European Journal of Operational Research 223 (2012) 346–359

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

An adaptive large neighborhood search heuristic for the Pollution-Routing Problem

Emrah Demir [a], Tolga Bektaş [a,*], Gilbert Laporte [b]

[a] School of Management and Centre for Operational Research, Management Science and Information Systems (CORMSIS), University of Southampton, Southampton, Highfield SO17 1BJ, United Kingdom
[b] Canada Research Chair in Distribution Management and Interuniversity Research Centre on Enterprise Networks, Logistics, and Transportation (CIRRELT), HEC Montréal 3000, Chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7

ARTICLE INFO

Article history:
Received 19 October 2011
Accepted 30 June 2012
Available online 7 July 2012

Keywords:
Vehicle routing
Fuel consumption
CO_2 emissions
Freight transportation
Heuristic algorithm

ABSTRACT

The Pollution-Routing Problem (PRP) is a recently introduced extension of the classical Vehicle Routing Problem with Time Windows which consists of routing a number of vehicles to serve a set of customers, and determining their speed on each route segment so as to minimize a function comprising fuel, emission and driver costs. This paper presents an adaptive large neighborhood search for the PRP. Results of extensive computational experimentation confirm the efficiency of the algorithm.

1. Introduction

The road transportation sector is a significant emitter of carbon dioxide (CO_2), the amount of which is directly proportional to fuel consumption (Kirby et al., 2000). Fuel consumption is dependent on a variety of parameters, such as vehicle speed, load and acceleration (Demir et al., 2011). The Pollution-Routing Problem (PRP) is an extension of the classical Vehicle Routing Problem with Time Windows (VRPTWs). It consists of routing a number of vehicles to serve a set of customers within preset time windows, and determining their speed on each route segment, so as to minimize a function comprising fuel, emission and driver costs. The PRP was introduced by Bektaş and Laporte (2011) who proposed a non-linear mixed integer mathematical model for the problem, which could be linearized. However, solving even medium scale PRP instances to optimality using such a model remains a challenge.

In this paper, we propose an extended Adaptive Large Neighborhood Search (ALNS) algorithm for the PRP. The algorithm integrates the classical ALNS scheme (Pisinger and Ropke, 2005, 2007; Ropke and Pisinger, 2006a) with a specialized speed optimization algorithm which computes optimal speeds on a given path so as to minimize fuel consumption, emissions and driver costs. The latter algorithm can also be used as a stand-alone routine to optimize speeds for the VRPTW. The remainder of this paper is organized as follows. In Section 2, we present the Pollution-Routing Problem. Section 3 describes a new iterative heuristic algorithm for the PRP. Section 4 presents the results of extensive numerical experiments. Conclusions are stated in Section 5.

2. Mathematical model for the Pollution-Routing Problem

We first formulate the PRP and discuss the extensions of the problem.

2.1. Description of the Pollution-Routing Problem

The PRP is defined on a complete directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{0, \ldots, n\}$ is the set of nodes, 0 is a depot and $\mathcal{A} = \{(i,j) : i,j \in \mathcal{N} \text{ and } i \neq j\}$ is the set of arcs. The distance from i to j is denoted by d_{ij}. A fixed-size fleet of vehicles denoted by the set $\mathcal{K} = \{1, \ldots, m\}$ is available, and each vehicle has capacity Q. The set $\mathcal{N}_0 = \mathcal{N} \setminus \{0\}$ is a customer set, and each customer $i \in \mathcal{N}_0$ has a non-negative demand q_i as well as a time interval $[a_i, b_i]$ for service. Early arrivals are permitted but the vehicle has to wait until time a_i before service can start. The service time of customer i is denoted by t_i.

2.2. Fuel and CO_2 emissions

The PRP is based on the *comprehensive emissions model* described by Barth et al. (2005), Scora and Barth (2006), and Barth and Boriboonsomsin (2008), which is an instantaneous model

* Corresponding author. Tel.: +44 02380598969.
E-mail addresses: E.Demir@soton.ac.uk (E. Demir), T.Bektas@soton.ac.uk (T. Bektaş), Gilbert.Laporte@cirrelt.ca (G. Laporte).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.06.044

estimating fuel consumption for a given time instant. According to this model, the fuel rate is given by

$$FR = \xi(kNV + P/\eta)/\kappa, \quad (1)$$

where ξ is fuel-to-air mass ratio, k is the engine friction factor, N is the engine speed, V is the engine displacement, and η and κ are constants. P is the second-by-second engine power output (in kilowatt), and can be calculated as

$$P = P_{tract}/\eta_{tf} + P_{acc}, \quad (2)$$

where η_{tf} is the vehicle drive train efficiency, and P_{acc} is the engine power demand associated with running losses of the engine and the operation of vehicle accessories such as air conditioning. P_{acc} is assumed to be zero. The parameter P_{tract} is the total tractive power requirements (in kilowatt) placed on the wheels:

$$P_{tract} = (M\tau + Mg\sin\theta + 0.5C_d\rho A v^2 + MgC_r\cos\theta)v/1000, \quad (3)$$

where M is the total vehicle weight (kilogram), v is the vehicle speed (meter/second), τ is the acceleration (meter/second^2), θ is the road angle, g is the gravitational constant, and C_d and C_r are the coefficient of the aerodynamic drag and rolling resistance, respectively. Finally, ρ is the air density and A is the frontal surface area of the vehicle.

For a given arc (i, j) of length d, let v be the speed of a vehicle speed traversing this arc. If all variables in FR except for the vehicle speed v remain constant on arc (i, j), the fuel consumption (in L) on this arc can be calculated as

$$F(v) = kNV\lambda d/v \quad (4)$$
$$+ P\lambda\gamma d/v, \quad (5)$$

where $\lambda = \xi/\kappa\psi$ and $\gamma = 1/1000\eta_{tf}\eta$ are constants and $\psi$is the conversion factor of fuel from gram/second to liter/second. Furthermore, let M be the load carried between nodes i and j. More specifically, $M = w + f$, where w is the curb weight (i.e., the weight of an empty vehicle) and f is the vehicle load. Let $\alpha = \tau + g\sin\theta + gC_r\cos\theta$ be a vehicle-arc specific constant and $\beta = 0.5C_d\rho A$ be a vehicle-specific constant. We omit the indices (i, j) on the variables v, d, f, and α to simplify the presentation. Then, $F(v)$ can be rewritten as

$$F(v) = \lambda(kNV + w\gamma\alpha v + \gamma\alpha f v + \beta\gamma v^3)d/v. \quad (6)$$

All other parameters and values are given in Table 1. The cost of fuel and CO_2 emissions per second can be calculated as $f_c FR/\psi$, where f_c is the unit cost of fuel and CO_2 emissions. Applying Eq. (6) to a low-duty vehicle for speeds varying from 20 kilometer/hour to 110 kilometer/hour for a road segment of $d = 100$ kilometer yields the fuel consumption curve shown in Fig. 1. The function depicted in Fig. 1 is the sum of two components, one induced by (4) and shown by the dashed line, and the other by (5) shown by the dotted line. One can see that the contribution of the first component of the function, namely kNV, will only be significant for low speed levels (less than 40 kilometer/hour), whereas that of P_{tract} is significant for higher speed levels. In the PRP model, Bektaş and Laporte (2011) consider speeds of 40 kilometer/hour and higher for which they only make use of P_{tract}. In this work, we will allow for lower speeds which yield higher fuel consumptions. This is accounted for by the kNV component of Eq. (1) which we will incorporate in our model.

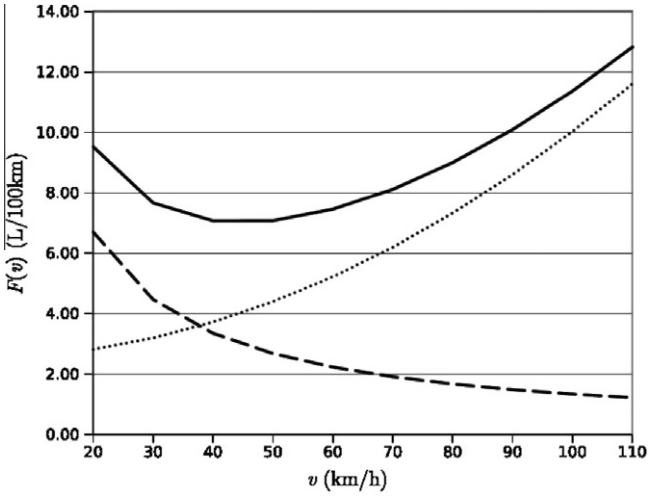

Fig. 1. Fuel consumption as a function of speed (Bektaş and Laporte, 2011).

One relevant study to the one considered here is by Jabali et al. (forthcoming), who describe a VRP that considers travel time, fuel, and CO_2 emissions costs in a time-dependent context, where the latter are estimated using emission functions provided in the MEET report (Hickman et al., 1999). The authors describe a Tabu Search algorithm to solve the problem and show, through computational experiments, that limiting vehicle speeds is effective in reducing emissions to a certain extent although costly in terms of total travel time.

Table 1
Parameters used in the PRP model.

Notation	Description	Typical values
w	Curb-weight (kilogram)	6350
ξ	Fuel-to-air mass ratio	1
k	Engine friction factor (kilojoule/rev/liter)	0.2
N	Engine speed (rev/second)	33
V	Engine displacement (liters)	5
g	Gravitational constant (meter/second^2)	9.81
C_d	Coefficient of aerodynamic drag	0.7
ρ	Air density (kilogram/meter^3)	1.2041
A	Frontal surface area (meter^2)	3.912
C_r	Coefficient of rolling resistance	0.01
n_{tf}	Vehicle drive train efficiency	0.4
η	Efficiency parameter for diesel engines	0.9
f_c	Fuel and CO_2 emissions cost per liter (£)	1.4
f_d	Driver wage per (£/second)	0.0022
κ	Heating value of a typical diesel fuel (kilojoule/gram)	44
ψ	Conversion factor (gram/second to liter/second)	737
v^l	Lower speed limit (meter/second)	5.5 (or 20 kilometer/hour)
v^u	Upper speed limit (meter/second)	25 (or 90 kilometer/hour)

2.3. Integer programming formulation

We now present the integer programming formulation for the PRP. The model works with a discretized speed function defined by R non-decreasing speed levels $\bar{v}^r$ $(r = 1, \ldots, R)$. Binary variables x_{ij} are equal to 1 if and only if arc (i, j) appears in solution. Continuous variables f_{ij} represent the total amount of flow on each arc $(i,j) \in \mathcal{A}$. Continuous variables y_j represent the time at which service starts at node $j \in \mathcal{N}_0$. Moreover, s_j represents the total time spent on a route that has a node $j \in \mathcal{N}_0$ as last visited before returning to the depot. Finally, binary variables z_{ij}^r indicate whether or not arc $(i,j) \in \mathcal{A}$ is traversed at a speed level r. An integer linear programming formulation of the PRP is shown below:

$$\text{Minimize} \sum_{(i,j)\in\mathcal{A}} kNV\lambda d_{ij} \sum_{r=1}^{R} z_{ij}^r / \bar{v}^r \tag{7}$$

$$+ \sum_{(i,j)\in\mathcal{A}} w\gamma\lambda\alpha_{ij} d_{ij} x_{ij} \tag{8}$$

$$+ \sum_{(i,j)\in\mathcal{A}} \gamma\lambda\alpha_{ij} d_{ij} f_{ij} \tag{9}$$

$$+ \sum_{(i,j)\in\mathcal{A}} \beta\gamma\lambda d_{ij} \sum_{r=1}^{R} z_{ij}^r (\bar{v}^r)^2 \tag{10}$$

$$+ \sum_{j\in\mathcal{N}_0} f_d s_j \tag{11}$$

subject to

$$\sum_{j\in\mathcal{N}} x_{0j} = m \tag{12}$$

$$\sum_{j\in\mathcal{N}} x_{ij} = 1 \quad \forall i \in \mathcal{N}_0 \tag{13}$$

$$\sum_{i\in\mathcal{N}} x_{ij} = 1 \quad \forall j \in \mathcal{N}_0 \tag{14}$$

$$\sum_{j\in\mathcal{N}} f_{ji} - \sum_{j\in\mathcal{N}} f_{ij} = q_i \quad \forall i \in \mathcal{N}_0 \tag{15}$$

$$q_j x_{ij} \leqslant f_{ij} \leqslant (Q - q_i) x_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{16}$$

$$y_i - y_j + t_i + \sum_{r\in\mathcal{R}} d_{ij} z_{ij}^r / \bar{v}^r \leqslant K_{ij}(1 - x_{ij}) \quad \forall i \in \mathcal{N}, \; j \in \mathcal{N}_0, \; i \neq j \tag{17}$$

$$a_i \leqslant y_i \leqslant b_i \quad \forall i \in \mathcal{N}_0 \tag{18}$$

$$y_j + t_j - s_j + \sum_{r\in\mathcal{R}} d_{j0} z_{j0}^r / \bar{v}^r \leqslant L(1 - x_{j0}) \quad \forall j \in \mathcal{N}_0 \tag{19}$$

$$\sum_{r=1}^{R} z_{ij}^r = x_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{20}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A} \tag{21}$$

$$f_{ij} \geqslant 0 \quad \forall (i,j) \in \mathcal{A} \tag{22}$$

$$y_i \geqslant 0 \quad \forall i \in \mathcal{N}_0 \tag{23}$$

$$z_{ij}^r \in \{0,1\} \quad \forall (i,j) \in \mathcal{A}, \; r = 1, \ldots, R. \tag{24}$$

This mathematical formulation of the PRP presented here is an extension of the one presented in Bektaş and Laporte (2011) to take into account for speeds 40 kilometer/hour or lower through the term (7) of the objective function. The objective function (7)–(10) is derived from (6). The terms (8) and (9) calculate the cost incurred by the vehicle curb weight and payload. Finally, the term (11) measures the total driver wages.

Constraints (12) state that each vehicle must leave the the depot. Constraints (13) and (14) are the degree constraints which ensure that each customer is visited exactly once. Constraints (15) and (16) define the arc flows. Constraints (17)–(19), where $K_{ij} = \max\{0, \; b_i + s_i + d_{ij}/l_{ij} - a_j\}$, and L is a large number, enforce the time window restrictions. Constraints (20) ensure that only one speed level is selected for each arc and $z_{ij}^r = 1$ if $x_{ij} = 1$.

The PRP is NP-hard since it is an extension of the classical Vehicle Routing Problem (VRP). Bektaş and Laporte (2011) have shown that a simplified version of this problem cannot be solved optimally for mid-size instances. For this reason, we have developed a heuristic to obtain good-quality solutions within short computational times.

3. An adaptive large neighborhood heuristic algorithm for the PRP

Our heuristic operates in two stages. In the first stage, it solves a VRPTW using an ALNS heuristic with operators partly borrowed from the literature. This metaheuristic is an extension of the Large Neighborhood Search (LNS) heuristic first proposed by Shaw (1998), and based on the idea of gradually improving an initial solution by using both destroy and repair operators. In other words, LNS consists of a series of removal and insertion moves. If the new solution is better than the current best solution, it replaces it and use as an input to the next iteration. The LNS heuristic can be embedded within any local search heuristic such as simulated annealing or tabu search.

In the second stage, a speed optimization algorithm (SOA) is run on the resulting VRPTW solution. Given a vehicle route, the SOA consists of finding the optimal speed on each arc of the route in order to minimize an objective function comprising fuel consumption costs and driver wages.

The proposed algorithm is designed as an iterative process whereby the ALNS uses fixed speeds as inputs to the VRPTW, following which the SOA is run on each route to improve the solution.

3.1. Adaptive large neighborhood search

The ALNS heuristic framework was put forward by Pisinger and Ropke (2005), Ropke and Pisinger (2006a), Pisinger and Ropke (2007) to solve variants of the vehicle routing problem. Rather than using one large neighborhood as in LNS, it applies several removal and insertion operators to a given solution. The neighborhood of a solution is obtained by removing some customers from the solution and reinserting them as in Milthers (2009). The removal and insertion operators are selected dynamically according to their past performance. To this end, each operator is assigned a *score* which is increased whenever it improves the current solution. The new solution is accepted if it satisfies some criteria defined by the local search framework (e.g., simulated annealing) applied at the outer level. The main features of the ALNS algorithm will be described in detail below.

3.1.1. Initialization

Several heuristic methods can be used to quickly obtain a feasible solution for the VRP. Cordeau et al. (2002) have analyzed and reviewed some of the classical heuristics based on four different criteria: accuracy, speed, simplicity, and flexibility. This comparison shows that the Clarke and Wright (1964) heuristic has the distinct advantage of being very quick and simple to implement. We have therefore used it in our algorithm to obtain an initial solution. It is noteworthy that while additional constraints incorporated into the CW heuristic "usually results in a sharp deterioration" (Cordeau et al., 2002), the quality of the initial solution is not so important since as a rule ALNS can easily recover from a poor initial solution. This algorithm was implemented while maintaining the feasibility of capacity and time window constraints.

3.1.2. Adaptive weight adjustment procedure

The selection of the removal and insertion operators is regulated by a roulette-wheel mechanism. Initially, all removal or

insertion operators are equally likely. Thus, for the twelve removal and five insertion operators, the probabilities are initially set to 1/12 and 1/5, respectively. During the algorithm, they are updated as $P_d^{t+1} = P_d^t(1 - r_p) + r_p\pi_i/\omega_i$, where r_p is the roulette wheel parameter, π_i is the score of operator i and ω_i is the number of times it was used during the last N_w iterations. The score of each operator measures how well the operator has performed at each iteration. If a new best solution is found, the score of an operator is increased by σ_1. If the solution is better than the current solution, the score is increased by σ_2. If the solution is worse than the current solution but accepted, the score is increased by σ_3.

3.1.3. Removal operators

We now present the twelve removal operators used in our algorithm. The first nine are either adapted or inspired by Pisinger and Ropke (2005), Ropke and Pisinger (2006a,b), Pisinger and Ropke (2007) and Shaw (1998), whereas the last three are new. The destroy phase mainly consists of removing s customers from the current solution and adding them into a *removal list* $\mathcal{L}$ as illustrated in Fig. 2.

A pseudo-code of the generic removal procedure is presented in Algorithm 1. The algorithm is initialized with a feasible solution X as input and returns a partially destroyed solution. The parameter ϕ defines the number of iterations of the search. In Algorithm 1, a chosen operator is used to remove a subset S of nodes from the solution. These nodes are inserted in a removal list $\mathcal{L}$. The algorithm iterates in a similar fashion for ϕ iterations.

Algorithm 1: The generic structure of the removal procedure

input : A feasible solution X, and maximal number of iterations ϕ
output: A partially destroyed solution X_p

```
Initialize removal list (L ← ∅)
for φ iterations do
    Apply remove operator to find the set S of nodes for removal
    L ← L ∪ S
    Remove the subset S of nodes from X
```

We now describe the removal operators used in our implementation:

1. **Random removal (RR)**: This operator starts with an empty removal list. It randomly removes s nodes from the solution, and runs for $\phi = s$ iterations. The idea of randomly selecting nodes helps diversify the search mechanism. The worst-case time complexity of the RR operator is $O(n)$.
2. **Worst-distance removal (WDR)**: This operator iteratively removes high cost customers, where the cost is defined as the sum of distances from the preceding and following customer on the tour, i.e., it removes node $j^* = \text{argmax}_{j\in\mathcal{N}}\{|d_{ij} + d_{jk}|\}$. The worst-case time complexity of the WDR operator is $O(n^2)$.
3. **Worst-time removal (WTR)**: This operator calculates, for each node j, the deviation of service start time from time a_j, and then removes the node with the largest deviation. The idea is to prevent long waits or delayed service start times. The algorithm starts with an empty removal list, and runs for $\phi = s^2$ iterations (for $i = 1, \ldots, s; j = 1, \ldots, s$). The operator selects $j^* = \text{argmax}_{j\in\mathcal{N}}\{|y_j - a_j|\}$ where y_j is the time at which service begins at node j. The solution is updated after each node removal. The worst-case time complexity of the WTR operator is $O(n^2)$.
4. **Route removal (RoR)**: This operator removes a full route from the solution. It randomly selects a route from the set of routes in the solution. The remove operator then repeatedly selects a node j from this route until all nodes are removed. The RoR operator can be implemented in $O(1)$ worst-case time.
5. **Shaw removal (SR)**: The aim of this operator is to remove a set of customers that are related in a predefined way and therefore are easy to change. The logic behind the operator was introduced by Shaw (1998). The algorithm starts by randomly selecting a node i and adds it to the removal list. Let $l_{ij} = -1$ if $i \in \mathcal{N}$ and $j \in \mathcal{N}$ are in the same route, and 1 otherwise. The operator selects a node $j^* = \text{argmin}_{j\in\mathcal{N}}\{\Phi_1 d_{ij} + \Phi_2|a_i - a_j| + \Phi_3 l_{ij} + \Phi_4|q_i - q_j|\}$, where $\Phi_1 - \Phi_4$ are weights which are normalized to find the best candidate from solution. The operator is applied $\phi = s^2$ times by selecting a node not yet in the removal list which is most similar to the one last added to the list. The worst-case time complexity of the SR operator is $O(n^2)$.
6. **Proximity-based removal (PR)**: The operator removes a set of nodes that are related in terms of distance. This operator is a special case of the Shaw removal operator with $\Phi_1 = 1$, and $\Phi_2 = \Phi_3 = \Phi_4 = 0$. The way the operator works is graph-

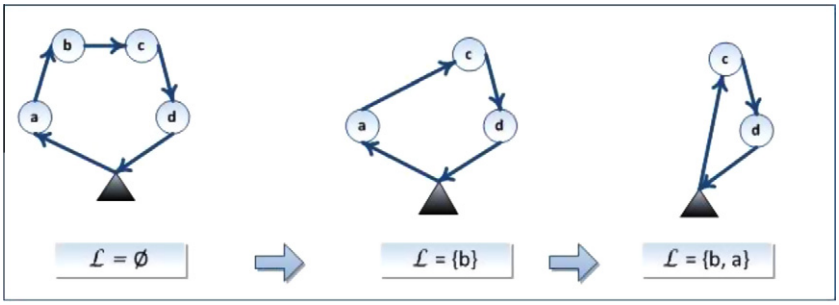

Fig. 2. Depiction of a removal operator.

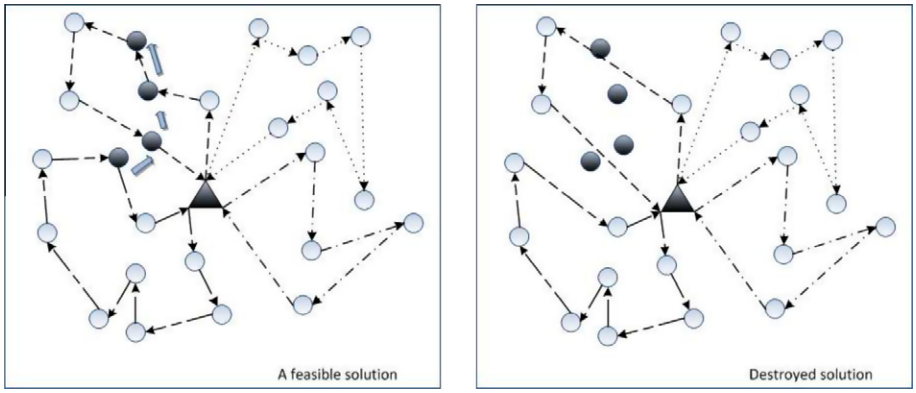

Fig. 3. Proximity-based removal operator.

ically illustrated in Fig. 3. The worst-case time complexity of the PR operator is $O(n^2)$.

7. **Time-based removal (TR)**: The operator is a special case of the Shaw removal with $\Phi_2 = 1$, and $\Phi_1 = \Phi_3 = \Phi_4 = 0$. The worst-case time complexity of the TR operator is $O(n^2)$.
8. **Demand-based removal (DR)**: This operator is a special case of the Shaw removal with $\Phi_4 = 1$, and $\Phi_1 = \Phi_2 = \Phi_3 = 0$. The worst-case time complexity of the DR operator is $O(n^2)$.
9. **Historical knowledge node removal (HR)**: The HR operator is similar to the neighbor graph removal operator used in Ropke and Pisinger (2006b). This operator keeps a record of the position cost of every node i, defined as the sum of the distances between its preceding and following nodes, and calculated as $d_i = d_{i-1,i} + d_{i,i+1}$ at every iteration of the algorithm. At any point in the algorithm, the best position cost d_i^* of node i is updated to be the minimum of all d_i values calculated until that point. The HR operator then picks a node j^* on a route with maximum deviation from its best position cost, i.e., $j^* = \text{argmax}_{j \in \mathcal{N}_0}\{d_j - d_j^*\}$. Node j^* is then added to the removal list. The worst-case time complexity of the HR operator is $O(n)$.
10. **Neighborhood removal (NR)**: This operator is based on the idea of removing nodes from routes which are extreme with respect to the average distance of a route. More specifically, in a given solution with a set of routes $\mathcal{B}$, the operator calculates, for each route $B = \{i_1, \ldots, i_{|B|}\}$ in $\mathcal{B}$ an average distance as $\bar{d}_B = \sum_{(i_1,i_2)\in B} d_{i_1 i_2}/|B|$ and selects a node $j^* = \text{argmax}_{B\in\mathcal{B}; j\in B}\{\bar{d}_B - \bar{d}_{B\setminus\{j\}}\}$. The worst-case time complexity of the NR operator is $O(n^2)$.
11. **Zone removal (ZR)**: The zone removal operator is based on removal of nodes in a predefined area in the Cartesian coordinate system in which nodes are located. The operator first compute the corner points of the area. The whole region is then split up into smaller areas. An area is randomly selected and all its node are removed. The removal operator selects $S = \{j^* | x(i^1) \leqslant x(j^*) \leqslant x(i^2)$ and $y(i^1) \leqslant y(j^*) \leqslant y(i^2)\}$, where $(x(i^1), x(i^2))$ are the x-coordinates of the selected zone i, and $(y(i^1), y(i^2))$ are the y-coordinates of the selected zone i. If the area does not contain any node, a new area is randomly selected and the process continues until s nodes are removed. The worst-case time complexity of the ZR operator is $O(n^2)$, although after an initial preprocessing of all areas, the worst-case time complexity can be reduced to $O(n)$.
12. **Node neighborhood removal (NNR)**: This operator initially selects a random node and then removes $s - 1$ nodes around it encapsulated in a rectangular area around the selected node. The choice of the rectangular neighborhood, as supposed to, say, a circular neighborhood, is motivated by the presence of grid-shaped networks encountered in several real-world settings as illustrated in Fig. 4. If the number of nodes within the selected zone is less than $s - 1$, then the area is enlarged by a given percentage. The worst-case time complexity of the NNR operator is $O(n^2)$.

3.1.4. Insertion operators

In this section, we present five insertion operators used in the ALNS algorithm. The first four of these operators are adapted from Ropke and Pisinger (2006a) whereas the last one is new. Insertion operators are used to repair a partially destroyed solution by inserting the nodes in the removal list back into the solution. These operators insert the removed nodes back into the existing routes when feasibility with respect to the capacity and time windows can be maintained, or they create new routes. We now briefly define the five insertion operators used in the main algorithm.

1. **Greedy insertion (GI)**: This operator repeatedly inserts a node in the best possible position of a route. The insertion cost is calculated as $d_i = d_{ji} + d_{ik} - d_{jk}$ for $j = 1, \ldots, s$ and $i = 1, \ldots, n$. The operator iterates $\phi = sn$ times. $S = \{j^*\}$ is selected such that $j^* = \text{argmin}_{B\in\mathcal{B}; j\in B}\{|d_i|\}$. The worst-case time complexity of the GI operator is $O(n^2)$.
2. **Regret insertion (RI)**: One problem with the greedy insertion operator is that it often postpones the insertion of the nodes until the later iterations when few feasible moves are possible. To counter this problem, this operator uses a 2-regret criterion. Let Δf_i denote the change in the objective value by inserting node i into its best and second best position for node i with respect to distance d_i as defined above. Let $i^* = \text{argmax}_{i\in\mathcal{L}}\{\Delta f_{i2} - \Delta f_{i1}\}$, where Δf_{i1} is the best feasible reinsertion and Δf_{i2} is the second best reinsertion of node i. For each node in the removal list, the operator is applied up to $\phi = s^2 n$ times. This operator is quite time consuming, but unnecessary computations can be avoided when computing Δf_i. The worst-case time complexity of the RI operator is $O(n^3)$.
3. **Greedy insertion with noise function (GIN)**: This operator is an extension of the greedy algorithm but uses a degree of freedom in selecting the best place for a node. This degree of freedom is achieved by modifying the cost for node i: $NewCost = ActualCost + \bar{d}\mu\epsilon$ where $\bar{d}$ is the maximum distance between nodes, μ is a noise parameter used for diversification and is set equal to 0.1, and ϵ is a random number between $[-1, 1]$. *New Cost* is calculated for each node in $\mathcal{L}$. The worst-case time complexity of the GIN operator is $O(n^2)$.

4. **Regret insertion with noise function (RIN)**: This operator is an extension of the 2-regret insertion algorithm but uses the same noise function as the GIN operator. The worst-case time complexity of the RIN operator is $O(n^3)$.
5. **Zone Insertion (ZI)**: This operator is similar to basic insertion but uses the time windows rather than distance to determine *best insertion* of each node. The zone algorithm determines the best position for each node and searches for another solution around it, feasible for the time window constraint. In other words, this operator tries to identify insertions that leave enough margin for future insertions. The worst-case time complexity of the ZI operator is $O(n^2)$.

ing as a local search framework is presented in Algorithm 2. In the algorithm, X_{best} shows the best solution found during the search, $X_{current}$ is the current solution obtained at the beginning of an iteration, and X_{new} is a temporary solution found at the end of iteration that can be discarded or become the current solution. The cost of solution X is denoted by $c(X)$. A solution X_{new} is always accepted if $c(X_{new}) < c(X_{current})$, and accepted with probability $e^{-(c(X_{new})-c(X_{current}))/T}$ if $c(X_{new}) > c(X_{current})$, where T is the *temperature*. The temperature is initially set at $c(X_{init})P_{init}$ where $c(X_{init})$ is the objective function value of the initial solution X_{init} and P_{init} is an initialization constant. The current temperature is gradually decreased during the course of the

Algorithm 2: The general framework of the ALNS with simulated annealing

input : a set of removal operators D, a set of insertion operators I, initialization constant P_{init}, cooling rate h

output: X_{best}

Generate an initial solution by using the Clarke and Wright algorithm
Initialize probability P^t_d for each destroy operator $d \in D$ and probability P^t_i for each insertion operator $i \in I$
Let T be the temperature and j be the counter initialized as $j \leftarrow 1$
Let $X_{current} \leftarrow X_{best} \leftarrow X_{init}$
repeat
  Select a removal operator $d^* \in D$ with probability P^t_d
  Let X^*_{new} be the solution obtained by applying operator d^* to $X_{current}$
  Select an insertion operator $i^* \in I$ with probability P^t_i
  Let X_{new} be the new solution obtained by applying operator i^* to X^*_{new}
  if $c(X_{new}) < c(X_{current})$ **then**
    $X_{current} \leftarrow X_{new}$
  else
    Let $\nu \leftarrow e^{-(c(X_{new})-c(X_{current}))/T}$
  Generate a random number $\epsilon \in [0, 1]$
  if $\epsilon < \nu$ **then**
    $X_{current} \leftarrow X_{new}$
  if $c(X_{current}) < c(X_{best})$ **then**
    $X_{best} \leftarrow X_{new}$
  $T \leftarrow hT$
  Update probabilities using the adaptive weight adjustment procedure
  $j \leftarrow j + 1$
until *the maximum number of iterations is reached*

3.1.5. Acceptance and stopping criteria

Simulated annealing was used as a local search framework for our ALNS algorithm. The ALNS algorithm with simulated annealing algorithm as hT, where $0 < h < 1$ is a fixed parameter. The algorithm returns the best found solution after a fixed number of iterations.

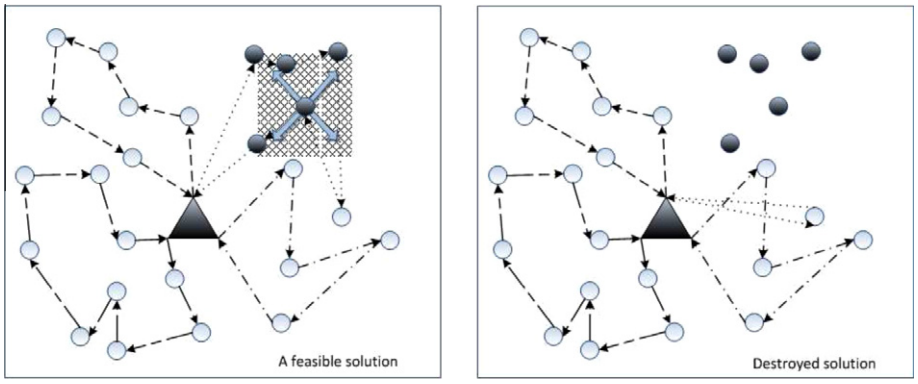

Fig. 4. Node neighborhood removal operator.

3.2. Speed optimization

In this section, we introduce and analyze the speed optimization problem (SOP). Given a vehicle route, the SOP consists of finding the optimal speed on each arc of the route between successive nodes so as to minimize an objective function comprising fuel consumption costs and driver wages. The objective of SOP is non-linear due to the function used to estimate fuel consumption of a vehicle, which is quadratic in speed. Fixing all vehicle speeds at their optimal values with respect to fuel consumption may lead to violations of time window constraints. Furthermore, driver costs and fuel consumption do not always move in the same direction (see Bektaş and Laporte, 2011). This makes the SOP a non-trivial problem.

3.2.1. Mathematical model

The SOP is defined on a feasible path $(0,\ldots,n+1)$ of nodes all served by a single vehicle, where 0 and $n+1$ are two copies of the depot. The model uses the variable w_i to denote the waiting time at each node i, the variable v_i to represent the speed at which a vehicle travels between nodes i and $i+1$, and the variable e_i for the arrival time at node i. The vehicle has a minimum and maximum speed, represented by v_i^l and v_i^u, between nodes i and $i+1$. The formulation of SOP is as follows:

$$\text{Minimize} \sum_{i=0}^{n} f_c F_i(v_i) + f_d e_{n+1} \tag{25}$$

subject to

$$e_{i+1} = e_i + w_i + t_i + d_i/v_i \quad i = 0,\ldots,n \tag{26}$$
$$a_i \leqslant e_i + w_i \leqslant b_i \quad i = 1,\ldots,n \tag{27}$$
$$v_i^l \leqslant v_i \leqslant v_i^u \quad i = 0,\ldots,n \tag{28}$$
$$w_i \geqslant 0 \quad i = 1,\ldots,n \tag{29}$$
$$e_i \geqslant 0 \quad i = 1,\ldots,n+1 \tag{30}$$
$$v_i \geqslant 0 \quad i = 1,\ldots,n \tag{31}$$
$$w_0 = e_0 = t_0 = 0, \tag{32}$$

where $F_i(v)$ is the total fuel consumption as derived in (6) but written using the load M_i, the acceleration τ_i and the road angle θ_i of arc $(i, i+1)$ for each $i = 0,\ldots,n$.

The objective function (25) minimizes the total cost of fuel consumption and driver wages. We recall that f_c is the fuel and CO_2 emissions cost per liter and f_d is the driver wage per second. Other parameters are as defined in Section 2.2. Constraints (26) ensure that the arrival time at node $i+1$ is the sum of the arrival time at node i, the waiting time at node i, the service time at node i and the travel time to node i. Constraints (27) guarantee that service at node i should start between a_i and b_i. Constraints (28) define upper and lower limits for speed. Constraints (29)–(31) impose the non-negativity restrictions on the variables.

We now describe a Speed Optimization Algorithm (SOA) for the SOP. This algorithm is an adapted version of that of Norstad et al. (2010) and Hvattum et al. (forthcoming), first proposed for ship routing. The algorithm is exact provided the cost function is convex. This is easily proved by adapting the arguments of Hvattum et al. (forthcoming) to our case.

At the beginning of the algorithm, the speeds v_i are calculated for each link by considering the beginning of the next time window and total available time. These speed values are used to find violations if any. The violation is calculated if the departure time is less than sum of a_i and t_i or arrival time to the node is greater than b_i. Otherwise, the violation is set to zero. If the minimal possible speed limit is greater than the optimal speed, the optimal speed is increased to the minimal speed. This will not violate the time window since increasing speed means that less time is spent on the arc, and this speed is feasible if the lower speed does not violate the time window. The optimal speeds and current speeds are then compared; if the current speed is less than the optimal value, it is replaced with the optimal value. The algorithm selects at each stage the arc with the largest time window violation and eliminates the violation. In order to apply our SOP algorithm, it remains to show that (25) is convex.

Proposition 1. *The objective function (25) is convex.*

Proof. Using Eqs. (26) and (32), the objective function (25) can be expanded as follows:

$$\sum_{i=0}^{n} f_c F_i(v) + f_d \left(\sum_{i=0}^{n} (w_i + t_i + d_i/v) \right) = \sum_{i=0}^{n} [f_c F_i(v) + f_d (w_i + t_i + d_i/v)].$$

Let $g_i(v) = f_c F_i(v)$ and $h_i(v) = f_d(w_i + t_i + d_i/v)$ for each $i = 0,\ldots,n$. Then, since $dg_i(v)/dv = -kNV\lambda d_i/v^2 + 2\beta\lambda\gamma v d_i$ and $d^2 g_i(v)/dv = 2kNV\lambda d_i/v^3 + 2\beta\gamma\lambda d_i > 0$, $g_i(v)$ is a convex function. Similarly, since $dh_i(v)/dv = -f_d d_i/v^2$ and $d^2 h_i(v)/dv = 2f_d d_i/v^3 > 0$, $h_i(v)$ is a convex function. Since the sum of convex functions is convex, the proof follows. □

Proposition 2. *Given an arc $(i, i+1)$ and other parameters as described in the Section 4.1, the optimal speed which minimizes fuel consumption costs and wage of driver is:*

$$v^* = \left(\frac{kNV}{2\beta\gamma} + \frac{f_d}{2\beta\lambda\gamma f_c} \right)^{1/3}, \tag{33}$$

and the optimal speed which minimizes fuel consumption costs is:

$$v^* = \left(\frac{kNV}{2\beta\gamma}\right)^{1/3}. \quad (34)$$

Proof. (33) follows from $d(g_i(v) + h_i(v))/dv = 0$ for each $i = 0, \ldots, n$. (34) follows from $dh_i(v)/dv = 0$ for each $i = 0, \ldots, n$. □

A pseudo-code of the SOA is shown in Algorithm 3. The SOA runs in two stages. In the first the stage, optimum speeds are calculated to minimize fuel, emission and driver costs. The first stage also calculates the minimal required travel time time of the depot, which is then set equal to upper bound of the time windows. In the second stage, speeds are revised to optimize fuel consumption. In Algorithm 3, the only difference between two stages is the optimal speed calculation in line 6, where optimal speeds are calculated using (33) for the first stage and using (34) for the second stage. The algorithm starts with a feasible route; it takes input parameters s and e, and returns speed optimized routes.

Algorithm 3: Speed Optimization Algorithm(s, e)

initialize: $violation \leftarrow 0$, $p \leftarrow 0$, $D \leftarrow \sum_{i=s}^{e-1} d_i$, $T \leftarrow \sum_{i=s}^{e} t_i$

1 **for** $i = s + 1$ **to** e **do**
2 **if** $\underline{e}_i \le a_i$ **then**
3 $v_{i-1} \leftarrow D/(\overline{e}_i - a_i - T)$
4 **else**
5 $v_{i-1} \leftarrow D/(\overline{e}_i - \underline{e}_i - T)$
6 $v^*_{i-1} \leftarrow$ *Optimal Speed*
7 **if** $\overline{e}_{i-1} + d_{i-1}/v_{i-1} < a_i$ *and* $\underline{e}_i \ge a_i + t_i$ *and* $i \ne |R| - 1$ **then**
8 $v_{i-1} \leftarrow d_{i-1}/(a_i - \overline{e}_{i-1})$
9
10 **else if** $\overline{e}_{i-1} + d_{i-1}/v_{i-1} < a_i$ *and* $\overline{e}_i \ge b_i + t_i$ *and* $i \ne |R| - 1$ **then**
11 $v_{i-1} \leftarrow d_{i-1}/(b_i - \overline{e}_{i-1})$
12
13 **if** $i = (N - 1)$ *and* $\overline{e}_i \ne \underline{e}_i$ **then**
14 $v_{i-1} \leftarrow d_{i-1}/(a_i - \overline{e}_{i-1})$
15 **if** $v^*_{i-1} < d_{i-1}/(b_{i+1} - a_i - t_i)$ **then**
16 $v^*_{i-1} \leftarrow d_{i-1}/(b_{i+1} - a_i - t_i)$
17 **if** $v^*_{i-1} > v_{i-1}$ **then**
18 $v_{i-1} \leftarrow v^*_{i-1}$
19 $\underline{e}_i \leftarrow \overline{e}_{i-1} + d_{i-1}/v_{i-1}$
20 **if** $i \ne e$ **then**
21 $\overline{e}_i \leftarrow \underline{e}_i + t_i$
22 $g_i \leftarrow \max\{0, \underline{e}_i - b_i, a_i + t_i - \overline{e}_i\}$
23 **if** $g_i > violation$ **then**
24 $violation \leftarrow g_i$
25 $p \leftarrow i$
26 **if** $violation > 0$ *and* $\underline{e}_p > b_p$ **then**
27 $\overline{e}_p \leftarrow b_p + t_p$
28 Speed Optimization Algorithm(s, p)
29 Speed Optimization Algorithm(p, e)
30 **if** $violation > 0$ *and* $\overline{e}_p < a_p + t_p$ **then**
31 $\overline{e}_p \leftarrow a_p + t_p$
32 Speed Optimization Algorithm(s, p)
33 Speed Optimization Algorithm(p, e)

4. Computational results

This section presents results of extensive computational experiments performed to assess the performance of our ALNS heuristic. We first describe the generation of the test instances and of the parameters. We then present the results.

4.1. Data and experimental setting

Nine sets of 20 instances each were generated. The size of the instances ranges from 10 to 200 nodes. The instances represent randomly selected cities from the UK and therefore use real geographical distances. Time windows and service times are randomly generated. All instances are available for download from http://www.apollo.management.soton.ac.uk/prplib.htm.

The proposed algorithm was implemented in C. All experiments were conducted on a server with 3 gigahertz speed and 1 gigabyte RAM. A preliminary analysis was conducted to fine-tune the parameters. No claim is made that our choice of parameter values is the best possible. However, the settings used generally worked well on our test problems. Our algorithm contains fifteen user controlled parameters which are shown in the Table 2.

The parameters used in the ALNS algorithm are grouped into three categories as described below.

1. The first group defines the selection procedure with the roulette wheel mechanism. We note that our setting of the parameters σ_1, σ_2 and σ_3 is contrary to the expected setting $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3$, normally used to reward an operator for good performance. In our implementation and similar to Pisinger and Ropke (2005), Pisinger and Ropke (2007), we have chosen an unconventional setting of these parameters whereby the discovery of a worse solution is rewarded more than the discovery of a better solution. This is to help diversify the search in the algorithm.
   To show the number of times each removal and insertion operator was called within the heuristic algorithm, we provide some information in Tables 3 and 4, respectively. These tables show, for each operator, the frequency of use in the algorithm as a percentage of the total number of iterations. The total time spent to run each operator is shown in parentheses. These results are obtained using one instance from each set.
   The results shown in Table 3 indicate that the frequency of using the different removal operators do not significantly vary from one another. As the instances get larger in size, the frequencies of SR and NR increase compared to other operators. We note, however, that the time consumed by SR and NR operators is significantly higher than other operators for instances with more than 75 nodes.
   As for the insertion operators, BGI is generally used slightly more than the other three as indicated by the results shown in Table 4. The times consumed by RI and RIN are significantly higher than those of the remaining operators.
2. The second group of parameters is used to calibrate the simulated annealing search framework and to define the initial temperature and cooling rate for the algorithm. Fig. 5 shows the behavior of the heuristic for a 100-node instance. The figure displays the way that best (X_{best}), current ($X_{current}$) and new (X_{new}) solutions change over 25000 iterations.
3. The third and last group of the parameters are specific to the way in which the removal or insertion operators work. Here, the most important parameter that significantly affects the solution quality is the allowable number of removable nodes as defined by a lower $\underline{s}$ and an upper $\bar{s}$ bound, calculated as a percentage of the total number of nodes in an instance. To give the reader an idea of the effect of these two parameters, we provide in Table 5 some statistics for varying values of $\underline{s}$ and $\bar{s}$ on a 100-node instance. The table shows the percent deviation of the

Table 2
Parameters used in the ALNS heuristic.

Group	Description	Typical values
I	Total number of iterations (N_i)	25000
	Number of iterations for roulette wheel (N_w)	450
	Roulette wheel parameter (r_p)	0.1
	New global solution (σ_1)	1
	Better solution (σ_2)	0
	Worse solution (σ_3)	5
II	Startup temperature parameter (P_{init})	100
	Cooling rate (h)	0.999
III	Lower limit of removable nodes ($\underline{s}$)	5–20% of $\lvert\mathcal{N}\rvert$
	Upper limit of removable nodes ($\bar{s}$)	12–30% of $\lvert\mathcal{N}\rvert$
	Zone parameter (z)	11
	First shaw parameter (Φ_1)	0.5
	Second shaw parameter (Φ_2)	0.25
	Third shaw parameter (Φ_3)	0.15
	Fourth shaw parameter (Φ_4)	0.25
	Noise parameter (μ)	0.1

Table 4
Number of iterations as a percentage of 25000 and the CPU times required by the insertion operators.

Instance sets	Insertion operators			
	GI	RI	GIN	RIN
10	32.1 (0.1)	28.4 (0.2)	16.8 (0.0)	22.7 (0.3)
15	22.6 (0.1)	27.0 (0.2)	21.2 (0.0)	29.2 (0.3)
20	27.0 (0.2)	28.9 (0.6)	19.0 (0.1)	25.0 (0.4)
25	31.0 (0.3)	33.5 (1.1)	12.9 (0.1)	22.6 (0.5)
50	27.6 (1.2)	29.5 (2.8)	20.1 (0.8)	22.8 (1.8)
75	30.3 (2.5)	30.9 (5.6)	18.3 (1.3)	20.5 (2.6)
100	28.5 (3.9)	28.4 (11.7)	19.2 (1.6)	24.0 (6.0)
150	27.8 (8.1)	25.6 (19.8)	26.4 (1.6)	20.1 (6.0)
200	26.2 (15.1)	23.4 (39.7)	25.0 (17.7)	25.3 (36.1)

Table 3
Number of iterations as a percentage of 25000 and the CPU times required by the removal operators.

Instance sets	Removal operators									
	RR	WDR	WTR	RoR	SR	PR	TR	DR	HR	NR
10	10.9 (0.0)	10.7 (0.0)	11.0 (0.0)	7.8 (0.0)	10.7 (0.0)	9.8 (0.0)	9.7 (0.0)	10.7 (0.0)	8.9 (0.0)	10.3 (0.0)
15	7.6 (0.0)	12.0 (0.0)	12.7 (0.0)	6.4 (0.0)	14.1 (0.0)	6.8 (0.0)	8.0 (0.0)	7.6 (0.0)	9.6 (0.0)	15.1 (0.0)
20	7.4 (0.0)	10.3 (0.0)	12.1 (0.0)	3.2 (0.0)	12.9 (0.0)	8.8 (0.0)	9.2 (0.0)	9.6 (0.0)	11.7 (0.0)	17.3 (0.1)
25	8.5 (0.0)	16.0 (0.0)	11.5 (0.0)	0.5 (0.0)	10.4 (0.1)	6.6 (0.0)	7.4 (0.0)	9.9 (0.0)	11.7 (0.0)	17.3 (0.1)
50	7.1 (0.0)	12.3 (0.1)	13.2 (0.1)	7.3 (0.0)	15.3 (0.4)	6.3 (0.1)	8.4 (0.0)	7.9 (0.0)	9.7 (0.0)	12.5 (0.4)
75	9.0 (0.0)	12.5 (0.1)	12.9 (0.2)	6.3 (0.0)	16.6 (1.0)	2.5 (0.0)	9.2 (0.0)	10.2 (0.0)	8.9 (0.0)	11.8 (1.3)
100	6.3 (0.1)	13.5 (0.2)	10.8 (0.2)	7.0 (0.0)	14.2 (3.0)	7.2 (0.1)	8.6 (0.1)	7.2 (0.1)	10.9 (0.0)	14.3 (2.9)
150	7.7 (0.0)	14.4 (0.3)	11.4 (0.4)	7.6 (0.0)	16.1 (6.2)	6.1 (0.1)	7.9 (0.2)	8.6 (0.2)	7.7 (0.1)	12.5 (6.7)
200	9.1 (0.1)	16.1 (0.5)	9.1 (0.5)	4.8 (0.1)	12.4 (17.5)	8.7 (0.3)	7.7 (0.3)	7.9 (0.2)	8.7 (0.1)	15.5 (17.6)

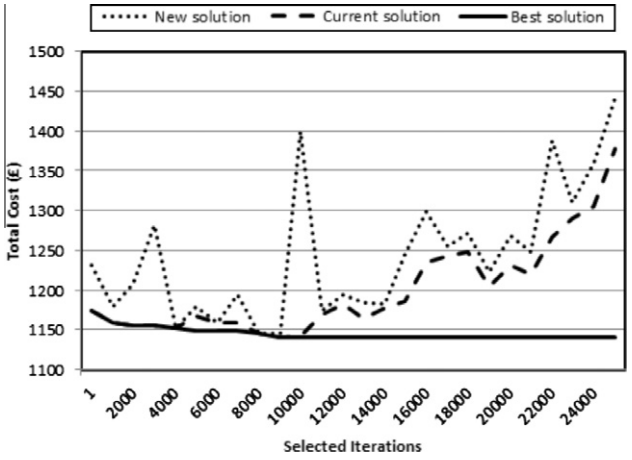

Fig. 5. Solution values obtained by ALNS for a 100-node instance.

Table 5
Effect of $\underline{s}$ and $\bar{s}$ on the quality of solutions obtained and measured as percentage deviation from the best known solution.

$\bar{s}$ \ $\underline{s}$	2	4	8	10	16
10	0.126	0.586	0.906	0.303	–
12	1.158	1.131	0.644	1.068	–
16	**0.000**	0.979	0.698	1.561	0.612
20	1.033	0.638	1.591	1.234	2.218
24	1.356	1.534	1.749	1.011	0.709
28	0.222	0.841	1.245	1.883	1.111
32	2.137	1.010	0.611	2.307	1.633
36	1.295	0.709	0.478	2.457	1.086
40	1.274	1.682	2.557	0.885	2.047
50	2.574	1.974	2.368	1.606	2.601
60	1.904	2.666	2.629	2.644	2.434
70	3.051	3.051	3.201	3.493	3.357

solutions found for each combination of $\underline{s}$ and $\bar{s}$ from the best known solution for this instance.

Other parameters in the third group include the following. The number of zones for zone removal and zone insertion operators is specified by z. Parameters $\Phi_1 - \Phi_4$ are specific to the Shaw removal operator as explained in Section 3.1.3. Finally, the last parameter (μ) is used to diversify the solution.

4.2. Results of the ALNS heuristic on the VRPTW

To help assess the quality of the ALNS heuristic, we present computational results in Tables 6–8 on Solomon's benchmark VRPTW instances, which come in three sets *r*, *c* and *rc* classified with respect to the geographical locations of the nodes. These tables compare the results of the original ALNS as reported in (Pisinger and Ropke, 2005) and denoted $ALNS_O$, to our ALNS heuristic, denoted $ALNS_I$. The extended version of the algorithm using the new operators is denoted $ALNS_E$. The comparisons are made in terms of best solution values obtained through 10 runs of each algorithm. These tables present, for each instance, the value of the best known or optimal solution compiled from several sources (e.g., Pisinger and Ropke, 2005; Milthers, 2009) under column "Best known value". All figures presented in Tables 6–8 use a single decimal place (Pisinger and Ropke, 2005; Milthers, 2009) as opposed to two decimal places (e.g., Pisinger and Ropke, 2007). For each variant of the algorithm, we then present the value of best solution obtained in column "Best value" and the corresponding average CPU time required to run the algorithm. As for the last three columns, the column titled Dev_{PR} (%) presents the percentage deviation of $ALNS_E$ from $ALNS_O$, the column titled Dev_I (%) shows the percentage deviation of $ALNS_I$ from those reported under "Best known value", and the column titled Dev_E (%) shows the percentage deviation of $ALNS_E$ from those reported under "Best known value". In particular, let $\upsilon(A)$ be the solution value produced by algorithm *A*. Then, Dev_{PR} (%) is calculated as 100 ($\upsilon(ALNS_O) - \upsilon(ALNS_E)$)/ $\upsilon(ALNS_O)$, Dev_I (%) is calculated as 100

Table 6
Results of the ALNS heuristic on benchmark VRPTW *r* instances.

Solomon instances	Best known value	$ALNS_O$		$ALNS_I$		$ALNS_E$		Dev_{PR} (%)	Dev_I (%)	Dev_E (%)
		Best value	CPU time s	Best value	CPU time s	Best value	CPU time s			
*r*101	1637.7	1637.7	30	1652.8	53	1637.7	48	0.00	−0.92	0.00
*r*102	1466.6	1467.7	33	1479.4	52	1466.6	46	0.07	−0.87	0.00
*r*103	1208.7	1208.7	34	1222.1	52	1208.7	46	0.00	−1.11	0.00
*r*104	971.5	976	34	985.3	51	971.5	45	0.46	−1.42	0.00
*r*105	1355.3	1355.3	31	1369.8	53	1355.3	47	0.00	−1.07	0.00
*r*106	1234.6	1234.6	33	1242.9	50	1234.6	46	0.00	−0.67	0.00
*r*107	1064.6	1064.6	33	1073.7	48	1064.6	42	0.00	−0.85	0.00
*r*108	932.1	933.7	36	939.5	52	936.1	44	−0.26	−0.26	−0.43
*r*109	1146.9	1146.9	31	1151.3	53	1146.9	46	0.00	−0.38	0.00
*r*110	1068	1075.6	33	1081.6	55	1073.9	49	0.16	−1.27	−0.55
*r*111	1048.7	1048.7	33	1057.3	48	1049.9	42	−0.11	−0.82	−0.11
*r*112	948.6	948.6	33	954.2	45	948.6	40	0.00	−0.59	0.00
Average								0.03	−0.85	−0.09
*r*201	1143.2	1148.5	45	1160.1	77	1143.2	71	0.46	−1.48	0.00
*r*202	1029.6	1036.9	54	1051.2	79	1032.2	72	0.45	−2.10	−0.25
*r*203	870.8	872.4	60	881	83	873.3	76	−0.10	−1.17	−0.29
*r*204	731.3	731.3	67	754.9	80	731.3	75	0.00	−3.23	0.00
*r*205	949.8	949.8	58	951.8	76	950.4	71	−0.06	−0.21	−0.06
*r*206	875.9	880.6	61	887.6	82	881	76	−0.05	−1.34	−0.58
*r*207	794	794	72	803.5	89	794	85	0.00	−1.20	0.00
*r*208	701.2	701.2	86	714.7	93	702.9	88	−0.24	−1.93	−0.24
*r*209	854.8	855.8	60	863.1	78	854.8	74	0.12	−0.97	0.00
*r*210	900.5	908.4	59	920.6	75	906.3	70	0.23	−2.23	−0.64
*r*211	746.7	752.3	67	769.2	80	751.6	74	0.09	−3.28	−0.66
Average								0.08	−1.74	−0.25

Table 7
Results of the ALNS heuristic on benchmark VRPTW *c* instances.

Solomon instances	Best known value	$ALNS_O$		$ALNS_I$		$ALNS_E$		Dev_{PR} (%)	Dev_I (%)	Dev_E (%)
		Best value	CPU time seconds	Best value	CPU time seconds	Best value	CPU time seconds			
*c*101	827.3	827.3	29	827.3	44	827.3	41	0.00	0.00	0.00
*c*02	827.3	827.3	32	827.3	43	827.3	40	0.00	0.00	0.00
*c*103	826.3	826.3	34	826.3	44	826.3	41	0.00	0.00	0.00
*c*104	822.9	822.9	36	822.9	42	822.9	40	0.00	0.00	0.00
*c*105	827.3	827.3	30	827.3	41	827.3	39	0.00	0.00	0.00
*c*106	827.3	827.3	31	827.3	45	827.3	41	0.00	0.00	0.00
*c*107	827.3	827.3	31	827.3	44	827.3	41	0.00	0.00	0.00
*c*108	827.3	827.3	32	827.3	44	827.3	42	0.00	0.00	0.00
*c*109	827.3	827.3	34	827.3	43	827.3	40	0.00	0.00	0.00
Average								0.00	0.00	0.00
*c*201	589.1	589.1	69	589.1	85	589.1	82	0.00	0.00	0.00
*c*202	589.1	589.1	74	589.1	89	589.1	85	0.00	0.00	0.00
*c*203	588.7	588.7	80	588.7	96	588.7	92	0.00	0.00	0.00
*c*204	588.1	588.1	84	588.1	98	588.1	91	0.00	0.00	0.00
*c*205	586.4	586.4	76	586.4	91	586.4	86	0.00	0.00	0.00
*c*206	586	586	72	586	86	586	81	0.00	0.00	0.00
*c*207	585.8	585.8	74	585.8	88	585.8	86	0.00	0.00	0.00
*c*208	585.8	585.8	74	585.8	94	585.8	88	0.00	0.00	0.00
Average								0.00	0.00	0.00

Table 8
Results of the ALNS heuristic on benchmark VRPTW *rc* instances.

Solomon instances	Best known value	$ALNS_O$		$ALNS_I$		$ALNS_E$		Dev_{PR} (%)	Dev_I (%)	Dev_E (%)
		Best value	CPU time seconds	Best value	CPU time seconds	Best value	CPU time seconds			
*rc*101	1619.8	1619.8	28	1623.7	48	1619.8	44	0.00	−0.24	0.00
*rc*102	1457.4	1463.5	30	1472.7	46	1463.5	42	0.00	−1.37	−0.42
*rc*103	1258	1267.0	31	1278.6	49	1267.1	43	−0.01	−1.64	−0.72
*rc*104	1132.3	1132.6	33	1139.8	48	1133.1	42	−0.04	−0.66	−0.07
*rc*105	1513.7	1513.7	30	1525.5	47	1513.7	43	0.01	−0.78	0.00
*rc*106	1372.7	1373.9	29	1381.4	44	1372.7	41	0.09	−0.63	0.00
*rc*107	1207.8	1209.3	30	1216.9	42	1209.3	40	0.00	−0.75	−0.12
*rc*108	1114.2	1114.2	31	1121.3	46	1114.2	43	0.00	−0.64	0.00
Average								0.01	−0.84	−0.17
*rc*201	1261.8	1262.6	42	1274.2	80	1262.7	74	−0.01	−0.98	−0.07
*rc*202	1092.3	1095.8	46	1102.3	76	1095.8	71	0.00	−0.92	−0.32
*rc*203	923.7	923.7	56	931.4	76	923.7	73	0.00	−0.83	0.00
*rc*204	783.5	785.8	68	790.8	79	783.8	76	0.29	−0.93	0.00
*rc*205	1154	1154	45	1163.1	68	1154	64	0.00	−0.79	0.00
*rc*206	1051.1	1051.1	52	1062.8	71	1051.1	68	0.00	−1.11	0.00
*rc*207	962.9	966.6	55	979.2	70	966.6	64	0.00	−1.69	−0.38
*rc*208	777.3	777.3	65	781.9	76	777.3	72	0.00	−0.59	0.00
Average								0.04	−0.98	−0.10

$(\upsilon(\text{Best}) - \upsilon(ALNS_I))/\upsilon(\text{Best})$ and Dev_E (%) is calculated as 100 $(\upsilon(\text{Best}) - \upsilon(ALNS_E))/\upsilon(\text{Best})$, where $\upsilon(\text{Best})$ is the best known solution value for each instance.

As shown in Tables 6–8, the extended ALNS heuristic performs very well on the VRPTW instances considered in our tests. For a majority of the instances, the heuristic is able to discover the best known solution. For the rest of the instances, the percentage deviations are no greater than 0.72%. The tables also show that the operators work well to improve the solutions produced by $ALNS_I$, particularly on the *r* instances, and help discover solutions which are slightly better than those reported in Pisinger and Ropke (2005) on some instances. In particular, the average deviation is reduced from 0.85% to 0.09% for instances *r*101–*r*108 and from 1.74% to 0.25% for instances *r*201–*r*208 with the use of the new operators. Similar improvements are achieved on the instances *rc* for which

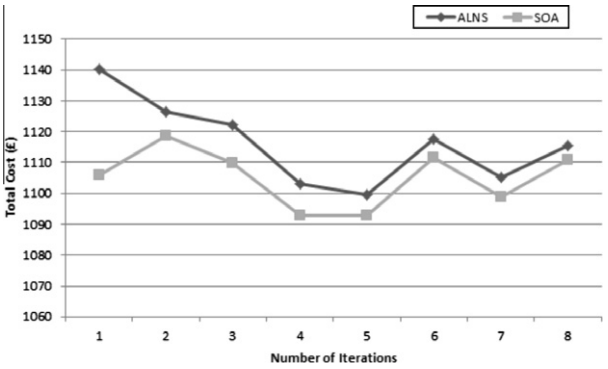

Fig. 6. ALNS and SOP algorithm for a 100-node instance.

Table 9
Performance improvement using SOA on solutions obtained by ALNS.

Instance sets	$ALNS_D$	$ALNS_D^+$	$ALNS_P$	$ALNS_P^+$	D_D%	D_P%
10	181.89	177.02	176.17	170.64	3.14	2.68
15	297.25	287.30	295.89	286.89	3.04	3.35
20	335.92	325.38	332.70	325.23	2.25	3.14
25	296.14	288.60	288.23	282.13	2.12	2.55
50	623.57	604.89	612.98	596.11	2.75	3.00
75	1016.26	985.99	1011.58	985.86	2.54	2.98
100	1304.37	1264.51	1262.33	1230.72	2.50	3.06
150	1524.99	1475.16	1512.60	1469.74	2.83	3.27
200	2241.34	2172.94	2212.31	2144.25	3.08	3.05
Average					2.69	3.01

the average deviation is reduced from 0.84% to 0.17% for instances *rc*101–*rc*108 and from 0.98% to 0.1% for instances *rc*201–*rc*208. These figures suggest that it is worthwhile to use the new operators to obtain good quality solutions. These results also confirm the effectiveness of our extended ALNS heuristic, given that it was designed to solve a problem more general than the classical VRPTW, whereas the best known solution values were obtained by means of specialized algorithms.

4.3. The effect of speed optimization

The SOA is extremely quick and able to improve upon the results produced by the ALNS. To give an example, Fig. 6 shows the

Table 10
Computational results for 10-node instances.

Instances	CPLEX			Our heuristic								Improvement %
	Solution cost £	Total CPU time s	LP relaxation	Solution cost £	Distance kilometers	# of vehicles	# of loops	ALNS time seconds	SOA time seconds	CPU time per loop seconds	Total CPU time seconds	
UK10_01	170.66	163.4	79.3	170.64	409.0	2	4	0.5	0.0	0.5	2.1	0.01
UK10_02	204.87	113.9	95.1	204.88	529.8	2	4	0.6	0.0	0.6	2.3	0.00
UK10_03	200.33	926.0	78.5	200.42	507.3	3	4	0.5	0.0	0.5	2.0	−0.04
UK10_04	189.94	396.5	81.9	189.99	480.1	2	4	0.5	0.0	0.5	2.2	−0.03
UK10_05	175.61	1253.7	77.3	175.59	447.0	2	4	0.6	0.0	0.6	2.3	0.01
UK10_06	214.56	347.5	101.8	214.48	548.0	2	4	0.6	0.0	0.6	2.2	0.04
UK10_07	190.14	191.0	98.5	190.14	494.7	2	5	0.6	0.0	0.6	2.9	0.00
UK10_08	222.16	139.8	111.0	222.17	567.8	2	4	0.5	0.0	0.5	2.1	0.00
UK10_09	174.53	54.0	68.2	174.54	457.0	2	4	0.6	0.0	0.6	2.2	0.00
UK10_10	189.83	76.9	88.7	190.04	486.7	2	5	0.5	0.0	0.5	2.6	−0.11
UK10_11	262.07	50.5	129.3	262.08	697.2	2	4	0.5	0.0	0.5	2.2	0.00
UK10_12	183.18	1978.7	77.5	183.19	460.3	2	4	0.5	0.0	0.5	2.2	−0.01
UK10_13	195.97	1235.1	91.4	195.97	510.5	2	4	0.6	0.0	0.6	2.2	0.00
UK10_14	163.17	84.1	83.0	163.28	397.8	2	4	0.6	0.0	0.6	2.4	−0.07
UK10_15	127.15	433.3	59.6	127.24	291.4	2	4	0.6	0.0	0.6	2.4	−0.07
UK10_16	186.63	680.8	89.4	186.73	451.1	2	4	0.5	0.0	0.5	1.9	−0.06
UK10_17	159.07	27.0	83.4	159.03	387.5	2	4	0.6	0.0	0.6	2.3	0.03
UK10_18	162.09	522.1	70.3	162.09	401.5	2	4	0.5	0.0	0.5	2.2	0.00
UK10_19	169.46	130.5	84.2	169.59	414.5	2	7	0.6	0.0	0.6	4.1	−0.08
UK10_20	168.80	1365.5	73.1	168.80	412.8	2	4	0.5	0.0	0.5	2.0	0.00

Table 11
Computational results for 100-node instances.

Instances	CPLEX			Our heuristic								Improvement %
	Solution cost £	Total CPU time seconds	LP relaxation	Solution cost £	Distance kilometers	# of vehicles	# of loops	ALNS time seconds	SOA time seconds	CPU time per loop seconds	Total CPU time seconds	
UK100_01	1389.05	10800*	572.22	1240.79	2914.4	14	4	23.0	0.0	23.0	92.1	10.67
UK100_02	1302.16	10800*	548.13	1168.17	2690.7	13	4	24.5	0.0	24.5	98.2	10.29
UK100_03	1231.44	10800*	518.00	1092.73	2531.8	13	8	26.0	0.0	26.0	207.9	11.26
UK100_04	1174.75	10800*	487.57	1106.48	2438.5	14	7	21.4	0.0	21.4	149.7	5.81
UK100_05	1121.71	10800*	474.66	1043.41	2328.5	14	8	19.9	0.0	19.9	159.0	6.98
UK100_06	1320.40	10800*	565.78	1213.61	2782.4	14	7	19.1	0.0	19.1	133.8	8.09
UK100_07	1177.87	10800*	494.67	1060.08	2463.9	12	4	25.7	0.0	25.7	102.6	10.00
UK100_08	1230.92	10800*	516.07	1106.78	2597.4	13	9	23.3	0.0	23.3	209.5	10.09
UK100_09	1092.20	10800*	439.48	1015.46	2219.2	13	6	25.7	0.0	25.7	154.0	7.03
UK100_10	1163.95	10800*	499.34	1076.56	2510.1	12	8	24.9	0.0	24.9	199.0	7.51
UK100_11	1343.18	10800*	560.36	1210.25	2792.1	15	5	21.4	0.0	21.4	107.1	9.90
UK100_12	1227.01	10800*	483.91	1053.02	2427.3	12	9	22.9	0.0	22.9	206.4	14.18
UK100_13	1333.10	10800*	535.11	1154.83	2693.1	13	4	22.0	0.0	22.0	87.9	13.37
UK100_14	1410.18	10800*	599.61	1264.50	2975.3	14	4	23.0	0.0	23.0	91.8	10.33
UK100_15	1453.81	10800*	608.94	1315.50	3072.1	15	5	22.2	0.0	22.2	110.9	9.51
UK100_16	1105.58	10800*	448.32	1005.03	2219.7	12	10	25.5	0.0	25.5	254.7	9.09
UK100_17	1389.99	10800*	594.54	1284.81	2960.4	15	7	21.8	0.0	21.8	152.8	7.57
UK100_18	1219.45	10800*	501.69	1106.00	2525.2	13	4	23.1	0.0	23.1	92.6	9.30
UK100_19	1115.82	10800*	458.01	1044.71	2332.6	13	4	22.7	0.0	22.7	91.0	6.37
UK100_20	1396.97	10800*	594.06	1263.06	2957.8	14	9	22.7	0.0	22.7	204.4	9.59

* Not solved to optimality in 10800 second.

Table 12
Computational results for 200-node instances.

Instances	CPLEX			Our heuristic								Improvement %
	Solution cost £	Total CPU time s	LP relaxation	Solution cost £	Distance kilometers	# of vehicles	# of loop	ALNS time seconds	SOA time seconds	CPU time per loop seconds	Total CPU time seconds	
UK200_01	2711.51	10800*	940.3	2111.70	4609.6	28	7	103.5	0.0	103.5	724.4	22.12
UK200_02	2726.99	10800*	897.2	1988.64	4444.4	24	9	113.4	0.0	113.4	1020.9	27.08
UK200_03	2621.57	10800*	903.7	2017.63	4439.9	27	4	101.0	0.0	101.0	404.1	23.04
UK200_04	3175.38	10800*	841.4	1934.13	4191.9	26	4	102.9	0.0	102.9	411.7	39.09
UK200_05	X	10800*	1008.6	2182.91	4861.9	27	9	102.9	0.0	102.9	926.4	X
UK200_06	X	10800*	811.1	1883.22	3980.4	27	4	112.6	0.0	112.6	450.3	X
UK200_07	3344.32	10800*	881.9	2021.95	4415.3	27	9	104.8	0.0	104.8	943.4	39.54
UK200_08	2980.33	10800*	951.8	2116.76	4664.4	27	4	107.6	0.0	107.6	430.6	28.98
UK200_09	2409.25	10800*	815.6	1894.18	4031.1	25	5	110.6	0.0	110.6	553.1	21.38
UK200_10	2871.22	10800*	1013.4	2199.95	4921.8	28	4	125.0	0.0	125.0	500.0	23.38
UK200_11	X	10800*	834.5	1941.19	4099.5	27	5	168.6	0.0	168.6	842.8	X
UK200_12	3388.33	10800*	994.0	2105.14	4808.5	25	11	64.6	0.0	64.6	711.0	37.87
UK200_13	X	10800*	986.9	2141.26	4760.3	25	4	111.1	0.0	111.1	444.6	X
UK200_14	X	10800*	894.4	2011.35	4369.9	27	4	112.7	0.0	112.7	450.6	X
UK200_15	3435.71	10800*	955.1	2110.86	4723.9	25	5	108.4	0.0	108.4	542.0	38.56
UK200_16	2691.36	10800*	935.3	2075.83	4545.9	27	4	113.9	0.0	113.9	455.6	22.87
UK200_17	3032.37	10800*	1028.4	2218.28	4972.8	26	4	102.3	0.0	102.3	409.2	26.85
UK200_18	3569.48	10800*	909.1	2004.68	4370.3	27	7	112.7	0.0	112.7	788.9	43.84
UK200_19	X	10800*	794.3	1844.90	3995.4	25	9	108.2	0.0	108.2	973.9	X
UK200_20	3561.79	10800*	984.1	2150.57	4805.4	27	5	106.2	0.0	106.2	531.1	39.62

X: No feasible solution found in 10800 second.

improvement by the SOA over routes found by the ALNS algorithm for a 100-node instance. In our implementation, the SOA terminates on a given route after three non-improving iterations. As Fig. 6 shows, the improvement provided by the SOA on a given solution ranges between 0.41% and 3.01%.

A more detailed analysis of the effect of the SOA is provided in Table 9. This table presents, for one instance from each of the nine sets, the solutions obtained by using a distance-minimizing objective shown under column $ALNS_D$, and results with the SOA as applied on $ALNS_D$ shown under column $ALNS_D^+$. The $ALNS_D$ and $ALNS_D^+$ solutions are produced by initially using a distance minimizing objective, but the solution values reported in Table 9 are recalculated using the PRP objective. Table 9 also shows the results with ALNS using the PRP objective (7)–(11) shown under column $ALNS_P$ and results with SOA as applied on $ALNS_P$ shown under column $ALNS_P^+$. The last two columns present the percentage difference between $ALNS_D$ and $ALNS_D^+$ under column D_D and the percentage difference between $ALNS_P$ and $ALNS_P^+$ under column D_P.

The results shown in Table 9 indicate that the SOA is generally able to improve the solutions found by the ALNS by 2–4%. The average improvement in the case of minimizing the PRP function is 3.01%, which is slightly higher as compared to the average improvement of 2.65% seen in the case of minimizing distance.

4.4. PRP heuristic results

This section presents the results obtained by the proposed heuristic on the nine sets of PRP instances generated. Each instance was solved once with the proposed heuristic and once with the PRP model solved with a truncated execution of CPLEX 12.1 (IBM ILOG, 2009) with its default settings. A common time-limit of 3 hours was imposed on the solution time for all instances.

The detailed results of these experiments are presented in Tables 10–12. These tables give the results for 10-, 100- and 200-node instances. Each table presents, for each instance, the solutions found by the heuristic and CPLEX. All columns in these tables are self-explanatory with the exception of the last, which shows the percentage deviation of the solution produced by the heuristic from the best solution obtained with CPLEX within the 3-hour time limit.

Table 13
Summary of comparisons between the proposed heuristic and CPLEX.

Instance sets	Average CPU time for CPLEX (s)	Average CPU time for the heuristic (seconds)	Average # of loops of the heuristic	Average time per loop (seconds)	Average (%)
10	508.5	2.3	4.3	0.5	−0.02
15	10800*	3.9	4.4	0.9	0.07
20	10800*	6.4	4.6	1.4	0.53
25	10800*	10.0	5.3	1.9	0.88
50	10800*	35.4	6.5	5.5	2.38
75	10800*	70.5	5.9	12.1	5.97
100	10800*	145.3	6.3	23.0	9.35
150	10800*	348.0	6.0	62.2	18.06
200	10800*	625.7	5.9	109.7	31.01

Table 10 shows that the heuristic finds the same solutions as those of CPLEX but in a substantially smaller amount of time. The fact that ALNS finds solutions which are better than the optimal solution provided by CPLEX for some instances is due to the discretization used in the mathematical formulation. The average time required by CPLEX to solve the 10-node instances to optimality is 508.5 seconds where the same statistic for the ALNS to produce the reported solutions is 2.3 second.

A summary of the results of the 180 instances tested is presented in Table 13. In this table, we present, for each instance set, the average cost, the average deviation between the solution produced by the heuristic and the best solution obtained with the PRP model within the 3-hour limit, the average CPU times for the heuristic and for CPLEX, and the average number of loops.

As can be observed from Table 13, the results indicate that the proposed algorithm runs quickly even for large size instances. The minimum and maximum number of loops, for all instances, are four and 13, respectively. Instances of up to 100 nodes are solved in 145 second on average. The algorithm requires just over 10 minutes of computation time to solve instances with 200 nodes and is able to produce much better results than CPLEX does in 3 hours. The improvements can be as high as 10% for instances of up to 100 nodes, around 18% for instances with 150 nodes and around 30% for instances with 200 nodes. CPLEX was not able to find opti-

mal solutions in 3 hours for 160 instances out of the total of 180 instances tested. The average CPU time per iteration of the algorithm is less than 2 minutes.

5. Conclusions

We have described a heuristic algorithm to solve the PRP. The algorithm iterates between a VRPTW and a speed optimization problem, the former solved through an enhanced ALNS and the latter solved using a polynomial time procedure. The enhanced ALNS uses new, as well as existing removal and insertion operators, which improve the solution quality. These operators can be used in ALNS for solving other types of problems. The SOA, on the other hand, improves the solution produced by the ALNS and minimizes fuel consumption costs and driver wages by optimizing vehicle speeds. The SOA has a negligible execution time, and is generic enough to be used as a stand-alone routine for other types of routing problems in order to optimize speed. To fully evaluate the effectiveness of the heuristic algorithm, we have generated different sets of instances based on real geographic data and have compiled a library of PRP instances. We have presented results of extensive computational experimentation using the proposed heuristic and have compared it against the solutions produced using the integer linear programming formulation of the PRP. The results show that the proposed algorithm is highly effective in finding good-quality solutions on instances with up to 200 nodes.

Acknowledgements

The authors gratefully acknowledge funding provided by the University of Southampton School of Management and by the Canadian Natural Sciences and Engineering Research Council under grant 39682-10. Thanks are due to the referees for their valuable comments.

References

Barth, M., Boriboonsomsin, K., 2008. Real-world CO_2 impacts of traffic congestion. Transportation Research Record: Journal of the Transportation Research Board 2058 (1), 163–171.

Barth, M., Younglove, T., Scora, G., 2005. Development of a Heavy-Duty Diesel Modal Emissions and Fuel Consumption Model. Technical report, UC Berkeley: California Partners for Advanced Transit and Highways (PATH).

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

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12 (4), 568–581.

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., Semet, F., 2002. A guide to vehicle routing heuristics. Journal of the Operational Research Society 53 (5), 512–522.

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

Hickman, J., Hassel, D., Joumard, R., Samaras, Z., Sorenson, S., 1999. MEET-Methodology for Calculating Transport Emissions and Energy Consumption. European Commission, DG VII. Technical report, ISBN 92-828-6785-4, Luxembourg. p. 362. <http://www.transport-research.info/Upload/Documents/200310/meet.pdf> (04.11.12).

Hvattum, L.M., Norstad, I., Fagerholt, K., Laporte, G., forthcoming. Analysis of an exact algorithm for the vessel speed optimization problem. Networks.

IBM ILOG. 2009. Copyright ©International Business Machines Corporation 1987.

Jabali, O., Van Woensel T., A.G., de Kok, forthcoming. Analysis of travel times and CO_2 emissions in time-dependent vehicle routing. Production and Operations Management.

Kirby, H.R., Hutton, B., McQuaid, R.W., Raeside, R., Zhang, X., 2000. Modelling the effects of transport policy levers on fuel efficiency and national fuel consumption. Transportation Research Part D: Transport and Environment 5 (4), 265–282.

Milthers, N.P.M., 2009. Solving VRP using Voronoi Diagrams and Adaptive Large Neighborhood Search. Master's thesis, University of Copenhagen, Denmark.

Norstad, I., Fagerholt, K., Laporte, G., 2010. Tramp ship routing and scheduling with speed optimization. Transportation Research Part C: Emerging Technologies 19 (5), 853–865.

Pisinger, D., Ropke, S., 2005. A General Heuristic for Vehicle Routing Problems. Technical Report, DIKU - Department of Computer Science, University of Copenhagen. <http://www.diku.dk/hjemmesider/ansatte/sropke/Papers/GeneralVRP_TechRep.pdf (04.11.12).

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

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

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

Scora, M., Barth, G., 2006. Comprehensive Modal Emission Model (CMEM), Version 3.01, User's Guide. Technical Report, 2006. <http://www.cert.ucr.edu/cmem/docs/CMEM_User_Guide_v3.01d.pdf> (04.11.12).

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming. Springer, New York, pp. 417–431.