Soft Computing (2019) 23:7359–7373
https://doi.org/10.1007/s00500-018-3382-8

METHODOLOGIES AND APPLICATION

Chinese and windy postman problem with variable service costs

Muhammed Emre Keskin[1] · **Mustafa Yılmaz**[1]

Published online: 11 July 2018

Abstract
Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of nodes denoted by $\mathcal{V}$, edges between nodes represented by $\mathcal{E}$, and costs associated with the edges, postman problem (PP) is to find the route having the minimum cost that begins and ends with a predefined starting point and spans each edge of the network. PP is a variant of the well-known arc routing problem. In many real-life applications of the PP, costs associated with the edges tend to reduce with each pass on the edges. We propose a new mathematical formulation to represent the postman problem with variable service costs. If the service costs are symmetric, the problem is named as the Chinese postman problem (CPP) with variable service costs (CPPVSC), and it is called as the windy postman problem with variable service costs (WPPVSC), otherwise. CPPVSC turns to be a variant of CPP, and it is an easy problem. We show that no edge can be traversed more than twice in the optimal solution. Moreover, we propose two heuristics for the solution of WPPVSC. Based on the extensive numerical experiments, we can say that both heuristics outperform the state-of-the-art commercial solvers.

Keywords Chinese postman problem · Windy postman problem · Variable service costs · Heuristic approaches

1 Introduction

Arc routing problem (ARP) is one of the well-studied problems of the operations research literature. There are many real-life applications of ARP such as garbage collection, post-delivery, snow plow, salinization of snowy roads (Assad and Golden 1995; Campbell and Langevin 2000; Eglese and Li 1992). PP is a variant of ARP in which the postman is required to pass through each edge of the network starting and ending with a predefined point with minimum possible cost. PP is undirected if traversing edges in each direction is possible, it is directed if traversing is permitted in only one direction, and it is mixed if a subset of the edges are directed. Similarly, if service costs do not depend on the traversing direction, PP is symmetric, and it is asymmetric otherwise. Finally, there are postman problems in which more than one vehicle is used (Hertz 2005). We direct interested readers to the study by Dror (2012) in which ARP and its variations (including variants of the postman problem) are analyzed in detail.

In real-life applications, service costs appointed to the edges may vary from pass to pass due to the natural conditions, traffic density or the weather conditions. For instance, service costs are related to the amount of snow on the roads if the implied application is snow plow. We basically assume that the roads are closed due to excessive amount of snow and the snow plow vehicle is required to pass through each road at least once in order to open them for a minimum extent. However, all the snow cannot be cleaned with a single pass of the vehicle. Hence, the amount of snow decreases with each pass of the vehicle. Similarly, postman learns the roads better with each pass implying that passing times tend to reduce throughout the post-delivery process. These variations in the service costs may alter the resulting cost-minimizing routes as well. Thus, variability of service costs should be taken into account that we undertake in this study.

PP with symmetric service costs belongs to class P, and it is easily solvable (Assad and Golden 1995). There is no need to pass through any edge more than twice for symmetric networks. PP with symmetric costs is known as CPP in the literature. On the other hand, more realistic version of the PP having asymmetric costs (Ávila et al. 2016),

Communicated by V. Loia.

✉ Muhammed Emre Keskin
emre.keskin@atauni.edu.tr

Mustafa Yılmaz
mustafay@atauni.edu.tr

[1] Department of Industrial Engineering, Atatürk University, 25030 Erzurum, Turkey

which is known as windy postman problem (WPP), is shown to be NP-hard by Guan (1984). It is possible and usually expected that some edges of the network will be traversed more than twice if there are nodes with odd degrees for WPP instances.

In this study, we concentrate on the PP in which service costs of the edges vary depending on the number of passes. Hence, costs do not vary with the time as they are in time-dependent ARPs (Gendreau et al. 2015), but with the passing numbers. If the costs are symmetric, i.e., the problem is CPPVSC, then it still belongs to class P. We show that no edge can be traversed more than twice in the optimal solution of CPPVSC. On the other hand, WPPVSC is naturally a difficult problem. We propose two heuristic solution strategies and show their accuracies and efficiencies on four sets of test instances from the literature and on one set of test instances we generate that includes relatively denser networks.

The rest of the paper is organized as follows. In the next section, a brief review of the related literature is given. Next, we provide the mathematical formulation of the postman problem with variable service costs (PPVSC). Later on, we analyze CPPVSC in detail. Heuristic solution strategies for solution of WPPVSC are given in the solution approaches section. Moreover, we expose the success of the heuristics in numerical results section. Finally, we conclude the paper and point out future research directions.

2 Related studies

Time-dependent routing problems are treated under two main titles in the literature which are node routing (traveling salesman problem Malandraki and Dial 1996; Li et al. 2005; Schneider 2002; Cordeau et al. 2012; Taş et al. 2016, vehicle routing problem Malandraki and Daskin 1992; Ichoua et al. 2003; Donati et al. 2008; Guan 1984; Dussault et al. 2013; Koç et al. 2016; Setak et al. 2015) and arc routing. One observation is that there are time-dependent traveling salesman and vehicle routing studies with deterministic or stochastic service costs, but arc routing studies assume only deterministic service costs. One reason behind this phenomenon may be the fact that time-dependent node routing problems have been studied more than the time-dependent arc routing problems. We direct the interested readers to the paper by Gendreau et al. (2015) that represents the state-of-the-art about the time-dependent routing problems. We provide a brief review of the time-dependent arc routing studies in the following.

Tagmouti et al. (2007) concentrates on the capacitated time-dependent arc routing problem. The authors transform the problem into the node routing problem and propose a solution method based on column generation. Tagmouti et al. (2010) also considers the same problem and introduces a variable neighborhood descent heuristic. Tagmouti et al. (2011) follows the same line of research, but the arc routing problem it takes assumes dynamic arc capacities. The authors adapt the variable neighborhood descent heuristic of Tagmouti et al. (2010) as the solution strategy. Besides, Black et al. (2013) utilizes variable neighborhood search and tabu search methods for prize collecting time-dependent arc routing problem. A variant of WPP is studied by Dussault et al. (2013) in which a snow plow application is considered. The authors assume that the service costs of the roads decrease if the snow plow vehicle has already passed. They offer a heuristic named cycle permutation local search as the solution method. Hence, subject of Dussault et al. (2013) is similar to our analysis, but it is assumed in Dussault et al. (2013) that service costs decrease only after the first pass while we generalize the idea and assume that each pass affects the service costs. Moreover, the authors assume a mixed network but allow passing through reverse directions by closing the roads during snow plow. These special conditions do not apply for our case. Another study is due to Sun et al. (2015) in which a new integer program for time-dependent Chinese postman problem is proposed. The authors generate valid constraints and employ a cutting plane algorithm as the solution method. Finally, Vincent and Lin (2015) offers an iterated greedy heuristic for the solution of prize collecting time-dependent arc routing problem. The authors illustrate effectiveness of their method on several test sets.

3 Mathematical model

Before giving the mathematical formulation of PPVSC, we, respectively, describe the sets, parameters and variables that are used in the formulation. First of all, we define $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as the graph of the nodes denoted by $\mathcal{V}$ and the edges represented by $\mathcal{E}$. We use i and j indexes for the nodes and (i, j) for the edges. Set of times of the passes on the edges is given by $\mathcal{T}$, and we let k stand for the order of the pass, while t denotes the total number of passes. Service cost of edge (i, j) k^{th} time is given by c_{ij}^k and we assume that $c_{ij}^{k_1} \geq c_{ij}^{k_2}$ if $k_1 \leq k_2$. That is, service costs do not become more costly as the number of the passes increases which is intuitive since the postman is expected to learn the roads better as the number of passes increases in the post-delivery application. Similarly, if each pass on the edges represents the shoveling of the accumulated snow on a specific edge, then the cost of the shoveling should not increase by the number of the passes on that edge as the snow becomes less and less at each pass of the snow plow vehicle. Moreover, we also assume that $c_{ij}^k > 0$ for any k, i.e., free rides are not allowed. It should be noted that by passing the edge (i, j), we mean to traverse it by going from i to j. The service cost of an edge is defined as the traversing cost of the edge which includes the cost of the shoveling operation taken place on the edge and the cost of the travel on the edge

Table 1 Sets, parameters and decision variables

	Definition
Sets	
$\mathcal{V}$	Set of nodes
$\mathcal{E}$	Set of edges
$\mathcal{T}$	Set of number of passes
Parameters	
c_{ij}^k	Service cost of the edge (i, j) kth time
d_{ij}^t	Total cost of traversing the edge (i, j) t times
Variables	
x_{ij}	Number of passes on edge (i, j)
y_{ij}^t	Indicates whether or not the number of passes on edge (i, j) is t

itself. We interchangeably use service cost and the traversing cost for the same concept throughout the study. We also refer to the service cost of the edges as the length of the edges especially in the solution approaches section in which we embrace the problem in the graph theory context. If service costs of the edges (i, j) and (j, i) at each pass are identical, i.e., $c_{ij}^k = c_{ji}^k$ $(i, j) \in \mathcal{E}, k \in \mathcal{T}$, then the costs possess a symmetric structure, and the problem becomes CPPVSC. On the other hand, if $c_{ij}^k \neq c_{ji}^k$ for any $(i, j) \in \mathcal{E}, k \in \mathcal{T}$, then the problem at hand is WPPVSC. It should be noted that the implied asymmetry among the costs can be easily justified. For instance, the slopes of the roads may cause traversing on a specific direction more difficult than the opposite direction, or as the name of the problem suggests, the wind may have more effect on specific directions. Another parameter that is used in the formulation is d_{ij}^t which denotes the total cost of traversing the edge (i, j) t times. Mathematically, $d_{ij}^t = \sum_{k=1}^{t} c_{ij}^k$. Note that $d_{ij}^{t_1} \leq d_{ij}^{t_2}$ for $t_1 \leq t_2$. It is easy to see that if the service costs (c_{ij}^k) are symmetric, then the total service costs (d_{ij}^t) are also symmetric, and if the service costs are asymmetric, then the total service costs are most likely asymmetric as well. Finally, we use two set of variables named x_{ij} and y_{ij}^t. x_{ij} stands for the number of passes on edge (i, j) and y_{ij}^t indicates whether or not the number of passes on edge (i, j) is exactly t, i.e., $y_{ij}^t = 1$ if $x_{ij} = t$ and $y_{ij}^t = 0$, otherwise. A summary of the set, parameter and variable definitions is provided in Table 1 for the sake of convenience.

Below, we give the mathematical formulation of the postman problem with variable service costs.

PPVSC:

$$\min \sum_{(i,j)\in\mathcal{E}} \sum_{t\in\mathcal{T}} d_{ij}^t y_{ij}^t \tag{1}$$

s.t.

$$\sum_{(i,j)\in\mathcal{E}} x_{ij} = \sum_{(j,i)\in\mathcal{E}} x_{ji} \qquad i \in \mathcal{V} \tag{2}$$

$$x_{ij} + x_{ji} \geq 1 \qquad (i, j) \in \mathcal{E} \tag{3}$$

$$t y_{ij}^t \leq x_{ij} \qquad (i, j) \in \mathcal{E}, t \in \mathcal{T} \tag{4}$$

$$x_{ij} \leq \sum_{t\in\mathcal{T}} t y_{ij}^t \qquad (i, j) \in \mathcal{E} \tag{5}$$

$$y_{ij}^t \in \{0, 1\} \qquad (i, j) \in \mathcal{E}, t \in \mathcal{T} \tag{6}$$

$$x_{ij} \geq 0 \text{ and integer} \qquad (i, j) \in \mathcal{E} \tag{7}$$

We minimize the total service cost in the objective function (1). In the first set of constraints (2), we ensure that total arrivals to and departures from each node are equal, implying that the postman, or the snow plow vehicle, should leave each node as many as the number of times she visits the node. Constraint (3) guarantees that each edge of the network is visited at least once. It should be noted that although it is possible to traverse the edges in each direction, it is not required. It can be, and is expected to be, the case in the solution that some of the edges are traversed through only one direction. Constraint (4) makes sure that if y_{ij}^t is 1, then x_{ij} is at least t. On the contrary, if $x_{ij} = t$, then the variable y_{ij}^t should be equal to 1 which is achieved by constraint (5). It can be observed that since the objective is in the direction of minimization, among the variables existing on the right hand side of constraint (5), only the related y_{ij}^t variable is set to 1 in the optimal solution. For instance, suppose that $x_{ij} = 6$ for a specific edge (i, j). Then, the right hand side of constraint (5) written for edge (i, j) should be at least 6. First, setting the right hand side of the constraint to a value that is larger than 6 increases the objective function unnecessarily. Hence, the right hand side will be exactly 6 in the optimal solution. This can be achieved by simply setting y_{ij}^6 to 1 which increases the objective function value by d_{ij}^6. Observe that any other solution leads to a larger objective function value. For example, if y_{ij}^4 and y_{ij}^2 are both set to 1 to achieve a total of 6 at the right hand side of the constraint, then their combined effect on the objective function value will be $d_{ij}^4 + d_{ij}^2$ which is greater than or equal to d_{ij}^6 since $d_{ij}^6 = \sum_{k=1}^{6} c_{ij}^k$ while $d_{ij}^4 + d_{ij}^2 = \sum_{k=1}^{4} c_{ij}^k + \sum_{k=1}^{2} c_{ij}^k$ and c_{ij}^k values are nonincreasing in k. Therefore, constraint (5) ensure that only the y_{ij}^t variable associated with the value of x_{ij} is equal to 1. As a consequence, if y_{ij}^t variable takes value 1, then x_{ij} is equated to t by collaborative efforts of constraints (4) and (5). This implies that it is possible to take x_{ij} variables as continuous in the model knowing that its value will always be integral. We exploit this phenomenon to significantly increase the solution efficiency as stated in the computational results section. Finally, constraint (6) and constraint (7) are the usual binary, nonnegativity and integrality restrictions.

4 Analysis of CPPVSC

It is a well-known fact that CPP is polynomially solvable. A polynomial time algorithm can be described as follows; if all the nodes have even degrees, then there is an Euler tour that passes through each edge only once and it is obviously optimal. If there are vertices with odd degrees, then their numbers should be even. A new complete graph consisting only the vertices with odd degrees can be formed in which the edge lengths correspond to the lengths of the shortest paths obtained on the original network. In other words, the length between nodes i and j on the new graph is equal to the length of the shortest path between nodes i and j on the original network. Then, a minimum weight perfect matching problem is solved in polynomial time in the newly formed complete graph. Edges of the shortest paths corresponding to the links selected in the perfect matching are then added to the original network to form a new network in which all the vertices have even degrees, and length of the Euler tour on that new network gives the minimum cost.

The algorithm mentioned above that solves classical CPP in polynomial time also solves CPPVSC as well. Note that it is not possible to equate arrivals to and departures from odd degree nodes by traversing the neighboring edges only once. This implies that there must be paths between the odd degree nodes including the edges traversed at least two times. Such paths with minimum possible travel costs can be found by constructing a complete network with odd degree nodes and searching for the minimum weight perfect matching on that network as described above. The only difference is that shortest paths between nodes should be calculated using c^2_{ij} values for each odd degree node pair. Moreover, although the passing costs tend to decrease by the number of passes, no edge is traversed more than twice. This is why we need only the c^2_{ij} values to calculate shortest path lengths. We express this observation in a more formal way in the proposition given below.

Proposition 1 *If the costs are symmetric, i.e., $c^k_{ij} = c^k_{ji}$ for $(i, j) \in \mathcal{E}, k \in \mathcal{T}$, then $x_{ij} \leq 2$ for $(i, j) \in \mathcal{E}$ in the optimal solution.*

Proof First of all, if all the vertices have even degrees than the Euler tour that passes each edge only once is optimal meaning that there is no need to pass any edge more than once even in the case that second time passes have very low costs. In that case,

$$x_{ij} = \begin{cases} 1 & \text{if } (i, j) \text{ is traversed through } i \text{ to } j, \\ 0 & \text{if } (i, j) \text{ is traversed through } j \text{ to } i. \end{cases}$$

Here, $x_{ij} \leq 2$ for $(i, j) \in \mathcal{E}$ trivially holds. Now, consider the case where some of the vertices have odd degrees. Then, a complete graph including the odd degree vertices can be formed. By the way of contradiction, assume that the minimum cost perfect matching, say matching 1, obtained on the complete graph of the odd degree vertices includes two edges (a, b) and (c, d) such that the shortest paths from a to b and from c to d on the original network share an edge (e, f) of the original network as depicted in Fig. 1.

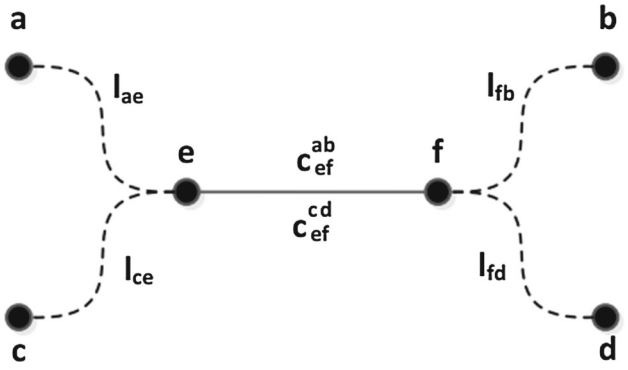

Fig. 1 Illustration of shortest paths between (a, b) and (c, d) pairs

Note that the Euler tour obtained on the network after adding the edges of the shortest paths corresponding to the selected links of the minimum cost perfect matching to the original network includes the edge (e, f) at least three times, i.e., $x_{ef} \geq 3$. For convenience, we let the edge lengths of the complete graph (which correspond to the lengths of the shortest paths on the original network) to be denoted by ℓ, for instance the edge length between a and b is given by ℓ_{ab}. Total cost of the matching 1 is equal to the sum of the ℓ_{ab}, ℓ_{cd} and the total remaining cost (TRC). Since we do not know the exact number and order of passes on edge (e, f), let the passing costs used for calculation of ℓ_{ab} and ℓ_{cd} be denoted by c^{ab}_{ef} and c^{cd}_{ef}, respectively. Obviously, $\ell_{ab} = \ell_{ae} + c^{ab}_{ef} + \ell_{fb}$ and $\ell_{cd} = \ell_{ce} + c^{cd}_{ef} + \ell_{fd}$. Then, total cost of the matching 1 is equal to TRC+$\ell_{ae} + \ell_{fb} + \ell_{ce} + \ell_{fd} + c^{ab}_{ef} + c^{cd}_{ef}$. Now, suppose that we form another matching, say matching 2, in which instead of edges (a, b) and (c, d), we select (a, c) and (b, d) and the remaining edges of the matching are the same with the ones of the matching 1. Edges (a, c) and (b, d) exist since the newly formed network is complete. Observe that matching 2 is also perfect and its total cost is equal to TRC+$\ell_{ac} + \ell_{bd}$. We know that there is a path between a and c which goes through node e having the length of $\ell_{ae} + \ell_{ce}$. Thus, $\ell_{ac} \leq \ell_{ae} + \ell_{ce}$ should hold. Similarly, $\ell_{bd} \leq \ell_{fb} + \ell_{fd}$ also holds. Nevertheless, this implies that TRC+$\ell_{ac} + \ell_{bd} \leq$ TRC+$\ell_{ae} + \ell_{ce} + \ell_{fb} + \ell_{fd} <$ TRC+$\ell_{ae} + \ell_{fb} + \ell_{ce} + \ell_{fd} + c^{ab}_{ef} + c^{cd}_{ef}$ which is a contradiction implying that the minimum cost perfect matching cannot include two edges such that corresponding shortest paths share an edge of the original network. Hence, no edge of the original network can be traversed more than twice in the optimal solution. □

As Proposition 1 suggests, CPPVSC is as much difficult as CPP which belongs to class P. Therefore, we provide no numerical results for CPPVSC.

5 Solution approaches

In this section, we provide two heuristic strategies for the solution of WPPVSC which are, respectively, called as pass iteration heuristic (PIH) and Lagrangian heuristic (LH).

5.1 Pass iteration heuristic

Observe that if c_{ij}^k values do not vary with k, i.e., service costs do not change from pass to pass, then WPPVSC reduces to WPP which is known to be NP-hard. Hence, WPPVSC is also NP-hard implying that we must resort to heuristic procedures for the solution of large instances of WPPVSC. It is known that traversing an edge more than twice is possible in WPP and it is even encouraged in WPPVSC since cost of the passing tends to reduce with each pass. Moreover, as the costs are asymmetric, known combinatorial algorithms like the blossom algorithm which finds the minimum cost perfect matching cannot be used. An examination of the literature reveals that metaheuristics have been widely used for solving WPP instances. Another idea for the solution is to utilize the power of the today's modern solvers which have gained significant strength in the last decade. In addition, the processor and the computer memories have significantly developed. Therefore, we propose a heuristic strategy which exploits the state-of-the-art solver Gurobi (2017). Solving big-sized instances of WPPVSC is difficult mostly due to the large number of binary variables included in the model. This observation calls to mind to decrease the number of binary decision variables by fixing a subset of them a priori in the model. As a result, a good quality solution can be found relatively easily since the number of binary variables involved would be much lower. Moreover, the obtained solution can be used as a starting point in a further attempt to find a better solution.

In order to decrease the number of binary variables, we concentrate on the cardinality of the set of number of the passes, $|\mathcal{T}|$. We try to compute the lowest possible number of passes that leads to the minimum cost by the following pass iteration idea. Suppose a new parameter ϕ stands for the number of maximum allowable number of passes and let the initial value of ϕ be 2 at the beginning of the algorithm implying that traversing any edge more than twice is not permitted. Hence, values of the y_{ij}^t variables for $(i, j) \in \mathcal{E}, t \in \mathcal{T}, t \geq 3$ are set to 0. Similarly, upper bounds of the x_{ij} variables are set to ϕ. After fixation of the variables, we have a much smaller model which is called as the restricted model in the sequel. We run the solver Gurobi to solve the restricted model for a predetermined amount of computation time, and the solver is expected to find good quality solutions since there are relatively small number of binary variables in the restricted model, especially for small ϕ values. In addition, since we do not aim to prove the optimality but to obtain good feasible solutions of the restricted model at each iteration, we increase the rate of the time that Gurobi uses to find primal feasible solutions. The solution obtained for the restricted model is also feasible for the original model as well. After finding the solution obtained for a specific value of ϕ, its value is increased by 1, i.e., $\phi \leftarrow \phi + 1$ and the solver is rerun for the new value of ϕ. Namely, we increase the upper bound of the x_{ij} variables by 1 and $y_{ij}^{\phi+1}$ variables are set free (their upper bounds are updated from 0 to 1) to form the newly restricted model. We expect an improvement in the objective function value since a slightly more flexible optimization framework is provided by letting the solver determine the values of more decision variables. However, at the same time, the solution of the restricted model becomes harder as the number of binary variables of the model increases with ϕ. Fortunately, we are able to accelerate the solution procedure of the new restricted model by providing the previously found solution as a starting point. Suppose the values of the decision variables coming from the previous restricted model are represented by $\bar{x}_{ij}$ and $\bar{y}_{ij}^t$ for $(i, j) \in \mathcal{E}, t \in \mathcal{T}$. We start x_{ij} and y_{ij}^t variables of the new restricted model from $\bar{x}_{ij}$ and $\bar{y}_{ij}^t$ values using the START attribute of the Gurobi solver. Namely, the comment line $x_{i,j}$.Set(GRB. DoubleAttr.Start, $\bar{x}_{i,j}$) and $y_{i,j}^t$.Set(GRB.DoubleAttr.Start, $\bar{y}_{i,j}^t$) are written for $(i, j) \in \mathcal{E}, t \in \mathcal{T}$ before running Gurobi for the solution of the new restricted model. Hence, the new restricted model will start from a good feasible solution coming from the previous restricted model, and hence, the solution procedure is likely to be more effective. The process continues until the improvement in the objective value after a unit increase in the value of ϕ becomes less than or equal to a final precision parameter ϵ, or the total solution time exceeds the predetermined time limit. Steps of the algorithm are formally summarized in Algorithm 1. Note that $O^{(\phi)}$ given in Algorithm 1 represents the objective function value of the solution found for a given value of ϕ. We name the algorithm as pass iteration heuristic.

Algorithm 1 PASS ITERATION HEURISTIC

Let $\phi = 2$, $DIF = 100$ and initiate value of ϵ
while ($DIF > \epsilon$ and time limit is not exceeded) **do**
- Let solver run for ϕ^{th} restricted model for a small amount of time, use solution found at the previous iteration to speed up the solution process
- Set $O^{(\phi)}$ to the best objective value obtained by the solver
- Set $DIF = O^{(\phi)} - O^{(\phi-1)}$, $\phi \leftarrow \phi + 1$

end while
Report the final solution and corresponding objective value

5.2 Lagrangian heuristic

Another approach suggests relaxing the coupling constraints in an attempt to ease the solution of the WPPVSC. Observe that constraints (4) and (5) include both continuous variables x_{ij} and binary variables y_{ij}. We relax constraints (4) and (5) in order to decompose the model to easy to solve subproblems one of which only contains continuous variables while the other solely depends on the binary variables. We carry the relaxed constraints to the objective function by multiplying with nonnegative Lagrange multipliers α and β. We let WPPVSC(α, β) denote the Lagrangian subproblem

WPPVSC(α, β):

$$LB(\alpha,\beta) = \min \sum_{(i,j)\in\mathcal{E}} \sum_{t\in\mathcal{T}} d_{ij}^t y_{ij}^t - \sum_{(i,j)\in\mathcal{E}} \sum_{t\in\mathcal{T}} \alpha_{ijt}\left(x_{ij} - t y_{ij}^t\right) - \sum_{(i,j)\in\mathcal{E}} \beta_{ij}\left(\sum_{t\in\mathcal{T}} t y_{ij}^t - x_{ij}\right)$$

s.t. (2), (3), (6), (7) (8)

where $LB(\alpha, \beta)$ denotes the optimal objective function value of the Lagrangian subproblem for a given Lagrange multiplier set $\{\alpha, \beta\}$.

5.2.1 Subproblems

Lagrangian subproblem WPPVSC(α, β) can be decomposed further into two subproblems. The terms of the objective function including only the continuous variables x together with the constraints involving only these variables form a linear programming (LP) subproblem which is easy to solve. The rest of the objective function along with the remaining constraints involving only binary variables y constitutes a binary integer programming subproblem. We call the former subproblem WPPVSC$_1(\alpha, \beta)$ and the latter subproblem WPPVSC$_2(\alpha, \beta)$, which are given below. We let Z_1 and Z_2 denote the optimal objective function values of the subproblems, respectively.

WPPVSC$_1(\alpha, \beta)$:

$$Z_1 = \min \sum_{(i,j)\in\mathcal{E}} \gamma_{ij} x_{ij} \tag{9}$$

s.t. (2), (3),

$$x_{ij} \geq 0 \qquad (i,j) \in \mathcal{E} \tag{10}$$

where $\gamma_{ij} = \beta_{ij} - \sum_{t\in\mathcal{T}} \alpha_{ijt}$ and

WPPVSC$_2(\alpha, \beta)$:

$$Z_2 = \min \sum_{(i,j)\in\mathcal{E}} \sum_{t\in\mathcal{T}} \theta_{ijt} y_{ij}^t$$

s.t. (6) (11)

where $\theta_{ijt} = d_{ij}^t - t\left(\beta_{ij} - \alpha_{ijt}\right)$.

Note that WPPVSC$_1(\alpha, \beta)$ is an LP and hence easily solvable. Besides, WPPVSC$_2(\alpha, \beta)$ can also be solved to optimality easily by inspection as follows. Since y_{ij}^t is binary, it can be set to zero or one. If $\theta_{ijt} \geq 0$, then $y_{ij}^t = 0$ and $\theta_{ijt} < 0$, then $y_{ij}^t = 1$ for $(i, j) \in \mathcal{E}, t \in \mathcal{T}$ since the objective function is in the direction of minimization.

5.2.2 Subgradient algorithm

The value $LB(\alpha, \beta)$ is a lower bound on the optimal objective value of the original problem WPPVSC for any Lagrange multiplier set $\{\alpha, \beta\}$. To find the best (largest) lower bound, we solve the Lagrangian dual problem

$$Z = \max_{\alpha,\beta\geq 0} LB(\alpha,\beta) \tag{12}$$

by using a subgradient optimization algorithm. At each iteration n of the subgradient optimization procedure, the current lower bound $LB^n(\alpha, \beta)$ is obtained by optimally solving subproblems WPPVSC$_1(\alpha, \beta)$ and WPPVSC$_2(\alpha, \beta)$. Then, Lagrange multipliers α^n, β^n are updated using the best available upper bound UB^*. Upper bounds UB^n are computed at each iteration n by constructing a feasible solution of the original problem from the solution of the subproblems, details of which are given in the next subsection. We report the UB^* and LB^* as the outputs of the subgradient procedure. We use three termination criteria for the procedure. The first one stops the subgradient algorithm if $UB^* - LB^* < \epsilon_1$ for a given small positive value. Second termination criterion depends on the size of the step size parameter π, which is employed in the subgradient algorithm. If there is no improvement in the value of best lower bound LB^* for consecutive N number of iterations, the value of π is halved. When π becomes smaller than a threshold level ϵ_2, the algorithm is stopped. An observation reveals that convergence of the subgradient algorithm slows down toward the end of the procedure. Hence, another criterion suggests putting an upper bound on the number of iterations in order not to spend too much time toward the end of the algorithm. The upper bound on the number of iterations is called *iterlim* in the body of the algorithm given below.

Although the subgradient algorithm is theoretically quite promising, its practical efficiency is prone to serious limitations due to the long computation times required by WPPVSC$_1(\alpha, \beta)$, especially for large instances. Since WPPVSC$_2(\alpha, \beta)$ is solved by inspection, it generally requires less than 1 minute even for the largest instances with 3000 nodes. On the other hand, the size of the LP subproblem WPPVSC$_1(\alpha, \beta)$ becomes too large to be solved recurrently in the body of the Lagrangian heuristic numerous times. Hence, we are still in need of making the problem size smaller. Besides, one can easily observe that if there is a feasible solution vector x for WPPVSC$_1(\alpha, \beta)$ with negative objective function value, values of the x variables can be increased indefinitely to produce an unbounded objective value. This surpasses Z_2 value coming from WPPVSC$_2(\alpha, \beta)$ and makes LB^n minus infinity and consequently slows the convergence rate of the subgradient algorithm. In order to make the problem size smaller and to put an upper bound on the x variables, which prevents WPPVSC$_1(\alpha, \beta)$ from being unbounded, we make use of the period iteration idea which is explained in the previous section. Namely, we begin the procedure as if the number of passes is limited by 2, i.e., $\phi = 2$, and we increase the limit of the number of passes by one after the convergence of the inner Lagrangian heuristic, until no improvement is observed in the reported feasible solution. Moreover, the value of ϕ is put as an upper bound for the value of x variables in the body of WPPVSC$_1(\alpha, \beta)$ which will prevent unbounded solutions, and speed up the convergence of LB^n. Therefore, the pass iteration idea is the backbone of both heuristics. The steps of the subgradient algorithm integrated with the pass iteration idea are summarized in Algorithm 2. Note that $O^{(\phi)}$ existing in the body of Algorithm 2 now represents the objective function value of the best feasible solution found by the Lagrangian heuristic for step ϕ.

Algorithm 2 LAGRANGIAN HEURISTIC

Let $\phi = 2$, $DIF = 100$ and initiate value of ϵ
while ($DIF > \epsilon$ and time limit is not exceeded) **do**
  Initialization: Set iteration counter $n = 0$, $\pi^0 = 2$, $LB^* = 0$, $UB^* = \infty$, $\alpha^n_{ijt}, \beta^n_{ij} = 0$ for all i, j, t
  while ($UB^* - LB^* \geq \epsilon_1$ and $\pi \geq \epsilon_2$ and $n \leq iterlim$) **do**
  – Solve WPPVSC$_1(\alpha, \beta)$ and WPPVSC$_2(\alpha, \beta)$, compute $LB^n = Z_1^n + Z_2^n$ and update $LB^* = \max\{LB^*, LB^n\}$
  – If LB^* is not updated during N iterations, set $\pi \leftarrow \pi/2$
  – Construct a feasible solution with objective value UB^n and update $UB^* = \min\{UB^*, UB^n\}$
  – Update Lagrange multipliers α and β:
  $\alpha^{n+1}_{ijt} = \max\left\{0, \alpha^n_{ijt} + \kappa^n\left(x_{ij} - ty^t_{ij}\right)\right\}$,
  $\beta^{n+1}_{ij} = \max\left\{0, \beta^n_{ij} + \kappa^n\left(\sum_{t\in T} ty^t_{ij} - x_{ij}\right)\right\}$ where $\kappa^n = \frac{\pi(UB^* - LB^n)}{A}$ with $A = \sum_{(i,j)\in E}\sum_{t\in T}\left[x_{ij} - ty^t_{ij}\right]^2 + \sum_{(i,j)\in E}\left[\sum_{t\in T} ty^t_{ij} - x_{ij}\right]^2$
  – $n \leftarrow n + 1$
  end while
  – Set $O^{(\phi)}$ to the best objective value obtained by the inner loop for step ϕ
  – Set $DIF = O^{(\phi)} - O^{(\phi-1)}$, $\phi \leftarrow \phi + 1$
end while
Report the final solution and corresponding objective value

5.2.3 Generating a feasible solution

As mentioned before, at each iteration, Lagrangian heuristic constructs a feasible solution from the solution of the Lagrangian problem. Since constraints (4) and (5) are relaxed in the Lagrangian problem, values of x and y variables coming from WPPVSC$_1(\alpha, \beta)$ and WPPVSC$_2(\alpha, \beta)$ (say $\bar{x}$ and $\bar{y}$) are not expected to be feasible for the original WPPVSC(α, β) problem. Nevertheless, a feasible y variable vector can be obtained near the solution of WPPVSC$_2(\alpha, \beta)$. Once the values corresponding to the $\bar{y}$ vector is set in the original problem WPPVSC(α, β), the model reduces to an LP and can easily be solved to optimality in order to generate feasible x vector as well. The approach we employ for finding the feasible solution depends on that idea. First of all, values of the y variables are set to the $\bar{y}$ values in the original WPPVSC(α, β) to reduce it to an LP. If the LP is feasible, then the solution of it (which will give x values), together with the $\bar{y}$ values coming from the solution of WPPVSC$_2(\alpha, \beta)$, constitutes a feasible solution for the original problem. On the other hand, if the LP is infeasible, this means it is impossible to satisfy the relaxed constraints with the $\bar{y}$ values obtained from WPPVSC$_2(\alpha, \beta)$. In such a case, instead of setting the values of y variables in the original problem WPPVSC(α, β) with the ones coming from WPPVSC$_2(\alpha, \beta)$, we initiate them from the values coming from WPPVSC$_2(\alpha, \beta)$ by Gurobi's START parameter and we let the solver run for a short time. By doing so, we let the solver to refine the $\bar{y}$ values coming from WPPVSC$_2(\alpha, \beta)$ so that they become feasible for the original problem. Besides, since a short computation time is allowed for the solver, it will probably generate feasible y values that are in a sense close to the infeasible ones obtained from the solution of WPPVSC$_2(\alpha, \beta)$. These steps are formally summarized in Algorithm 3.

6 Computational results

In this section, the selection of the parameters of the PPVSC formulation is given first, and then the efficiency and accuracy of PIH and LH are illustrated on extensive test instances.

Algorithm 3 Generation of a feasible solution

- Let $\bar{y}$ be the vector coming from WPPVSC$_2(\alpha, \beta)$
- Set $y_{ij}^t \leftarrow \bar{y}_{ij}^t$ for all $(i, j) \in E$, $t \in T$ using Gurobi.DoubleAttr.Set feature of the Gurobi to reduce WPPVSC(α, β) to an LP
- If LP is feasible solve it to obtain UB^n and x_{ij} values which will produce a feasible solution with y_{ij}^t values
- If LP is infeasible, instead of setting the values of y_{ij}^t variables (with the ones of $\bar{y}_{ij}^t$), let Gurobi initiate them from $\bar{y}_{ij}^t$ values using Gurobi.DoubleAttr.Start feature of the Gurobi
- Let Gurobi run for a short time for WPPVSC(α, β) (having y_{ij}^t values initiated from the values of $\bar{y}_{ij}^t$) to generate a feasible solution close to the solution of WPPVSC$_2(\alpha, \beta)$ if possible
- Report UB^n if a feasible solution is found

6.1 Selection of the parameters

We use 5 different sets of test instances. First set, which we call test bed 1, is due to Golden et al. (1983) including 23 problems denoted by gdb1,...,gdb23. Second set, named test bed 2, is given by Benavent et al. (1992). It includes 34 problems which are denoted as val1A,...,val10D. Third set, entitled as test bed 3, includes 24 problems denoted by egl-e1-A,...,egl-s4-C and is provided in the paper by Li and Eglese (1996). Problem size slightly increases as we go from test bed 1 to test bed 2, and from test bed 2 to test bed 3. These three sets are originally developed for capacitated arc routing problems. We neglect the capacities and take only the arc traveling costs as the initial service costs. We generate the fourth set in order to have denser instances and call it test bed 4. Node places are randomly chosen so that the resulting network is a connected one. Vertices that are closer than 15 meters are accepted as neighbors, and the initial service costs are equated to the length of the edges. There are 12 problems in test bed 4 that are called as 100-1,...,400-3. Finally, the fifth set named test bed 5, generated for WPP, is given in a website (http://www.uv.es/corberan/instancias.htm) and includes 120 problems denoted WA0531,...,WB3065. We take values of the first pass costs, namely c_{ij}^1 parameters, from the sets. Values of the other cost parameters, i.e., c_{ij}^k $k \in \mathcal{T}, k \geq 2$, are generated using the formula $c_{ij}^k = \left\lceil \frac{c_{ij}^{k-1}}{2} \right\rceil + \text{Rand}(\frac{c_{ij}^{k-1}}{2})$ where Rand(n) denotes a real number randomly generated within the interval $(0,n)$. Note that c_{ij}^k values compulsorily possess a nonincreasing structure, and $c_{ij}^k > 0$ for any k by this selection mechanism.

6.2 Accuracy and efficiency of heuristics

In this section, we assess the performance of PIH and LH by comparing the minimum costs found by them with those

Table 2 Performance of Gurobi, PIH and LH on test bed 1

Name	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
gdb1	12	22	272	8.54	0.00	272	1.04	272	19.87
gdb2	12	26	296	4.45	0.00	296	0.40	296	16.19
gdb3	12	22	235	3.94	0.00	235	0.42	235	87.49
gdb4	11	19	241	2.27	0.00	241	0.24	241	36.33
gdb5	13	26	318	6.89	0.00	318	2.20	318	65.00
gdb6	12	22	263	2.33	0.00	263	0.28	263	36.08
gdb7	12	22	256	6.18	0.00	256	0.62	256	36.20
gdb8	27	46	239	17.94	0.00	239	3.05	239	640.41
gdb9	27	51	226	43.56	0.00	226	10.91	226	800.00
gdb10	12	25	247	6.07	0.00	247	0.44	247	16.41
gdb11	22	45	341	37.09	0.00	341	7.24	341	378.65
gdb12	13	23	341	12.10	0.00	341	1.29	341	31.62
gdb13	10	28	467	4.27	0.00	467	0.16	467	23.53
gdb14	7	21	91	0.17	0.00	91	0.05	91	0.21
gdb15	7	21	51	0.12	0.00	51	0.05	51	0.10
gdb16	8	28	114	3.69	0.00	114	0.43	114	28.22
gdb17	8	28	81	3.68	0.00	81	3.65	81	58.60
gdb18	9	36	131	0.48	0.00	131	0.11	131	0.19
gdb19	8	11	54	0.17	0.00	54	0.04	54	2.59
gdb20	11	22	109	2.16	0.00	109	0.16	109	6.43
gdb21	11	33	139	2.57	0.00	139	0.21	139	15.37
gdb22	11	44	184	4.70	0.00	184	0.35	184	24.10
gdb23	11	55	200	0.62	0.00	200	0.15	200	0.58

Table 3 Performance of Gurobi, PIH and LH on test bed 2

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
val1A	24	39	147	22.04	0.00	147	3.98	147	117.65
val1B	24	39	160	42.18	0.00	160	8.88	160	251.89
val1C	24	39	149	23.14	0.00	149	4.69	149	346.14
val2A	24	34	198	14.70	0.00	198	6.19	198	187.63
val2B	24	34	197	15.00	0.00	197	0.49	197	51.37
val2C	24	34	206	15.13	0.00	206	1.01	206	62.80
val3A	24	35	77	6.04	0.00	77	0.75	77	40.32
val3B	24	35	77	12.76	0.00	77	0.89	77	44.02
val3C	24	35	75	7.35	0.00	75	0.49	75	51.19
val4A	41	69	343	58.21	0.00	343	10.76	343	811.89
val4B	41	69	345	63.29	0.00	345	11.17	345	372.92
val4C	41	69	330	55.25	0.00	330	12.83	330	1422.40
val4D	41	69	330	222.67	0.00	330	217.99	330	3602.35
val5A	34	65	340	683.85	0.00	340	322.99	340	1512.14
val5B	34	65	380	1382.08	0.00	380	851.04	380	3299.50
val5C	34	65	365	102.59	0.00	365	70.62	365	834.89
val5D	34	65	363	355.28	0.00	363	296.53	363	2375.12
val6A	31	50	191	28.93	0.00	191	22.51	191	883.63
val6B	31	50	202	28.49	0.00	202	12.34	202	522.85
val6C	31	50	201	32.16	0.00	201	8.59	201	245.58
val7A	40	66	247	19.29	0.00	247	4.87	247	593.66
val7B	40	66	243	23.80	0.00	243	8.07	243	1871.66
val7C	40	66	252	32.13	0.00	252	3.05	252	206.30
val8A	30	63	329	34.59	0.00	329	21.26	329	1049.56
val8B	30	63	349	115.14	0.00	349	148.41	349	1772.29
val8C	30	63	353	228.96	0.00	353	293.17	353	2360.08
val9A	50	92	285	298.19	0.00	285	487.51	285	3605.18
val9B	50	92	288	1130.00	0.00	288	692.44	288	3604.42
val9C	50	92	289	536.19	0.00	289	679.51	289	2976.42
val9D	50	92	292	834.49	0.00	292	519.44	292	3602.25
val10A	50	97	387	654.30	0.00	387	396.99	387	3523.46
val10B	50	97	388	332.85	0.00	388	164.40	388	2495.59
val10C	50	97	374	352.60	0.00	374	559.97	374	2444.45
val10D	50	97	376	2013.45	0.00	376	475.60	376	3608.37

found by the state-of-the-art MILP solver Gurobi (2017) on the described test sets. We code PPVSC, PIH and LH in Visual Studio environment by C# language and we carry out all experiments using a single Intel i7-4770 core. We let Gurobi, PIH and LH run for at most 1 hour for each of the test problems instances and the minimum objective function values found in the allowed computation times are reported. We, respectively, tabulate the results corresponding to test beds 1, 2, 3 and 4 Tables 2, 3, 4 and 5. Results corresponding to test bed 5 are tabulated in Tables 6 and 7. The first column of the tables includes the instance name, while the second and third columns keep the numbers of vertices and the edges (which are denoted by $|\mathcal{V}|$ and $|\mathcal{E}|$), respectively. Similarly, the fourth, fifth and sixth columns keep the objective value of the best solution found by Gurobi, cpu time Gurobi uses, and the percent deviation between the best solution found by Gurobi and the best possible objective function value reported by Gurobi (which are denoted by O^{G}, T^{G} and Gap), respectively. In addition, the seventh and eighth columns, respectively, keep the objective value of the best solution found by PIH and the cpu time it uses (which are denoted by O^{PIH} and T^{PIH}). Finally, the best possible objective function value found by LH, and the cpu time LH uses (which are denoted by O^{LH} and T^{LH}) are given under columns ninth and tenth, respectively.

As can be understood from Table 2, Gurobi, PIH and LH are able to find the optimal solution for each instance of test bed 1. However, the average CPU time used by Gurobi is

Table 4 Performance of Gurobi, PIH and LH on test bed 3

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
egl-e1-A	77	98	3245	521.11	0.00	3245	122.84	3245	1402.03
egl-e1-B	77	98	3170	952.39	0.00	3170	147.15	3170	2037.39
egl-e1-C	77	98	2884	2176.99	0.00	2884	231.94	2884	2962.77
egl-e2-A	77	98	3149	110.76	0.00	3149	750.73	3149	3605.55
egl-e2-B	77	98	3185	498.80	0.00	3185	373.34	3185	3207.93
egl-e2-C	77	98	3199	185.68	0.00	3199	21.77	3199	1017.33
egl-e3-A	77	98	3159	259.16	0.00	3159	96.77	3159	3601.11
egl-e3-B	77	98	3087	3600.02	1.36	3077	615.17	3077	3606.81
egl-e3-C	77	98	2885	108.56	0.00	2885	124.90	2885	1416.23
egl-e4-A	77	98	3169	626.68	0.00	3169	484.53	3169	3602.27
egl-e4-B	77	98	3126	1436.33	0.00	3126	160.18	3126	2108.84
egl-e4-C	77	98	3400	153.94	0.00	3400	39.22	3400	1185.61
egl-s1-A	140	190	4977	3600.02	0.44	4977	926.75	4977	3620.54
egl-s1-B	140	190	4805	969.26	0.00	4805	748.08	4805	3603.62
egl-s1-C	140	190	5140	3118.91	0.00	5140	903.73	5140	3616.98
egl-s2-A	140	190	4905	3600.52	0.31	4905	758.82	4905	3625.02
egl-s2-B	140	190	4970	1056.90	0.00	4970	1295.19	4974	3607.69
egl-s2-C	140	190	4710	3600.02	0.42	4710	784.55	4710	3617.10
egl-s3-A	140	190	4679	3600.39	0.49	4679	1587.11	4683	3613.97
egl-s3-B	140	190	4946	3608.86	0.59	4937	873.29	4937	3606.62
egl-s3-C	140	190	4985	3602.55	0.66	4978	808.14	4978	3636.82
egl-s4-A	140	190	4854	3601.72	0.93	4854	1659.16	4855	3615.02
egl-s4-B	140	190	4782	3600.86	1.05	4782	901.80	4782	3652.79
egl-s4-C	140	190	4850	3605.10	0.56	4850	1244.70	4851	3643.07

5.18 times larger than the average CPU time used by PIH. On the other hand, LH uses 13.36 times larger computation time than Gurobi uses. This is mainly due to the fact that LH has to go through thousands of iterations before convergence. Hence, all three methods can be said to be 100% accurate, but PIH is the most efficient one.

Performance of Gurobi, PIH and LH on test bed 2 is similar to their performances on test bed 1. They are all able to find the optimal solution for each instance and the average time used by Gurobi is 1.55 times of the average time used by PIH, and the average time used by LH is 5.19 times of the average time used by Gurobi, as can be observed in Table 3. From the results of test beds 1 and 2, one may extract that although LH is able to find the optimal solution for each instance, it uses much more computation time than the other alternatives, implying that both Gurobi and PIH are preferable over LH for small instances.

Story is slightly different for test bed 3. There are 10 instances for which Gurobi is unable to prove the optimality, and for 3 of them PIH and LH are able to find better solutions. However, results of Gurobi are better than those of LH four 4 instances, too. If Gurobi and LH are compared on the basis of the average objective function values, it can be said that LH edges out. Hence, PIH is the most accurate one in three solution alternatives for test bed 3. Moreover, PIH is the most efficient one as well, since the average time Gurobi requires is 3.08 times larger than the required time by PIH while the computation time LH requires is 1.52 times larger than the computation time of Gurobi. These results can be seen in Table 4.

Performance of PIH and LH gets better with the increasing problem size. This observation becomes clearer in the results for test bed 4 given in Table 5. Gurobi and LH spend all the allowable time for each of the instances, while PIH converges before time limit exceeds for 4 instances. On average, times, respectively, spent by Gurobi and LH are 1.21 and 1.22 times more than that of PIH spends. The average objective function value found by Gurobi is 16.20 and 16.37% larger than those of the average objective function values, respectively, found by PIH and LH. Results of LH are slightly better than those of PIH. Besides, PIH results are better than Gurobi results for each of the 12 instances, and better than LH for 5 instances while LH results are better than PIH results for 7 instances implying that PIH and LH clearly dominate Gurobi.

Dominance of PIH and LH over Gurobi becomes even clearer for test bed 5. Not only PIH is more successful than Gurobi for all instances but one of test bed 5 (for one instance both methods produce the optimal solution), costs found by

Table 5 Performance of Gurobi, PIH and LH on test bed 4

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
100-1	100	2475	317021	3602.38	6.88	305730	3601.89	307333	3602.71
100-2	100	1650	209973	3603.20	6.43	204245	3601.80	205256	3601.24
100-3	100	1237	158425	3604.78	7.27	152457	3602.84	153082	3600.78
200-1	200	9950	209416	3645.88	63.36	173817	3036.61	173769	3607.34
200-2	200	6633	141325	3608.72	20.12	115952	3606.36	116126	3604.16
200-3	200	4975	108805	3608.01	22.11	87379	3603.72	87132	3603.41
300-1	300	22425	481052	3640.60	94.77	403254	1462.02	400912	3610.68
300-2	300	14950	325927	3641.01	94.84	267719	2560.67	268213	3607.86
300-3	300	11212	242387	3643.83	94.81	200275	3608.19	200068	3609.36
400-1	400	39900	846706	3678.43	99.98	726444	2056.62	721338	3625.45
400-2	400	26600	571130	3611.05	94.73	482577	3610.45	481733	3613.70
400-3	400	19950	434758	3698.63	94.78	362852	1310.06	362626	3616.69

Table 6 Performance of Gurobi, PIH and LH on test bed 5: WA instances

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
WA0531	500	837	526315	3600.09	3.62	522597	3601.08	518980	3600.93
WA0532	500	813	508309	3600.34	6.33	491922	3600.85	490201	3600.61
WA0535	500	837	497432	3604.55	8.61	476078	3601.20	472106	3600.62
WA0541	500	1028	599841	3600.05	9.79	585447	3600.92	581830	3600.85
WA0542	500	1040	590969	3600.23	9.50	576809	3617.58	571395	3600.59
WA0545	500	1036	596434	3600.71	10.55	572662	3237.24	568513	3600.74
WA0551	500	1270	693002	3600.04	8.82	679055	3604.68	673884	3600.62
WA0552	500	1260	2893862	3600.05	77.52	702787	3611.54	696354	3600.78
WA0555	500	1275	2696731	3600.17	77.75	643733	3641.25	641803	3600.72
WA0561	500	1506	829877	3600.08	8.07	811987	3601.36	804740	3604.29
WA0562	500	1518	3279636	3600.06	77.15	799887	3602.26	794791	3600.90
WA0565	500	1506	3395852	3600.05	79.28	757111	3612.56	751686	3600.83
WA1031	1000	1671	1059972	3600.09	11.44	1041916	3603.78	1034520	3603.32
WA1032	1000	1641	3736045	3602.47	75.68	1015864	3600.32	1011364	3602.81
WA1035	1000	1659	5837968	3600.07	85.82	954437	3600.40	932692	3601.77
WA1041	1000	2052	1217486	3600.09	10.75	1211749	3600.93	1186170	3602.25
WA1042	1000	2068	5207878	3600.14	79.63	2987800	3601.42	1171889	3601.62
WA1045	1000	2064	7715237	3600.11	87.26	1131592	3608.23	1094231	3603.02
WA1051	1000	2545	1477959	3600.13	10.73	1463340	3702.16	1432094	3601.97
WA1052	1000	2535	7350097	3600.23	82.50	2480708	3654.91	1393460	3604.64
WA1055	1000	2525	7336528	3600.12	83.92	1322024	3605.25	1295547	3602.08
WA1061	1000	3018	1703625	3600.74	9.53	1677616	3600.54	1654378	3602.95
WA1062	1000	3012	6524686	3600.18	77.66	1590455	3602.29	1579747	3602.22
WA1065	1000	3012	5872427	3600.19	77.71	5425571	2407.16	1418604	3602.13
WA1531	1500	2478	1474932	3600.13	12.04	1461648	3606.66	1433134	3602.71
WA1532	1500	2505	6954940	3600.33	81.45	1459543	3601.11	1443959	3602.90
WA1535	1500	2517	3093363	3600.18	60.44	1411375	3600.97	1377267	3603.61
WA1541	1500	3136	1743176	3600.69	11.93	1699579	3600.64	1689637	3603.23
WA1542	1500	3144	10336634	3602.12	85.63	5918295	1356.45	1649516	3603.08
WA1545	1500	3136	9996651	3600.26	86.66	4551919	3686.20	1497854	3603.11
WA1551	1500	3825	1997436	3600.14	11.60	1950970	3600.94	1957118	3604.15

Table 6 continued

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
WA1552	1500	3805	9592521	3600.26	81.88	5300897	3690.30	1926235	3603.36
WA1555	1500	3805	11209483	3600.14	85.61	5825372	2566.47	1806734	3605.22
WA1561	1500	4518	2287970	3600.34	10.71	2237133	3600.88	2246242	3604.90
WA1562	1500	4518	15360299	3600.28	86.86	7647217	1210.57	6848054	3603.80
WA1565	1500	4530	15331122	3600.17	87.98	6771771	3205.39	2058273	3603.90
WA2031	2000	3330	1708825	3600.15	9.63	1691668	3600.81	1675073	3605.52
WA2032	2000	3324	4981646	3619.67	68.91	1727972	3605.66	1702635	3605.67
WA2035	2000	3303	4679736	3603.65	67.81	1659809	3600.75	1630306	3607.40
WA2041	2000	4164	8026129	3611.92	77.40	5259674	3611.87	1990598	3604.68
WA2042	2000	4156	10912406	3600.18	84.03	3446495	3600.79	1941857	3605.24
WA2045	2000	4124	4651913	3600.72	65.70	4444667	3627.12	1789054	3604.79
WA2051	2000	5065	2318224	3605.51	11.50	2282783	3600.93	2281921	3606.37
WA2052	2000	5100	13571873	3600.18	85.08	6583986	3610.80	5191846	3605.14
WA2055	2000	5085	12705618	3613.07	85.29	7186907	1207.78	2116000	3605.19
WA2061	2000	6030	2647701	3600.50	10.85	2626700	3600.96	2588047	3604.30
WA2062	2000	6018	13895892	3602.27	83.13	7972703	1811.65	7967479	3605.72
WA2065	2000	6036	16774775	3603.37	87.04	8569951	2427.53	8323327	3605.71
WA3031	3000	5046	2197198	3600.18	7.51	2172332	3600.71	2158868	3609.08
WA3032	3000	4986	2755940	3600.16	26.24	2157508	3601.37	2154480	3608.64
WA3035	3000	5007	3748736	3600.40	48.55	2095780	3600.83	2079312	3611.05
WA3041	3000	6200	2483660	3600.25	9.84	2446106	3600.98	2449366	3610.07
WA3042	3000	6184	7107276	3600.24	69.15	4254863	3600.88	2397035	3609.70
WA3045	3000	6228	6562561	3600.31	67.40	4891235	3600.87	2334723	3609.11
WA3051	3000	7580	2914344	3600.67	11.29	2852102	3602.28	2849108	3612.24
WA3052	3000	7575	13953992	3600.31	82.33	7320404	1597.12	2730075	3610.00
WA3055	3000	7585	14421139	3600.35	83.66	7003448	3601.02	2632234	3610.46
WA3061	3000	9066	3197836	3600.55	10.28	3162669	3601.68	3163197	3607.07
WA3062	3000	9036	17578286	3600.27	83.96	7799893	3601.17	8644888	3610.54
WA3065	3000	9042	17428674	3600.74	84.88	8298485	3172.07	8716880	3610.56

Gurobi is more than 3, 4, 5, and even 6 times of the ones found by PIH for a large number of instances. For example, costs found by Gurobi are 3.68, 4.12, 5.55 and 6.12 times of the cost found by PIH, respectively, for instances WA1032, WA0552, WA1055 and WA1035. On the average, cost of the results found by Gurobi is 1.94 times larger than the cost associated with PIH results. Besides, Gurobi uses all the allowable time for all instances but one while PIH converges before the allowable time limit for several instances. On the other hand, LH outperforms PIH for most of the instances. PIH produces better results for only 6 WA instances and 3 WB instances while LH is better for remaining 54 WA instances and 57 WB instances. Average costs for respective WA and WB instances found by LH are given by 2170222 and 76642, while they are, respectively, 3067783 and 83335.6 for PIH results implying that LH is much more successful than PIH for test bed 5 instances. As a concluding remark, we can say that PIH is more successful than Gurobi for PPVSC instances and its success becomes clearer with increasing problem size. In addition, LH is competitive with PIH for small- and moderate-sized instances, and it becomes more and more successful as the problem size gets larger.

7 Conclusion and future research paths

This paper presents a mathematical formulation for the postman problem with variable service costs. It is assumed that the travel costs of the edges depend on the number of passes and they tend to decrease with each pass. The problem is named Chinese postman problem with variable service costs if the service costs are symmetric, and it is named as windy postman problem with variable service costs, otherwise. CPPVSC is solvable in polynomial time, and it is shown in the paper that no edge is traversed more than twice in the optimal solution. On the other hand, WPPVSC is NP-hard since it can be reduced to WPP which is known to be NP-hard. Two heuristics named PIH and LH are proposed for the solution

Table 7 Performance of Gurobi, PIH and LH on test bed 5: WB instances

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
WB0531	500	881	37258	3627.97	4.55	36768	2403.44	36600	3600.60
WB0532	500	874	40452	3600.60	21.90	36920	3601.45	36871	3600.97
WB0535	500	887	34752	3601.68	4.58	34528	1802.55	34369	3600.47
WB0541	500	1115	48368	3602.93	10.92	47053	3600.95	46680	3600.97
WB0542	500	1124	69094	3600.23	37.79	47412	3603.50	47016	3600.96
WB0545	500	1121	45567	3601.59	12.17	43354	3608.00	43157	3600.92
WB0551	500	1325	63345	3602.89	9.73	62184	3610.79	61416	3601.36
WB0552	500	1309	87103	3600.32	37.80	59033	3600.90	58556	3600.72
WB0555	500	1319	102448	3601.09	51.37	54557	3601.53	54420	3600.74
WB0561	500	1555	79196	3600.24	7.76	77632	3601.05	77088	3601.46
WB0562	500	1517	245114	3600.46	72.83	71063	3601.34	70600	3601.17
WB0565	500	1526	361310	3600.99	83.15	65613	3600.75	65401	3600.82
WB1031	1000	1757	87352	3602.94	49.16	51768	3601.99	51404	3601.92
WB1032	1000	1743	255903	3603.62	83.28	50901	3601.45	50442	3601.50
WB1035	1000	1752	236405	3657.20	82.93	48195	3604.75	47834	3603.17
WB1041	1000	2243	70643	3610.02	11.58	69218	3611.78	68300	3602.42
WB1042	1000	2182	355016	3604.84	83.27	66368	3610.54	65768	3601.71
WB1045	1000	2252	414237	3625.11	86.59	62313	3602.76	62201	3603.17
WB1051	1000	2659	275569	3607.24	71.44	86235	3631.58	85424	3601.93
WB1052	1000	2670	543570	3660.82	85.74	85518	3603.86	84444	3602.23
WB1055	1000	2653	344386	3606.07	79.74	78810	3602.23	77179	3601.91
WB1061	1000	3098	108016	3693.42	8.26	105606	3602.85	105510	3601.81
WB1062	1000	3088	360918	3614.79	73.09	104675	3725.23	103930	3603.80
WB1065	1000	3110	360574	3600.33	75.93	94128	3605.90	93553	3602.80
WB1531	1500	2631	66616	3642.08	26.19	57043	3862.18	56614	3603.11
WB1532	1500	2647	281796	3605.60	82.41	58054	3606.20	57497	3602.81
WB1535	1500	2654	74193	3610.34	36.43	55779	3602.64	55732	3602.91
WB1541	1500	3352	279758	3627.18	75.90	76054	3602.36	74712	3606.73
WB1542	1500	3334	364169	3629.08	81.61	75346	3606.65	74340	3603.31
WB1545	1500	3346	453199	3732.50	86.30	70851	3600.61	70446	3603.69
WB1551	1500	3983	603453	3868.82	85.45	97263	3635.73	96457	3603.22
WB1552	1500	3999	596189	3669.33	85.63	95482	3600.77	94256	3604.20
WB1555	1500	3998	384511	3602.87	79.64	89730	3618.80	87530	3606.92
WB1561	1500	4585	641210	3629.62	83.28	298495	3612.53	115549	3604.47
WB1562	1500	4656	438045	3614.09	75.89	116087	3603.46	114062	3604.72
WB1565	1500	4670	662906	3728.98	85.42	108206	3600.85	105882	3604.48
WB2031	2000	3503	64439	3661.75	13.17	62753	3604.16	62678	3603.91
WB2032	2000	3467	63848	3620.75	16.03	61392	3010.78	61382	3605.25
WB2035	2000	3498	69739	3651.03	24.16	60433	3600.89	60443	3605.69
WB2041	2000	4399	85890	3645.78	18.86	77887	3600.79	77206	3606.20
WB2042	2000	4406	401985	3608.14	82.82	77357	3600.93	76801	3606.77
WB2045	2000	4445	249878	3603.22	73.67	74553	3600.99	74286	3604.96
WB2051	2000	5288	436244	3635.65	79.78	98074	3601.17	96857	3608.35
WB2052	2000	5301	351784	3630.81	75.52	179479	3601.40	95677	3607.38
WB2055	2000	5296	282865	3687.69	71.65	90808	3600.95	89866	3606.57
WB2061	2000	6090	699231	3701.12	84.86	115859	3601.20	115446	3605.90
WB2062	2000	6129	315699	3761.49	67.08	215765	3601.14	113240	3606.18

Table 7 continued

Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	O^{G}	T^{G}	Gap	O^{PIH}	T^{PIH}	O^{LH}	T^{LH}
WB2065	2000	6093	678693	3601.94	85.80	107448	3603.71	106749	3606.60
WB3031	3000	5176	61959	2796.42	0.00	61959	1174.07	61960	3605.62
WB3032	3000	5202	61555	3611.40	0.25	61503	1680.15	61503	3600.19
WB3035	3000	5223	63120	3604.97	0.21	63113	1420.17	63128	3622.22
WB3041	3000	6539	161688	3600.43	54.08	81867	3601.03	81845	3608.57
WB3042	3000	6538	82061	3600.61	12.45	78617	3601.48	78533	3609.08
WB3045	3000	6603	81759	3600.39	16.02	75887	3076.89	75865	3611.43
WB3051	3000	7838	100367	3605.52	9.54	98571	3601.09	97806	3613.60
WB3052	3000	7776	105182	3600.61	18.42	94299	3602.03	94085	3611.76
WB3055	3000	7761	109642	3600.48	23.45	92296	3602.04	92219	3610.86
WB3061	3000	8975	113260	3600.41	9.67	110949	3601.24	110287	3611.05
WB3062	3000	9085	142357	3600.65	26.78	115023	3602.70	113568	3611.35
WB3065	3000	8973	333780	3600.72	71.17	106002	3601.26	105826	3617.95

of WPPVSC instances. In PIH, Gurobi runs for a predetermined amount of time and for a given number of allowable passes at each iteration and the number of allowable passes is increased by one at the next iteration in which the solution obtained at the previous iteration is used as a starting point. In LH, two constraints are relaxed and carried to the objective function after multiplied by positive Lagrangian multipliers and the multipliers are optimized through a subgradient algorithm. A feasible solution is obtained in each step of the algorithm making use of the subproblem solutions. Results of PIH and LH are compared with results of Gurobi, and it is observed that although using the solver as the solution method for small sized WPPVSC instances is possible, PIH dominates Gurobi for instances with moderate and large sizes, and LH outperforms PIH for large instances.

This work can be extended in several directions. First of all, variable service cost idea can be imposed on other variants of PP such as capacitated PP, PP with multiple vehicles, or hierarchical PP. Another idea is to analyze the PP with stochastic service costs. Finally, PP with multiple depots can be addressed under variable service cost assumption.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Assad AA, Golden BL (1995) Arc routing methods and applications. Handb Oper Res Manag Sci 8:375–483

Ávila T, Corberán Á, Plana I, Sanchis JM (2016) A branch-and-cut algorithm for the profitable windy rural postman problem. Eur J Oper Res 249(3):1092–1101

Benavent E, Campos V, Corberán A, Mota E (1992) The capacitated arc routing problem: lower bounds. Networks 22(7):669–690

Black D, Eglese R, Wøhlk S (2013) The time-dependent prize-collecting arc routing problem. Comput Oper Res 40(2):526–535

Campbell JF, Langevin A (2000) Roadway snow and ice control. In: Dror M (ed) Arc routing. Springer, Berlin, pp 389–418

Cordeau JF, Ghiani G, Guerriero E (2012) Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem. Transp Sci 48(1):46–58

Donati AV, Montemanni R, Casagrande N, Rizzoli AE, Gambardella LM (2008) Time dependent vehicle routing problem with a multi ant colony system. Eur J Oper Res 185(3):1174–1191

Dror M (2012) Arc routing: theory, solutions and applications. Springer, Dordrecht

Dussault B, Golden B, Groër C, Wasil E (2013) Plowing with precedence: a variant of the windy postman problem. Comput Oper Res 40(4):1047–1059

Eglese R, Li L (1992) Efficient routeing for winter gritting. J Oper Res Soc 43(11):1031–1034

Gendreau M, Ghiani G, Guerriero E (2015) Time-dependent routing problems: a review. Comput Oper Res 64:189–197

Golden BL, DeArmon JS, Baker EK (1983) Computational experiments with algorithms for a class of routing problems. Comput Oper Res 10(1):47–59

Guan M (1984) On the windy postman problem. Discrete Appl Math 9(1):41–46

Gurobi optimizer 6.0.: high-end libraries for math programming (2017). http://www.gurobi.com/. Accessed Jan 2017

Hertz A (2005) Recent trends in arc routing. In: Golumbic MC, Hartman IBA (eds) Graph theory, combinatorics and algorithms. Springer, Berlin, pp 215–236

Ichoua S, Gendreau M, Potvin JY (2003) Vehicle dispatching with time-dependent travel times. Eur J Oper Res 144(2):379–396

Koç Ç, Bektaş T, Jabali O, Laporte G (2016) Thirty years of heterogeneous vehicle routing. Eur J Oper Res 249(1):1–21

Li LY, Eglese RW (1996) An interactive algorithm for vehicle routeing for winter-gritting. J Oper Res Soc 47(2):217–228

Li F, Golden B, Wasil E (2005) Solving the time dependent traveling salesman problem. In: Golden B, Raghavan S (eds) The next wave in computing, optimization, and decision technologies. Springer, Berlin, pp 163–182

Malandraki C, Daskin MS (1992) Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. Transp Sci 26(3):185–200

Malandraki C, Dial RB (1996) A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. Eur J Oper Res 90(1):45–55

Schneider J (2002) The time-dependent traveling salesman problem. Phys A Stat Mech Appl 314(1):151–155

Setak M, Habibi M, Karimi H, Abedzadeh M (2015) A time-dependent vehicle routing problem in multigraph with fifo property. J Manuf Syst 35:37–45

Sun J, Meng Y, Tan G (2015) An integer programming approach for the chinese postman problem with time-dependent travel time. J Comb Optim 29(3):565–588

Tagmouti M, Gendreau M, Potvin JY (2007) Arc routing problems with time-dependent service costs. Eur J Oper Res 181(1):30–39

Tagmouti M, Gendreau M, Potvin JY (2010) A variable neighborhood descent heuristic for arc routing problems with time-dependent service costs. Comput Ind Eng 59(4):954–963

Tagmouti M, Gendreau M, Potvin JY (2011) A dynamic capacitated arc routing problem with time-dependent service costs. Transp Res Part C Emerg Technol 19(1):20–28

Taş D, Gendreau M, Jabali O, Laporte G (2016) The traveling salesman problem with time-dependent service times. Eur J Oper Res 248(2):372–383

Test instances for arc routing problems. http://www.uv.es/corberan/instancias.htm (2017). Accessed Jan 2017

Vincent FY, Lin SW (2015) Iterated greedy heuristic for the time-dependent prize-collecting arc routing problem. Comput Ind Eng 90:54–66