ARTICLE IN PRESS

Computers & Operations Research ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computers & Operations Research

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

An ILP improvement procedure for the Open Vehicle Routing Problem

Majid Salari, Paolo Toth *, Andrea Tramontani

DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

ARTICLE INFO

Keywords:
Integer Linear Programming
Local search
Heuristics
Open Vehicle Routing Problem

ABSTRACT

We address the Open Vehicle Routing Problem (OVRP), a variant of the "classical" (capacitated and distance constrained) Vehicle Routing Problem (VRP) in which the vehicles are not required to return to the depot after completing their service. We present a heuristic improvement procedure for OVRP based on Integer Linear Programming (ILP) techniques. Given an initial feasible solution to be possibly improved, the method follows a destruct-and-repair paradigm, where the given solution is randomly destroyed (i.e., customers are removed in a random way) and repaired by solving an ILP model, in the attempt of finding a new improved feasible solution. The overall procedure can be considered as a general framework which could be extended to cover other variants of Vehicle Routing Problems. We report computational results on benchmark instances from the literature. In several cases, the proposed algorithm is able to find the new best known solution for the considered instances.

1. Introduction

We address the Open Vehicle Routing Problem (OVRP), a variant of the "classical" (capacitated and distance constrained) Vehicle Routing Problem (VRP) in which the vehicles are not required to return to the depot after completing their service. OVRP can be formally stated as follows. We are given a central *depot* and a set of n *customers*, which are associated with the nodes of a complete undirected graph $G=(V,E)$ (where $V=\{0,1,\dots,n\}$, node 0 represents the depot and $V\backslash\{0\}$ is the set of customers). Each edge $e \in E$ has an associated finite *cost* $c_e \geq 0$ and each customer $v \in V\backslash\{0\}$ has a *demand* $q_v > 0$ (with $q_0=0$). A fleet of m identical *vehicles* is located at the depot, each one with a *fixed cost* F, a *capacity* Q and a *total distance-traveled* (*duration*) limit D. The customers must be served by at most m Hamiltonian paths (*open routes*), each path associated with one vehicle, starting at the depot and ending at one of the customers. Each route must have a duration (computed as the sum of the edge costs in the route) not exceeding the given limit D of the vehicles, and can visit a subset S of customers whose total demand $\sum_{v \in S} q_v$ does not exceed the given capacity Q. The problem consists of finding a feasible solution covering (i.e., visiting) exactly once all the customers and having a minimum overall cost, computed as the sum of the traveled edge costs plus the fixed costs associated with the vehicles used to serve the customers. OVRP is known to be $\mathcal{NP}$-hard in the strong sense, as it generalizes the Bin Packing Problem and the Hamiltonian Path Problem.

In this paper we present a heuristic improvement procedure for OVRP based on Integer Linear Programming (ILP) techniques. Given an initial feasible solution to be possibly improved, the procedure iteratively performs the following steps: (a) randomly select several customers from the current solution, and build the restricted solution obtained from the current one by extracting (i.e., short-cutting) the selected customers; (b) reallocate the extracted customers to the restricted solution by solving an ILP problem, in the attempt of finding a new improved feasible solution. This method has been proposed by De Franceschi et al. [7] and deeply investigated by Toth and Tramontani [27] in the context of the classical VRP. Here, we consider a simpler version of this approach, which exploits no particular feature of the addressed problem. The method follows a destruct-and-repair paradigm, where the current solution is randomly destroyed (i.e., customers are removed in a random way) and repaired by following ILP techniques. Hence, the overall procedure can be considered as a general framework which could be extended to cover other variants of Vehicle Routing Problems.

The notion of using ILP techniques to improve a feasible solution of a combinatorial optimization problem has emerged in several papers in the last few years. Addressing the split delivery VRP, Archetti et al. [2] developed a heuristic algorithm that integrates tabu search with ILP by solving integer programs to explore promising parts of the solution space identified by a tabu search heuristic. A similar approach has been presented by Archetti et al. [1] for an inventory routing problem. Hewitt et al. [15] proposed to solve the capacitated fixed charge network flow problem by combining exact and heuristic approaches. In this

* Corresponding author. Tel.: +39 051 2093028; fax: +39 051 2093073.
E-mail addresses: majid.salari2@unibo.it (M. Salari), paolo.toth@unibo.it (P. Toth), andrea.tramontani@unibo.it (A. Tramontani).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.02.010

case as well a key ingredient of the method is to use ILP to improve feasible solutions found during the search. Finally, the idea of exploiting ILP to explore promising neighborhoods of feasible solutions has been also investigated in the context of general purpose integer programs; see, e.g., Fischetti and Lodi [10] and Danna et al. [6]. The methods presented in [6,10] are currently embedded in the commercial mixed integer programming solver Cplex [16].

The paper is organized as follows. Section 2 recalls the main works proposed in the literature for OVRP. In Section 3 we describe a neighborhood for OVRP and the ILP model which allows to implicitly define and explore the presented neighborhood. The implementation of the heuristic improvement procedure is given in Section 4, while Section 5 reports the computational experiments on benchmark capacitated OVRP instances from the literature (with/without distance constraints), comparing the presented method with the most effective metaheuristic techniques proposed for OVRP. Some conclusions are finally drawn in Section 6.

2. Literature review

The classical VRP is a fundamental combinatorial optimization problem which has been widely studied in the literature (see, e.g., Toth and Vigo [28] and Cordeau et al. [5]). At first glance, having open routes instead of closed ones looks like a minor change, and in fact OVRP can be also formulated as a VRP on a directed graph, by fixing to 0 the cost of each arc entering the depot. However, if the undirected case is considered, the open version turns out to be more general than the closed one. Indeed, as shown by Letchford et al. [17], any closed VRP on n customers in a complete undirected graph can be transformed into an OVRP on n customers, but there is no transformation in the reverse direction. Further, there are many practical applications in which OVRP naturally arises. This happens, of course, when a company does not own a vehicle fleet, and hence customers are served by hired vehicles which are not required to come back to the depot (see, e.g., Tarantilis et al. [26]). But the *open model* also arises in pick-up and delivery applications, where each vehicle starts at the depot, delivers to a set of customers and then it is required to visit the same customers in reverse order, picking up items that have to be backhauled to the depot. An application of this type is described in Schrage [23]. Further areas of application, involving the planning of train services and of school bus routes, are reported by Fu et al. [13].

OVRP has recently received an increasing attention in the literature. Exact branch-and-cut and branch-cut-and-price approaches have been proposed, respectively, by Letchford et al. [17] and Pessoa et al. [19], addressing the capacitated problem with no distance constraints and no empty routes allowed (i.e., $D=\infty$ and exactly m vehicles must be used). Heuristic and metaheuristic algorithms usually take into account both capacity and distance constraints, and consider the number of routes as a decision variable. In particular, an unlimited number of vehicles is supposed to be available (i.e., $m=\infty$) and the objective function is generally to minimize the number of used vehicles first and the traveling cost second, assuming that the fixed cost of an additional vehicle always exceeds any traveling cost that could be saved by its use (i.e., considering $F=\infty$). However, several authors address as well the variant in which there are no fixed costs associated with the vehicles (i.e., $F=0$) and hence the objective function is to minimize the total traveling cost with no attention to the number of used vehicles (see, e.g., Tarantilis et al. [26]). Considering capacity constraints only (i.e., taking $D=\infty$), Sariklis and Powell [22] propose a two-phase heuristic which first assigns customers to clusters and then builds a Hamiltonian path for each cluster, Tarantilis et al. [24] describe a population-based heuristic, while Tarantilis et al. [25,26] present threshold accepting metaheuristics. Taking into account both capacity and distance constraints, Brandão [3], Fu et al. [13,14] and Derigs and Reuter [8] propose tabu search heuristics, Li et al. [18] describe a record-to-record travel heuristic, Pisinger and Ropke [20] present an adaptive large neighborhood search heuristic which follows a destruct-and-repair paradigm, while Fleszar et al. [12] propose a variable neighborhood search heuristic.

3. Reallocation model

Let z be a feasible solution of the OVRP defined on G. For any given node subset $\mathcal{F} \subset V\backslash\{0\}$, we define $z(\mathcal{F})$ as the *restricted solution* obtained from z by *extracting* (i.e., by short-cutting) all the nodes $v \in \mathcal{F}$. Let $\mathcal{R}$ be the set of routes in the restricted solution, $\mathcal{I}=\mathcal{I}(z,\mathcal{F})$ the set of all the edges in $z(\mathcal{F})$, and $\mathcal{S}=\mathcal{S}(\mathcal{F})$ the set of all the *sequences* which can be obtained through the recombination of nodes in $\mathcal{F}$ (i.e., the set of all the elementary paths in $\mathcal{F}$). Each edge $i \in \mathcal{I}$ is viewed as a potential *insertion point* which can allocate one or more nodes in $\mathcal{F}$ through at most one sequence $s \in \mathcal{S}$. We say that the insertion point $i=(a,b) \in \mathcal{I}$ allocates the nodes $\{v_j \in \mathcal{F} : j=1,\dots,h\}$ through the sequence $s=(v_1,v_2,\dots,v_h) \in \mathcal{S}$, if the edge (a,b) in the restricted solution is replaced by the edges $(a,v_1),(v_1,v_2),\dots,(v_h,b)$ in the new feasible solution. Since the restricted routes, as well as the final ones, are open paths starting at the depot, in addition to the edges of the restricted solution we also consider the insertion points (called *appending insertion points* in the following) $i=(p_r,0)$, where p_r denotes the last customer visited by route $r \in \mathcal{R}$, which allow to append any sequence to the last customer of any restricted route. Further, empty routes in the restricted solution are associated with insertion points (0,0).

For each sequence $s \in \mathcal{S}$, $c(s)$ and $q(s)$ denote, respectively, the cost of the elementary path corresponding to s and the sum of the demands of the nodes in s. For each insertion point $i=(a,b) \in \mathcal{I}$ and for each sequence $s=(v_1,v_2,\dots,v_h) \in \mathcal{S}$, γ_{si} denotes the extra-cost (i.e., the extra-distance) for assigning sequence s to insertion point i in its best possible orientation (i.e., $\gamma_{si} := c(s)-c_{ab}+\min\{c_{av_1}+c_{v_hb},c_{av_h}+c_{v_1b}\}$). Note that, for the appending insertion points $i=(p_r,0)$, γ_{si} is computed as $c(s)+\min\{c_{p_rv_1},c_{p_rv_h}\}$. The extra-cost for assigning the sequence s to the insertion point $i=(0,0)$ associated with an empty route is simply $c(s)+\min\{c_{0v_1},c_{0v_h}\}$. For each route $r \in \mathcal{R}$, $\mathcal{I}(r)$ denotes the set of insertion points associated with r, while $\tilde{q}(r)$ and $\tilde{c}(r)$ denote, respectively, the total demand and the total distance computed for route r, still in the restricted solution.

For each $i \in \mathcal{I}$, $\mathcal{S}_i \subseteq \mathcal{S}$ denotes a sequence subset containing the sequences which can be allocated to the specific insertion point i. The definition of $\mathcal{S}_i$ will be discussed later in this section. Then, a neighborhood of the given solution z can be formulated (and explored) by solving an ILP problem (denoted as the *Reallocation Model*) based on the decision variables

$$x_{si} = \begin{cases} 1 & \text{if sequence } s \in \mathcal{S}_i \text{ is allocated to insertion point } i \in \mathcal{I}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

which reads as follows:

$$\sum_{r \in \mathcal{R}} \tilde{c}(r) + \min \sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{S}_i} \gamma_{si} x_{si} \tag{2}$$

subject to

$$\sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{S}_i(v)} x_{si} = 1, \quad v \in \mathcal{F}, \tag{3}$$

$$\sum_{s \in \mathcal{S}_i} x_{si} \le 1, \quad i \in \mathcal{I}, \tag{4}$$

$$\sum_{i \in \mathcal{I}(r)} \sum_{s \in \mathcal{S}_i} q(s) x_{si} \le Q - \tilde{q}(r), \quad r \in \mathcal{R}, \tag{5}$$

$$\sum_{i \in \mathcal{I}(r)} \sum_{s \in \mathcal{S}_i} \gamma_{si} x_{si} \le D - \tilde{c}(r), \quad r \in \mathcal{R}, \tag{6}$$

$$x_{si} \in \{0,1\}, \quad i \in \mathcal{I}, \ s \in \mathcal{S}_i, \tag{7}$$

where, for any $i \in \mathcal{I}$ and $v \in \mathcal{F}$, $\mathcal{S}_i(v) \subseteq \mathcal{S}_i$ denotes the set of sequences covering customer v which can be allocated to insertion point i. The objective function (2), to be minimized, gives the traveling cost of the final OVRP solution. Constraints (3) impose that each extracted node belongs to exactly one of the selected sequences, i.e., that it is covered exactly once in the final solution. Constraints (4) avoid to allocate two or more sequences to the same insertion point. Finally, constraints (5) and (6) impose that each route in the final solution fulfills the capacity and distance restrictions, respectively. Note that, if there is a non-null fixed cost F associated with the vehicles, it can be taken into account by simply adding F to the cost of the edges incident at the depot node.

The Reallocation Model (2)–(7) defines a neighborhood of a given solution z which depends on the extracted nodes $\mathcal{F}$ and on the subsets $\mathcal{S}_i$ ($i \in \mathcal{I}$). In particular, for any given $\mathcal{F}$, the choice of $\mathcal{S}_i$ is a key factor in order to allow an effective exploration of the solution space in the neighborhood of the given solution. The subsets $\mathcal{S}_i$ are built by following a column generation approach: we initialize the Linear Programming (LP) relaxation of the Reallocation Model (LP-RM) with a subsets of variables with small insertion cost, and afterwards we iteratively solve the column generation problem associated with LP-RM, adding other variables with *small* reduced cost. The overall procedure for building the subsets $\mathcal{S}_i$ can be described as follows.

1. (Initialization) For each insertion point $i = (a_i, b_i) \in \mathcal{I}$, initialize subset $\mathcal{S}_i$ with the *basic* sequence extracted from i (i.e., the, possibly empty, sequence of nodes connecting node a_i and b_i in the given solution z) plus the feasible singleton sequence with the minimum insertion cost (i.e., the sequence (v), with $v \in \mathcal{F}$, with the minimum extra-cost among all the singleton sequences which can be allocated to i without violating the capacity and distance restrictions for the restricted route containing i). Initialize LP-RM with the initial set of variables corresponding to the current subsets $\mathcal{S}_i$, and solve LP-RM.
2. (Column generation) For each insertion point $i \in \mathcal{I}$, solve the *column generation problem* associated with i, adding to $\mathcal{S}_i$ all the sequences s corresponding to elementary paths in $\mathcal{F}$, whose associated variables x_{si} have a reduced cost rc_{si} under a given threshold $RC_{\max}$ (i.e., variables x_{si} such that $rc_{si} \le RC_{\max}$). If at least one sequence/variable has been added, solve the new LP-RM and repeat step 2. Otherwise terminate.

For any fixed insertion point $i \in \mathcal{I}$, the column generation problem associated with i in LP-RM is a Resource Constrained Elementary Shortest Path Problem (RCESPP), which usually arises in the Set Partitioning formulation of the classical VRP (see, e.g., Feillet et al. [9] and Righini and Salani [21]). Here, for each insertion point $i \in \mathcal{I}$, we solve the corresponding RCESPP through a simple greedy heuristic, with the aim of finding as many variables with small reduced cost as possible. Hashing techniques are used to avoid the generation of duplicated variables.

Note that each subset $\mathcal{S}_i$ contains the *basic* sequence extracted from insertion point i, and hence the current solution can always be obtained as a new feasible solution of the Reallocation Model.

3.1. Column generation for the Reallocation Model

Let π_v^1, π_i^2, π_r^3 and π_r^4 be the dual variables associated, respectively, with constraints (3)–(6) in LP-RM, where $v \in \mathcal{F}$, $i \in \mathcal{I}$ and $r \in \mathcal{R}$, and denote with $\tilde{\pi} = (\tilde{\pi}_v^1, \tilde{\pi}_i^2, \tilde{\pi}_r^3, \tilde{\pi}_r^4)$ the optimal dual solution of LP-RM. For any fixed $i = (a_i, b_i) \in \mathcal{I}$, consider the directed graph $\tilde{G}(i,\tilde{\pi}) = (V_i, A_i)$, with $V_i := \{a_i, b_i\} \cup \mathcal{F}$ and $A_i := \{(v,w) : v \in V_i, w \in V_i\} \backslash \{(a_i,b_i),(b_i,a_i)\}$.Associate with each arc $a = (v,w) \in A_i, w \neq 0$, a weight θ_a equal to the cost of the corresponding edge $e = (v,w)$ in the graph G, while set $\theta_a := 0$ for each arc $a = (v,0) \in A_i$, if $0 \in V_i$. Associate with each arc $a \in A_i$ a cost $c'_a = \theta_a(1 - \tilde{\pi}_{r_i}^4)$, and associate with each node $v \in \mathcal{F}$ a weight q_v and a cost $q'_v = -(\tilde{\pi}_v^1 + q_v \tilde{\pi}_{r_i}^3)$. Then, let $P = (V_P, A_P)$ be an elementary path $(a_i, v_1, \ldots, v_h, b_i)$ connecting nodes a_i and b_i in $\tilde{G}(i,\tilde{\pi})$, where $V_P := \{v_1, \ldots, v_h\} \subseteq V_i$ and $A_P := \{(a_i,v_1), \ldots, (v_h,b_i)\} \subseteq A_i$. We say that P is a feasible path if

$$\sum_{v \in V_P} q_v \le Q - \tilde{d}(r_i), \quad \sum_{a \in A_P} \theta_a \le D - \tilde{c}(r_i) + c_i,$$

where c_i denotes the cost of insertion point $i = (a_i, b_i)$, while the cost of the path is

$$c'(P) = \sum_{a \in A_P} c'_a + \sum_{v \in V_P} q'_v.$$

Any sequence $s = (v_1, \ldots, v_h) \in \mathcal{S}$ is clearly associated with the elementary path $(a_i, v_1, \ldots, v_h, b_i)$ in $\tilde{G}(i,\tilde{\pi})$. The reduced cost rc_{si} of variable x_{si} in LP-RM is defined by

$$rc_{si} := \gamma_{si} - \sum_{v \in V_P} \tilde{\pi}_v^1 - \tilde{\pi}_i^2 - q(s)\tilde{\pi}_{r_i}^3 - \gamma_{si}\tilde{\pi}_{r_i}^4$$

and can easily be rewritten as

$$rc_{si} := -\tilde{\pi}_i^2 - c_i(1 - \tilde{\pi}_{r_i}^4) + \sum_{a \in A_P} c'_a + \sum_{v \in V_P} q'_v.$$

Hence, the following proposition holds:

Proposition 1. *For any $i = (a_i, b_i) \in \mathcal{I}$, the column generation problem associated with i in LP-RM is the problem of finding an elementary feasible path P from a_i to b_i in $\tilde{G}(i,\tilde{\pi})$, with cost $c'(P) < \tilde{\pi}_i^2 + c_i(1 - \tilde{\pi}_{r_i}^4)$.*

As described above, the column generation problem for LP-RM associated with any insertion point $i \in \mathcal{I}$ is a Resource Constrained Elementary Shortest Path Problem (RCESPP) defined on graph $\tilde{G}(i,\tilde{\pi})$, whose size strictly depends on $|\mathcal{F}|$. The orientation of $\tilde{G}(i,\tilde{\pi})$ is required only when the considered $i = (a_i, b_i) \in \mathcal{I}$ is an appending insertion point (i.e., b_i is the depot node). Even in this case, the column generation problem could be addressed on a mixed graph, where only the edges incident at the depot are replaced by directed arcs (of different cost and weight) entering and leaving the depot. In the general case, $\tilde{G}(i,\tilde{\pi})$ contains negative cycles (i.e., cycles in which the sum of the costs c'_a associated with the arcs and the costs q'_v associated with the nodes is negative): indeed, while dual variables $\pi_i^2, \pi_r^3, \pi_r^4$ are non-positive, dual variables π_v^1 are free and usually assume positive values. Positive values of variables π_v^1 can lead to negative node costs q'_v and to negative cycles in graph $\tilde{G}(i,\tilde{\pi})$. Therefore, the column generation problem in LP-RM is strongly $\mathcal{NP}$-hard.

In order to find a promising set of variables for the Reallocation Model in a short computing time, we solve the RCESPP associated with each insertion point through a simple heuristic. We say that a node $v \in \mathcal{F}$ is *feasible* for $i \in \mathcal{I}$ if the singleton sequence (v) can be allocated to i without violating the capacity and distance restrictions on the restricted route r_i. For any given insertion point $i = (a_i, b_i) \in \mathcal{I}$, we first build a *reduced* graph $\tilde{G}(i,\tilde{\pi})$, obtained by considering only nodes a_i, b_i and the *nf* feasible nodes of $\mathcal{F}$ with smallest insertion cost (i.e., the *nf* feasible nodes $v_k \in \mathcal{F}, k = 1, \ldots, nf$, whose associated singleton sequences (v_k) have the smallest

Please cite this article as: Salari M, et al. An ILP improvement procedure for the Open Vehicle Routing Problem. Computers and Operations Research (2010), doi:10.1016/j.cor.2010.02.010

extra-cost for i). At each iteration of the column generation step described in Section 3, nf is uniformly randomly generated in $[nf_{min}, nf_{max}]$. Then, on the reduced graph $\tilde{G}(i, \tilde{\pi})$, we apply the following simple heuristic:

1. Find an initial feasible path $P=(a_i, v, b_i)$, in $\tilde{G}(i, \tilde{\pi})$.
2. Evaluate all the 1–1 *feasible* exchanges between each node $w \in V_i \backslash V_P$ and each node $v \in V_P$, and select the best one (with respect to the cost of the corresponding path); if this exchange leads to an improvement, perform it and repeat step 2.
3. Evaluate all the *feasible* insertions of each node $w \in V_i \backslash V_P$ in each arc $(v_1, v_2) \in A_P$ and select the best one; if no feasible insertion exists, terminate; otherwise, force such an insertion even if it leads to a worse path and repeat step 2.

Whenever a new path in $\tilde{G}(i, \tilde{\pi})$ is generated, the corresponding sequence is added to $\mathcal{S}_i$ if the reduced cost of x_{si} is smaller than a given threshold RC_{max}.

4. Heuristic improvement procedure

The Reallocation Model described in the previous section allows for exploring a neighborhood of a given feasible solution, depending on the choice of the extracted customers in $\mathcal{F}$. We propose a heuristic improvement procedure for OVRP, based on model (2)–(7), which iteratively explores different neighborhoods of the current solution. Given an initial feasible solution z_0 for OVRP (taken from the literature or found by any heuristic method), the procedure works as follows.

1. (Initialization) Set $kt := 0$ and $kp := 0$. Take z_0 as the incumbent solution and initialize the current solution z_c as $z_c := z_0$.
2. (Node selection) Build set $\mathcal{F}$ by selecting each customer with a probability p.
3. (Node extraction) Extract the nodes selected in the previous step from the current solution z_c and construct the corresponding restricted OVRP solution $z_c(\mathcal{F})$, obtained by short-cutting the extracted nodes.
4. (Reallocation) Define the subsets $\mathcal{S}_i$ $(i \in \mathcal{I}(z_c, \mathcal{F}))$ as described in Section 3. Build the corresponding Reallocation Model (2)–(7) and solve the model by using a general-purpose ILP solver. Once an optimal ILP solution has been found, construct the corresponding new OVRP solution and possibly update z_c and z_0.
5. (Termination) Set $kt := kt+1$. If $kt = KT_{max}$, terminate.
6. (Perturbation) If z_c has been improved in the last iteration, set $kp := 0$; otherwise set $kp := kp+1$. If $kp = KP_{max}$, "perturb" the current solution z_c and set $kp := 0$. In any case, repeat step 2.

The procedure performs KT_{max} iterations and at each iteration explores a randomly generated neighborhood of the current solution z_c. However, if z_c is not improved for KP_{max} consecutive iterations, we introduce a random perturbation (see Step 6) in order to move to a different area of the solution space, so as to enforce the diversification of the search. In particular, when performing a Perturbation Step, we randomly extract np customers from z_c (with np uniformly randomly chosen in $[np_{min}, np_{max}]$ and with each customer having the same probability to be extracted), and reinsert each extracted customer, in turn, in its best feasible position. If a customer cannot be inserted in any currently non-empty route (due to the capacity and/or distance restrictions), a new route is created to allocate the customer. In general, when performing the Perturbation Step, several customers cannot be inserted in the non-empty routes of the current solution, and hence the new perturbed solution can use more vehicles than the current one.

5. Computational results

The performance of the Heuristic Improvement Procedure (HIP) described in the previous sections was evaluated on the 16 benchmark instances usually addressed in the literature, taken from Christofides et al. [4] (instances `C1–C14`) and from Fisher [11] (instances `F11–F12`), and on the 8 large scale benchmark instances proposed by Li et al. [18], and also addressed by Derigs and Reuter [8] (instances `O1–O8`). The number of customers of `C1–C14` and `F11–F12` ranges from 50 to 199. `C1–C5`, `C11–C12` and `F11–F12` have only capacity constraints, while `C6–C10` and `C13–C14` are the same instances as `C1–C5` and `C11–C12`, respectively, but with both capacity and distance constraints. Instances `O1–O8` have no distance restrictions and a number of customers varying from 200 to 480. As usual, for the problems with distance constraints, the route duration limit D is taken as the original value for the classical VRP multiplied by 0.9.

HIP needs an initial solution to be given, which in principle could be computed through any available constructive heuristic algorithm. We decided to run HIP starting from an extremely good feasible solution available from the literature (in several cases, the best known solution reported in the literature), with the aim of attempting to improve it (this is of course impossible if the initial solution is provably optimal, as it is the case for some of them). In particular, we considered as initial solutions the ones obtained by Fu et al. [13,14], Pisinger and Ropke [20], Derigs and Reuter [8] and Fleszar et al. [12].

HIP has been tested on a Pentium IV 3.4 GHz with 1 GByte RAM, running under Microsoft Windows XP Operative System, and has been coded in C++ with Microsoft Visual C++ 6.0 compiler. The ILP solver used in the experiments is ILOG Cplex 10.0 [16]. HIP setting depends on the parameters RC_{max}, p, nf_{min}, nf_{max}, np_{min}, np_{max}, and on the number of iterations KP_{max} and KT_{max}. Although these parameters could be tuned considering the edge costs and the particular characteristics of each tested instance, we preferred to run all the experiments with a fixed set of parameters: $RC_{max}=1$, $p=0.5$ (i.e., 50% of the customers are selected on average), $nf_{min}=15$, $nf_{max}=25$, $np_{min}=15$, $np_{max}=25$, $KP_{max}=50$ and $KT_{max}=5000$ (i.e., we perform globally 5000 iterations, and the current solution is perturbed if it cannot be improved for 50 consecutive iterations). Further, since several authors address the problem considering as objective function the minimization of the number of vehicles first and of the traveling cost second (i.e., assuming $F=\infty$), while other authors considered as objective function the minimization of the traveling cost (i.e., $F=0$), we decided to run HIP without allowing to change the number of vehicles used in the initial solution. However, as stated in Section 4, the Perturbation Step often requires additional routes to be created (to preserve the feasibility of the solution). In such cases, we add a small penalty θ to the cost of the edges incident at the depot, in order to force HIP to "recover" the solution in the following iterations. After some preliminary tests, we decided to fix $\theta = 12$ for the considered instances. Finally, HIP is a randomized algorithm and hence the computational results may depend on the randomization. For each tested instance (and each initial solution), we considered five runs of the algorithm corresponding to five different seeds for generating the random numbers.

The computational results are reported in Tables 1–3. All the CPU times are expressed in seconds, and all the solution costs have been computed in double precision.

Table 1 reports the computational results on the 16 instances `C1–C14` and `F11–F12` obtained by starting from the solutions

Table 1
Computational results on the “classical” 16 benchmark instances starting from the solutions by Fu et al. [13,14].

Pb	*m*	P.best	Initial		Run 1		Run 2		Run 3		Run 4		Run 5		Best		Worst		Average	
			Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev
C2	10	567.14	567.14	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	567.14	0.00	567.14	0.00	567.14	0.00
C3	8	*639.74	641.88	0.33	*639.74	0.00	640.42	0.11	640.42	0.11	640.42	0.11	640.42	0.11	*639.74	0.00	640.42	0.11	640.28	0.08
C4	12	733.13	738.94	0.79	733.13	0.00	733.13	0.00	733.13	0.00	733.13	0.00	733.13	0.00	733.13	0.00	733.13	0.00	733.13	0.00
C5	17	869.25	878.95	1.12	**868.81**	−0.05	**868.81**	−0.05	**868.81**	−0.05	**868.81**	−0.05	**868.81**	−0.05	**868.81**	−0.05	**868.81**	−0.05	**868.81**	−0.05
C6	6	412.96	412.96	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	412.96	0.00	412.96	0.00	412.96	0.00
C7	11	568.49	568.49	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	568.49	0.00	568.49	0.00	568.49	0.00
C8	9	644.63	646.31	0.26	644.63	0.00	644.63	0.00	644.63	0.00	644.63	0.00	644.63	0.00	644.63	0.00	644.63	0.00	644.63	0.00
C9	14	756.14	761.28	0.68	756.14	0.00	756.14	0.00	756.38	0.03	756.38	0.03	756.14	0.00	756.14	0.00	756.38	0.03	756.24	0.01
C10	17	875.07	903.10	3.20	878.54	0.40	879.13	0.46	877.47	0.27	880.25	0.59	879.68	0.53	877.47	0.27	880.25	0.59	879.01	0.45
C11	7	682.12	717.15	5.14	683.64	0.22	685.20	0.45	685.20	0.45	682.83	0.10	682.83	0.10	682.83	0.10	685.20	0.45	683.94	0.27
C12	10	*534.24	534.71	0.09	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00
C13	12	896.50	917.90	2.39	**894.19**	−0.26	897.37	0.10	896.66	0.02	897.37	0.10	**896.14**	−0.04	**894.19**	−0.26	897.37	0.10	**896.35**	−0.02
C14	11	591.87	600.66	1.49	591.87	0.00	591.87	0.00	591.87	0.00	591.87	0.00	591.87	0.00	591.87	0.00	591.87	0.00	591.87	0.00
F12	7	769.66	777.07	0.96	**769.55**	−0.01	770.38	0.09	**769.55**	−0.01	770.38	0.09	770.38	0.09	**769.55**	−0.01	770.38	0.09	770.05	0.05
Avg.				1.18		0.02		0.08		0.06		0.07		0.05		0.00		0.09		0.06

Pb	Initial t.time	Run 1 t.time	Run 1 b.time	Run 2 t.time	Run 2 b.time	Run 3 t.time	Run 3 b.time	Run 4 t.time	Run 4 b.time	Run 5 t.time	Run 5 b.time	Average t.time	Average b.time
C2	7.8	84.2	*84.2*	86.5	*86.5*	80.2	*80.2*	81.9	*81.9*	89.6	*89.6*	84.5	84.5
C3	23.2	119.9	106.0	110.9	74.7	117.9	56.8	139.9	132.0	118.8	88.3	121.5	91.6
C4	6.8	156.6	21.2	157.8	0.6	154.7	0.4	198.1	0.8	164.1	0.7	166.3	4.7
C5	61.9	220.3	10.3	220.0	11.6	228.5	32.7	277.8	12.9	225.5	12.5	234.4	16.0
C6	0.6	45.1	*45.1*	43.2	*43.2*	49.0	*49.0*	44.7	*44.7*	42.2	*42.2*	44.8	44.8
C7	6.0	83.1	*83.1*	87.7	*87.7*	83.2	*83.2*	79.8	*79.8*	79.5	*79.5*	82.7	82.7
C8		136.2	0.1	135.9	0.1	144.9	3.2	177.7	2.6	144.7	0.1	147.9	1.2
C9	46.6	255.5	102.7	265.5	7.9	247.7	195.7	312.1	299.6	259.2	18.3	268.0	124.8
C10	51.9	460.2	323.9	477.7	35.9	513.0	453.1	505.9	474.5	497.8	366.5	490.9	330.8
C11	23.1	198.8	165.8	199.8	40.4	229.8	225.7	329.6	204.2	201.4	200.0	231.9	167.2
C12	4.2	94.0	1.6	98.2	73.2	99.1	89.5	106.2	16.6	107.8	92.1	101.1	54.6
C13	82.1	1165.3	475.0	1004.9	201.4	1180.5	685.0	1146.5	725.2	1396.5	1230.9	1178.7	663.5
C14	2.5	354.7	293.8	339.2	88.6	321.3	33.7	366.6	337.3	453.9	435.8	367.1	237.8
F12	28.4	148.2	77.8	158.0	74.9	154.8	70.7	169.3	30.9	168.2	67.5	159.7	64.4
Avg.	24.7	251.6	127.9	241.8	59.1	257.5	147.1	281.2	174.5	282.1	194.6	262.8	140.6

CPU times are expressed in seconds.

Table 2
Computational results on the “classical” 16 benchmark instances starting from the best available solutions.

Pb	*m*	P.best	Initial		Run 1		Run 2		Run 3		Run 4		Run 5		Best		Worst		Average	
			Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev
C2	10	567.14	567.14	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	567.14	0.00	567.14	0.00	567.14	0.00
C4	12	733.13	733.13	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	733.13	0.00	733.13	0.00	733.13	0.00
C5	16	879.37	896.08	1.90	892.37	1.48	892.37	1.48	892.37	1.48	892.37	1.48	892.37	1.48	892.37	1.48	892.37	1.48	892.37	1.48
C5	17	869.24	869.24	0.00	**868.93**	−0.04	**868.93**	−0.04	**869.00**	−0.03	**868.93**	−0.04	**868.93**	−0.04	**868.93**	−0.04	**869.00**	−0.03	**868.94**	−0.03
C6	6	412.96	412.96	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	412.96	0.00	412.96	0.00	412.96	0.00
C7	10	583.19	583.19	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	583.19	0.00	583.19	0.00	583.19	0.00
C7	11	568.49	568.49	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	568.49	0.00	568.49	0.00	568.49	0.00
C8	9	644.63	644.63	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	644.63	0.00	644.63	0.00	644.63	0.00
C9	13	757.84	757.84	0.00	**757.73**	−0.01	**757.69**	−0.02	**757.70**	−0.02	**757.73**	−0.01	**757.73**	−0.01	**757.69**	−0.02	**757.73**	−0.01	**757.72**	−0.02
C9	14	756.14	756.14	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	756.14	0.00	756.14	0.00	756.14	0.00
C10	17	875.07	875.07	0.00	**874.71**	−0.04	**874.71**	−0.04	**874.71**	−0.04	**874.71**	−0.04	**874.71**	−0.04	**874.71**	−0.04	**874.71**	−0.04	**874.71**	−0.04
C11	7	682.12	682.12	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	682.12	0.00	682.12	0.00	682.12	0.00
C13	11	904.04	904.04	0.00	**899.16**	−0.54	**899.16**	−0.54	**899.16**	−0.54	**899.16**	−0.54	**899.16**	−0.54	**899.16**	−0.54	**899.16**	−0.54	**899.16**	−0.54
C13	12	896.50	917.90	2.39	**894.19**	−0.26	897.37	0.10	896.66	0.02	897.37	0.10	**896.14**	−0.04	**894.19**	−0.26	897.37	0.10	**896.35**	−0.02
C14	11	591.87	591.87	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	591.87	0.00	591.87	0.00	591.87	0.00
C14	12	581.81	581.81	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	581.81	0.00	581.81	0.00	581.81	0.00
F12	7	769.66	769.66	0.00	**769.55**	−0.01	**769.55**	−0.01	–	0.00	**769.55**	−0.01	–	0.00	**769.55**	−0.01	769.66	0.00	**769.59**	−0.01
Avg.				0.25		0.03		0.05		0.05		0.06		0.05		0.03		0.06		0.05

Pb	Source	t.time	b.time	t.time	b.time	t.time	b.time	t.time	b.time	t.time	b.time	t.time	b.time
C2	[12,14,20]	84.2	*84.2*	86.5	*86.5*	80.2	*80.2*	81.9	*81.9*	89.6	*89.6*	84.5	84.5
C4	[12,20]	151.2	*151.2*	137.5	*137.5*	123.2	*123.2*	164.5	*164.5*	153.5	*153.5*	146.0	146.0
C5	[20]	450.2	276.5	480.4	237.5	463.4	237.5	434.0	237.5	518.3	237.5	469.3	245.3
C5	[8]	275.2	130.2	247.5	195.4	278.1	21.4	260.5	205.0	255.8	166.9	263.4	143.8
C6	[12,14,20]	45.1	*45.1*	43.2	*43.2*	49.0	*49.0*	44.7	*44.7*	42.2	*42.2*	44.8	44.8
C7	[20]	80.6	*80.6*	86.1	*86.1*	73.1	*73.1*	81.2	*81.2*	85.2	*85.2*	81.2	81.2
C7	[14]	83.1	*83.1*	87.7	*87.7*	83.2	*83.2*	79.8	*79.8*	79.5	*79.5*	82.7	82.7
C8	[12]	136.8	*136.8*	142.2	*142.2*	137.2	*137.2*	130.0	*130.0*	143.9	*143.9*	138.0	138.0
C9	[20]	412.5	33.9	404.6	330.6	372.9	0.2	355.4	11.0	413.0	6.9	391.7	76.5
C9	[8]	243.4	*243.4*	213.9	*213.9*	221.9	*221.9*	227.6	*227.6*	267.1	*267.1*	234.8	234.8
C10	[8]	454.7	2.6	390.1	2.4	344.0	1.7	395.6	4.2	387.6	0.7	394.4	2.3
C11	[12,20]	178.3	*178.3*	183.4	*183.4*	181.0	*181.0*	183.4	*183.4*	165.3	*165.3*	178.3	178.3
C13	[12]	959.3	6.3	1022.0	133.3	1030.3	6.6	1027.2	4.6	980.5	78.6	1003.9	45.9
C13	[14]	1165.3	475.0	1004.9	201.4	1180.5	685.0	1146.5	725.2	1396.5	1230.9	1178.7	663.5
C14	[12,20]	276.3	*276.3*	293.8	*293.8*	263.6	*263.6*	294.4	*294.4*	301.1	*301.1*	285.8	285.8
C14	[8]	364.2	*364.2*	304.0	*304.0*	310.5	*310.5*	290.4	*290.4*	354.1	*354.1*	324.6	324.6
F12	[12]	142.2	56.6	157.2	103.4	143.9	*143.9*	137.2	64.2	129.7	*129.7*	142.0	99.6
Avg.		323.7	154.4	310.9	163.7	313.9	154.1	313.8	166.4	339.0	207.8	320.2	169.3

CPU times are expressed in seconds.

Table 3
Computational results on the eight large scale benchmark instances starting from the solutions by Derigs and Reuter [8].

Pb	m	P.best	Initial		Run 1		Run 2		Run 3		Run 4		Run 5		Best		Worst		Average	
			Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev
O1	5	6018.52	6018.52	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	6018.52	0.00	6018.52	0.00	6018.52	0.00
O2	9	4584.55	4584.69	0.00	**4573.53**	−0.24	**4573.53**	−0.24	**4573.53**	−0.24	**4573.53**	−0.24	**4573.53**	−0.24	**4573.53**	−0.24	**4573.53**	−0.24	**4573.53**	−0.24
O3	7	7731.46	7731.46	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	7731.46	0.00	7731.46	0.00	7731.46	0.00
O4	10	7260.59	7260.59	0.00	**7259.81**	−0.01	**7253.91**	−0.09	**7253.91**	−0.09	**7253.20**	−0.10	**7251.74**	−0.12	**7251.74**	−0.12	**7259.81**	−0.01	**7254.51**	−0.08
O5	9	9167.19	9167.19	0.00	**9165.40**	−0.02	**9156.74**	−0.11	**9157.42**	−0.11	**9159.22**	−0.09	**9159.22**	−0.09	**9156.74**	−0.11	**9165.40**	−0.02	**9159.6**	−0.08
O6	9	9803.80	9805.45	0.02	–	0.02	–	0.02	–	0.02	–	0.02	9804.25	0.00	9804.25	0.00	9805.45	0.02	9805.21	0.01
O7	10	10348.57	10348.57	0.00	**10344.37**	−0.04	**10344.37**	−0.04	**10344.37**	−0.04	**10344.37**	−0.04	**10344.37**	−0.04	**10344.37**	−0.04	**10344.37**	−0.04	**10344.37**	−0.04
O8	10	12420.16	12420.16	0.00	–	0.00	–	0.00	–	0.00	–	0.00	–	0.00	12420.16	0.00	12420.16	0.00	12420.16	0.00
Avg.				0.00		−0.04		−0.06		−0.06		−0.06		−0.06		−0.06		−0.04		−0.05
Pb			t.time		t.time	b.time	t.time	b.time	t.time	b.time	t.time	b.time	t.time	b.time					t.time	b.time
O1			467.0		182.2	*182.2*	191.5	*191.5*	174.0	*174.0*	168.5	*168.5*	175.9	*175.9*					178.4	178.4
O2			467.0		284.0	34.6	298.6	233.0	302.8	89.2	313.0	63.3	395.5	360.6					318.8	156.1
O3			4047.0		304.6	*304.6*	279.6	*279.6*	300.6	*300.6*	295.5	*295.5*	296.8	*296.8*					295.4	295.4
O4			927.0		438.9	34.4	405.5	387.5	437.6	72.6	421.7	219.9	406.3	385.4					422.0	220.0
O5			1186.0		499.6	41.4	479.1	270.8	513.9	496.5	550.3	173.6	479.5	210.5					504.5	238.6
O6			1231.0		581.3	*581.3*	590.4	*590.4*	637.4	*637.4*	620.1	*620.1*	590.8	361.1					604.0	558.1
O7			3190.0		653.0	8.6	631.8	23.8	661.6	420.9	743.5	306.6	619.7	387.2					661.9	229.4
O8			1969.0		623.6	*623.6*	635.2	*635.2*	668.9	*668.9*	653.7	*653.7*	647.9	*647.9*					645.9	645.9
Avg.			1685.5		445.9	226.3	438.9	326.5	462.1	357.5	470.8	312.7	451.6	353.2					453.9	315.2

CPU times are expressed in seconds.

provided by Fu, Eglese and Li and obtained through the algorithm proposed in [13]. In some cases, several solutions are provided for the same instance, obtained by using slightly different versions of their algorithm, with the same number of routes and different traveling cost. Among the different solutions for the same instance, we considered as initial solution for HIP the best one provided. For instances C1 and F11, all the solutions available from [13,14] are provably optimal (see, e.g., Letchford et al. [17]) and cannot be further improved. Thus, these instances were not considered in this set of experiments. The upper part of the table reports the solutions found by HIP. The first column gives the instance name (*Pb*). Columns 2 and 3 report the number of vehicles used in the initial solution (m) and the cost of the best known solution using the same number of vehicles (*P.best*). Columns 4 and 5 report the cost of the initial solution (*cost*) and the corresponding percentage deviation w.r.t. the best known value (*%dev*), computed as $100^*(cost-P.best)/P.best$. Then, for each of the 5 runs of the algorithm, we report the final solution cost provided by HIP and the corresponding percentage deviation (again computed w.r.t. the best known value). When HIP was not able to improve on the initial solution, we mark with a "—" the final solution cost. Finally, we report the best, the worst and the average result out of the five different runs. Final solution costs equal to the previously best known ones are underlined, new best solutions are in bold face, while provably optimal solutions, taken from Letchford et al. [17], are marked with an *. The lower part of the table gives the computing times. First, we report the overall CPU time of the algorithm corresponding to the initial solution, obtained on a Pentium IV 3 GHz. These times have been taken from [14]. However, the cost of the initial solution for instance C8 is better than the ones reported in [14], and hence for this initial solution we did not report the corresponding computing time. Then, for each run of the algorithm, we report the overall computing time required to perform all the 5000 iterations (*t.time*) and the CPU time required to reach the final solution (*b.time*). For a "fair" calculation of the average values, when HIP was not able to improve on the initial solution we considered *b.time* equal to the overall computing time. Finally, the last two columns give the average CPU times (i.e., average *t.time* and average *b.time*) out of the five different runs.

Table 2 reports the computational results on the same instances by starting from the best available solutions among the ones obtained by Fu et al. [13,14], Pisinger and Ropke [20], Derigs and Reuter [8] and Fleszar et al. [12]. The table has the same structure as Table 1, but column 2 in the lower part of the table reports the source of the initial solution used in the experiments. For instances C5, C7, C9, C13 and C14, the best available solutions for the case $F=\infty$ and the case $F=0$ are different. In such cases, we considered both the solutions as initial solutions for HIP. For instances C1, C3, C12 and F11, all the solutions available from [8,12,20] are provably optimal and hence these instances were not considered in this set of experiments.

Finally, Table 3 reports the computational results on the 8 large scale instances O1–O8 by starting from the solutions provided by Derigs and Reuter [8]. The table has the same structure as Table 1, but the CPU time related to the initial solution (column 2 in the lower part of the table) was obtained on a Pentium IV 2.8 GHz.

The tables show that HIP is able to improve even extremely good quality solutions, obtained by some of the most effective metaheuristic techniques proposed for OVRP. It is worth noting that the solutions and the CPU times provided by Fu et al. [13,14] and reported in Table 1 are the best ones from among 20 runs of the corresponding randomized algorithm with different seeds. Hence, taking into account the different performance of the processors used for testing the different algorithms, the overall computing time required by HIP is comparable with the others reported in the tables, and in several cases the final improved solution is found very quickly. Our test-bed concerns in practice 35 different, non-provably optimal, initial solutions which could be possibly improved, corresponding to 22 different instances. By considering the best result from among the five different runs executed for each of these 35 initial solutions, HIP improves on the initial solution in 22 cases. For these cases, HIP reaches 6 times the previously best known solution (provably optimal in two cases), while finds 12 times a new best solution. Considering the 13 initial solutions which HIP does not improve, it is worth noting that all these solutions are the best known ones in the literature (for the case $F=\infty$ or $F=0$). Looking at the different runs executed for each initial solution, we can note that in some cases the results depend on the seed used for the random generator. However, the method is overall quite consistent since, by considering all the tested initial solutions, the average computing time and the average final percentage deviation are only slightly affected by the choice of the seed.

In order to look for possible better solutions, we performed some additional experiments. In particular, after the first 5000 iterations, we ran HIP for 2000 more iterations with a slightly different parameter setting. Starting from the solutions provided by Fu et al. [14], for instance C5 with 17 vehicles, after 5220 iterations and 237.4 s HIP found a solution of cost 868.44 that corresponds to a further improvement on the previous best known solution. Finally, still starting from the solutions by Fu et al. [14], we ran HIP with a different tuning of parameter p, to investigate how the neighborhood size affects the overall performance of the method, both in terms of quality of the solutions found and of CPU time. Let $z_{avg}(\overline{p})$ be the average final solution cost obtained on the 14 instances C2–C14 and F12 with $p=\overline{p}$, and let $ttime_{avg}(\overline{p})$ be the corresponding average CPU time in seconds. With $p=0.3$, 0.5 and 0.7 we obtained the following results: $z_{avg}(0.3)=684.55$ and $ttime_{avg}(0.3)=71.9$, $z_{avg}(0.5)=681.94$ and $ttime_{avg}(0.5)=262.8$, $z_{avg}(0.7)=683.32$ and $ttime_{avg}(0.7)=460.0$. As expected, the average CPU time consistently increases with the number of extracted customers, while the best solution costs are obtained with the default setting of p (i.e., $p=0.5$), thus indicating that extracting too many customers leads in general to worse solutions (i.e., $z_{avg}(0.7)>z_{avg}(0.5)$). This is not completely surprising, and it is essentially due to the column generation heuristic, which falls in troubles in finding good variables for the Reallocation Model when the current solution has been almost completely "destroyed" by the removal of too many customers.

As previously seen, the proposed algorithm is able to improve on high-quality initial solutions. However, a natural question concerns the effectiveness of the method if the initial solution is instead a "bad-quality" solution. To answer this question, we implemented a modified version of the tabu search algorithm proposed by Fu et al. [13] (we refer the reader to [13] for a detailed description of this algorithm). More precisely, we first computed an initial random (and typically infeasible) solution, and then we applied only 200 iterations of the tabu search algorithm, with the aim of quickly finding a feasible solution, possibly "far" from the good ones. The computational results provided by HIP on the 16 instances C1–C14 and F11–F12 when starting from such initial solutions are reported in Table 4.

The table has the same structure as Table 1 and shows that HIP is quite effective even when the initial solution is not a good-quality solution. First, we can note that all the solutions are improved by all the five different runs. Further, even in this case the method is quite consistent, as all the five different runs provide on average very similar results, both in terms of quality of the solutions found and of CPU time. Finally, considering all the instances and all the different runs, the average behavior of the

Table 4
Computational results on the "classical" 16 benchmark instances starting from "bad initial solutions".

Pb	m	P.best	Initial		Run 1		Run 2		Run 3		Run 4		Run 5		Best		Worst		Average	
			Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev	Cost	%dev
C1	5	*416.06	467.80	12.44	417.37	0.31	417.36	0.31	*416.06	0.00	*416.06	0.00	417.37	0.31	*416.06	0.00	417.37	0.31	416.84	0.19
C2	11	564.06	657.07	16.49	564.06	0.00	564.06	0.00	564.06	0.00	564.06	0.00	564.06	0.00	564.06	0.00	564.06	0.00	564.06	0.00
C3	8	*639.74	768.93	20.19	642.14	0.38	642.14	0.38	642.98	0.51	642.14	0.38	643.75	0.63	642.14	0.38	643.75	0.63	642.63	0.45
C4	12	733.13	1069.38	45.86	738.05	0.67	741.75	1.18	748.63	2.11	742.11	1.22	744.15	1.50	738.05	0.67	748.63	2.11	742.94	1.34
C5	17	869.24	1449.20	66.72	887.40	2.09	879.89	1.23	882.12	1.48	887.48	2.10	887.85	2.14	879.89	1.23	887.85	2.14	884.95	1.81
C6	6	412.96	444.98	7.75	416.84	0.94	412.96	0.00	416.85	0.94	416.84	0.94	412.96	0.00	412.96	0.00	416.85	0.94	415.29	0.56
C7	11	568.49	654.27	15.09	568.49	0.00	568.49	0.00	568.49	0.00	568.49	0.00	569.51	0.18	568.49	0.00	569.51	0.18	568.69	0.04
C8	9	644.63	752.98	16.81	647.56	0.45	645.16	0.08	645.16	0.08	645.16	0.08	645.16	0.08	645.16	0.08	647.56	0.45	645.64	0.16
C9	14	756.14	896.61	18.58	756.81	0.09	756.81	0.09	757.78	0.22	756.38	0.03	759.60	0.46	756.38	0.03	759.60	0.46	757.48	0.18
C10	17	875.07	983.97	12.44	901.18	2.98	898.16	2.64	897.99	2.62	886.75	1.33	887.69	1.44	886.75	1.33	901.18	2.98	894.35	2.20
C11	7	682.12	835.93	22.55	690.83	1.28	689.24	1.04	691.10	1.32	692.63	1.54	691.36	1.35	689.24	1.04	692.63	1.54	691.03	1.31
C12	10	*534.24	545.25	2.06	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00	*534.24	0.00
C13	12	896.50	1025.11	14.35	902.87	0.71	912.53	1.79	904.17	0.86	905.14	0.96	905.79	1.04	902.87	0.71	912.53	1.79	906.10	1.07
C14	12	581.81	641.66	10.29	581.92	0.02	581.92	0.02	581.92	0.02	581.92	0.02	581.92	0.02	581.92	0.02	581.92	0.02	581.92	0.02
F11	4	*177.00	201.27	13.71	*177.00	0.00	*177.00	0.00	*177.00	0.00	*177.00	0.00	*177.00	0.00	*177.00	0.00	*177.00	0.00	*177.00	0.00
F12	7	769.66	919.22	19.43	783.41	1.79	784.12	1.88	782.66	1.69	785.35	2.04	784.12	1.88	782.66	1.69	785.35	2.04	783.93	1.85
Avg.				19.67		0.73		0.66		0.74		0.67		0.69		0.45		0.98		0.70

Pb	Initial t.time	Run 1 t.time	Run 1 b.time	Run 2 t.time	Run 2 b.time	Run 3 t.time	Run 3 b.time	Run 4 t.time	Run 4 b.time	Run 5 t.time	Run 5 b.time	Average t.time	Average b.time
C1	0.0	70.2	20.6	72.3	14.9	65.8	33.6	61.9	14.8	61.4	25.0	66.3	21.8
C2	0.1	86.2	62.2	78.2	28.7	94.3	32.0	83.0	58.5	77.2	52.1	83.8	46.7
C3	0.1	136.4	69.1	110.7	98.4	110.3	37.6	119.8	68.3	112.0	36.0	117.8	61.9
C4	0.2	177.8	127.5	159.2	88.3	188.9	148.5	186.4	64.3	177.5	79.1	178.0	101.5
C5	0.4	271.1	269.7	269.1	224.8	291.6	173.5	262.1	152.7	268.2	238.2	272.4	211.8
C6	0.0	48.5	0.6	44.0	26.8	51.1	0.6	50.6	0.5	45.9	10.7	48.0	7.8
C7	0.1	85.0	68.3	75.6	15.5	81.3	13.7	75.3	43.2	75.7	23.0	78.6	32.7
C8	0.1	153.1	87.3	160.8	123.6	169.6	29.2	153.3	51.3	151.7	110.5	157.7	80.4
C9	0.2	295.2	138.5	298.2	200.5	317.3	250.9	281.9	258.4	304.3	260.2	299.4	221.7
C10	0.4	705.2	665.0	721.8	524.1	729.1	719.1	828.4	678.8	584.2	556.8	713.7	628.8
C11	0.1	219.8	145.9	176.5	45.1	248.5	217.7	227.1	145.8	227.3	194.7	219.8	149.8
C12	0.1	99.6	23.0	89.8	61.3	96.2	12.5	102.2	74.2	96.6	27.9	96.9	39.8
C13	0.1	1113.8	393.4	1359.0	1270.8	1105.3	787.1	845.0	603.9	1244.1	562.0	1133.4	723.4
C14	0.1	363.1	213.6	327.1	124.8	486.7	421.3	305.6	159.2	452.1	325.2	386.9	248.8
F11	0.1	97.9	59.1	79.8	74.7	88.9	25.7	91.1	53.0	88.2	31.6	89.2	48.8
F12	0.2	190.9	36.4	176.7	80.3	152.2	43.5	178.8	140.4	181.7	43.9	176.1	68.9
Avg.	0.1	257.1	148.8	262.4	187.7	267.3	184.2	240.8	160.5	259.3	161.1	257.4	168.4

CPU times are expressed in seconds.

Table 5
Current best known solution costs for the tested OVRP benchmark instances.

Inst.	n	D	Best known solution						
			$F=\infty$				$F=0$		
			m	LB	Cost	Best heuristics	m	Cost	Best heuristics
C1	50		5	416.1	*416.06	[3,8,12–14,18,20]	6	412.96	[24–26]
C2	75		10	559.62	567.14	[8,12–14,18,20]	11	564.06	[24–26]
C3	100		8	639.7	*639.74	[8,12,18]	9	639.57	[26]
C4	150		12	730.2	733.13	[8,12,18,20]			
C5	199		16	848.5	879.37	[25]	17	**868.44**	
C6	50	180	6		412.96	[3,8,12–14,18,20]			
C7	75	144	10		583.19	[20]	11	568.49	[8,13,14,18]
C8	100	207	9		644.63	[3,8,12,18]			
C9	150	180	13		**757.69**		14	756.14	[8]
C10	199	180	17		**874.71**				
C11	120		7	657.1	682.12	[8,12,20]	10	678.54	[26]
C12	100		10	534.2	*534.24	[8,12,18,20,24–26]			
C13	120	648	11		**899.16**		12	**894.19**	
C14	100	936	11		591.87	[8,12,18,20]	12	581.81	[8]
F11	71		4	177.0	*177.00	[8,13,14,18,20]			
F12	134		7	762.9	**769.55**				
O1	200		5		6018.52	[8,18]			
O2	240		9		**4573.53**				
O3	280		7		7731.46	[8]			
O4	320		10		**7251.74**				
O5	360		8		9197.61	[18]	9	**9156.74**	
O6	400		9		9803.80	[18]			
O7	440		10		**10344.37**				
O8	480		10		12420.16	[8]			

algorithm is satisfactory: starting from a set of initial solutions with an average percentage deviation (w.r.t. the best known value) of 19.67, HIP finds a set of final solutions with an average percentage deviation of 0.70 in an average overall computing time of 257.4 s.

The current best known solution costs for the tested instances are given in summary in Table 5, where we also report the number of customers n and the route duration limit D associated with the vehicles. Solution costs are given both for the case $F=\infty$ (i.e., when the objective is to minimize the number of used vehicles first and the traveling cost second) and the case $F=0$ (i.e., when the objective is to minimize the traveling cost). As usual, the best known solution cost for the case $F=0$ is reported only if the traveling cost is smaller than the corresponding one for the case $F=\infty$. For each instance whose best known solution was not improved by HIP we report the algorithms providing the corresponding best known costs. Previously best known solution costs reached also by HIP (starting from a worse solution) are underlined, while new best solution costs found by HIP are in bold face. For the capacitated instances, in the case $F=\infty$, we also report the best known lower bound LB taken from [17,19].

6. Conclusions and future directions

We addressed the Open Vehicle Routing Problem (OVRP), a variant of the "classical" Vehicle Routing Problem (VRP) in which the vehicles are not required to return to the depot after completing their service. OVRP has recently received an increasing attention in the literature, and several heuristic and metaheuristic algorithms have been proposed for this problem, as well as exact approaches.

We presented a heuristic improvement procedure for OVRP based on Integer Linear Programming (ILP) techniques. Given an initial solution to be possibly improved, the method follows a destruct-and-repair paradigm, where the given solution is randomly destroyed (i.e., customers are removed in a random way) and repaired by solving an ILP model, in the attempt of finding a new improved solution.

Computational results on 24 benchmark instances from the literature showed that the proposed improvement method can be used as a profitable tool for finding good-quality OVRP solutions, and that even extremely good quality solutions found by the most effective metaheuristic techniques proposed for OVRP can be improved. Out of 30 best known solutions which are not provably optimal, in 10 cases the proposed method was able to improve on the best known solution reported in the literature.

Future directions of work could involve more sophisticated criteria for removing customers from the current solution, as well as more sophisticated algorithms for solving the column generation problem related to the ILP model. On the other side, the overall procedure can be considered as a general framework and it could be extended to cover other variants of Vehicle Routing Problems, as, for example, Vehicle Routing Problems with heterogenous vehicles and multi-depot Vehicle Routing Problems.

Acknowledgments

This work has been partially supported by MIUR (Ministero Istruzione, Università e Ricerca), Italy. We wish to thank Zhuo Fu, Richard Eglese, Leon Li, Krzysztof Fleszar, Ibrahim H. Osman, Khalil S. Hindi, David Pisinger, Stefan Ropke, Ulrich Derigs and Katharina Reuter who provided the initial solutions used in the computational experiments. We also thank the referees for their constructive and useful comments and remarks.

Appendix

New best known solutions found by HIP (see Table A1).

Please cite this article as: Salari M, et al. An ILP improvement procedure for the Open Vehicle Routing Problem. Computers and Operations Research (2010), doi:10.1016/j.cor.2010.02.010

Table A1

Problem `c5`. *m*: 17
cost: 868.44
Solution:

1:	0	105	26	149	195	179	54	130	165	55	25	170	67		
2:	0	112	183	6	96	99	104	59	93	85	61				
3:	0	53	40	21	73	171	74	72	197	75	133	22	41	145	
4:	0	28	184	76	196	116	77	3	158	79	129	169	121	29	
5:	0	58	152	137	2	115	178	144	57	15	43				
6:	0	166	83	199	114	8	174	46	124	168	47	36	143	49	64
7:	0	27	132	69	162	101	70	30	160	128	20	188	66		
8:	0	111	50	102	157	33	185	81	120	164	34	78			
9:	0	154	138	12	109	177	150	80	68	134	163	24			
10:	0	180	198	110	4	155	139	187	39	56	186	23			
11:	0	146	52	153	106	194	7	182	88	148	62	159			
12:	0	13	117	151	92	37	98	100	193	91	191	141	16	86	
13:	0	156	147	60	118	5	84	173	113	17	45	125			
14:	0	167	127	190	31	189	10	108	90	32	131				
15:	0	89	18	82	48	123	19	107	175	11	126	63	181		
16:	0	94	95	97	87	172	42	142	14	192	119	44	140	38	
17:	0	176	1	122	51	9	103	161	71	135	35	136	65		

Problem `c9`. *m*: 13
cost: 757.69
Solution:

1:	0	108	37	52	15	107	44	137	92	42	93	65		
2:	0	139	18	110	133	25	95	67	13	136	40	88		
3:	0	46	102	6	57	132	98	23	69	7	61	114		
4:	0	100	2	83	131	20	59	101	3	121	36	115		
5:	0	56	144	146	109	148	87	150	141	66	41	94	19	64
6:	0	27	81	138	48	112	60	8	26	113	140	82	31	
7:	0	38	62	9	130	50	118	21	79	74	34	104	30	
8:	0	90	71	123	122	124	45	91	72	33	125	106	73	
9:	0	12	47	68	14	58	96	24	97	86	43	99		
10:	0	63	17	145	147	142	4	149	143	135	111	55	134	
11:	0	77	32	119	51	1	120	22	80	70	28	116		
12:	0	103	5	76	49	10	54	105	75	39	89	117		
13:	0	78	11	126	16	127	53	129	29	128	84	35	85	

Problem `c10`. *m*: 17
cost: 874.71
Solution:

1:	0	28	184	116	68	150	80	134	163	24	29	121		
2:	0	69	162	101	70	30	20	188	128	160	131	32	181	
3:	0	111	50	102	157	33	81	120	135	35	136	65		
4:	0	105	26	149	195	179	110	155	4	139	187	39		
5:	0	146	153	82	48	124	168	47	36	143	49	64		
6:	0	6	96	104	99	93	85	193	91	191	141	44	140	38
7:	0	147	60	118	5	84	173	61	16	86	113	17		
8:	0	152	58	137	2	178	115	145	41	22	133	74	171	
9:	0	27	132	176	1	122	51	9	103	161	71	66		
10:	0	112	13	117	97	87	144	57	172	42	142	43	15	
11:	0	154	138	12	109	177	54	130	165	55	25	170	67	
12:	0	183	94	95	59	151	92	37	98	100	192	119	14	
13:	0	167	127	190	31	88	148	123	19	107	175	11		
14:	0	156	89	166	18	83	199	114	8	125	45	174	46	
15:	0	76	196	77	3	158	185	79	129	169	78	34	164	

Table A1. (*continued*)

16:	0	52	106	194	7	182	62	159	10	189	108	90	126	63
17:	0	53	40	180	198	21	73	72	197	75	56	186	23	

Problem `c13`. *m*: 11
cost: 899.16
Solution:

1:	0	87	92	37	38	39	42	41	44	47	46	49	
2:	0	112	84	7	9	10	11	15	14	13	12	8	
3:	0	98	68	79	80	53	55	58	56	60	63		
4:	0	119	81	117	113	83	6	5	4	3	2	1	
5:	0	67	69	70	71	74	72	75	78	77	76	73	
6:	0	110	52	54	57	59	65	61	62	64	66		
7:	0	21	20	23	26	28	32	35	34	36	29		
8:	0	95	93	94	97	115	40	43	45	48	51	50	
9:	0	88	82	111	86	85	89	91	90	114	18	118	108
10:	0	109	17	16	19	25	22	24	27	31	30	33	
11:	0	120	105	106	107	104	103	116	100	99	101	102	96

Problem `c13`. *m*: 12
cost: 894.19
Solution:

1:	0	109	17	16	19	25	22	24	27	31	30	33	
2:	0	82	111	86	85	89	92	91	90	114	18	118	108
3:	0	88	87	95	102	105	106	107	104	101	99	100	116
4:	0	21	20	23	26	28	32	35	34	36	29		
5:	0	67	69	70	71	74	75	72	78	77	76	73	
6:	0	112	84	7	9	10	11	15	14	13	12	8	
7:	0	119	81	117	113	83	6	5	4	3	2	1	
8:	0	103	68	79	80	53	55	58	56	60	63		
9:	0	98	52	54	57	59	65	61	62	64	66		
10:	0	93	94	97	110	40	43	45	48	51	50		
11:	0	96	115	37	38	39	42	41	44	47	46	49	
12:	0	120											

Problem `F12`. *m*: 7
cost: 769.55
Solution:

1:	0	73	74	77	64	76	134	32	34	48	49	62	50	51	52	53	102
		56	57	105	93	94	45	39	44	43	40	3	41	42	2	4	5
		6	7	8	9	10	12	11	14	88	15	13	16				
2:	0	20	82	19	65	130	119	117	116	131	115	114					
3:	0	91	21	25	26	27	28	30	29	92	90	89	87	86	85	84	83
4:	0	17	18	132	125	111	110	122	123	124	128	129	113				
5:	0	81	112	126	127	121	120	109	108	107	106						
6:	0	22	24	23	72	47	75	1	61	60	59	31	58	54	55	103	104
		101	35	36	99	100	98	97	96	38	37	95					
7:	0	46	118	71	66	78	63	79	67	133	33	80	68	69	70		

Problem `O2`. *m*: 9
cost: 4573.53
Solution:

1:	0	16	56	55	95	135	134	133	132	131	130	170	171	172	173	174	175
		176	177	178	179	180	181	182	183	184	185						
2:	0	42	43	44	84	83	82	122	121	161	162	202	201	240	239	238	237
		236	235	234	233	232	231	230	229	228	227						
3:	0	18	17	57	97	96	136	137	138	139	140	141	142	143	144	145	146
		147	148	149	150	151	152	153	154	155	156	157	158	159			
4:	0	13	53	52	51	50	49	48	47	46	45	85	86	87	127	126	125

		124	123	163	164	165	166	206	205	204	203						
5:	0	15	14	54	94	93	92	91	90	89	88	128	129	169	168	167	207
		208	209	210	211	212	213	214	215	216	217	218					
6:	0	25	24	23	22	21	20	60	61	62	63	64	65	66	67	68	69
		70	71	72	73	74	75	76	77	78	79	80					
7:	0	1	41	81	120	160	200	199	198	197	196	195	194	193	192	191	190
		189	188	187	186	226	225	224	223	222	221	220	219				
8:	0	19	59	58	98	99	100	101	102	103	104	105	106	107	108	109	110
		111	112	113	114	115	116	117	118	119							
9:	0	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	2
		3	4	5	6	7	8	9	10	11	12						

Problem o4. *m*: 10
cost: 7251.74
Solution:

1:	0	39	79	78	77	76	75	74	114	115	116	117	118	119	159	160	121
		122	162	161	201	241	281	282	242	202	203	243	283	284	244	204	205
		245	285														
2:	0	28	68	69	70	71	72	73	113	112	111	110	109	108	148	149	150
		151	152	153	154	155	156	157	158	198	199	200	240	280	320	319	279
		239	238	278	318												
3:	0	24	64	65	105	104	103	102	142	143	144	145	146	186	226	266	306
		307	267	227	228	268	308	309	269	229	230	270	310	311	271	231	232
		272	312														
4:	0	15	16	17	18	19	20	60	59	58	57	56	55	54	94	95	96
		97	98	99	100												
5:	0	14	13	53	93	133	134	135	136	176	175	174	173	213	253	293	294
		254	214	215	255	295	296	256	216	217	257	297	298	258	218	219	220
		260	300	299	259												
6:	0	25	26	27	67	66	106	107	147	187	188	189	190	191	192	193	194
		195	196	197	237	277	317	316	276	236	235	275	315	314	274	234	233
		273	313														
7:	0	11	51	50	49	48	47	46	45	44	43	42	41	80	120	81	82
		83	84	124	123	163	164	165	166	167	207	206	246	286	287	247	
8:	0	21	22	23	63	62	61	101	141	140	139	138	137	177	178	179	180
		181	182	183	184	185	225	265	305	304	264	224	223	263	303	302	262
		222	221	261	301												
9:	0	10	9	8	7	6	5	4	3	2	1	40	38	37	36	35	34
		33	32	31	30	29											
10:	0	12	52	92	91	90	89	88	87	86	85	125	126	127	128	129	130
		131	132	172	171	170	169	168	208	248	288	289	249	209	210	250	290
		291	251	211	212	252	292										

Problem o5. *m*: 9
cost: 9156.74
Solution:

1:	0	3	39	38	37	73	108	107	106	105	104	103	102	138	139	140	141
		142	143	144	109	145	180	179	215	251	287	323	359	360	324	288	252
		216	181	217	253	289	325										
2:	0	19	56	57	58	59	60	96	95	94	93	92	91	127	128	129	130
		131	132	168	167	166	165	164	200	236	272	308	344	345	309	273	237
		201	202	238	274	310	346	347	311	275	239	203	204	240	276	312	348
3:	0	9	10	11	12	13	49	48	47	46	45	44	43	42	78	79	80
		81	82	83	84	85	121	120	119	118	117	116	115	114	150	151	152
		153	189	225	261	297	333	332	296	260	224	188	187	223	259	295	331
4:	0	18	55	54	53	52	88	89	90	126	125	124	123	159	160	161	162
		163	199	235	271	307	343	342	306	270	234	198	197	233	269	305	341
		340	304	268	232	196	195	231	267	303	339						
5:	0	8	7	6	5	4	40	41	77	76	75	74	110	111	112	113	149
		148	147	146	182	218	254	290	326	327	291	255	219	183	184	220	256

Table A1. (*continued*)

		292	328	329	293	257	221	185	186	222	258	294	330				
6:	0	17	16	15	14	50	51	87	86	122	158	194	230	266	302	338	337
		301	265	229	193	157	156	155	154	190	226	262	298	334	335	299	263
		227	191	192	228	264	300	336									
7:	0	20	21	22	23	24	25	61	62	63	64	100	99	98	97	133	134
		135	136	172	171	170	169	205	241	277	313	349	350	314	278	242	206
		207	243	279	315	351	352	316	280	244	208	209	245	281	317	353	
8:	0	34	35	33	32	31	30	29	28	28	26						
9:	0	2	1	36	72	71	70	69	68	67	66	65	101	137	173	174	175
		176	177	178	214	250	286	322	358	357	321	285	249	213	212	248	284
		320	356	355	319	283	247	211	210	246	282	318	354				

Problem o7. *m*: 10
cost: 10344.37
Solution:

1:	0	19	18	17	61	105	104	148	149	150	151	152	153	197	196	195	194
		193	192	191	190	189	233	234	235	236	237	238	239	240	241	285	329
		373	417	416	372	328	284	283	327	371	415	414	370	326	282	281	325
		369	413														
2:	0	39	83	127	171	172	216	215	214	213	212	211	210	209	208	207	251
		252	253	254	298	342	386	430	429	385	341	297	296	340	384	428	427
		383	339	295	294	338	382	426									
3:	0	37	38	82	81	125	126	170	169	168	167	166	165	164	163	162	161
		160	159	158	157	156	155	199	200	201	202	203	204	205	206	250	249
		248	247	291	335	379	423	424	380	336	292	293	337	381	425		
4:	0	23	24	25	26	27	28	29	30	31	32	33	34	35	36	80	79
		78	77	76	75	74	73	72	71	70	69	68	67	111	112	113	114
		115	116	117	118	119	120	121	122	123	124						
5:	0	20	21	22	66	65	64	63	62	106	107	108	109	110	154	198	242
		243	244	245	246	290	334	378	422	421	377	333	289	288	332	376	420
		419	375	331	287	286	330	374	418								
6:	0	11	10	9	8	7	6	5	49	50	51	52	53	97	96	140	141
		142	186	185	184	228	229	230	231	275	319	363	407	406	362	318	274
		273	317	361	405	404	360	316	272	271	315	359	403				
7:	0	4	3	2	46	47	48	92	93	94	95	139	138	137	136	180	181
		182	183	227	226	225	224	268	267	266	310	354	398	399	355	311	312
		356	400	401	357	313	269	270	314	358	402						
8:	0	42	43	44	1	45	88	87	86	130	131	132	89	90	91	135	134
		133	176	175	219	220	177	178	179	223	222	221	264	263	262	306	307
		351	395	439	440	396	352	308	265	309	353	397					
9:	0	12	13	14	15	16	60	59	58	57	56	55	54	98	99	100	101
		102	103	147	146	145	144	143	187	188	232	276	320	364	408	409	365
		321	277	278	322	366	410	411	367	323	279	280	324	368	412		
10:	0	40	41	85	84	128	129	173	174	218	217	261	260	259	258	257	256
		255	299	343	387	431	432	388	344	300	301	345	389	433	434	390	346
		302	303	347	391	435	436	392	348	304	305	349	393	437	438	394	350

References

[1] Archetti C, Bertazzi L, Hertz A, Speranza MG. A hybrid heuristic for an inventory-routing problem. Technical Report no. 317, University of Brescia, Brescia, Italy, 2009.
[2] Archetti C, Speranza MG, Savelsbergh MWP. An optimization-based heuristic for the split delivery vehicle routing problem. Transportation Science 2008;42:22–31.
[3] Brandão J. A tabu search algorithm for the open vehicle routing problem. European Journal of Operational Research 2004;157:552–64.
[4] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C, editors. Combinatorial optimization. Chichester: Wiley; 1979. p. 313–38.
[5] Cordeau J-F, Laporte G, Savelsbergh MWP, Vigo D. Vehicle routing. In: Barnhart C, Laporte G, editors. Transportation. Handbooks in operations research and management science, vol. 14. Amsterdam: Elsevier; 2007. p. 367–428.
[6] Danna E, Rothberg E, Le Pape C. Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming 2005;102(Ser. A):71–90.
[7] De Franceschi R, Fischetti M, Toth P. A new ILP-based refinement heuristic for vehicle routing problems. Mathematical Programming 2006;105:471–99.
[8] Derigs U, Reuter K. A simple and efficient tabu search heuristic for solving the open vehicle routing problem. Journal of the Operational Research Society 2009; 60:1658–69.
[9] Feillet D, Dejax P, Gendreau M, Gueguen C. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 2004;44:216–29.
[10] Fischetti M, Lodi A. Local branching. Mathematical Programming 2003;98(Ser. B):23–47.
[11] Fisher M. Optimal solutions of vehicle routing problems using minimum k-trees. Operations Research 1994;42:626–42.
[12] Fleszar K, Osman IH, Hindi KS. A variable neighbourhood search for the open vehicle routing problem. European Journal of Operational Research 2009;195:803–9.
[13] Fu Z, Eglese R, Li LYO. A new tabu search heuristic for the open vehicle routing problem. Journal of the Operational Research Society 2005;56:267–74.
[14] Fu Z, Eglese R, Li LYO. Corrigendum: a new tabu search heuristic for the open vehicle routing problem. Journal of the Operational Research Society 2006;57:1018.
[15] Hewitt M, Nemhauser GL, Savelsbergh MWP. Combining exact and heuristic approaches for the capacitated fixed charge network flow problem. INFORMS Journal on Computing 2009, 1-12, doi: 10.1287/ijoc.1090.0348.
[16] IBM ILOG Cplex 〈http://www.ilog.com〉.
[17] Letchford AN, Lysgaard J, Eglese RW. A branch-and-cut algorithm for the capacitated open vehicle routing problem. Journal of the Operational Research Society 2007;58:1642–51.
[18] Li F, Golden B, Wasil E. The open vehicle routing problem: algorithms, large-scale test problems, and computational results. Computers & Operations Research 2007;34:2918–30.
[19] Pessoa A, Poggi de Aragão M, Uchoa E. Robust branch-cut-and-price algorithms for vehicle routing problems. In: Golden B, Raghavan S, Wasil E, editors. The vehicle routing problem: latest advances and new challenges. New York: Springer; 2008. p. 297–325.
[20] Pisinger D, Ropke S. A general heuristic for vehicle routing problems. Computers & Operations Research 2007;34:2403–35.
[21] Righini G, Salani M. New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks 2008 155–70.
[22] Sariklis D, Powell S. A heuristic method for the open vehicle routing problem. Journal of the Operational Research Society 2000;51:564–73.
[23] Schrage L. Formulation and structure of more complex/realistic routing and scheduling problems. Networks 1981;11:229–32.
[24] Tarantilis CD, Diakoulaki D, Kiranoudis CT. Combination of geographical information system and efficient routing algorithms for real life distribution operations. European Journal of the Operational Research 2004;152:437–53.
[25] Tarantilis CD, Ioannou G, Kiranoudis CT, Prastacos GP. A threshold accepting approach to the open vehicle routing problem. RAIRO Operations Research 2004;38:345–60.
[26] Tarantilis CD, Ioannou G, Kiranoudis CT, Prastacos GP. Solving the open vehicle routing problem via a single parameter metaheuristic algorithm. Journal of the Operational Research Society 2005;56:588–96.
[27] Toth P, Tramontani A. An integer linear programming local search for capacitated vehicle routing problems. In: Golden B, Raghavan S, Wasil E, editors. The vehicle routing problem: latest advances and new challenges. New York: Springer; 2008. p. 275–95.
[28] Toth P, Vigo D. The vehicle routing problem. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia: SIAM; 2002.