, e.g., Ropke and Pisinger 2006a, Hemmelmayr et al. 2012, Aksen et al. 2014).

In this paper, we use the same general framework as in Pisinger and Ropke (2007), Ropke and Pisinger (2006a) and Demir et al. (2012). A formal description of our method is provided in Algorithm 1 below where we use S_i to denote the *initial* solution, S_b to denote the *best* solution encountered so far, S_c to denote the *current* solution and S_n to denote the *incumbent* solution.

Our implementation of the simulated annealing acceptance rule is the same as that of Ropke and Pisinger (2006a) and Demir et al. (2012): given a current solution S_c with a total cost $TC(S_c)$, the incumbent solution S_n is always accepted if it has a lower cost than that of the current solution, i.e., if $TC(S_n) \leq TC(S_c)$. Otherwise, the incumbent solution S_n is accepted with probability $e^{-(TC(S_n)-TC(S_c))/T}$, where T is the current *temperature*. The temperature starts at a positive value, equal to $\eta \cdot TC(S_i)$, then decreases over time as it gets multiplied by the *cooling rate* $\varsigma \in [0, 1]$ at each iteration. The ALNS heuristic stops when a number Δ of iterations has been reached. All user-controlled parameters are denoted by Greek letters. A detailed description of each part of Algorithm 1 is provided in the following subsections.

3.1. Construction of the initial solution

An initial feasible solution is generated using a modified version of the sequential insertion heuristic (SIH) introduced by Solomon (1987). The SIH starts creating a first route from a "seed" customer which is the one closest to the depot. In each subsequent step, the SIH either adds one of the currently unassigned nodes to one of the existing (partial) routes or creates a new route with only that node (leaving from the depot and returning to it immediately afterwards).

Let $(j_0, ..., j_{n+1})$ be a current partial route, where j_0 and j_{n+1} are two copies of the depot, i.e., $j_0 = j_{n+1} = 0$. The cost of inserting unassigned node i between adjacent nodes j_k and $j_k + 1$ for $k \in \{0, 1, ..., n\}$ is $C^i_{j_k,j_k+1}$, which is calculated as

Algorithm 1: The overall ALNS framework

Input: Set of removal operators Ω^-, set of insertion operators Ω^+, cooling rate ς and constants $\underline{\alpha}, \overline{\alpha}, \eta, \underline{\gamma}, \overline{\gamma}, \beta, \delta, k, \lambda, \sigma_1, \sigma_2, \sigma_3$ and Δ.
Output: A feasible solution S_b

$S_i \leftarrow$ Generate an initial solution using $\underline{\alpha}$ and $\overline{\alpha}$
For each removal operator $i \in \Omega^-$, initialize probability $\phi_i^- \leftarrow \frac{1}{|\Omega^-|}$
For each insertion operator $j \in \Omega^+$, initialize probability $\phi_j^+ \leftarrow \frac{1}{|\Omega^+|}$
$T \leftarrow \eta \cdot TC(S_i)$
$S_b \leftarrow S_i$
$S_c \leftarrow S_i$
repeat
  Select a removal operator $i \in \Omega^-$ with probability ϕ_i^-
  Select a insertion operator $j \in \Omega^+$ with probability ϕ_j^+ (DSOP is used as a subroutine for some of the operators)
  $S_n \leftarrow$ Obtain incumbent solution by applying operators i and j to S_c (possibly using $\underline{\gamma}, \overline{\gamma}, \beta, \delta$ and κ)
  if $TC(S_n) < TC(S_c)$ **then**
    $S_c \leftarrow S_n$
  else
    $r \leftarrow$ Generate a random number in $[0, 1]$
    if $r < e^{-(TC(S_n)-TC(S_c))/T}$ **then**
      $S_c \leftarrow S_n$
  if $TC(S_c) < TC(S_b)$ **then**
    $S_b \leftarrow S_c$
  $T \leftarrow \varsigma \cdot T$
  Update ϕ_i^-, ϕ_i^+ using constants $\lambda, \sigma_1, \sigma_2$ and σ_3.
  until *The maximum number of iterations* Δ *is reached*;

$$C^i_{j_k,j_k+1} = \begin{cases} d_{j_k,i} + d_{i,j_k+1} - \alpha d_{j_k,j_k+1} & \text{if route } (j_0, ..., j_k, i, j_{k+1}, ..., j_{n+1}) \text{ is feasible,} \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

In equation (5), α is a diversification parameter used to obtain different initial solutions in separate runs of the ALNS heuristic. Specifically, every time we compute the insertion cost, a new α value is randomly drawn from the interval $[\underline{\alpha}, \overline{\alpha}]$ according to a continuous uniform distribution. Note that the insertion cost is infinite if inserting customer i between nodes j_k and j_{k+1} would violate the route feasibility conditions of §2.2. At each iteration, the SIH considers each unassigned node i one by one and calculates the cost of inserting it in each possible position in the current set of partial routes, as well as the cost of creating a new route to serve this customer, i.e., $C^i_{0,0}$. The unassigned node with the least insertion cost is then selected and inserted in the position where it minimizes the insertion cost. After all customers have been inserted into a feasible position, the DSOP of Franceschetti et al. (2013) is run to optimize the travel speeds and departure times on each route. For a complete description of the DSOP algorithm we refer to Franceschetti et al. (2013).

3.2. Roulette wheel mechanism

The selection of the removal and insertion operators is regulated by a roulette wheel mechanism in which a weight is assigned to each operator and the probability of selection is proportional to these weights. Let Ω^- and Ω^+ denote the set of removal and insertion operators, respectively, and let ρ_j^- and ρ_j^+ denote the weights of the j^{th} removal and insertion operator, respectively. At each iteration of the ALNS algorithm, the probability ϕ_j^- of choosing operator j is calculated as $\phi_j^- = \rho_j^- / \sum_{i=1}^{|\Omega^-|} \rho_i^-$ and $\phi_j^+ = \rho_j^+ / \sum_{i=1}^{|\Omega^+|} \rho_i^+$ respectively for the removal and insertion operators. The basic idea is to adjust the weight of each operator based on its past performance. At the beginning all insertion and removal operators have weight set equal to one. As in Pisinger and Ropke (2007) let define a *segment* as a set of 100 iterations. At the end of each segment the weights of the removal and insertion operators are updated as follows. Consider removal operator $j \in \Omega^-$ and insertion operator $k \in \Omega^+$, their weights are recalculated as

$$\rho_j^- \leftarrow \rho_j^-(1-\lambda) + \lambda \frac{\Psi_j}{n_j^-} \quad \text{or} \quad \rho_k^+ \leftarrow \rho_k^+(1-\lambda) + \lambda \frac{\Psi_k}{n_k^+}, \tag{6}$$

where $\lambda \in [0,1]$ is the roulette wheel parameter, n_j^- or n_k^+ is the number of times removal operator j and insertion operator k have been used since the start of the segment, and Ψ_j and Ψ_k are the scores of operators j and k at the end of the segment. At the beginning of each segment the scores Ψ_j and Ψ_k are set to zero for all $j \in \Omega^-$ and $k \in \Omega^+$. At the end of each iteration, if a new solution is accepted, the scores of the removal and insertion operators which were used in this iteration are increased by either σ_1, σ_2 and σ_3. Specifically, σ_1 is used when a new best solution is found i.e., if $TC(S_n) \leq TC(S_b)$, σ_2 is used when the incumbent solution is better than the current solution, i.e., if $TC(S_n) \leq TC(S_c)$ and σ_3 is used when the incumbent solution is worse than the current solution, i.e., if $TC(S_n) > TC(S_c)$. In other words the weights of the operators is updated at the end of each segment as a weighted average of their past values and a score of performance over the last segment.

3.3. Removal and insertion operators

Our proposed ALNS heuristic uses eight removal operators and four insertion operators. In the removal phase, a removal operator is used to remove a number of customer nodes from the current solution and put them on a removal list $\mathcal{L}$. Subsequently, in the insertion phase, an insertion operator is used to reinsert all nodes from $\mathcal{L}$ into the partially destroyed solution, so as to obtain the incumbent solution. The DSOP is then run to optimize the travel speeds and the departure times of the vehicles on each route.

3.3.1. Removal operators

We first provide a description of our eight removal operators. The first five, namely the RR, WNR, PSR, CR and NGR, are adapted from existing work (see Shaw (1998), Ropke and Pisinger (2006a), Demir et al. (2012), Ribeiro and Laporte (2012), Demir et al. (2013)). The last three, namely the LSG, WSR and LWT, are new operators which we specifically designed for the TDPRP. Most of the removal operators operate by sorting the customer nodes according to a given metric. A fixed number of nodes are then removed from the current solution such that nodes with a smaller index in the sorted list are more likely to be chosen. In practice, this choice is implemented using a randomization process as follows. Consider a ranking of n customer nodes according to a given metric. For each node, a random number y is drawn from a continuous uniform distribution on $[0,1]$. The node chosen for removal is the $\lfloor y^\delta n \rfloor$-th one on the list, where δ is a positive fixed input parameter. Note that high (low) δ values lead to a higher (lower)

probability of choosing nodes with smaller indices. This randomization process is performed in order to diversify the search, as in Ribeiro and Laporte (2012).

We now provide a detailed description of the removal operators used in our ALNS algorithm. In what follows, γ, which is the number of nodes removed from the current solution, is a value randomly generated in each iteration from a discrete uniform distribution on $[\underline{\gamma}, \overline{\gamma}]$.

Random removal operator (RR)

The RR operator randomly selects γ customer nodes and removes them from the current solution S_c.

Worst node removal operator (WNR)

The WNR operator sorts the customer nodes according to their removal cost calculated as $TC(S_c) - TC(S_c \setminus i)$, where $TC(S_c \setminus i)$ is the cost of solution S_c after removing node i, which is calculated after running the DSOP in order to re-optimize the speeds and the departure times in the route which included i. Given the ranking of customer nodes based on their removal cost (from highest to lowest), γ nodes are chosen using the randomization process described above.

Proximity-based Shaw removal operator (PSR)

The PSR operator randomly selects a node in N_0 and adds it to the removal list $\mathcal{L}$. Then the customer nodes that are not in $\mathcal{L}$ are sorted based on their distance to this node (from the closest to the furthest), and a node is chosen from this ranking using the randomization process described above. Next, a node in $\mathcal{L}$ is selected at random and the customer nodes which are in $N_0 \setminus \mathcal{L}$ are then sorted based on their distance to this node (from the closest to the furthest). The next node added to $\mathcal{L}$ is then chosen from this ranking using the randomization process described above, etc. The process repeats until γ nodes have been removed from the current solution S_c.

Cluster removal operator (CR)

The CR operator starts by randomly choosing a route r from the current solution, then divides its nodes into two subsets using a modified version of the Kruskal's algorithm for the *Minimum Spanning Tree Problem* (Kruskal 1956) which stops as soon as two connected components are found. After the two clusters have been created, the CR operator randomly selects one of the two subsets, removes all its nodes from the current solution and adds them to the removal list $\mathcal{L}$. If the total number of removed nodes is less than γ, the CR operator selects a random node $j \in \mathcal{L}$ and looks for a node closest to j but belonging to a different route, say route r'. Route r' is then partitioned into two clusters and the process is repeated until at least γ nodes have been removed from the current solution.

Neighbor graph removal operator (NGR)

The NGR operator chooses γ nodes to remove using some historical information saved in a *neighbor graph*. To each arc (i, j) in the original graph is associated a weight in the neighbor graph which corresponds to the total cost of the best solution found so far, when node i is visited just before node j by the same vehicle. At the beginning of the ALNS heuristic all arcs have an infinite weight. These weights are then updated at the end of each iteration. Given a current solution S_c the NGR operator calculates the cost of customer node i by summing the arcs weights in the *neighbor graph* of the arcs going into and out of node i. The customer nodes are then sorted based on that cost metric (from the highest to the lowest) and γ nodes are chosen for removal using the randomization process described above.

Largest speed gap removal operator (LSG)

Given the current solution S_c, the LSG operator assigns to every customer node a value equal to the absolute value of the difference between the travel speed on the outgoing arc and the travel speed

on the incoming arc. If a vehicle starts traversing an arc during the congestion period and reaches the end of the arc during the free-flow period, the speed considered for comparison is the speed used in the last part of the arc, i.e. the free-flow speed. The customer nodes are then sorted based on this value (from highest to lowest) and γ nodes are chosen for removal using the randomization process described above. The reasoning behind this newly developed operator is that changing the position of the nodes that mark a decrease or an increase in the vehicle speed may help even out the travel speeds, and therefore reduce GHG emissions.

Worse speed removal operator (WSR)

Given the current solution S_c, the WSR operator assigns a positive value to every customer node as follows. If the vehicle arrives at the node before the end of the congestion period, this value is set to infinity; otherwise it is set equal to the traveling speed on the incoming arc. The customer nodes are then sorted based on these values (from highest to lowest), and γ nodes are chosen for removal using the randomization process described above. The idea behind this operator is to remove the nodes that are reached during the congestion period or by a vehicle traveling at a very high speed. As with the previous operator, removing these nodes may even out the travel speeds, thereby reducing GHG emissions.

Longest waiting time removal operator (LWT)

The LWT operator assigns a value to every customer node equal the total waiting time at that node, that is, the sum of pre- and post-service waiting times. The customer nodes are then sorted based on these values (from highest to lowest), and γ nodes are chosen for removal using the randomization process described above. The idea behind this operator is that, since the driver is paid during the waiting times, reducing them may lower the drivers' wage and therefore the total cost.

3.3.2. Insertion operators

Let S_p denote the partial solution obtained after removing a number of customer nodes from the current solution S_c using one of the removal operators listed above and let $\mathcal{L}$ be the removal list, where the nodes are listed in the order in which they were removed. We now describe the insertion operators that we use to construct the incumbent solution S_n. The first three operators, namely the BGI, MGI and k-RIH operators, are adapted from Shaw (1998), Ropke and Pisinger (2006b), Demir et al. (2012), Ribeiro and Laporte (2012), Demir et al. (2013), the last one, namely the β-HIH operator, is a new operator we propose.

Best greedy insertion operator (BGI)

The BGI operator selects the nodes in $\mathcal{L}$ one at time (in the order they are listed). For each node, it calculates its insertion cost between every pair of adjacent nodes in the partially destroyed solution S_p, as well as in a new route from a depot and returning to it immediately afterwards. This insertion cost is equal to the increase in total cost after inserting the node (calculated after running the DSOP) if the resulting solution satisfies the route feasibility conditions from §2.2, and is infinite otherwise. The node is then inserted in the position with the least insertion cost. The process is repeated until all nodes in $\mathcal{L}$ have been inserted back.

Modified greedy insertion operator (MGI)

The MGI operator works in a similar way as the BGI operator, except that the insertion cost of node $i \in \mathcal{L}$ between adjacent nodes j and k in S_p is calculated as $d_{ji} + d_{ik} - d_{jk}$ if the insertion satisfies the route feasibility conditions from §2.2, and is infinite otherwise. Compared to BGI operator, this operator is much faster since calculating the insertion cost does not require solving the DSOP. Here the DSOP is run only once, after all nodes in $\mathcal{L}$ have been inserted.

κ-regret insertion operator (κ-RIH)

For each node in $\mathcal{L}$, the κ-RIH operator first calculates the cost of inserting it between every pair of adjacent nodes in the partial destroyed solution S_p, as well as in a new route from a depot and returning to it immediately afterwards, in the same way as the BGI operator. Then, for each node $i \in \mathcal{L}$ the κ-RIH operator calculates the *Regret Value* $RV_i = C_{i,\kappa} - C_{i,1}$, where $C_{i,\kappa}$ is the cost of inserting node i into the position with the κ^{th} lower cost, where κ is a fixed integer parameter greater than one. The node with the maximum regret value is then removed from $\mathcal{L}$ and inserted where it generates the lowest insertion cost. The process is repeated until all nodes in $\mathcal{L}$ have been inserted in S_p.

Note that the DSOP is run whenever an insertion cost is calculated and whenever an unassigned node is inserted into the partial solution. The main advantage of this operator is that it improves on the myopic behavior of the operators previously introduced (see Potvin and Rousseau 1993, Ropke and Pisinger 2006b) because it considers inserting a node in a non-cost-minimizing position, thereby creating more diversification in the solution.

Hybrid insertion operator (β-HIH)

The β-HIH operator first modifies the removal list $\mathcal{L}$ by reversing the order of the nodes from the list with probability 1/2. If the length of the removal list $\mathcal{L}$ is less or equal to a fixed integer positive parameter β, the β-HIH operator tries to insert the whole list of nodes between every pair of adjacent nodes and also considers creating a new route with these nodes only. The insertion cost of the sequence of nodes is equal to the increase in total cost if the resulting route satisfies the route feasibility conditions from §2.2 and is infinite otherwise. If the minimum insertion cost is finite, the sequence of nodes is inserted in the position of lowest insertion cost. If not, or if the removal list $\mathcal{L}$ is longer than β, the β-HIH operator randomly selects nodes one at a time from $\mathcal{L}$, and inserts them each in the position where their insertion cost is minimized. This operator improves the myopic behavior of the BGI and MGI as it tries to insert multiple nodes all at once. Furthermore, we believe that this operator is particularly beneficial in the following cases: (*i*) when the nodes in $\mathcal{L}$ are close to each other (which tends to be the case when the (PSR) removal operator is used), or (*ii*) when the nodes in $\mathcal{L}$ have tight upper lime window limits (which tends to be the case when the (WSR) removal operator is used). Note that the DSOP is run whenever an insertion cost is calculated and after an unassigned node is inserted in the partial solution.

4. Computational Experiments

In this section we present the results of computational experiments we have conducted to assess the performance of our metaheuristic on the TDPRP. We used all instances from the PRP Library (PRPLIB), available at `http://www.apollo.management.soton.ac.uk/prplib.htm` as well as all instances proposed by Kramer et al. (2015b) which are adapted from the PRP Library. The PRPLIB contains 180 problem instances, partitioned into nine sets of 20 instances, such that the instances in each set have the same number of customer nodes, which varies between 10 and 200 nodes. Each instance comes with the following information: (*i*) the distances between each pair of nodes (these are based on actual geographical distances between randomly selected cities from the United Kingdom), (*ii*) the demand at each node, (*iii*) the service time window at each node, (*iv*) the service duration at each node and (*v*) the maximum vehicle traveling speed which is assumed to be the same on every arc. Kramer et al. (2015b) created two new sets of instances, namely set B and set C, by modifying the time window limits from the PRPLIB, while keeping all other parameters unchanged. Specifically, in set B the length of the service time window interval at each customer is randomly selected in the interval 2,000 and 5,000. Similarly, in set C the length of the service time window interval at each node is randomly selected between 2,000 and

15,000. As a result, the instances in set B have the tightest time windows, followed by the instances in set C, followed by the original PRPLIB instances. Once the lengths of the service time windows intervals are defined, the authors set the lower time window limit in such a way that it is feasible to reach each customer and return back to the depot before the upper time window limit at the depot, as in the orignal PRPLIB instances. The lower time window limit is randomly selected within this interval, the upper time window limit is calculated by summing the lower time window and the time window interval. Finally the upper time window at the depot for the return trip is set equal to the same value as in the PRPLIB original instances. Note that the time windows in sets B and C were chosen to guarantee feasibility of the problem instances in the absence of traffic congestion. In the setting of the TDPRP, the drop in vehicle speed during the initial period of traffic congestion can render the problem instance infeasible depending on its length therefore we calculated the maximum feasible value of the traffic congestion period length as explained in §2.2. In our numerical experiments, we use the same cost function parameters as in Demir et al. (2012) and Franceschetti et al. (2013), which are reported in Table 1. Unless stated otherwise, we assume that the driver is paid from the beginning of the planning horizon, i.e., drivers' wage policy (a).

TABLE 1: Cost function parameters

Notation	Description	Unit	Value
A	Weight module constant	£ m/(kJ,s^2)	$1.176E{-}8$
B	Engine module constant	£ s^2	0.00142
C	Speed module constant	£ kg/(kJ,m)	$1.98E{-}7$
D	Drivers' wage	$£/s$	0.0022
μ	Curb weight	kg	6,350

Our algorithm was coded in Java and run on a server with 64-bit GNU/Linux operating system, 96 GB of RAM and one processor Intel Xeon X5675 running at 3.07 GHz.

4.1. Parameter tuning

The implementation of the ALNS heuristic contains 14 user-controlled parameters which are provided in Table 2, along with the value we used in our numerical experiments. As in Demir et al. (2012), we divide the parameters into four groups. Group (i) includes the parameters that control the generation of the initial solution. Group (ii) includes the parameters that control the roulette wheel mechanism which governs the choice of the operators at each iteration. Group (iii) includes the parameters that control the simulated annealing search framework and the ones that calibrate the initial temperature and cooling rate. Finally, group (iv) includes all parameters used by the removal and insertion operators.

In order to set the value of these parameters, we conducted some parameter-tuning numerical experiments. All tests are performed on a tuning set consisting on the following 27 instances: nine instances from the PRPLIB, i.e., UK10_01, UK10_02, UK10_03, UK100_01, UK100_02, UK100_03, UK200_01, UK200_02, UK200_03, nine instances from Kramer et al. (2015b) Set B, i.e., UK10_01-B, UK10_02-B, UK10_03-B, UK100_01-B, UK100_02-B, UK100_03-B, UK200_01-B, UK200_02-B, UK200_03-B, and nine instances from Kramer et al. (2015b) Set C, i.e., UK10_01-C, UK10_02-C, UK10_03-C, UK100_01-C, UK100_02-C, UK100_03-C, UK200_01-C, UK200_02-C, UK200_03-C. We considered multiple sets of values and for each parameter setting we apply our ALNS to all instances from the tuning set ten times. In particular, we considered multiple sets of values for the $(\sigma_1, \sigma_2, \sigma_3)$, λ and η parameters as in Ropke and Pisinger (2006a) and Demir et al. (2012). The results from these tuning experiments are shown in the following subsections. The values of all other parameters are set as in Table 2. To ensure comparison fairness we used the same initial solution and seeds for the random numbers across runs with different parameter values. For the rest of our numerical experiments the congestion speed and the maximum free flow speed

TABLE 2: Parameters used in the ALNS heuristic

Group	Notation	Description	Value
(i)	$\underline{\alpha}$	Minimum α value	0.7
	$\overline{\alpha}$	Maximum α value	2
(ii)	λ	Roulette wheel parameter	0.2
	σ_1	Operators score when a new best solution found	30
	σ_2	Operators score when a new incumbent solution is accepted which is better than current solution	10
	σ_3	Operators score when a new incumbent solution is accepted which is worse than current solution	20
(iii)	Δ	Total number of iterations	$25,000$
	η	Initial temperature parameter	0.001
	ς	Cooling rate	0.99975
(iv)	$\underline{\gamma}$	Minimum number of nodes to remove	$\lfloor \log_{10}(N) \rfloor$
	$\overline{\gamma}$	Maximum number of nodes to remove	$\lfloor \log_{1.35}(N) \rfloor$
	δ	Randomization parameter	4
	κ	Insertion control parameter for κ-RIH operator	4
	β	Insertion control parameter for β-HIH operator	3

are set to $v_c = 10$ km/h and to $v^{max} = 90$ km/h, respectively. The congestion period is set at 75% of the maximum feasible congestion period length a^{max} as defined in §2.2.

4.1.1. Tuning of the roulette wheel mechanism parameters

In order to tune the control parameters used in the roulette wheel mechanism, i.e., σ_1, σ_2 and σ_3 defined in §3.2, we ran some numerical tests on all instances from the tuning set. Note that in all the problem instances we used, the value of $a^{\max}$ was finite. The values of parameters λ and η are set to 0.5 and 0.001, respectively. We considered the following four value combinations for $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3)$: $(30, 20, 10)$, $(30, 10, 20)$, $(10, 10, 10)$ and $(10, 30, 20)$. The comparison of the four different combinations is reported in Table 3. For each instance of the tuning set the columns *Best* report the best solution found out of ten runs using a given combination, the columns $\%Dev$ report the percentage deviation with respect to best solution value calculated across all combinations.

The results reported in Table 3 show that, in terms of percentage deviation, $(30, 10, 20)$ overall performs the best and $(10, 30, 20)$ performs the worst, even though the difference across all combinations is relatively small. Based on this analysis, we decided to use $\sigma_1 = 30, \sigma_2 = 10, \sigma_3 = 20$ for the rest of our numerical experiments. Given the definitions of the sigma values (see §3.2), the chosen combination set is such that the highest weight is assigned when a new best solution was found and interestingly, the second highest weight is given when the incumbent solution was worse than the current solution. This indicates that there is value in encouraging diversification amongst operators, in order to avoid a situation where early good performers end up dominating the roulette wheel mechanism at the detriment of other possibly promising operators.

4.1.2. Tuning of the roulette wheel control parameter

In order to tune the parameter λ, which controls the assignment of weights to the operators, we ran some tests on all instances from the tuning set using four different parameter values: 0, 0.25, 0.5 and 0.75. When λ is equal to zero the dynamic adjustment of the operators weights is removed and the score of each operator remains constant i.e., equal to one. A higher value of λ means that the updating of the operators' score is more sensitive to their performance in the most recent segment of iterations. Setting λ

TABLE 3: Tuning of the roulette wheel mechanism parameters ($\sigma_1, \sigma_2, \sigma_3$)

Instance	$(\sigma_1, \sigma_2, \sigma_3) = 30, 20, 10)$		$(\sigma_1, \sigma_2, \sigma_3) = (30, 10, 20)$		$(\sigma_1, \sigma_2, \sigma_3) = (10, 10, 10)$		$(\sigma_1, \sigma_2, \sigma_3) = (10, 30, 20)$		Min
	Best	*%Dev*	*Best*	*%Dev*	*Best*	*%Dev*	*Best*	*%Dev*	
UK10_1	323.19	0.00	323.19	0.00	323.19	0.00	323.19	0.00	323.19
UK10_1-B	245.96	0.00	245.96	0.00	245.96	0.00	245.96	0.00	245.96
UK10_1-C	223.70	0.00	223.70	0.00	223.70	0.00	223.70	0.00	223.70
UK10_2	326.49	0.00	326.49	0.00	326.49	0.00	326.49	0.00	326.49
UK10_2-B	303.09	0.00	303.09	0.00	303.09	0.00	303.09	0.00	303.09
UK10_2-C	277.15	0.00	277.15	0.00	277.15	0.00	277.15	0.00	277.15
UK10_3	318.82	0.00	318.82	0.00	318.82	0.00	318.82	0.00	318.82
UK10_3-B	301.12	0.00	301.12	0.00	301.12	0.00	301.12	0.00	301.12
UK10_3-C	242.18	0.00	242.18	0.00	242.18	0.00	242.18	0.00	242.18
UK100_1	1,760.04	1.40	1,760.04	1.40	1,735.73	0.00	1,743.96	0.47	1,735.73
UK100_1-B	1,625.09	0.00	1,625.96	0.05	1,645.07	1.23	1,645.34	1.25	1,625.09
UK100_1-C	1,517.59	0.00	1,522.66	0.33	1,522.55	0.33	1,531.49	0.92	1,517.59
UK100_2	1,630.97	0.55	1,622.12	0.00	1,626.24	0.25	1,627.98	0.36	1,622.12
UK100_2-B	1,636.56	0.76	1,624.24	0.00	1,627.38	0.19	1,629.30	0.31	1,624.24
UK100_2-C	1,476.71	0.39	1,476.25	0.36	1,471.02	0.00	1,483.18	0.83	1,471.02
UK100_3	1,553.82	0.34	1,549.33	0.05	1,551.54	0.19	1,548.54	0.00	1,548.54
UK100_3-B	1,548.69	0.46	1,541.59	0.00	1,548.47	0.45	1,554.63	0.85	1,541.59
UK100_3-C	1,349.59	0.00	1,351.60	0.15	1,352.47	0.21	1,353.67	0.30	1,349.59
UK200_1	2,934.45	0.00	2,940.28	0.20	2,937.98	0.12	2,945.07	0.36	2,934.45
UK200_1-B	2,872.91	0.46	2,868.41	0.30	2,859.78	0.00	2,871.29	0.40	2,859.78
UK200_1-C	2,656.57	0.16	2,663.14	0.41	2,659.17	0.26	2,652.39	0.00	2,652.39
UK200_2	2,816.70	0.00	2,822.32	0.20	2,831.87	0.54	2,820.49	0.13	2,816.70
UK200_2-B	2,699.62	0.93	2,685.81	0.41	2,692.83	0.68	2,674.73	0.00	2,674.73
UK200_2-C	2,520.46	0.31	2,529.00	0.64	2,525.04	0.49	2,512.79	0.00	2,512.79
UK200_3	2,974.57	0.34	2,985.75	0.72	2,965.63	0.04	2,964.37	0.00	2,964.37
UK200_3-B	2,849.79	0.28	2,841.80	0.00	2,877.88	1.27	2,874.08	1.14	2,841.80
UK200_3-C	2,614.45	0.32	2,612.35	0.24	2,606.20	0.00	2,617.56	0.44	2,606.20
Average		0.25		0.20		0.23		0.29	

equal to zero means that the weights of the operators remain constant, so that each one is always equally likely to be chosen in each iteration. The comparison is reported in Table 4. The values of *Best* and *%Dev* are as described for Table 3.

The results in Table 4 show that dynamically changing the weights of the operators so that their probability of being chosen depends on past performance does not always lead to a better solution, as evidenced by the many problem instances for which setting λ equal to zero led to the minimum total cost. We found, however, that overall the best performing value of λ was 0.5 so based on these results we decided to use this value in our computational experiments.

4.1.3. Tuning of the initial temperature parameter

In order to tune the parameter η, which controls the initial temperature, we ran some tests on all instances from the tuning set using three different parameter values: 0.001, 0.01, and 0.1. A higher value of η means that the initial temperature is higher, leading to a higher probability of accepting the incumbent solution in each iteration, given a fixed cost difference value with the current solution. The comparison is reported in Table 5. The values of *Best* and *Dev* are as described for Table 3.

Our results suggest that it is preferable to use a small values of η, since 0.001 generally leads to lower total cost values. Based on this analysis, we decided to set the value of η equal to 0.001 in our computational experiments.

TABLE 4: Tuning of the roulette wheel control parameter (λ)

Instance	$\lambda = 0$		$\lambda = 0.25$		$\lambda = 0.5$		$\lambda = 0.75$		Min
	Best	*%Dev*	*Best*	*%Dev*	*Best*	*%Dev*	*Best*	*%Dev*	
UK10_1	323.19	0.00	323.19	0.00	323.19	0.00	323.19	0.00	323.19
UK10_1-B	245.96	0.00	245.96	0.00	245.96	0.00	245.96	0.00	245.96
UK10_1-C	223.70	0.00	223.70	0.00	223.70	0.00	223.70	0.00	223.70
UK10_2	326.49	0.00	326.49	0.00	326.49	0.00	326.49	0.00	326.49
UK10_2-B	303.09	0.00	303.09	0.00	303.09	0.00	303.09	0.00	303.09
UK10_2-C	277.15	0.00	277.15	0.00	277.15	0.00	277.15	0.00	277.15
UK10_3	318.82	0.00	318.82	0.00	318.82	0.00	318.82	0.00	318.82
UK10_3-B	301.12	0.00	301.12	0.00	301.12	0.00	301.12	0.00	301.12
UK10_3-C	242.18	0.00	242.18	0.00	242.18	0.00	242.18	0.00	242.18
UK100_1	1,752.73	0.63	1,741.82	0.00	1,760.04	1.05	1,749.56	0.44	1,741.82
UK100_1-B	1,649.30	1.56	1,624.03	0.00	1,625.96	0.12	1,641.58	1.08	1,624.03
UK100_1-C	1,532.31	0.63	1,526.56	0.26	1,522.66	0.00	1,526.91	0.28	1,522.66
UK100_2	1,633.19	0.68	1,629.91	0.48	1,622.12	0.00	1,629.22	0.44	1,622.12
UK100_2-B	1,623.40	0.00	1,630.78	0.45	1,624.24	0.05	1,630.33	0.43	1,623.40
UK100_2-C	1,475.42	0.00	1,479.28	0.26	1,476.25	0.06	1,476.82	0.09	1,475.42
UK100_3	1,542.84	0.00	1,547.13	0.28	1,549.33	0.42	1,547.39	0.29	1,542.84
UK100_3-B	1557.64	1.04	1,558.92	1.12	1,541.59	0.00	1,559.29	1.15	1,541.59
UK100_3-C	1348.78	0.27	1,345.10	0.00	1,351.60	0.48	1,351.97	0.51	1,345.10
UK200_1	2,933.64	0.01	2,945.82	0.42	2,940.28	0.23	2,933.46	0.00	2,933.46
UK200_1-B	2,868.76	0.03	2,869.96	0.07	2,868.41	0.01	2,868.00	0.00	2,868.00
UK200_1-C	2,658.43	0.21	2,666.13	0.50	2,663.14	0.38	2,652.99	0.00	2,652.99
UK200_2	2,825.35	0.70	2,835.73	1.07	2,822.32	0.59	2,805.77	0.00	2,805.77
UK200_2-B	2,678.78	0.00	2,689.21	0.39	2,685.81	0.26	2,682.31	0.13	2,678.78
UK200_2-C	2,533.40	0.17	2,540.91	0.47	2,529.00	0.00	2,532.08	0.12	2,529.00
UK200_3	2,979.76	0.00	2,979.73	0.00	2,985.75	0.20	2,979.76	0.00	2,979.73
UK200_3-B	2,850.16	0.29	2,886.67	1.58	2,841.80	0.00	2,870.18	1.00	2,841.80
UK200_3-C	2,592.89	0.00	2,619.10	1.01	2,612.35	0.75	2,613.01	0.78	2,592.89
Average		0.23		0.31		0.17		0.25	

4.2. Computational time analysis

In this section we analyze the speed performance of our ALNS heuristic using instances from the tuning set. Table 6 shows the average computational (CPU) time across ten runs of the ALNS heuristic. The column entitled *CPU time* reports the average CPU time for running the algorithm (in minutes). The column entitled *RO time* reports the average CPU time (in minutes) collectively spent by all the removal operators during one run of the algorithm. The column *IO time* reports the average CPU time (in minutes) collectively spent by all the insertion operators during one run of the algorithm, finally the column *DSOP time* reports the average CPU time (in minutes) spent solving the DSOP, which is used as a subroutine. All these values are displayed in Table 6. Note that these are average values calculated across all instances from the tuning set with the same number of nodes. Table 6 shows that our ALNS algorithm generates a solution in a reasonable time: less than 10 minutes on average for problems with 100 nodes and about 36 minutes for problems with 200 nodes. Moreover we see that, on average, the insertion operators require more time than the removal operators. This is because three out of four insertion operators (namely BGI, κ-RIH and β-HIH) run the DSOP as a subroutine versus only one out of the eight removal operators (namely WNR). Running the DSOP is a time-consuming process as evidenced by the fact that it requires more than 50% of the total CPU time. We further analyze the time required by each operator in the next section.

4.3. Relative performance of the operators

In this section, we study the relative performance of the removal and insertion operators both in terms of speed and the solution quality. To this end, we solve all instances from the tuning set. For each one, we

TABLE 5: Tuning of the initial temperature parameter (η) for the simulated annealing acceptance rule

Instance	$\eta = 0.001$		$\eta = 0.01$		$\eta = 0.1$		Min
	Best	*%Dev*	*Best*	*%Dev*	*Best*	*%Dev*	
UK10_1	323.19	0.00	323.19	0.00	323.19	0.00	323.19
UK10_1-B	245.96	0.00	245.96	0.00	245.96	0.00	245.96
UK10_1-C	223.70	0.00	223.70	0.00	223.70	0.00	223.70
UK10_2	326.49	0.00	326.49	0.00	326.49	0.00	326.49
UK10_2-B	303.09	0.00	303.09	0.00	303.09	0.00	303.09
UK10_2-C	277.15	0.00	277.15	0.00	277.15	0.00	277.15
UK10_3	318.82	0.00	318.82	0.00	318.82	0.00	318.82
UK10_3-B	301.12	0.00	301.12	0.00	301.12	0.00	301.12
UK10_3-C	242.18	0.00	242.18	0.00	242.18	0.00	242.18
UK100_1	1,760.04	1.13	1,740.32	0.00	1,746.13	0.33	1,740.32
UK100_1-B	1,625.96	0.00	1,642.22	1.00	1,633.27	0.45	1,625.96
UK100_1-C	1,522.66	0.65	1,515.76	0.20	1,512.77	0.00	1,512.77
UK100_2	1,622.12	0.00	1,624.09	0.12	1,637.82	0.97	1,622.12
UK100_2-B	1,624.24	0.00	1,628.48	0.26	1,649.84	1.58	1,624.24
UK100_2-C	1,476.25	1.19	1,474.25	1.05	1,458.88	0.00	1,458.88
UK100_3	1,549.33	0.24	1,545.65	0.00	1,563.84	1.18	1,545.65
UK100_3-B	1,541.59	0.00	1,548.50	0.45	1,557.97	1.06	1,541.59
UK100_3-C	1,351.60	0.00	1,353.13	0.11	1,370.81	1.42	1,351.60
UK200_1	2,940.28	0.00	2,973.39	1.13	2,977.64	1.27	2,940.28
UK200_1-B	2,868.41	0.00	2,902.70	1.20	2,928.85	2.11	2,868.41
UK200_1-C	2,663.14	0.00	2,708.00	1.68	2,720.90	2.17	2,663.14
UK200_2	2,822.32	0.00	2,833.12	0.38	2,852.90	1.08	2,822.32
UK200_2-B	2,685.81	0.00	2,712.10	0.98	2,741.27	2.07	2,685.81
UK200_2-C	2,529.00	0.00	2,552.37	0.92	2,582.37	2.11	2,529.00
UK200_3	2,985.75	0.04	2,985.37	0.03	2,984.51	0.00	2,984.51
UK200_3-B	2,841.80	0.00	2,894.06	1.84	2,916.88	2.64	2,841.80
UK200_3-C	2,612.35	0.21	2,606.94	0.00	2,657.37	1.93	2,606.94
Average		0.13		0.42		0.83	

TABLE 6: Average computational time spent in each stage of the algorithm

# nodes	CPU time (min)	RO time (min)	IO time (min)	DSOP time (min)
10	0.11	0.01	0.10	0.09
100	9.09	0.87	8.21	7.43
200	36.07	3.41	32.66	30.61
Average	15.09	1.43	13.66	12.71

ran our algorithm ten times. The first column in Table 7 reports the name of the operators. The columns $\%Usage$ report the percentage of total iterations in which an operator is used. The columns $\%IBest$ report the percentage of iterations in which the incumbent solution was better than the best solution for which that particular operator was used. Finally the columns CPU report the normalized CPU time of each operator, where the normalization is done using the minimum value of the average CPU time across all operators of one type: for the removal (insertion) operators, the minimum average CPU time was obtained by the RR (MGI) operator. Hence, for example, a value of 1.55 for normalized average CPU time for the CR operator means that it was on average 1.55 times slower than the RR operator. All values reported in Table 7 are average values calculated across the instances with the same number of nodes.

In terms of quality of the solution, as measured by the proportion of improvements to the best solution, the best performing removal operators are the CR, NGR operators followed by our newly developed LSG

TABLE 7: Relative performance of the operators

	10 nodes			100 nodes			200 nodes			Average		
	% Usage	% IBest	Avg CPU	% Usage	% IBest	Avg CPU	% Usage	% IBest	Avg CPU	% Usage	% IBest	Avg CPU
Removal												
RR	12%	8%	1.00	13%	2%	1.00	12%	3%	1.00	12%	5%	1.00
WNR[2]	15%	14%	5.35	13%	14%	225.59	13%	13.3%	714.32	13%	14%	315.09
PSR	11%	9%	1.17	13%	14%	2.06	13%	14%	2.48	12%	12%	1.90
CR	14%	19%	1.50	14%	24%	1.85	13%	21%	1.77	14%	22%	1.71
NGR	12%	18%	1.35	11%	12%	3.29	11%	12%	6.16	11%	14%	3.60
LSG[1]	14%	16%	1.17	12%	10%	2.24	13%	12%	3.00	13%	13%	2.14
WSR[1]	11%	9%	1.20	13%	13%	2.37	13%	12%	2.77	12%	12%	2.11
LWT[1]	13%	6%	1.24	12%	11%	2.23	12%	12%	2.86	12%	10%	2.11
Insertion												
BGI[2]	25%	34%	18.89	25%	28%	70.16	25%	31%	95.33	25%	31%	61.46
MGI	26%	2%	1.00	26%	3%	1.00	25%	3%	1.00	26%	3%	1.00
KRI[2]	25%	30%	38.62	24%	33%	181.80	25%	34%	243.01	25%	32%	154.48
HIH[1,2]	24%	35%	23.07	25%	36%	73.69	25%	32%	98.94	25%	34%	65.24

[1] newly developed operator, [2] uses the DSOP as a subroutine. %Usage = # of iterations in which an operator is used divided by total # of iterations(Δ). % IBest = # of iterations in which the operator generated produced an incumbent solution better than the best solution divided by total # of iterations in which an incumbent solution is better than the best solution. Norm. Avg. CPU time = average CPU time required by the removal (insertion) operator divided by the minimum average CPU time across all removal (insertion) operators.

operator. Regarding the insertion operators, the one which leads to the most in the best solution is our β-HIH operator. In contrast relatively very few improvements are achieved with RR removal operator and the MGI insertion operator. In terms of speed, we see that, as expected, all operators which use the DSOP as a subroutine are much slower than those which do not. Overall, there seems to be a trade-off between speed and quality of the solution: the RR and MGI operators are the fastest but the worst performing ones.

Next we further study the performance of the operators, focusing on the new ones we developed. For this purpose we ran two different versions of our ALNS heuristic algorithm on all the 100-, and 200-node instances. In Version #1, we included all the removal and insertion operators described in §3.3. In Version #2, we included only the removal and insertion operators adapted from the literature, that is, we excluded the three removal operators (LSG, WSR and LWR) and the one insertion operator (β-HIH) we developed. Hence, comparing the performance of Versions #1 and #2 measures the value of the removal and insertion operators we proposed. To ensure fairness, we used the same initial solution and seeds for the random numbers across versions. All results were calculated assuming a congestion period equal to the 75% of the maximum feasible congestion period length a^{max} and a congestion speed of 10 km/h. For each instance we ran our algorithm ten times. The complete results for all the 100-, and 200-node instances are reported in Tables 12 and 13 in Appendix B. We also report the p-values obtained from comparing the minimum cost values across the two versions using a one-tailed Wilcoxon signed rank test (see Rey and Neuhäuser (2011)). A summary of the results is reported in Table 8. The columns *Average* and *Maximum* respectively report the average and the maximum percentage cost reduction over ten runs obtained from using the new removal and insertion operators. To calculate these values, we use the best solution value across the ten runs for each instance/version, that is, we calculated the percentage cost reduction as $Dev = 100[TC(S_b^{Version\#2}) - TC(S_b^{Version\#1})]/TC(S_b^{Version\#1})$.

Table 8 shows that, in most cases, using our newly developed operators leads to an improvement in the performance of our ALNS algorithm. This confirms that our newly developed operators improve the quality of the solution given by the ALNS heuristic. A possible explanation is that, while the operators adapted from the literature are mainly aimed at minimizing the total distance traveled by the vehicles, the new ones aim at reducing the negative effects of fluctuating speeds in transport networks, which negatively impact emissions. As discussed in Franceschetti et al. (2013), these negative effects become

TABLE 8: Value of the newly developed operators

# of nodes	% Cost Reduction (Dev)	
	Average	Maximum
UK100	-0.03	0.86
UK100-B	0.16	0.78
UK100-C	0.26	1.26
UK200	0.10	0.96
UK200-B	0.11	1.07
UK200-C	0.14	1.17

Average = average percentage cost reduction across all instances with the same number of nodes; *Maximum* = maximum percentage cost reduction across all instances with the same number of nodes where $Dev = [TC(S_b^{Version\#2}) - TC(S_b^{Version\#1})]/TC(S_b^{Version\#1})$.

more relevant in the presence of traffic congestion and therefore they cannot be ignored.

4.4. Performance on TDPRP instances

In this section we study the performance of our ALNS heuristic on TDPRP instances. First we use a set of small-size (i.e., 10-, 20-, and 25-node) instances from the PRPLIB to compare our algorithm to the solution found in Franceschetti et al. (2013) using the MIP of Franceschetti et al. (2013), which was ran using CPLEX. Note that the MIP formulation of Franceschetti et al. (2013) requires that the number of vehicles be fixed at a value K, while, in our model, we optimize the number of vehicles to use without imposing a maximum value constraint. To ensure the fairness in our comparisons, we modified the original MIP formulation from Franceschetti et al. (2013) by replacing constraint $\sum_{j \in N} x_{0j} = K$ with $\sum_{j \in N} x_{0j} \leq |N_0|$, where $|N_0|$ is the number of customer nodes in the network. We used this modified MIP version to compute the results displayed in Tables 9 and 10. All results were calculated assuming a congestion period of 1 hour and a congestion speed of 10 km/h. For each instance we ran our algorithm ten times. Table 9 presents the comparison for the case where the driver is paid from the beginning of the planning horizon and Table 10 presents the results for the case where the driver is paid from her departure time. The columns *ALNS* report total cost, i.e. TC, of the best solution found by our metaheuristic. The columns *CPLEX* report the total cost calculated using the MIP formulation. Finally, the columns entitled *Dev.* report the percentage difference between the two cost values, calculated as $[TC(S_b^{ALNS}) - TC(S^{CPLEX})]/TC(S_b^{ALNS})$.

The results in Table 9 and 10 show that most of the instances have very small deviations (in absolute value), which suggests that our ALNS heuristic performs very well. The maximum positive deviation is 0.43%, which is the worst performance of our solution method. In a few cases the deviation is negative, implying that the solution obtained with our ALNS heuristic is actually better than that obtained by Franceschetti et al. (2013). This is due to the fact that the MIP formulation of Franceschetti et al. (2013) uses a discrete set of values for the free-flow travel speeds whereas our ALNS heuristic allows for continuous values. The average CPU times required by our heuristic to solve a single instance of 10-, 15-, and 20-node are respectively 10.8 seconds, 18.69 seconds, and 25.15 seconds. In contrast, the average CPU times required by CPLEX to solve a single instance of 10-, 15- and 20-node using CPLEX are respectively 41.55 seconds, 503.44 seconds, and 8058.76 seconds. Hence, our algorithm is significantly faster than the CPLEX solving the MIP, expecially for large instances.

Next, we solve all instances from the PRPLIB and from Kramer et al. (2015b) in order to further assess the performance of our algorithm. For every instance, we ran our algorithm ten times. Note that, for the instances with more than 20-node, solving the MIP formulation from Franceschetti et al. (2013) using CPLEX is computationally too expensive; therefore we do not have values to compare our solution

TABLE 9: Total cost comparison between ALNS and CPLEX (driver paid from the beginning of the planning horizon)

Instance	UK_10			UK_15			UK_20		
	ALNS $TC(£)$	CPLEX $TC(£)$	Dev. (%)	ALNS $TC(£)$	CPLEX $TC(£)$	Dev. (%)	ALNS $TC(£)$	CPLEX $TC(£)$	Dev. (%)
1	323.19	323.19	0.00	455.08	455.12	-0.01	499.38	499.08	0.06
2	326.49	326.50	0.00	353.27	353.30	-0.01	524.15	524.19	-0.01
3	318.82	318.84	-0.01	384.40	384.47	-0.02	323.78	326.90*	-0.96
4	299.62	299.65	-0.01	423.66	423.67	0.00	497.05	497.11	-0.01
5	244.79	244.82	-0.01	489.98	490.02	-0.01	468.02	468.02	0.00
6	350.17	350.24	-0.02	377.51	377.61	-0.03	511.35	511.43	-0.02
7	277.567	277.78	-0.08	377.77	377.80	-0.01	351.12	351.80	-0.19
8	357.49	357.58	-0.03	256.24	255.13	0.43	463.88	463.81	0.01
9	240.87	240.89	-0.01	363.98	363.67	0.09	551.73	560.12*	-1.52
10	273.58	273.58	0.00	353.38	353.51	-0.04	448.76	448.81	-0.01
11	401.99	402.20	-0.05	445.59	445.64	-0.01	529.98	530.10	-0.02
12	296.45	296.45	0.00	475.79	475.79	0.00	481.69	481.68	0.00
13	312.42	312.43	0.00	388.67	388.67	0.00	498.43	498.55	-0.02
14	266.97	267.00	-0.01	510.50	510.54	-0.01	582.00	582.00	0.00
15	221.95	221.95	0.00	352.21	352.29	-0.02	477.83	477.87	-0.01
16	248.57	248.60	-0.01	329.10	329.16	-0.02	487.35	487.42	-0.01
17	252.25	252.25	0.00	409.44	409.45	0.00	528.48	533.38*	-0.93
18	240.22	240.27	-0.02	449.22	449.22	0.00	530.27	530.30	-0.01
19	292.66	292.71	-0.02	303.88	304.93	-0.35	524.89	524.92	0.00
20	259.48	259.56	-0.03	300.34	300.34	0.00	514.90	514.99	-0.02
Average			-0.02			0.00			-0.18

$Dev = [TC_{ALNS} - TC_{CPLEX}]/TC_{ALNS}$.

TABLE 10: Total cost comparison between ALNS and CPLEX (driver paid from departure time)

Instance	UK_10			UK_15			UK_20		
	ALNS $TC(£)$	CPLEX $TC(£)$	Dev. (%)	ALNS $TC(£)$	CPLEX $TC(£)$	Dev. (%)	ALNS $TC(£)$	CPLEX $TC(£)$	Dev. (%)
1	214.77	214.77	0.00	321.32	321.37	-0.01	399.27	403.96	-1.18
2	247.81	247.82	-0.01	248.97	249.03	-0.03	420.10	420.10	0.00
3	226.56	226.56	0.00	310.91	310.98	-0.02	233.60	233.69	-0.04
4	202.90	202.94	-0.02	339.13	339.89	-0.22	383.96	384.10	-0.04
5	193.44	193.44	0.00	367.37	367.45	-0.02	358.75	358.75	0.00
6	257.34	257.34	0.00	290.11	290.11	0.00	383.62	383.71	-0.03
7	235.31	235.43	-0.05	294.10	294.23	-0.04	273.56	278.42	-1.78
8	268.75	268.91	-0.06	202.24	202.29	-0.02	354.53	354.72	-0.05
9	181.77	181.79	-0.01	289.06	289.09	-0.01	407.59	407.64	-0.01
10	237.89	237.94	-0.02	261.60	261.69	-0.04	326.06	326.06	0.00
11	333.25	335.68	-0.73	311.81	311.96	-0.05	445.26	442.93	0.52
12	202.75	201.66	0.54	368.49	368.73	-0.07	371.91	372.24	-0.09
13	216.29	216.33	-0.02	298.58	298.58	0.00	381.71	381.91	-0.05
14	217.83	217.87	-0.02	409.23	409.33	-0.02	431.74	431.74	0.00
15	161.55	161.65	-0.06	263.12	263.12	0.00	376.80	376.82	-0.01
16	183.86	183.91	-0.03	232.46	232.57	-0.05	378.19	378.29	-0.03
17	200.31	199.12	0.59	328.28	328.34	-0.02	453.57	449.61	0.87
18	195.62	195.62	0.00	332.55	332.55	0.00	424.71	418.90	1.37
19	228.14	228.15	0.00	206.94	207.00	-0.03	401.93	405.26	-0.83
20	180.38	180.38	0.00	212.24	212.24	0.00	409.60	409.73	-0.03
Average			0.01			-0.03			-0.07

to. In other words, our ALNS approach is the first solution method capable of yielding a solution for instances with more than 20 customer nodes.

Tables 14-22 in Appendix C display the computational results for the 10-, 15-, 20-, 25-, 50-, 75-, 100-, 150- and 200-node instances, under the assumption that the driver is paid from the beginning of the planning horizon. The columns *PRPLIB* report the results calculated using the instances from the PRPLIB library, the columns *KSVC14 Set B* report the results calculated using the Set B instances from Kramer et al. (2015b), similarly the columns *KSVC14 Set C* report the results calculated using the Set C instances from Kramer et al. (2015b). The first column lists the instance number, the columns a report the length of the congestion period, the columns *CPU* report the average CPU time (in seconds) out of ten runs, the columns *Avg.* report the average solution cost out of ten runs, the columns *Best* report the total cost of the best solution found out of ten runs, finally the column *Gap* report the average gap, calculated as $Gap = 100(Avg. - Best)/Best$.

We see from Tables 14-22 that our ALNS algorithm is very robust as in most cases, the percentage gap between best and average solution is of the order of 1 to 2%. The comparison of the total cost across the three sets of problem instances (the original PRPLIB and sets B and C) speaks to the impact of the time windows on the total costs. Instances from the original PRPLIB, which are those with the loosest time window intervals, have the lowest total cost, while instances of set B, which are those with the tightest time windows interval, have the highest total cost. Also we see that instances with tighter time windows take longer to solve, since it generally takes longer to find a feasible incumbent at each iteration.

4.5. Performance on PRP instances

In this section we compare the performance of our ALNS heuristic to other solution methods for the PRP which, as discussed earlier, is a special case of the TDPRP where the congestion period is zero. We compare our results on all the 100-node instances from the PRPLIB to the results from Demir et al. (2012), Koç et al. (2014) and Kramer et al. (2015b) in Table 11. The first column lists the instance name, the second one reports the best solution calculated with our ALNS heuristic (out of ten runs). The columns *DBL12*, *KBJL14* and *KSVC14* provide the results reported in Demir et al. (2012), Koç et al. (2014) and Kramer et al. (2015b), respectively. Finally, the columns entitled *Dev.* report the percentage cost deviation between our algorithm and the solution methods from the literature, namely $[TC(S_b^{ALNS}) - TC(S_b^{DBL12})]/TC(S_b^{ALNS})$, $[TC(S_b^{ALNS}) - TC(S_b^{KBJL14})]/TC(S_b^{ALNS})$ and $[TC(S_b^{ALNS}) - TC(S_b^{KSVC14})]/TC(S_b^{ALNS})$.

Table 11 shows that our ALNS heuristic is able to compete with the best heuristics for the PRP, even though it was designed for a more general version of the problem, namely the TDPRP. In fact, it is very remarkable that our algorithm provides the best solution of all methods for four of the instances (namely UK100_02, UK100_13, UK100_16, and UK100_20). A full characterization of the best solution we obtained for each problem instance is available upon request. In particular Table 11 indicate that our algorithm significantly outperforms that of Demir et al. (2012). The reason may be that there are some significant differences between the ALNS of Demir et al. (2012) and our algorithm, in terms of the number of operators used, the inclusion of speed factor in removal phase, scoring mechanism and the number of removable nodes. Further, the ALNS of Demir et al. (2012) uses SOP algorithm at the end of 25,000 iterations to improve the solution quality whereas we use DSOP algorithm at each insertion phase. We believe that the more frequent application of the DSOP drives the performance improvement (however it also leads to an increase in the average CPU time).

TABLE 11: Computational results on 100-node PRP instances

Instance	ALNS	DBL12		KBJL14		KSVC14	
	$TC(£)$	$TC(£)$	Dev. (%)	TC (£)	Dev. (%)	TC (£)	Dev. (%)
UK100_01	1,216.18	1240.79	-2.02	1,212.72	0.28	**1,209.11**	0.58
UK100_02	**1,146.55**	1168.17	-1.89	1,149.16	-0.23	1,146.79	-0.02
UK100_03	1,089.74	1092.73	-0.27	1,080.87	0.81	**1,078.75**	1.01
UK100_04	1,086.95	1106.48	-1.80	1,085.66	0.12	**1,075.29**	1.07
UK100_05	1,041.57	1043.41	-0.18	1,033.19	0.80	**1,028.86**	1.22
UK100_06	1,194.73	1213.61	-1.58	1192.67	0.17	1,193.38	0.11
UK100_07	1,051.99	1060.08	-0.77	**1,044.58**	0.70	1,045.02	0.66
UK100_08	1,090.29	1106.78	-1.51	1,092.67	-0.22	**1,089.84**	0.04
UK100_09	991.38	1015.46	-2.43	992.36	-0.10	**988.41**	0.30
UK100_10	1,067.70	1076.56	-0.83	1,063.05	0.44	**1,059.95**	0.73
UK100_11	1,204.81	1210.25	-0.45	1,200.53	0.35	**1,196.50**	0.69
UK100_12	1,039.43	1053.02	-1.31	1,030.17	0.89	**1,027.38**	1.16
UK100_13	**1,129.73**	1154.83	-2.22	1,132.02	-0.20	1,132.03	-0.20
UK100_14	1,245.03	1264.50	-1.56	**1,241.31**	0.30	1,242.68	0.19
UK100_15	1,306.61	1315.50	-0.68	1,311.36	-0.36	**1,300.13**	0.50
UK100_16	**980.46**	1005.03	-2.51	986.57	-0.62	981.86	-0.14
UK100_17	1,267.22	1284.81	-1.39	**1,257.44**	0.77	1,258.16	0.71
UK100_18	1,086.44	1106.00	-1.80	1,088.89	-0.23	**1,073.38**	1.20
UK100_19	1,016.82	1044.71	-2.74	1,024.17	-0.72	**1,015.95**	0.09
UK100_20	**1,237.87**	1263.06	-2.04	1,249.84	-0.97	1,240.00	-0.17
Average			-1.50		0.10		0.49

5. Conclusions

We have developed a metaheuristic algorithm for the TDPRP, which extends the PRP to settings with traffic congestion. This problem is of high practical relevance since congestion is an important problem for many cities and the amount of greenhouse gas emissions significantly increase at lower vehicle speeds. Our algorithm is based on an application of the classical ALNS heuristic and uses the departure and speed optimization procedure (DSOP) from Franceschetti et al. (2013) as a subroutine. Our implementation of the ALNS combines newly developed with pre-existing removal and insertion operators and we show that the new operators significantly improve the solution quality for medium- and large-size instances. Our numerical results show that our algorithm performs well and is relatively fast on instances with up to 200 nodes.

Acknowledgements

The work was partly supported by the Dutch Institute for Advanced Logistics under the project 4C4D and to the Canadian Natural Sciences and Engineering Research Council under grant 2015-061809. The authors gratefully acknowledge funding provided by the Technology University of Eindhoven. Thanks are due to the referees for their constructive feedback and to Kyle Hyndman for his valuable advice.

Appendix A: Calculation of the maximum congestion period length

In this section we show how to calculate the maximum length of the congestion period, denoted a^{max} in such a way that the problem is feasible. To this end, for each customer node $i \in N_0$ we calculate the

maximum length of the congestion period, denoted a_i^{max} so that it is possible for a vehicle to arrive at the customer node by the upper time window limit u_i and return to the depot by the upper time window limit u_0. There are three cases, depending on the distances, service time and time window limit values:

- Case 1: It is possible for a vehicle to arrive by u_i and u_0 in congestion, that is, $\frac{d_{0,i}}{v_c} < u_i$ and $\max\{\frac{d_{0,1}}{v_c}, l_i\} + h_i + \frac{d_{i,0}}{v_c} < u_0$. In this case, no matter what the length of the congestion period, the vehicle always arrives on time, i.e., $a_i^{max} = \infty$.

- Case 2: It is possible for a vehicle to arrive by u_i in congestion but not by u_0, that is, $\frac{d_{0,i}}{v_c} < u_i$ and $\max\{\frac{d_{0,1}}{v_c}, l_i\} + h_i + \frac{d_{i,0}}{v_c} > u_0$. In this case, the vehicle always arrives on time at the customer node but to arrive at the depot back on time, at least the return trip must be driven (partially) at the free flow speed. The arrival time back at the depot is given by

$$\mu_0^i = \begin{cases} \max\left\{a + \frac{d_{0,i} - av_c}{v^{max}}, l_i\right\} + h_i + \frac{d_{i,0}}{v^{max}} & \text{if } a < \frac{d_{0,i}}{v_c} \\ \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i + \frac{d_{i,0}}{v^{max}} & \text{if } \frac{d_{i,0}}{v_c} < a < \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i \\ a + \frac{d_{i,0} - \left[a - \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} - h_i\right] v_c}{v^{max}} & \text{if } \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i < a < \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i + \frac{d_{i,0}}{v_c}. \end{cases}$$

  To have $\mu_0^i \le u_0$, in the first row, we need

$$a \le \frac{u_0 v^{max} - h_i v^{max} - d_{0,i} - d_{i,0}}{v^{max} - v_c}$$

  because we can assume that $l_i + h_i + \frac{d_{i,0}}{v^{max}} \le u_0$ otherwise the problem is infeasible for all values of a.

  In the second row, if $\frac{d_{0,i}}{v_c} + h_i + \frac{d_{i,0}}{v^{max}} > u_0$, then for all values of a which falls into the range $\left[\frac{d_{i,0}}{v_c}, \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} + h_i\right]$, the vehicle would arrive at the depot late. Hence the outbound trip must take place in the transient region, in other words, we must have $a < \frac{d}{v_c}$.

  In the third row, we need

$$a \le \frac{u_0 v^{max} - \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} v_c - h_i v_c - d_{i,0}}{v^{max} - v_c}.$$

  Therefore

$$\begin{aligned} a_i^{max} &= \min\left\{\frac{u_0 v^{max} - h_i v^{max} - d_{0,i} - d_{i,0}}{v^{max} - v_c}, \frac{u_0 v^{max} - \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} v_c - h_i v_c - d_{i,0}}{v^{max} - v_c}\right\} \\ &= \frac{u_0 v^{max} - \max\left\{h_i v^{max}, \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} v_c + h_i v_c\right\} - d_{i,0}}{v^{max} - v_c}. \end{aligned}$$

- Case 3: It is not possible for the vehicle to arrive by u_i in congestion, that is, $\frac{d_{0,i}}{v_c} > u_i$. In this case, a must be less than $\frac{d_{0,i}}{v_c}$ so outbound trip is driven in the transient zone and the return trip

in free flow. The earliest possible arrival times at both nodes as a function of a are

$$\begin{aligned}\mu_i &= a + \frac{d_{0,i} - av_c}{v^{max}} \\ \mu_0 &= \max\left\{a + \frac{d_{0,i} - av_c}{v^{max}}, l_i\right\} + h_i + \frac{d_{i,0}}{v^{max}}.\end{aligned}$$

So we need

$$\begin{aligned}\mu_i \le u_i \Leftrightarrow a \le \frac{u_i v^{max} - d_{i,0}}{v^{max} - v_c} \\ \mu_0 \le u_0 \Leftrightarrow a \le \frac{u_0 v^{max} - h_i v^{max} - d_{0,i} - d_{i,0}}{v^{max} - v_c}.\end{aligned}$$

Again, note that $l_i + h_i + \frac{d_{i,0}}{v^{max}} \le u_0$ since otherwise the problem is infeasible for all values of a. Therefore

$$\begin{aligned}a_i^{max} &= \min\left\{\frac{u_i v^{max} - d_{0,i}}{v^{max} - v_c}, \frac{u_0 v^{max} - h_i v^{max} - d_{0,i} - d_{i,0}}{v^{max} - v_c}\right\} \\ &= \frac{\min\{u_i v^{max}, (u_0 - h_i)v^{max} - d_{i,0}\} - d_{0,i}}{v^{max} - v_c}.\end{aligned}$$

Combining all three cases, we obtain

$$a_i^{max} = \begin{cases} \frac{\min\{u_i v^{max}, (u_0 - h_i)v^{max} - d_{i,0}\} - d_{0,i}}{v^{max} - v_c} & \text{if } \frac{d_{0,i}}{u_i} \ge v_c \\ \frac{u_0 v^{max} - \max\left\{h_i v^{max}, \max\left\{\frac{d_{0,i}}{v_c}, l_i\right\} v_c + h_i v_c\right\} - d_{i,0}}{v^{max} - v_c} & \text{if } \frac{d_{0,i}}{v_c} < u_i \text{ and } \max\{\frac{d_{0,i}}{v_c}, l_i\} + h_i + \frac{d_{i,0}}{v_c} \ge u_0 \\ \infty & \text{if } \frac{d_{0,i}}{v_c} < u_i \text{ and } \max\{\frac{d_{0,i}}{v_c}, l_i\} + h_i + \frac{d_{i,0}}{v_c} < u_0 \end{cases} \quad \text{for } i = 1, ..., n.$$

Finally, we have

$$a^{max} = \min_{i \in N_0} a_i^{max}.$$

Appendix B: Value of the newly developed operators

In this section we provide complete results for our analysis of the value of the newly developed operators over all the instances from the tuning set. In Table 12 the the column entitled *Avg.* report the best solution found out of ten runs using Version 1 and Version 2 of the algorithm, respectively. The columns entitled $S_b^{Version\#1}$ and $S_b^{Version\#2}$ report the best solution found out of ten runs using Version 1 and Version 2 of the algorithm, respectively. Finally, the column entitled *Dev.* reports the percentage cost reduction between the two best solutions, $Dev = [TC(S_b^{Version\#2}) - TC(S_b^{Version\#1})]/TC(S_b^{Version\#1})$.

TABLE 12: Comparisons of Version # 1 and Version #2 on the 100-node instances

Instance	Version#1		Version#2		Dev. (%)	p-value
	Avg.	$TC(S_b^{Version\#1})$	Avg.	$TC(S_b^{Version\#2})$		
UK100_01	1,775.85	1,739.33	1,780.9	1,742.6	0.19	0.25
UK100_02	1,661.87	1,622.12	1,668.2	1,636.0	0.86	0.36
UK100_03	1,573.72	1,549.33	1,559.3	1,544.6	-0.31	0.14
UK100_04	1,565.55	1,547.92	1,557.5	1,541.5	-0.41	0.03
UK100_05	1,574.33	1,552.31	1,569.8	1,548.2	-0.26	0.29
UK100_06	1,791.34	1,759.96	1,778.4	1,763.3	0.19	0.08
UK100_07	1,528.59	1,518.92	1,533.9	1,520.9	0.13	0.17
UK100_08	1,537.15	1,530.04	1,535.4	1,527.3	-0.18	0.19
UK100_09	1,421.93	1,409.89	1,425.0	1,406.2	-0.27	0.40
UK100_10	1,555.16	1,520.60	1,551.2	1,525.9	0.35	0.19
UK100_11	1,740.56	1,727.62	1,741.4	1,716.6	-0.64	0.22
UK100_12	1,532.78	1,520.53	1,534.7	1,513.8	-0.44	0.40
UK100_13	1,692.68	1,680.68	1,696.7	1,671.8	-0.53	0.48
UK100_14	1,789.21	1,763.17	1,792.8	1,752.2	-0.62	0.32
UK100_15	1,887.12	1,854.64	1,885.7	1,860.9	0.34	0.48
UK100_16	1,506.27	1,489.38	1,516.1	1,499.1	0.65	0.01
UK100_17	1,801.15	1,776.99	1,806.0	1,772.0	-0.28	0.40
UK100_18	1,585.78	1,570.27	1,583.8	1,564.8	-0.35	0.36
UK100_19	1,500.69	1,464.38	1,495.0	1,473.2	0.60	0.32
UK100_20	1,857.40	1,839.29	1,875.0	1,844.8	0.30	0.02
UK100_01-B	1,660.94	1,625.96	1,650.1	1,618.52	-0.46	0.07
UK100_02-B	1,641.51	1,624.24	1,642.3	1,628.30	0.25	0.40
UK100_03-B	1,576.26	1,541.59	1,581.4	1,553.45	0.77	0.32
UK100_04-B	1,515.30	1,498.92	1,518.6	1,502.78	0.26	0.12
UK100_05-B	1,550.37	1,522.14	1,549.9	1,532.18	0.66	0.44
UK100_06-B	1,711.88	1,695.03	1,705.1	1,692.01	-0.18	0.04
UK100_07-B	1,549.15	1,542.97	1,544.9	1,542.97	0.00	0.05
UK100_08-B	1,553.68	1,537.51	1,552.9	1,537.38	-0.01	0.40
UK100_09-B	1,444.56	1,424.88	1,441.3	1,433.54	0.61	0.32
UK100_10-B	1,556.28	1,531.50	1,550.8	1,534.67	0.21	0.06
UK100_11-B	1,679.03	1,668.58	1,676.6	1,657.75	-0.65	0.29
UK100_12-B	1,437.48	1,421.41	1,442.8	1,432.53	0.78	0.08
UK100_13-B	1,677.96	1,653.45	1,672.1	1,654.87	0.09	0.22
UK100_14-B	1,735.30	1,698.22	1,733.3	1,703.44	0.31	0.29
UK100_15-B	1,791.38	1,761.05	1,788.0	1,772.05	0.62	0.32
UK100_16-B	1,427.73	1,410.04	1,429.2	1,417.47	0.53	0.36
UK100_17-B	1,737.80	1,711.61	1,726.7	1,710.87	-0.04	0.02
UK100_18-B	1,552.92	1,541.33	1,550.4	1,530.91	-0.68	0.22
UK100_19-B	1,470.30	1,441.51	1,464.0	1,438.03	-0.24	0.19
UK100_20-B	1,700.02	1,665.86	1,707.3	1,671.92	0.36	0.48
UK100_01-C	1,550.51	1,522.66	1,539.41	1,520.63	-0.13	0.05
UK100_02-C	1,502.72	1,476.25	1,500.84	1,479.63	0.23	0.48
UK100_03-C	1,366.37	1,351.60	1,365.79	1,354.21	0.19	0.29
UK100_04-C	1,429.74	1,405.74	1,424.31	1,401.10	-0.33	0.25
UK100_05-C	1,371.39	1,352.85	1,457.59	1,365.79	0.96	0.14
UK100_06-C	1,533.18	1,513.62	1,538.48	1,514.30	0.05	0.25
UK100_07-C	1,377.41	1,357.25	1,366.88	1,355.90	-0.10	0.01
UK100_08-C	1,441.77	1,419.95	1,433.48	1,410.85	-0.64	0.07
UK100_09-C	1,323.80	1,313.72	1,324.94	1,316.35	0.20	0.40
UK100_10-C	1,393.27	1,369.83	1,398.65	1,382.40	0.92	0.17
UK100_11-C	1,560.26	1,533.17	1,569.97	1,552.52	1.26	0.08
UK100_12-C	1,274.46	1,259.72	1,279.01	1,256.71	-0.24	0.17
UK100_13-C	1,496.27	1,473.94	1,487.29	1,471.82	-0.14	0.04
UK100_14-C	1,593.11	1,570.15	1,587.42	1,573.18	0.19	0.10
UK100_15-C	1,690.93	1,671.20	1,681.10	1,668.82	-0.14	0.03
UK100_16-C	1,271.56	1,246.90	1,269.81	1,253.98	0.57	0.25
UK100_17-C	1,622.98	1,606.50	1,620.18	1,609.12	0.16	0.32
UK100_18-C	1,377.57	1,351.86	1,378.82	1,364.85	0.96	0.44
UK100_19-C	1,323.38	1,305.83	1,319.67	1,310.40	0.35	0.17
UK100_20-C	1,581.79	1,556.33	1,585.28	1,571.01	0.94	0.29

TABLE 13: Comparisons of Version # 1 and Version #2 on the 200-node instances

Instance	Version#1		Version#2		Dev. (%)	p-value
	Avg.	$TC(S_b^{Version\#1})$	Avg.	$TC(S_b^{Version\#2})$		
UK200_01	2,961.79	2,940.28	2,954.23	2,937.10	-0.11	0.17
UK200_02	2,846.00	2,822.32	2,843.39	2,831.90	0.34	0.32
UK200_03	2,999.55	2,985.75	2,995.80	2,969.51	-0.54	0.05
UK200_04	2,798.91	2,775.31	2,798.59	2,776.50	0.04	0.32
UK200_05	3,033.96	2,998.85	3,024.76	3,006.11	0.24	0.10
UK200_06	2,746.86	2,734.37	2,744.59	2,731.15	-0.12	0.44
UK200_07	2,924.40	2,907.60	2,923.74	2,908.68	0.04	0.25
UK200_08	3,042.72	3,025.49	3,045.86	3,019.79	-0.19	0.22
UK200_09	2,732.12	2,706.26	2,730.27	2,691.77	-0.54	0.25
UK200_10	3,214.45	3,185.81	3,219.50	3,195.82	0.31	0.44
UK200_11	2,808.32	2,793.15	2,818.76	2,808.07	0.53	0.01
UK200_12	2,941.13	2,906.05	2,937.60	2,901.12	-0.17	0.36
UK200_13	3,047.95	3,010.19	3,049.56	3,028.57	0.61	0.40
UK200_14	2,890.13	2,876.86	2,894.91	2,877.53	0.02	0.06
UK200_15	2,967.42	2,949.03	2,974.70	2,945.18	-0.13	0.22
UK200_16	2,936.96	2,906.59	2,928.26	2,906.58	0.00	0.19
UK200_17	3,146.65	3,105.67	3,137.16	3,097.24	-0.27	0.10
UK200_18	2,892.18	2,875.36	2,892.02	2,877.08	0.06	0.44
UK200_19	2,687.20	2,656.33	2,700.42	2,680.44	0.91	0.08
UK200_20	3,117.70	3,073.94	3,122.30	3,103.60	0.96	0.22
UK200_01-B	2,896.23	2,868.41	2,888.74	2,859.37	-0.32	0.10
UK200_02-B	2,716.90	2,685.81	2,714.33	2,688.98	0.12	0.29
UK200_03-B	2,876.03	2,841.80	2,883.89	2,865.50	0.83	0.00
UK200_04-B	2,735.03	2,706.47	2,742.09	2,718.28	0.44	0.19
UK200_05-B	2,951.49	2,925.49	2,941.19	2,910.40	-0.52	0.07
UK200_06-B	2,711.00	2,676.30	2,709.43	2,683.69	0.28	0.29
UK200_07-B	2,772.41	2,749.17	2,766.31	2,748.82	-0.01	0.25
UK200_08-B	2,924.04	2,877.81	2,918.86	2,871.83	-0.21	0.36
UK200_09-B	2,609.52	2,572.47	2,622.01	2,599.93	1.07	0.08
UK200_10-B	3,001.31	2,979.99	2,999.26	2,975.82	-0.14	0.19
UK200_11-B	2,731.97	2,710.23	2,740.33	2,727.12	0.62	0.07
UK200_12-B	2,906.92	2,878.31	2,902.10	2,876.88	-0.05	0.32
UK200_13-B	2,865.27	2,845.21	2,863.58	2,850.51	0.19	0.48
UK200_14-B	2,832.61	2,805.29	2,822.18	2,788.13	-0.61	0.10
UK200_15-B	2,914.38	2,871.20	2,907.77	2,884.42	0.46	0.17
UK200_16-B	2,852.73	2,829.41	2,848.78	2,826.42	-0.11	0.25
UK200_17-B	2,982.58	2,949.17	2,979.86	2,946.95	-0.08	0.32
UK200_18-B	2,856.53	2,811.88	2,864.31	2,834.02	0.79	0.22
UK200_19-B	2,572.62	2,549.55	2,568.53	2,543.70	-0.23	0.36
UK200_20-B	2,979.89	2,956.62	2,975.72	2,944.95	-0.39	0.25
UK200_01-C	2,683.23	2,663.14	2,679.85	2,664.49	0.05	0.25
UK200_02-C	2,565.09	2,529.00	2,562.30	2,542.72	0.54	0.44
UK200_03-C	2,631.72	2,612.35	2,635.16	2,596.04	-0.62	0.36
UK200_04-C	2,526.65	2,488.33	2,529.98	2,493.34	0.20	0.44
UK200_05-C	2,718.89	2,699.56	2,727.39	2,706.16	0.24	0.32
UK200_06-C	2,461.60	2,440.44	2,461.71	2,435.57	-0.20	0.40
UK200_07-C	2,610.89	2,574.81	2,611.17	2,573.59	-0.05	0.48
UK200_08-C	2,637.10	2,619.46	2,637.88	2,608.67	-0.41	0.22
UK200_09-C	2,395.47	2,377.58	2,398.71	2,379.78	0.09	0.25
UK200_10-C	2,705.21	2,660.76	2,714.17	2,685.24	0.92	0.08
UK200_11-C	2,531.30	2,510.59	2,534.32	2,503.82	-0.27	0.44
UK200_12-C	2,672.57	2,631.50	2,668.76	2,655.95	0.93	0.32
UK200_13-C	2,637.66	2,619.02	2,632.95	2,607.96	-0.42	0.29
UK200_14-C	2,571.10	2,543.72	2,563.38	2,546.83	0.12	0.19
UK200_15-C	2,659.14	2,642.61	2,654.70	2,635.56	-0.27	0.14
UK200_16-C	2,625.74	2,591.23	2,622.02	2,605.65	0.56	0.25
UK200_17-C	2,689.45	2,663.41	2,710.96	2,694.62	1.17	0.40
UK200_18-C	2,593.43	2,562.59	2,586.86	2,557.18	-0.21	0.25
UK200_19-C	2,376.19	2,347.99	2,376.72	2,348.70	0.03	0.48
UK200_20-C	2,660.56	2,617.58	2,658.52	2,627.90	0.39	0.48

TABLE 14: Computational results for the 10-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK10_01	17235	0.23	323.19	323.19	0.00	3261	0.25	245.96	245.96	0.00	11614	0.19	223.70	223.70	0.00
UK10_02	14722	0.11	326.49	326.49	0.00	5232	0.29	303.09	303.09	0.00	9672	0.18	277.15	277.15	0.00
UK10_03	16506	0.19	318.82	318.82	0.00	5070	0.23	301.12	301.12	0.00	8570	0.22	242.18	242.18	0.00
UK10_04	15085	0.14	299.62	299.62	0.00	3035	0.18	273.40	273.40	0.00	12189	0.15	242.72	242.72	0.00
UK10_05	14219	0.16	244.79	244.79	0.00	2987	0.13	293.37	293.37	0.00	7444	0.15	216.28	216.28	0.00
UK10_06	15581	0.15	350.17	350.17	0.00	4348	0.28	332.91	332.91	0.00	11099	0.19	316.94	316.94	0.00
UK10_07	14913	0.16	277.567	277.567	0.00	5199	0.27	312.93	312.93	0.00	8880	0.24	230.92	230.92	0.00
UK10_08	14030	0.16	357.49	357.49	0.00	5940	0.23	338.91	338.91	0.00	6421	0.17	304.96	304.96	0.00
UK10_09	14411	0.20	240.87	240.87	0.00	2143	0.13	262.32	262.32	0.00	5429	0.26	201.03	201.03	0.00
UK10_10	15883	0.16	273.58	273.58	0.00	3788	0.25	287.76	287.76	0.00	9751	0.19	282.59	282.59	0.00
UK10_11	13744	0.19	401.99	401.99	0.00	6210	0.21	424.88	424.88	0.00	5479	0.18	307.50	307.50	0.00
UK10_12	14474	0.12	296.45	296.45	0.00	2506	0.17	252.67	252.67	0.00	7538	0.24	217.42	217.42	0.00
UK10_13	15015	0.20	312.42	312.42	0.00	2936	0.25	272.80	272.80	0.00	5704	0.21	223.68	223.68	0.00
UK10_14	14354	0.21	266.97	266.97	0.00	4396	0.19	282.26	282.26	0.00	5853	0.22	226.29	226.29	0.00
UK10_15	14118	0.18	221.95	221.95	0.00	3952	0.29	202.25	202.25	0.00	6718	0.31	175.99	175.99	0.00
UK10_16	14706	0.21	248.57	248.57	0.00	4275	0.30	244.85	244.85	0.00	11516	0.12	242.34	242.34	0.00
UK10_17	14546	0.24	252.25	252.25	0.00	3704	0.25	289.13	289.13	0.00	11071	0.20	239.61	239.61	0.00
UK10_18	13922	0.18	240.44	240.22	0.09	7403	0.28	250.73	250.73	0.00	6519	0.17	202.56	202.56	0.00
UK10_19	15881	0.20	292.66	292.66	0.00	4422	0.23	328.44	328.44	0.00	10008	0.19	255.66	254.00	0.65
UK10_20	16429	0.22	259.48	259.48	0.00	2866	0.26	208.74	208.74	0.00	5102	0.21	203.27	203.27	0.00
Average		0.18			0.00		0.23			0.00		0.20			0.03

TABLE 15: Computational results for the 15-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK15_01	15199	0.30	455.08	455.08	0.00	2277	0.26	389.36	389.36	0.00	9769	0.31	372.99	372.99	0.00
UK15_02	15849	0.28	353.27	353.27	0.00	3101	0.38	294.92	294.92	0.00	10520	0.34	299.24	299.24	0.00
UK15_03	13385	0.30	384.40	384.40	0.00	3425	0.34	439.37	439.37	0.00	7172	0.45	338.24	338.24	0.00
UK15_04	15096	0.36	423.66	423.66	0.00	2366	0.41	398.91	398.91	0.00	4516	0.48	359.07	359.07	0.00
UK15_05	14807	0.20	489.98	489.98	0.00	2910	0.30	451.55	451.55	0.00	8215	0.23	401.37	400.99	0.09
UK15_06	16607	0.27	377.51	377.51	0.00	2607	0.49	336.27	336.27	0.00	3860	0.36	270.69	265.48	1.96
UK15_07	14655	0.278	377.77	377.77	0.00	6947	0.39	399.75	399.75	0.00	7777	0.29	338.52	338.48	0.01
UK15_08	15450	0.45	256.24	256.24	0.00	4414	0.31	258.06	258.06	0.00	1793	0.37	216.25	216.25	0.00
UK15_09	13818	0.25	363.98	363.98	0.00	5599	0.38	343.09	343.09	0.00	6204	0.41	309.45	309.45	0.00
UK15_10	16375	0.53	353.38	353.38	0.00	3579	0.63	315.52	315.52	0.00	3518	0.55	257.51	257.51	0.00
UK15_11	15653	0.27	445.59	445.59	0.00	5666	0.45	432.86	432.86	0.00	5793	0.26	349.25	349.25	0.00
UK15_12	14632	0.26	475.79	475.79	0.00	3463	0.50	402.96	402.96	0.00	3597	0.41	359.89	359.81	0.02
UK15_13	13572	0.33	388.67	388.67	0.00	3954	0.44	366.13	359.11	1.95	5068	0.48	304.26	304.26	0.00
UK15_14	14453	0.26	510.50	510.50	0.00	3008	0.37	461.63	461.63	0.00	12835	0.25	480.28	480.28	0.00
UK15_15	14232	0.34	352.21	352.21	0.00	2549	0.30	344.31	344.31	0.00	4455	0.23	277.62	277.62	0.00
UK15_16	14909	0.31	329.23	329.10	0.04	3961	0.48	355.25	355.25	0.00	8637	0.45	281.78	281.78	0.00
UK15_17	14480	0.38	409.44	409.44	0.00	2888	0.38	416.82	416.82	0.00	4061	0.27	326.41	326.41	0.00
UK15_18	13258	0.20	449.22	449.22	0.00	3245	0.49	480.58	480.58	0.00	6777	0.23	386.81	386.81	0.00
UK15_19	16683	0.27	303.88	303.88	0.00	4321	0.33	272.32	272.32	0.00	9592	0.34	234.87	234.87	0.00
UK15_20	14999	0.41	300.46	300.34	0.04	3607	0.46	288.72	288.72	0.00	15273	0.44	330.74	330.74	0.00
Average		0.31			0.00		0.41			0.10		0.36			0.10

TABLE 16: Computational results for the 20-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK20_01	14172	0.38	499.38	499.38	0.00	3329	0.76	443.83	443.83	0.00	4174	0.51	384.78	384.78	0.00
UK20_02	13177	0.35	528.66	524.15	0.86	2825	0.68	442.01	442.01	0.00	6782	0.38	396.39	396.39	0.00
UK20_03	15013	0.45	324.49	323.78	0.22	2427	0.89	322.18	322.18	0.00	2241	0.49	262.94	262.94	0.00
UK20_04	14356	0.50	498.80	497.05	0.35	2190	0.57	477.44	473.04	0.93	2949	0.44	398.49	398.29	0.05
UK20_05	14317	0.44	468.02	468.02	0.00	2138	0.85	430.68	430.68	0.00	6311	0.65	353.44	353.44	0.00
UK20_06	14768	0.56	511.35	511.35	0.00	2489	0.53	489.45	488.05	0.29	8083	0.61	439.02	438.30	0.16
UK20_07	14035	0.432	351.28	351.12	0.05	4218	0.85	322.34	322.34	0.00	5263	0.63	288.38	288.38	0.00
UK20_08	15799	0.32	463.88	463.88	0.00	4179	0.63	444.28	444.28	0.00	6754	0.41	372.60	371.36	0.33
UK20_09	15044	0.61	551.73	551.73	0.00	2581	0.69	505.46	505.46	0.00	5052	0.44	415.50	412.74	0.67
UK20_10	15506	0.33	448.76	448.76	0.00	2491	0.40	409.23	409.23	0.00	3878	0.66	368.13	366.88	0.34
UK20_11	14227	0.56	529.98	529.98	0.00	4654	0.71	577.65	577.65	0.00	7483	0.54	484.97	482.60	0.49
UK20_12	14873	0.37	483.78	481.69	0.44	4372	0.67	507.48	507.48	0.00	6039	0.53	372.33	371.85	0.13
UK20_13	14865	0.34	499.17	498.43	0.15	5605	0.46	463.22	462.01	0.26	5370	0.52	399.24	399.17	0.02
UK20_14	14243	0.30	582.00	582.00	0.00	4950	0.49	562.47	562.47	0.00	10299	0.31	550.03	550.03	0.00
UK20_15	14215	0.44	479.60	477.83	0.37	2475	0.65	501.69	501.69	0.00	7532	0.72	390.77	390.77	0.00
UK20_16	13984	0.39	487.35	487.35	0.00	5377	0.44	495.68	495.68	0.00	6281	0.64	433.79	433.79	0.00
UK20_17	13819	0.40	529.55	528.48	0.20	4100	0.44	506.67	506.67	0.00	9380	0.54	473.28	472.95	0.07
UK20_18	13842	0.36	530.27	530.27	0.00	2769	0.92	481.52	481.52	0.00	3385	0.62	417.44	417.44	0.00
UK20_19	14612	0.46	535.03	524.89	1.93	3585	0.58	505.39	503.42	0.39	4965	0.46	408.34	391.49	4.31
UK20_20	15132	0.39	514.90	514.90	0.00	2251	0.68	506.34	506.34	0.00	8863	0.42	432.47	432.47	0.00
Average		0.42			0.23		0.64			0.09		0.53			0.33

TABLE 17: Computational results for the 25-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK25_01	14063	0.78	445.63	445.63	0.00	3605	1.51	436.05	436.05	0.00	4333	0.78	375.92	374.04	0.50
UK25_02	14190	0.66	489.75	488.15	0.33	3311	0.85	480.40	479.35	0.22	4300	1.02	434.87	433.22	0.38
UK25_03	14546	0.81	318.74	315.19	1.13	4304	0.85	338.68	335.71	0.88	10544	0.81	318.74	315.19	1.13
UK25_04	13838	0.57	370.59	370.59	0.00	2910	1.17	407.78	407.61	0.04	5424	0.86	333.84	333.84	0.00
UK25_05	14255	0.61	521.27	519.88	0.27	3604	0.80	476.93	476.93	0.00	5174	0.66	419.80	415.37	1.07
UK25_06	13940	0.71	424.71	424.69	0.01	5560	0.75	459.06	457.89	0.26	5853	0.57	381.82	379.00	0.74
UK25_07	14036	0.63	501.93	501.29	0.13	2981	0.78	484.32	478.23	1.27	5728	0.53	422.71	413.78	2.16
UK25_08	14269	0.42	541.72	536.40	0.99	3096	0.69	506.74	506.20	0.11	4891	0.80	468.35	444.33	5.41
UK25_09	16804	0.66	461.14	459.80	0.29	4956	1.23	398.08	397.78	0.07	7237	0.84	381.78	381.78	0.00
UK25_10	13762	0.59	503.99	502.92	0.21	2826	1.07	485.72	485.55	0.04	4942	0.84	449.93	447.18	0.62
UK25_11	13825	0.58	550.48	544.82	1.04	3613	1.00	528.90	525.23	0.70	4854	0.68	439.06	432.83	1.44
UK25_12	13922	0.62	615.51	608.80	1.10	4080	1.18	598.78	598.78	0.00	3216	0.86	481.92	481.92	0.00
UK25_13	15083	1.04	364.32	364.32	0.00	3710	0.83	376.37	376.27	0.03	7280	0.73	339.61	332.13	2.25
UK25_14	14409	0.43	592.72	592.72	0.00	2373	1.10	536.68	530.20	1.22	3930	0.47	455.51	449.73	1.29
UK25_15	14111	0.50	586.54	586.54	0.00	2683	1.52	536.93	536.93	0.00	8559	0.69	505.11	499.11	1.20
UK25_16	14399	0.61	495.37	495.37	0.00	4869	0.47	501.71	493.77	1.61	4331	0.76	432.81	430.49	0.54
UK25_17	15606	0.53	730.12	730.12	0.00	3127	0.59	629.71	629.61	0.02	5876	0.58	573.67	573.14	0.09
UK25_18	13969	0.61	598.44	598.44	0.00	2791	0.63	526.71	525.61	0.21	6148	1.05	496.93	496.86	0.01
UK25_19	14273	0.73	614.34	614.13	0.03	4066	1.15	609.32	609.11	0.03	4424	0.64	494.91	494.77	0.03
UK25_20	14385	0.55	536.17	523.97	2.33	4644	0.63	531.26	522.64	1.65	4203	0.79	431.94	427.92	0.94
Average		0.63			0.39		0.94			0.42		0.75			0.99

TABLE 18: Computational results for the 50-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK50_01	14248	1.97	906.52	893.38	1.47	3120	2.80	898.98	888.02	1.23	5707	2.45	784.08	773.99	1.30
UK50_02	14320	1.88	883.78	883.78	0.00	2185	2.95	883.11	878.97	0.47	3005	2.55	784.16	780.19	0.51
UK50_03	13682	2.22	903.69	882.91	2.35	3153	3.84	859.50	853.86	0.66	3145	2.40	790.98	772.22	2.43
UK50_04	14361	1.89	1,094.15	1,053.01	3.91	3508	2.67	1,031.20	1,023.97	0.71	2386	2.31	912.30	897.25	1.68
UK50_05	13238	1.97	869.93	869.12	0.09	2212	2.52	906.55	904.56	0.22	4710	2.72	822.31	811.11	1.38
UK50_06	14561	2.14	854.32	850.56	0.44	4315	2.51	840.39	835.75	0.56	3897	2.73	761.56	757.60	0.52
UK50_07	13021	2.55	750.19	749.82	0.05	2721	2.61	766.69	758.05	1.14	3732	2.66	716.61	709.87	0.95
UK50_08	13021	2.18	800.25	791.62	1.09	2472	2.85	814.55	808.37	0.76	6309	2.29	719.51	716.66	0.40
UK50_09	13712	1.90	1,022.50	1,010.13	1.22	2996	3.15	947.13	934.55	1.35	4932	2.06	843.01	834.22	1.05
UK50_10	14211	1.82	1,001.97	983.27	1.90	2136	2.24	908.81	891.55	1.94	4247	2.56	851.15	840.88	1.22
UK50_11	14114	1.80	940.21	930.87	1.00	1758	3.25	889.54	882.66	0.78	3886	2.34	775.13	765.58	1.25
UK50_12	15202	2.14	873.39	860.31	1.52	2375	3.03	812.41	808.57	0.47	3591	3.11	724.02	717.22	0.95
UK50_13	13681	2.16	816.14	805.05	1.38	2527	2.96	804.72	802.70	0.25	2775	2.13	772.31	765.40	0.90
UK50_14	14305	1.81	1,010.24	994.91	1.54	2620	2.26	963.46	942.18	2.26	3600	2.52	800.86	790.81	1.27
UK50_15	13965	2.03	866.43	859.81	0.77	2900	3.11	830.96	823.67	0.88	4988	2.71	741.83	730.94	1.49
UK50_16	14004	1.70	896.91	847.73	5.80	3302	3.08	786.29	783.49	0.36	5734	2.68	751.64	742.74	1.20
UK50_17	16262	2.74	731.20	719.36	1.65	2954	3.35	721.16	713.27	1.11	3194	2.61	635.65	623.10	2.02
UK50_18	14503	1.93	1,013.90	1,005.45	0.84	3647	3.23	940.13	917.49	2.47	2148	2.53	842.90	840.04	0.34
UK50_19	14557	2.01	891.23	874.23	1.95	2924	3.31	823.62	813.09	1.30	4799	2.15	760.31	753.74	0.87
UK50_20	14530	2.59	1,006.38	1,003.11	0.33	2205	2.94	939.03	912.27	2.93	3474	2.41	836.70	820.50	1.97
Average		2.07			1.47		2.93			1.09		2.50			1.19

TABLE 19: Computational results for the 75-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK75_01	13791	3.76	1,429.58	1,414.82	1.04	2473	5.70	1,392.11	1,379.86	0.89	4081	5.03	1,224.90	1,207.52	1.44
UK75_02	14724	3.91	1,262.61	1,247.01	1.25	2623	7.13	1,192.99	1,179.96	1.10	3439	4.33	1,055.24	1,048.61	0.63
UK75_03	14357	4.13	1,294.05	1,282.06	0.93	2597	6.97	1,266.17	1,246.06	1.61	2702	6.16	1,079.22	1,064.03	1.43
UK75_04	14487	4.64	1,175.53	1,166.10	0.81	2577	6.00	1,161.13	1,155.14	0.52	2977	5.59	1,113.08	1,104.62	0.77
UK75_05	13790	4.48	1,290.63	1,258.37	2.56	2281	7.13	1,258.48	1,246.75	0.94	3058	5.36	1,080.27	1,072.16	0.76
UK75_06	13871	3.98	1,356.91	1,327.90	2.18	3677	6.90	1,315.16	1,307.21	0.61	2911	5.40	1,170.27	1,152.94	1.50
UK75_07	14600	4.19	1,421.00	1,413.67	0.52	2615	5.89	1,361.21	1,322.89	2.90	2142	5.28	1,147.00	1,137.82	0.81
UK75_08	13756	3.81	1,407.70	1,390.00	1.27	2662	5.17	1,308.47	1,296.98	0.89	2180	5.16	1,239.08	1,231.22	0.64
UK75_09	13485	4.10	1,338.12	1,324.47	1.03	2343	6.48	1,290.23	1,266.64	1.86	3328	4.94	1,141.67	1,126.06	1.39
UK75_10	13678	3.72	1,403.51	1,371.05	2.37	1827	5.42	1,314.71	1,288.71	2.02	2492	5.17	1,197.91	1,182.73	1.28
UK75_11	14280	3.49	1,057.66	1,024.76	3.21	2982	6.38	1,015.20	1,006.75	0.84	4024	5.14	922.08	914.49	0.83
UK75_12	14166	3.81	1,302.20	1,286.17	1.25	1988	5.84	1,177.70	1,161.57	1.39	2612	4.72	1,103.37	1,093.29	0.92
UK75_13	13206	3.64	1,384.41	1,363.84	1.51	2595	5.24	1,345.63	1,343.85	0.13	2813	5.01	1,187.50	1,165.09	1.92
UK75_14	13577	4.09	1,341.10	1,308.85	2.46	2954	5.59	1,270.97	1,251.24	1.58	6717	4.59	1,196.13	1,175.82	1.73
UK75_15	13208	3.93	1,415.43	1,383.70	2.29	2882	5.28	1,433.11	1,412.61	1.45	2605	4.93	1,202.93	1,174.59	2.41
UK75_16	13455	4.08	1,296.93	1,292.30	0.36	2992	5.44	1,301.61	1,284.41	1.34	4129	4.92	1,202.49	1,175.82	2.27
UK75_17	13742	3.82	1,301.64	1,269.42	2.54	2879	5.19	1,346.36	1,337.42	0.67	2870	4.99	1,162.26	1,139.92	1.96
UK75_18	14189	4.47	1,282.90	1,266.32	1.31	2653	6.06	1,220.37	1,194.25	2.19	4924	4.89	1,112.84	1,100.81	1.09
UK75_19	14518	3.81	1,266.01	1,236.67	2.37	3094	6.46	1,191.36	1,178.82	1.06	4954	5.29	1,084.82	1,071.01	1.29
UK75_20	14258	3.84	1,359.03	1,329.19	2.25	2902	6.58	1,326.19	1,304.07	1.70	4061	4.91	1,181.01	1,165.13	1.36
Average		3.98			1.68		6.04			1.28		5.09			1.32

TABLE 20: Computational results for the 100-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK100_01	13788	9.01	1,775.85	1,739.33	2.10	3277.00	11.18	1,660.94	1,625.96	2.15	2988.00	11.17	1,550.51	1,522.66	1.83
UK100_02	13269	9.02	1,661.87	1,622.12	2.45	2096.00	13.11	1,641.51	1,624.24	1.06	3631.00	11.24	1,502.72	1,476.25	1.79
UK100_03	13663	8.50	1,573.72	1,549.33	1.57	2413.00	12.83	1,576.26	1,541.59	2.25	2425.00	12.00	1,366.37	1,351.60	1.09
UK100_04	14037	9.61	1,565.55	1,547.92	1.14	2130.00	12.53	1,515.30	1,498.92	1.09	2385.00	12.23	1,429.74	1,405.74	1.71
UK100_05	13713	9.30	1,574.33	1,552.31	1.42	3117.00	13.31	1,550.37	1,522.14	1.85	3447.00	11.03	1,371.39	1,352.85	1.37
UK100_06	14217	8.80	1,791.34	1,759.96	1.78	2987.00	14.65	1,711.88	1,695.03	0.99	3542.00	10.24	1,533.18	1,513.62	1.29
UK100_07	13631	9.89	1,528.59	1,518.92	0.64	2402.00	13.82	1,549.15	1,542.97	0.40	2165.00	11.44	1,377.41	1,357.25	1.49
UK100_08	12826	9.31	1,537.15	1,530.04	0.46	2780.00	10.94	1,553.68	1,537.51	1.05	2425.00	11.48	1,441.77	1,419.95	1.54
UK100_09	13288	9.44	1,421.93	1,409.89	0.85	2526.00	13.17	1,444.56	1,424.88	1.38	3812.00	11.71	1,323.80	1,313.72	0.77
UK100_10	13524	8.79	1,555.16	1,520.60	2.27	2403.00	12.85	1,556.28	1,531.50	1.62	5118.00	9.75	1,393.27	1,369.83	1.71
UK100_11	13845	8.71	1,740.56	1,727.62	0.75	2699.00	13.59	1,679.03	1,668.58	0.63	2841.00	11.28	1,560.26	1,533.17	1.77
UK100_12	13783	8.37	1,532.78	1,520.53	0.81	2070.00	13.36	1,437.48	1,421.41	1.13	2548.00	11.37	1,274.46	1,259.72	1.17
UK100_13	13822	9.13	1,692.68	1,680.68	0.71	3108.00	13.64	1,677.96	1,653.45	1.48	2703.00	12.11	1,496.27	1,473.94	1.52
UK100_14	13292	7.61	1,789.21	1,763.17	1.48	2281.00	12.97	1,735.30	1,698.22	2.18	2988.00	10.65	1,593.11	1,570.15	1.46
UK100_15	13688	8.26	1,887.12	1,854.64	1.75	3635.00	12.13	1,791.38	1,761.05	1.72	4231.00	10.06	1,690.93	1,671.20	1.18
UK100_16	14651	9.89	1,506.27	1,489.38	1.13	3167.00	14.35	1,427.73	1,410.04	1.25	2628.00	12.23	1,271.56	1,246.90	1.98
UK100_17	14651	8.50	1,801.15	1,776.99	1.36	2317.00	12.13	1,737.80	1,711.61	1.53	3214.00	12.23	1,622.98	1,606.50	1.03
UK100_18	14451	8.99	1,585.78	1,570.27	0.99	2307.00	12.13	1,552.92	1,541.33	0.75	2128.00	9.82	1,377.57	1,351.86	1.90
UK100_19	13983	9.09	1,500.69	1,464.38	2.48	2436.00	14.13	1,470.30	1,441.51	2.00	6323.00	11.76	1,323.38	1,305.83	1.34
UK100_20	13961	8.17	1,857.40	1,839.29	0.98	2788.00	12.28	1,700.02	1,665.86	2.05	3541.00	11.52	1,581.79	1,556.33	1.64
Average		8.92			1.36		12.96			1.43		11.27			1.48

TABLE 21: Computational results for the 150-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK150_01	13748	22.14	2,142.57	2,113.38	1.38	2810	27.54	2,106.50	2,079.48	1.30	3653	25.98	1,927.06	1,902.26	1.30
UK150_02	13588	20.02	2,405.09	2,386.41	0.78	1728	27.00	2,317.50	2,290.05	1.20	2915	23.89	2,145.32	2,104.92	1.92
UK150_03	13396	19.86	2,059.18	2,053.40	0.28	1945	31.13	2,089.86	2,065.12	1.20	2548	25.67	1,860.00	1,838.40	1.17
UK150_04	13440	18.74	2,363.72	2,335.53	1.21	2236	24.72	2,308.84	2,277.51	1.38	2283	25.49	2,187.80	2,154.78	1.53
UK150_05	14113	21.19	2,205.49	2,184.24	0.97	2373	28.47	2,079.40	2,054.00	1.24	3805	24.96	1,933.89	1,908.61	1.32
UK150_06	13351	21.64	2,178.35	2,161.14	0.80	2517	23.39	2,169.21	2,140.02	1.36	2622	23.07	1,942.99	1,924.33	0.97
UK150_07	13658	19.80	2,449.80	2,413.44	1.51	2558	29.49	2,387.58	2,361.18	1.12	3287	25.81	2,206.74	2,183.08	1.08
UK150_08	13375	18.60	2,242.10	2,218.11	1.08	2491	31.00	2,155.34	2,140.44	0.70	3212	24.96	2,064.26	2,020.67	2.16
UK150_09	13692	19.66	2,409.45	2,384.49	1.05	2194	26.89	2,278.69	2,248.08	1.36	3039	23.09	2,089.03	2,075.83	0.64
UK150_10	12793	19.98	2,257.46	2,242.52	0.67	2038	29.63	2,252.27	2,238.52	0.61	2910	26.04	2,072.90	2,055.90	0.83
UK150_11	13575	18.44	2,409.85	2,374.14	1.50	2457	28.20	2,305.03	2,289.52	0.68	2161	25.36	2,159.53	2,137.02	1.05
UK150_12	13564	18.88	2,513.77	2,500.93	0.51	2468	21.20	2,378.83	2,347.08	1.35	3567	22.72	2,260.26	2,214.06	2.09
UK150_13	13863	19.45	2,364.80	2,339.61	1.08	2725	26.66	2,254.47	2,238.72	0.70	2208	24.75	2,023.20	1,999.61	1.18
UK150_14	13679	19.34	2,364.25	2,340.97	0.99	1712	28.53	2,348.70	2,328.91	0.85	3935	21.56	2,107.89	2,092.72	0.72
UK150_15	14571	19.30	2,159.74	2,139.43	0.95	2133	29.99	2,044.73	2,022.52	1.10	3295	22.07	1,880.75	1,862.98	0.95
UK150_16	13000	18.87	2,306.00	2,283.59	0.98	2786	27.47	2,283.51	2,256.25	1.21	2743	24.33	2,091.01	2,059.69	1.52
UK150_17	13343	18.11	2,355.58	2,317.54	1.64	2158	26.18	2,293.13	2,271.06	0.97	2942	23.97	2,138.93	2,103.37	1.69
UK150_18	13884	20.07	2,391.15	2,369.61	0.91	2237	28.31	2,222.45	2,201.73	0.94	3300	24.97	2,047.83	2,029.08	0.92
UK150_19	13406	19.10	2,524.14	2,495.35	1.15	2004	25.28	2,415.49	2,385.03	1.28	3070	22.51	2,240.82	2,214.74	1.18
UK150_20	13787	19.75	2,526.38	2,500.41	1.04	2193	26.90	2,428.96	2,410.32	0.77	2386	23.77	2,184.68	2,150.95	1.57
Average		19.65			1.02		27.40			1.07		24.25			1.29

TABLE 22: Computational results for the 200-node instances.

Instance	PRPLIB					KSVC14 Set B					KSVC14 Set C				
	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap Gap	a (s)	CPU (min)	Avg. (£)	Best (£)	Gap (%)
UK200_1	13305	32.53	2,961.79	2,940.28	0.73	2325	48.88	2,896.23	2,868.41	0.97	2691	46.04	2,683.23	2,663.14	0.75
UK200_2	13991	34.86	2,846.00	2,822.32	0.84	2612	51.25	2,716.90	2,685.81	1.16	3836	43.85	2,565.09	2,529.00	1.43
UK200_3	14094	33.45	2,999.55	2,985.75	0.46	2010	50.17	2,876.03	2,841.80	1.20	4221	43.46	2,631.72	2,612.35	0.74
UK200_4	13983	37.10	2,798.91	2,775.31	0.85	2218	50.81	2,735.03	2,706.47	1.06	3525	45.47	2,526.65	2,488.33	1.54
UK200_5	13171	30.36	3,033.96	2,998.85	1.17	2381	49.00	2,951.49	2,925.49	0.89	1906	43.00	2,718.89	2,699.56	0.72
UK200_6	13959	36.87	2,746.86	2,734.37	0.46	1835	50.34	2,711.00	2,676.30	1.30	1832	46.59	2,461.60	2,440.44	0.87
UK200_7	14070	35.10	2,924.40	2,907.60	0.58	2176	49.54	2,772.41	2,749.17	0.85	3446	45.24	2,610.89	2,574.81	1.40
UK200_8	14137	36.32	3,042.72	3,025.49	0.57	2562	48.46	2,924.04	2,877.81	1.61	2702	46.71	2,637.10	2,619.46	0.67
UK200_9	13892	36.47	2,732.12	2,706.26	0.96	2183	47.66	2,609.52	2,572.47	1.44	3302	46.05	2,395.47	2,377.58	0.75
UK200_10	13997	31.76	3,214.45	3,185.81	0.90	2133	52.53	3,001.31	2,979.99	0.72	2449	43.09	2,705.21	2,660.76	1.67
UK200_11	13968	37.36	2,808.32	2,793.15	0.54	2586	49.04	2,731.97	2,710.23	0.80	3635	48.73	2,531.30	2,510.59	0.82
UK200_12	12883	32.68	2,941.13	2,906.05	1.21	2204	49.16	2,906.92	2,878.31	0.99	2521	43.99	2,672.57	2,631.50	1.56
UK200_13	13614	34.68	3,047.95	3,010.19	1.25	2330	46.83	2,865.27	2,845.21	0.71	2636	41.72	2,637.66	2,619.02	0.71
UK200_14	13775	33.21	2,890.13	2,876.86	0.46	1868	48.91	2,832.61	2,805.29	0.97	2320	47.20	2,571.10	2,543.72	1.08
UK200_15	13349	32.47	2,967.42	2,949.03	0.62	2580	47.42	2,914.38	2,871.20	1.50	2697	45.42	2,659.14	2,642.61	0.63
UK200_16	13617	32.94	2,936.96	2,906.59	1.04	2117	49.07	2,852.73	2,829.41	0.82	2765	41.19	2,625.74	2,591.23	1.33
UK200_17	13540	33.99	3,146.65	3,105.67	1.32	2466	47.35	2,982.58	2,949.17	1.13	2066	42.28	2,689.45	2,663.41	0.98
UK200_18	13492	34.20	2,892.18	2,875.36	0.58	2376	45.67	2,856.53	2,811.88	1.59	2978	42.25	2,593.43	2,562.59	1.20
UK200_19	13875	37.65	2,687.20	2,656.33	1.16	1790	53.31	2,572.62	2,549.55	0.90	2394	45.93	2,376.19	2,347.99	1.20
UK200_20	13913	34.50	3,117.70	3,073.94	1.42	2156	47.72	2,979.89	2,956.62	0.79	2353	42.50	2,660.56	2,617.58	1.64
Average		34.42			0.84		49.16			1.07		44.54			1.07

Appendix C: Computational Results

References

D. Aksen, O. Kaya, F. S. Salman, and Ö. Tüncel. An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research*, 239(2):413–426, 2014.

M. Barth and K. Boriboonsomsin. Energy and emissions impacts of a freeway-based dynamic eco-driving system. *Transportation Research Part D: Transport and Environment*, 14(6):400–410, 2009.

T. Bektaş and G. Laporte. The pollution-routing problem. *Transportation Research Part B: Methodological*, 45(8): 1232–1250, 2011.

S. Dabia, E. Demir, and T. Van Woensel. An exact approach for a special variant of the pollution-routing problem. *Forthcoming in Transportation Science*, 2016.

E. Demir, T. Bektaş, and G. Laporte. A comparative analysis of several vehicle emission models for road freight transportation. *Transportation Research Part D: Transport and Environment*, 6(5):347–357, 2011.

E. Demir, Bektaş T., and Laporte G. An adaptive large neighborhood search heuristic for the Pollution-Routing Problem. *European Journal of Operational Research*, 223(2):346–359, 2012.

E. Demir, T. Bektaş, and G. Laporte. The bi-objective Pollution-Routing Problem. *European Journal of Operational Research*, 232(3):464–478, 2013.

E. Demir, Bektaş T., and Laporte G. A review of recent research on green road freight transportation. *European Journal of Operational Research*, 237:775–793, 2014.

E. Demir, Y. Huang, S. Scholts, and T. Van Woensel. A selected review on the negative externalities of the freight transportation: Modeling and pricing. *Transportation Research Part E: Logistics and Transportation Review*, 77:95–114, 2015.

D.J. Forkenbrock. External costs of intercity truck freight transportation. *Transportation Research Part A*, 33 (7-8):505–526, 1999.

D.J. Forkenbrock. Comparison of external costs of rail and truck freight transportation. *Transportation Research Part A*, 35(4):321–337, 2001.

A. Franceschetti, D. Honhon, T. Van Woensel, T. Bektaş, and G. Laporte. The time-dependent pollution routing problem. *Transportation Research Part B*, 56:265–293, 2013.

V. C. Hemmelmayr, J.-F. Cordeau, and T. G. Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, 39(12):3215–3228, 2012.

O. Jabali, T. Van Woensel, and A.G. de Kok. Analysis of travel times and CO_2 emissions in time-dependent vehicle routing. *Production and Operations Management*, 21(6):1060–1074, 2012.

Ç. Koç, T. Bektaş, O. Jabali, and G. Laporte. The fleet size and mix pollution-routing problem. *Transportation Research Part B: Methodological*, 70:239–254, 2014.

R. Kramer, N. Maculan, A. Subramanian, and T. Vidal. A speed and departure time optimization algorithm for the pollution-routing problem. *European Journal of Operational Research*, 247(3):782–787, 2015a.

R. Kramer, A. Subramanian, T. Vidal, and L.D.A.F. Cabral. A matheuristic approach for the pollution-routing problem. *European Journal of Operational Research*, 243(2):523–539, 2015b.

J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34 (8):2403–2435, 2007.

J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

D. Rey and M. Neuhäuser. Wilcoxon-signed-rank test. In *International Encyclopedia of Statistical Science*, pages 1658–1659. Springer, 2011.

G.M. Ribeiro and G. Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 39(3):728–735, 2012.

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

S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006b.

G. Scora and M. Barth. Comprehensive modal emission model (CMEM), version 3.01, user's guide. Technical report, 2006. Available at: http://www.cert.ucr.edu/cmem/docs/CMEM_User_Guide_v3.01d.pdf (accessed on June 1, 2015).

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Lecture Notes in Computer Science*, volume 1520, pages 417–431, Springer, Berlin, 1998.

M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.