J Heuristics (2013) 19:201–232
DOI 10.1007/s10732-011-9186-y

An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem

Puca Huachi Vaz Penna · Anand Subramanian · Luiz Satoru Ochi

Received: 15 May 2010 / Accepted: 29 July 2011 / Published online: 8 September 2011

Abstract This paper deals with the Heterogeneous Fleet Vehicle Routing Problem (HFVRP). The HFVRP is $\mathcal{NP}$-hard since it is a generalization of the classical Vehicle Routing Problem (VRP), in which clients are served by a heterogeneous fleet of vehicles with distinct capacities and costs. The objective is to design a set of routes in such a way that the sum of the costs is minimized. The proposed algorithm is based on the Iterated Local Search (ILS) metaheuristic which uses a Variable Neighborhood Descent procedure, with a random neighborhood ordering (RVND), in the local search phase. To the best of our knowledge, this is the first ILS approach for the HFVRP. The developed heuristic was tested on well-known benchmark instances involving 20, 50, 75 and 100 customers. These test-problems also include dependent and/or fixed costs according to the vehicle type. The results obtained are quite competitive when compared to other algorithms found in the literature.

Keywords Heterogeneous Fleet Vehicle Routing Problem · Fleet size and mix · Metaheuristic · Iterated Local Search

P.H.V. Penna
Instituto do Noroeste Fluminense de Educação Superior, Universidade Federal Fluminense, Rua João Jazbik s/n, Santo Antônio de Pádua, RJ 28470-000, Brazil

A. Subramanian
Departamento de Engenharia de Produção, Centro de Tecnologia, Universidade Federal da Paraíba, Campus I—Bloco G, Cidade Universitária, João Pessoa, PB 58051-970, Brazil
e-mail: anand@ct.ufpb.br

P.H.V. Penna (✉) · A. Subramanian · L.S. Ochi
Instituto de Computação, Universidade Federal Fluminense, Rua Passo da Pátria 156, Bloco E, 3° andar, São Domingos, Niterói, RJ 24210-240, Brazil
e-mail: ppenna@ic.uff.br

L.S. Ochi
e-mail: satoru@ic.uff.br

1 Introduction

The Vehicle Routing Problem (VRP) is one of the best known problems in the field of Operations Research. Inspired by real world applications, several variants were proposed over the years. Our interest relies on the Heterogeneous Fleet Vehicle Routing Problem (HFVRP). This variant is a generalization of the classical VRP allowing vehicles with different capacities, instead of a homogeneous fleet. This situation can be often found in practice and the HFVRP models this kind of applications.

According to Hoff et al. (2010), in industry, a fleet of vehicles is rarely homogeneous. Generally, either an acquired fleet is already heterogeneous or they become heterogeneous over the time when vehicles with different features are incorporated into the original fleet. In addition, insurance, maintenance and operating costs usually have distinct values according to the level of depreciation or usage time of the fleet. Moreover, from both a tactical and an operational point of view, a mixed vehicle fleet also increases the flexibility in terms of distribution planning.

The HFVRP practical importance can be verified by the variety of case studies found in the literature. Prins (2002) described an application in the French furniture industry involving 775 destination stores in which the heterogeneous fleet is composed by 71 vehicles. Cheung and Hang (2003) examined a case faced by transportation of air-cargo freight forwarders where the vehicle fleet is heterogeneous. Additional constraints such as backhauls and time windows were also taken into account by the authors. Tarantilis and Kiranoudis (2001) considered a real world case regarding the distribution of fresh milk in Greece that is performed using a heterogeneous fixed fleet. Tarantilis and Kiranoudis (2007) presented two planning problems where the first one dealt with the distribution of perishable foods for a major dairy company while the second one dealt with the distribution of ready concrete for a construction company. In these two cases, the fleet was admitted to be fixed and heterogeneous. A comprehensive survey on industrial aspects of combined fleet composition and routing in maritime and road-based transportation was recently performed by Hoff et al. (2010).

There are a couple of HFVRP variants often found in the literature. They are basically related to the fleet limitation (limited or unlimited) and the costs considered (dependent and/or fixed). The HFVRP with unlimited fleet, also known as the Fleet Size and Mix (FSM), was proposed by Golden et al. (1984) and it consists of determining the best fleet composition and its optimal routing scheme. Another HFVRP version, called Heterogeneous VRP (HVRP), was proposed by Taillard (1999) and it consists in optimizing the use of the available fixed fleet.

The HFVRP is $\mathcal{NP}$-hard since it includes the classical VRP as a special case, when all vehicles are identical. Therefore, (meta)heuristic algorithms are a suitable approach for obtaining high quality solutions in an acceptable computation time.

In this paper, we present a hybrid heuristic based on the Iterated Local Search (ILS) metaheuristic which uses a Variable Neighborhood Descent procedure, with a random neighborhood ordering (RVND), in the local search phase. According to Lourenço et al. (2003), ILS contains several of the desirable features of a metaheuristic such as simplicity, robustness, effectiveness and the ease of implementation. The authors also described a number of well-succeeded ILS implementations for different Combinatorial Optimization problems such as the Traveling Salesman Problem

(TSP), Job Shop, Flow Shop, MAX-SAT, etc. Surprisingly, to date there are relatively few applications of this metaheuristic to VRPs (see, for example, Bianchi et al. 2006; Ibaraki et al. 2008; Prins 2009a; Subramanian et al. 2010; Chen et al. 2010). Nevertheless, the computational results found by these researchers who have made use of an ILS approach to solve some VRP variant are quite encouraging. To the best of our knowledge, this is the first ILS approach developed for the HFVRP. The proposed heuristic is an extension of the one presented by Subramanian et al. (2010) for the VRP with Simultaneous Pickup and Delivery. Five HFVRP variants were tackled and the results obtained were compared with other solution approaches found in the literature.

The remainder of this paper is organized as follows. Section 2 describes the HFVRP and its main variants. Section 3 reviews some works related to the HFVRP. Section 4 provides a brief outline of the ILS metaheuristic. Section 5 explains the proposed hybrid heuristic. Section 6 contains the results obtained and a comparison with those reported in the literature. Section 7 presents the concluding remarks of this work.

2 Problem description

The HFVRP is defined in the literature as follows. Let $G = (V, A)$ be a directed graph where $V = \{0, 1, \ldots, n\}$ is a set composed by $n + 1$ vertices and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the set of arcs. The vertex 0 denotes the depot, where the vehicle fleet is located, while the set $V' = V \setminus \{0\}$ is composed by the remaining vertices that represent the n customers. Each customer $i \in V'$ has a non-negative demand q_i. The fleet is composed by m different types of vehicles, with $M = \{1, \ldots, m\}$. For each $u \in M$, there are m_u available vehicles, each with a capacity Q_u. Every vehicle is associated with a fixed cost denoted by f_u. Finally, for each arc $(i, j) \in A$ there is an associated cost $c_{ij}^u = d_{ij} r_u$, where d_{ij} is the distance between the vertices (i, j) and r_u is a dependent (variable) cost per distance unit, of a vehicle u.

A route is defined by the pair (R, u), with $R = (i_1, i_2, \ldots, i_{|R|})$ and $i_1 = i_{|R|} = 0$ and $\{i_2, \ldots, i_{|R|-1}\} \subseteq V'$, that is, each route is a circuit in G, including the depot, associated with a vehicle $u \in M$. A route (R, u) is feasible, if the customers demands do not exceed the capacity of the vehicle. The cost associated to a route is the sum of the fixed cost of the corresponding vehicle and the cost of the traversed arcs. This way, the HFVRP consists in finding feasible routes in such a way that each customer is visited exactly once; the maximum number of routes defined for a vehicle $u \in M$ do not exceed m_u; and the sum of the costs is minimized.

The present work deals with the five following variants:

i. HVRPFD, limited fleet, with fixed and dependent costs;
ii. HVRPD, limited fleet, with dependent costs but without fixed costs, i.e., $f_u = 0, \forall k \in M$;
iii. FSMFD, unlimited fleet, i.e., $m_u = +\infty, \forall k \in M$, with fixed and dependent costs;
iv. FSMF, unlimited fleet, with fixed costs but without dependent costs, i.e., $c_{ij}^{u_1} = c_{ij}^{u_2} = c_{ij}, \forall u_1, u_2 \in M, u_1 \neq u_2, \forall (i, j) \in A$;
v. FSMD, unlimited fleet, with dependent costs but without fixed costs.

3 Literature review

The first HFVRP variant studied in the literature was the FSM, initially proposed by Golden et al. (1984). The authors developed two heuristics where the first one is based on the savings algorithm of Clarke and Wright (1964), while the second one makes use of a giant tour scheme. They also proposed a mathematical formulation for the FSMF and presented some lower bounds.

Some exact approaches were developed for the FSM. Yaman (2006) suggested valid inequalities and presented lower bounds for the FSMF. Choi and Tcha (2007) obtained lower bounds for all FSM variants by means of a column generation algorithm based on a set covering formulation. Pessoa et al. (2009) proposed a Branch-Cut-and-Price (BCP) algorithm also capable of solving all FSM variants. The same authors also employed a BCP algorithm over an extended formulation to solve the FSM and other VRPs such as the Open VRP and the Asymmetric VRP (Pessoa et al. 2008). More recently, Baldacci and Mingozzi (2009) put forward a set-partitioning based algorithm that uses bounding procedures based on linear relaxation and Lagrangian relaxation to solve the five HFVRP variants mentioned in Sect. 2. Their solution method is capable of solving instances with up to 100 customers and, to our knowledge, this is the best exact approach proposed in the HFVRP literature.

Some authors implemented heuristic procedures based on Evolutionary Algorithms. Ochi et al. (1998a) developed a hybrid evolutionary heuristic that combines a Genetic Algorithm (GA) (Holland 1975) with Scatter Search (Glover et al. 2003) to solve the FSMF. A parallel version, based on the island model, of the same algorithm was presented by Ochi et al. (1998b). A hybrid GA that applies a local search as a mutation method was proposed by Liu et al. (2009) to solve the FSMF and the FSMD. A Memetic Algorithm (MA) (Moscato and Cotta 2003) was proposed by Lima (2004) for solving FSMF. Two heuristic procedure based on the same metaheuristic were developed by Prins (2009b) to solve all FSM variants and the HVRPD.

Renaud and Boctor (2002) proposed a sweep-based heuristic for the FSMF that integrates classical construction and improvement VRP approaches. Imran et al. (2009) developed a Variable Neighborhood Search (VNS) (Mladenovic and Hansen 1997) algorithm that makes use of a procedure based on Dijkstra's and sweep algorithms for generating an initial solution and several neighborhood structures in the local search phase. The authors considered all FSM variants.

A couple of Tabu Search (TS) (Glover 1986) heuristics were proposed to solve the FSMF and the FSMD. Gendreau et al. (1999) suggested a TS algorithm that incorporates a GENIUS approach and an AMP. Lee et al. (2008) developed an algorithm that combines TS with a SP approach. More recently, Brandão (2009) proposed a deterministic TS that makes use of different procedures for generating initial solutions.

The HVRP was proposed by Taillard (1999). The author developed an algorithm based on AMP, TS and column generation which was also applied to solve the FSM.

Prins (2002) dealt with the HVRP by implementing a heuristic that extends a series of VRP classical heuristics followed by a local search procedure based on the Steepest Descent Local Search and TS.

Tarantilis et al. (2003) solved the HVRPD by means of a threshold accepting approach that consists of an adaptation of the SA procedure in which a worse solution is only accepted if it is within a given threshold. The same authors (Tarantilis et al. 2004) also proposed another threshold accepting procedure to solve the same variant. Li et al. (2007) put forward a record-to-record travel algorithm that, also as the threshold method, consists of a deterministic variant of the SA. The authors considered both HVRPFD and HVRPD.

A HFVRP comprehensive survey containing all the five variants mentioned here can be found in Baldacci et al. (2008).

4 A brief overview of the ILS metaheuristic

The proposed general heuristic is mostly based on the ILS framework. Before describing the solution method, a brief outline of this metaheuristic is provided.

Consider a local optimum solution that has been found by a local search algorithm. Instead of restarting the same procedure from a completely new solution, the ILS metaheuristic applies a local search repeatedly to a set of solutions obtained by perturbing previously visited local optimal solutions. The essential idea of ILS resides in the fact that it focuses on a smaller subset, instead of considering the total space of solutions. This subset is defined by the local optimum of a given optimization procedure (Lourenço et al. 2003). To implement an ILS algorithm, four procedures should be specified: (i) `GenerateInitialSolution`, where an initial solution is constructed; (ii) `LocalSearch`, which improves the solution initially obtained; (iii) `Perturb`, where a new starter point is generated through a perturbation of the solution returned by the `LocalSearch`; (iv) `AcceptanceCriterion`, that determines from which solution the search should continue. Algorithm 1 describes how these components are combined to build the ILS framework.

Algorithm 1: ILS

```
Procedure ILS
    s0 ← GenerateInitialSolution
    s* ← LocalSearch(s0)
    while Stopping criterion is not met do
        s' ← Perturb(s*, history)
        s*' ← LocalSearch(s')
        s* ← AcceptanceCriterion(s*, s*', history)
    end
end
```

The modification realized in the perturbation phase is used to escape from a current locally optimal solution. Frequently, the move is randomly chosen within a larger neighborhood than the one utilized in the local search, or a move that the local search cannot undo in just one step. In principle, any local search method can be used. However, ILS performance, in terms of the solution quality and computational effort, strongly depends on the chosen procedure. The acceptance criterion is used to decide the next solution that should be perturbed. The choice of this criterion is important because it controls the balance between intensification and diversification. The search history is employed for deciding if some previously found local optimum should be chosen. The performance of the ILS procedure strongly depends on the intensity of the perturbation mechanisms. If it is small, not many new solutions will be explored, while if it is too large, it will adopt almost randomly starting points.

5 The ILS-RVND heuristic

This section describes the ILS-RVND heuristic and its steps are summarized in Algorithm 2. For the HVRP, the given number of vehicles of each type is initially considered, while for the FSM, one vehicle of each type is first considered (line 2). Let v be the number of vehicles (line 3). The multi-start heuristic executes *MaxIter* iterations (lines 4–24), where at each iteration a solution is generated by means of a constructive procedure (line 5). The parameter *MaxIterILS* represents the maximum number of consecutive perturbations allowed without improvements (line 8). This value is calculated based on the number of customers and vehicles and on a given parameter

Algorithm 2: ILS-RVND

```
1  Procedure ILS-RVND(MaxIter, β)
2      Initialize fleet
3      v ← total number of vehicles
4      for i ← 1 to MaxIter do
5          s ← GenerateInitialSolution(v)
6          s′ ← s
7          iterILS ← 0
8          MaxIterILS ← ComputeMaxIterILS(n, v, β)
9          while (iterILS ≤ MaxIterILS) do
10             s ← RVND(s)
11             if (unlimited fleet) then
12                 UpdateFleet()
13             end
14             if (f(s) < f(s′)) then
15                 s′ ← s
16                 iterILS ← 0
17             end
18             s ← Perturb(s′)
19             iterILS ← iterILS + 1
20         end
21         if (f(s′) < f*) then
22             s* ← s′
23         end
24     end
25     return s*
26 end
```

β (see Sect. 6.1). The main ILS loop (lines 9–20) aims to improve the generated initial solution using a RVND procedure (line 10) in the local search phase combined with a set of perturbation mechanisms (line 18). Notice that the perturbation is always performed on the best current solution (s') of a given iteration (acceptance criterion).

The next subsections provide a detailed explanation of the main components of the ILS-RVND heuristic.

5.1 Constructive procedure

The constructive procedure makes use of two insertion criteria, namely the Modified Cheapest Feasible Insertion Criterion (MCFIC) and the Nearest Feasible Insertion Criterion (NFIC). Also, two insertion strategies were employed, specifically the Sequential Insertion Strategy (SIS) and the Parallel Insertion Strategy (PIS).

The pseudocode of the constructive procedure is presented in Algorithm 3. Let the Candidate List (CL) be initially composed by all customers (line 2). Each route is filled with a seed customer k, randomly selected from the CL (lines 4–7). An insertion criterion and an insertion strategy is chosen at random (lines 8–9). An initial solution is generated using the selected combination of criterion and strategy (lines 10–14). If the solution s is infeasible we restart the constructive procedure (lines 15–17). If the fleet is unlimited (FSM), an empty route associated to each type of vehicle is added to the constructed solution s (line 18). These empty routes are necessary to allow a possible fleet resizing during the local search phase.

Algorithm 3: GenerateInitialSolution

```
1  Procedure GenerateInitialSolution(v)
2      Initialize the CL
3      Let s = {s^1, ..., s^v} be the set composed by v empty routes
4      for v' ← 1 to v do
5          s^{v'} ← k ∈ CL selected at random
6          Update CL      // CL ← CL - {k}
7      end
8      InsertionCriterion ← MCFIC or NFIC   // (chosen at random)
9      InsertionStrategy ← SIS or PIS   // (chosen at random)
10     if (InsertionStrategy = SIS) then
11         s ← SequentialInsertion(v, CL, InsertionCriterion)
12     else
13         s ← ParallelInsertion(v, CL, InsertionCriterion)
14     end
15     if (s is infeasible) then
16         Go to line 2
17     end
18     Add an empty route associated to each type of vehicle in s      // Only for FSM
19     return s
20 end
```

5.1.1 Insertion criteria

The cost of inserting an unrouted customer $k \in \text{CL}$ in a given route using the MCFIC is expressed in Eq. 1, where function $g(k)$ represents the insertion cost. The value of $g(k)$ is computed by the sum of two terms. The first computes the insertion cost of the client k between every pair of adjacent customers i and j while the second corresponds to a surcharge used to avoid late insertions of clients located far away from the depot. The cost back and forth from the depot is weighted by a factor γ.

$$g(k) = \left(c^u_{ik} + c^u_{kj} - c^u_{ij}\right) - \gamma\left(c^u_{0k} + c^u_{k0}\right) \tag{1}$$

The NFIC directly computes the distance between a customer $k \in \text{CL}$ and every customer i that has been already included into the partial solution, that is $g(k) = c^u_{ik}$ (Eq. 2). It is assumed that the insertion of k is always performed after i.

$$g(k) = c^u_{ik} \tag{2}$$

In both criteria, the insertion associated with the least-cost is done, i.e. $\min\{g(k) | k \in \text{CL}\}$.

5.1.2 Insertion strategies

In the SIS, only a single route is considered for insertion at each iteration. The pseudocode of the SIS is presented in Algorithm 4. If the insertion criterion corresponds to the MCFIC, a value of γ is chosen at random within the discrete interval $\{0.00, 0.05, 0.10, \ldots, 1.65, 1.70\}$ (line 2). This interval was defined in Subramanian et al. (2010) after some preliminary experiments. While the CL is not empty and there is at least one customer $k \in \text{CL}$ that can be added to the current partial solution without violating any constraint (lines 6–17), each route is filled with a customer selected using the corresponding insertion criterion (lines 7–15). If the fleet is unlimited and the solution s is still incomplete, a new vehicle, chosen at random from the available types, is added and the procedure restarts from line 6 (lines 18–22).

PIS differs from SIS because all routes are considered while evaluating the least-cost insertion. Algorithm 5 illustrates the pseudocode of the PIS. While the CL is not empty and there is at least one customer $k \in \text{CL}$ that can be included in s (lines 5–12), the insertions are evaluated using the selected insertion criterion and the customer associated with the least-cost insertion is then included in the correspondent route v (lines 6–10). The remainder of the code operates just as the SIS.

5.2 Local search

The local search is performed by a VND (Mladenovic and Hansen 1997) procedure, which utilizes a random neighborhood ordering (RVND). Let $N = \{N^{(1)}, \ldots, N^{(r)}\}$ be the set of neighborhood structures. Whenever a given neighborhood of the set N fails to improve the incumbent solution, the RVND randomly chooses another neighborhood from the same set to continue the search throughout the solution space. In this case, N is composed only by inter-route neighborhood structures.

Algorithm 4: SequentialInsertion

```
1  Procedure SequentialInsertion(v, CL, InsertionCriterion)
2      if (InsertionCriterion = MCFIC) then
3          γ ← random value within a given interval
4      end
5      v0 ← 1
6      while (CL ≠ ∅ and at least one customer k ∈ CL can be added to s) do
7          for v' ← v0 ... v and CL ≠ ∅ do
8              if (at least one customer k ∈ CL can be inserted into the vehicle v') then
9                  Evaluate the value of each cost g(k) for k ∈ CL
10                 g^min ← min{g(k)|k ∈ CL}
11                 k' ← customer k associated to g^min
12                 s^v' ← s^v' ∪ {k'}
13                 Update CL
14             end
15         end
16         Update v0
17     end
18     if (CL > 0 and the vehicle fleet is unlimited) then
19         Add a new vehicle chosen at random        // v ← v + 1
20         Update v0      // v0 ← v
21         Go to line 6
22     end
23     return s
24 end
```

Algorithm 5: ParallelInsertion

```
1  Procedure ParallelInsertion(v, CL, InsertionCriterion)
2      if (InsertionCriterion = MCFIC) then
3          γ ← random value within a given interval
4      end
5      while (CL ≠ ∅ and at least one customer k ∈ CL can be added to s) do
6          Evaluate the value of each cost g(k) for k ∈ CL
7          g^min ← min{g(k)|k ∈ CL}
8          k' ← customer k associated to g^min
9          v' ← route associated to g^min
10         s^v' ← s^v' ∪ {k'}
11         Update CL
12     end
13     if (CL > 0 and the vehicle fleet is unlimited) then
14         Add a new vehicle chosen at random; {v ← v + 1}
15         Go to line 5
16     end
17     return s
18 end
```

Algorithm 6: RVND

```
1  Procedure RVND(s)
2      Update ADSs
3      Initialize the Inter-Route Neighborhood List (NL)
4      while (NL ≠ 0) do
5          Choose a neighborhood N^(η) ∈ NL at random
6          Find the best neighbor s' of s ∈ N^(η)
7          if (f(s') < f(s)) then
8              s ← s'
9              s ← IntraRouteSearch(s)
10             Update Fleet    // Only for FSM
11             Update NL
12         else
13             Remove N^(η) from the NL
14         end
15         Update ADSs
16     end
17     return s
18 end
```

The pseudocode of the RVND procedure is presented in Algorithm 6. Firstly, a Neighborhood List (NL) containing a predefined number of inter-route moves is initialized (line 3). In the main loop (lines 4–16), a neighborhood $N^{(\eta)} \in$ NL is chosen at random (line 5) and then the best admissible move is determined (line 6). In case of improvement, an intra-route local search is performed, the fleet is updated and the NL is populated with all the neighborhoods (lines 7–12). Otherwise, $N^{(\eta)}$ is removed from the NL (line 13). It is important to mention that the fleet is only updated for the FSM. This update assures that there is exactly one empty vehicle of each type. A set of Auxiliary Data Structures (ADSs) (see Sect. 5.2.1) is updated at the beginning of the process (line 6) and whenever a neighborhood search is performed (line 5).

Let N' be a set composed by r' intra-route neighborhood structures. Algorithm 7 describes how the intra-route search procedure was implemented. At first, a neighborhood list NL$'$ is initialized with all the intra-route neighborhood structures (line 2). Next, while NL$'$ is not empty a neighborhood $N'^{(\eta)} \in$ NL$'$ is randomly selected and a local search is exhaustively performed until no more improvements are found (lines 3–11).

5.2.1 Auxiliary Data Structures (ADSs)

In order to enhance the neighborhood search, some ADSs were adopted. The following arrays store useful information regarding each route.

- **SumDemand[ ]**—sum of the demands. For example, if SumDemand[2] = 100, it means that the sum of the demands of all customers of route 2 corresponds to 100.
- **MinDemand[ ]**—minimum demand. For example, if MinDemand[3] = 5, it means that 5 is the least demand among all customers of route 3.

Algorithm 7: IntraRouteSearch

```
Procedure IntraRouteSearch(s)
    Initialize the Intra-Route Neighborhood List (NL')
    while (NL' ≠ 0) do
        Choose a neighborhood N'^(η) ∈ NL' at random
        Find the best neighbor s' of s ∈ N'^(η)
        if f(s') < f(s) then
            s ← s'
        else
            Remove N'^(η) from the NL'
        end
    end
    return s
end
```

– **MaxDemand[ ]**—maximum demand.
– **MinPairDemand[ ]**—minimum sum of demands of two adjacent customers. For example, if MinPairDemand[1] = 10, it means that the least sum of the demands of two adjacent customers of route 1 corresponds to 10.
– **MaxPairDemand[ ]**—maximum sum of demands of two adjacent customers.
– **CumulativeDemand[ ][ ]**—cumulative load at each point of the route. For example, if CumulativeDemand[2][4] = 78, it means that the sum of the demands of the first four customers of route 2 corresponds to 78.
– **NeighborhoodStatus[ ][ ]**—informs if the route has been modified after the neighborhood has failed to find an improvement move involving the same route. For example, if NeighborhoodStatus[1][3] = true, it means that the last time the neighborhood $N^{(1)}$ was applied, no improvement move involving route 3 was found, but this route was later modified by another neighborhood structure or by a perturbation move. If NeighborhoodStatus[1][3] = false, it means that the route 3 did not suffer any change after the last time $N^{(1)}$ was unsuccessful to find an improvement move involving this route.

To update the information of the ADSs one should take into account only the routes that were modified. Let $\bar{n}$ be the total number of customers in the modified routes. Except for the NeighborhoodStatus, the ADSs updating is as follows. For each modified route, a verification is performed along the whole tour to update the corresponding values of the ADSs. Hence, the computational complexity is of the order of $\mathcal{O}(\bar{n})$. As for the NeighborhoodStatus, each route, the information regarding all inter-routes neighborhoods are updated which results in a computational complexity is of the order of of $\mathcal{O}(\bar{v}|N|)$, where $\bar{v}$ is the number of modified routes.

5.2.2 Inter-route neighborhood structures

Seven VRP neighborhood structures involving inter-route moves were employed. Five of them are based on the λ-interchanges scheme (Osman 1993), which consists of exchanging up to λ customers between two routes. To limit the number of possibilities we have considered $\lambda = 2$. Another one is based on the Cross-exchange operator (Taillard et al. 1997), which consists of exchanging two segments of different routes. Finally, a new neighborhood structure called, K-Shift, which consists of transferring a set of consecutive customers from a route to another one, was implemented.

The solution spaces of the seven neighborhoods are explored exhaustively, that is, all possible combinations are examined, and the best improvement strategy is considered. The computational complexity of each one of these moves is $\mathcal{O}(n^2)$.

In order to accelerate the local search, a set of conditions can be specified to avoid the examination of moves that are known to be infeasible. It is in this context that the ADSs play an important role. The information stored in these data structures are used in the process of identifying unnecessary moves during the local search. Each neighborhood structure has its own conditions but the idea is essentially the same. Moreover, all of them share the condition that, given a inter-route movement involving the routes r_1 and r_2 , it is worth examining a move using neighborhood $N^{(\eta)}$ only if NeighborhoodStatus[η][r_1] = true or NeighborhoodStatus[η][r_2] = true.

Only feasible moves are admitted, i.e., those that do not violate the maximum load constraints. Therefore, every time an improvement occurs, the algorithm checks whether this new solution is feasible or not. This checking is trivial and it can be performed in a constant time by just verifying if the sum of the customers demands of a given route does not exceed the vehicle's capacity when the same is leaving (or arriving on) the depot.

The inter-route neighborhood structures are described next. In addition, the particular conditions of each neighborhood that must be satisfied to avoid evaluating some infeasible moves are presented as well.

Shift(1,0) – $N^{(1)}$—A customer k is transferred from a route r_1 to a route r_2. If MinDemand[r_1] + SumDemand[r_2] $> Q_{u(r_2)}$ it means that transferring any customer from r_1 to r_2 implies an infeasible solution. This fact is easy to verify because if even the customer with the least demand cannot be transferred to the other route, it is clear that the remaining customers also cannot. In addition, if d_k + SumDemand[r_2] $>$ $Q_{u(r_2)}$ then there is no point in evaluating the transfer of $k \in r_1$ to any position in r_2, since the vehicle load of r_2 will always be violated. Thus, a verification should be performed to avoid the evaluation of these infeasible moves.

Swap(1,1) – $N^{(2)}$—Permutation between a customer k from a route r_1 and a customer l, from a route r_2. To avoid evaluating infeasible moves one should verify if MinDemand[r_1] − MaxDemand[r_2] + SumDemand[r_2] $\leq Q_{u(r_2)}$ and d_k + SumDemand[r_2] − MaxDemand[r_2] $\leq Q_{u(r_2)}$.

Shift(2,0) – $N^{(3)}$—Two adjacent customers, k and l, are transferred from a route r_1 to a route r_2. This move can also be seen as an arc transfer. In this case, the move examines the transfer of both arcs (k,l) and (l,k). Before starting to evaluate the customers transfer from r_1 to r_2 one should very if following conditions are met: MinPairDemand[r_1] + SumDemand[r_2] $\leq Q_{u(r_2)}$.

Swap(2,1) – $N^{(4)}$—Permutation of two adjacent customers, k and l, from a route r_1 by a customer k' from a route r_2. As in Shift(2,0), both arcs (k,l) and (l,k) are considered. The evaluation of some infeasible moves are avoided by checking if MinPairDemand[r_1] − MaxDemand[r_2] + SumDemand[r_2] $\leq Q_{u(r_2)}$.

Swap(2,2) – $N^{(5)}$—Permutation between two adjacent customers, k and l, from a route r_1 by another two adjacent customers k' and l', belonging to a route r_2. All the four possible combinations of exchanging arcs (k,l) and (k',l') are considered. To avoid evaluating some infeasible moves the following conditions must be satisfied: MinPairDemand$[r_1]$ − MaxDemand$[r_2]$ + SumDemand$[r_2] \leq Q_{u(r_2)}$.

Cross – $N^{(6)}$—The arc between adjacent clients k and l, belonging to a route r_1, and the one between k' and l', from a route r_2, are both removed. Next, an arc is inserted connecting k and l' and another is inserted linking k' and l. The vehicle loads of both routes are computed in constant time using the ADSs SumDemand and, CumulativeDemand.

***K*-Shift** – $N^{(7)}$—A subset of consecutive customers K is transferred from a route r_1 to the end of a route r_2. In this case, it is assumed that the dependent and fixed costs of r_2 is smaller than those of r_1. It should be pointed out that the move is also applied if r_2 is an empty route.

5.2.3 *Intra-route neighborhood structures*

Five well-known intra-route neighborhood structures were adopted. The set N' is composed by Or-*opt* (Or 1976), 2-*opt* and exchange moves. Since we evaluate all possible moves, the computational complexity of these local search procedures is $\mathcal{O}(\bar{n}^2)$. Their description is as follows.

Reinsertion—One, customer is removed and inserted in another position of the route.

Or-opt2—Two adjacent customers are removed and inserted in another position of the route.

Or-opt3—Three adjacent customers are removed and inserted in another position of the route.

2-opt—Two nonadjacent arcs are deleted and another two are added in such a way that a new route is generated.

Exchange—Permutation between two customers.

5.3 Perturbation mechanisms

A set P of three perturbation mechanisms were adopted. Whenever the `Perturb()` function is called, one of the moves described below is randomly selected. It is worth emphasizing that these perturbations are different from those employed by Subramanian et al. (2010).

Multiple-Swap(1,1) – $P^{(1)}$—Multiple random Swap(1,1) moves are performed in sequence. After some preliminary experiments, the number of successive moves was empirically set to be chosen from the interval $\{0.5v, 0.6v, \ldots, 1.4v, 1.5v\}$.

Multiple-Shift(1,1) – $P^{(2)}$—Multiple Shift(1,1) moves are performed in sequence randomly. The Shift(1,1) consists in transferring a customer k from a route r_1 to a route r_2, whereas a customer l from r_2 is transferred to r_1. In this case, the number of moves is randomly selected from the same interval of $P^{(1)}$. This perturbation is more stronger than the previous one since it admits a larger number of moves.

Split – $P^{(3)}$—A route r is divided into smaller routes. Let $M' = \{2, \dots, m\}$ be a subset of M composed by all vehicle types, except the one with the smallest capacity. Firstly, a route $r \in s$ (let $s = s'$) associated with a vehicle $u \in M'$ is selected at random. Next, while r is not empty, the remaining customers of r are sequentially transferred to a new randomly selected route $r' \notin s$ associated with a vehicle $u' \in \{1, \dots, u-1\}$ in such a way that the capacity of u' is not violated. The new generated routes are added to the solution s while the route r is removed from s. The procedure described is repeated multiple times where the number of repetitions is chosen at random from the interval $\{1, 2, \dots, v\}$. This perturbation was applied only for the FSM, since it does not make sense for the HVRP.

Every time a perturbation move is applied, we verify if the perturbed solution is feasible or not. If it is feasible then the perturbation phase is terminated. Otherwise, the same perturbation move is reapplied until a feasible move is performed.

6 Computational results

The algorithm ILS-RVND was coded in C++ (g++ 4.4.3) and executed in an Intel® Core™ i7 Processor 2.93 GHz with 8 GB of RAM memory running Ubuntu Linux 10.04 (kernel version 2.6.32-22). The developed heuristic we tested in well-known instances, namely those proposed by Golden et al. (1984) and Taillard (1999). The latter introduced dependent costs and established a limit for the number of vehicles of each type. Table 1 describes the characteristics of these instances. The values of *MaxIter* and *MaxIterILS* were calibrated as described in Sect. 6.1. The impact of the perturbation mechanisms is shown in Sect. 6.2. For each instance, the proposed algorithm was executed 30 times and the results are presented in Sect. 6.4. New improved solutions are reported in Appendix.

In Sect. 6.4 we also compare the results of our solution approach with the best known algorithms presented in the literature, namely those of Taillard (1999), Tarantilis et al. (2004), Choi and Tcha (2007), Li et al. (2007), Imran et al. (2009), Liu et al. (2009), Brandão (2009) and Prins (2009b). These algorithms were respectively executed in a Sun Sparc workstation 50 MHz, Pentium II 400 MHz, Pentium IV 2.6 GHz, Athlon 1.0 GHz, Pentium M 1.7 GHz, Pentium IV 3.0 GHz, Pentium M 1.4 GHz and Pentium IV M 1.8 GHz. Since they were executed in different computers, a comparison in terms of CPU time becomes quite difficult. Nevertheless, in an attempt of performing an approximate comparison, we made use of the list of computers available in Dongarra (2010), where the author reports the speed, in Millions of Floating-Point Operations per Second (Mflop/s), of various computers. As for the models that are not in the list, we adopted the speed of those with similar configuration. Therefore, we assume that the speed of the computers are: 27 Mflop/s—Sun Sparc workstation 50 MHz; 262 Mflop/s—Pentium II 400 MHz; 2266 Mflop/s—Pentium IV 2.6 GHz; 1168 Mflop/s—Athlon 1.0 GHz; 1477 Mflop/s—Pentium M 1.7 GHz; 3181 Mflop/s—Pentium IV 3.0 GHz; 1216 Mflop/s—Pentium M 1.4 GHz; 1564 Mflop/s—Pentium IV M 1.8 GHz. In the case of our Intel i7 2.93 GHz, we ran the program utilized by Dongarra (2010) and we obtained a speed of 5839 Mflop/s.

Table 1 HFVRP Instances

Inst. No.	n	Veh. type A				Veh. type B				Veh. type C				Veh. type D				Veh. type E				Veh. type F			
		Q_A	f_A	r_A	m_A	Q_B	f_B	r_B	m_B	Q_C	f_C	r_C	m_C	Q_D	f_D	r_D	m_D	Q_E	f_E	r_E	m_E	Q_F	f_F	r_F	m_F
3	20	20	20	1.0	20	30	35	1.1	20	40	50	1.2	20	70	120	1.7	20	120	225	2.5	20				
4	20	60	1000	1.0	20	80	1500	1.1	20	150	3000	1.4	20												
5	20	20	20	1.0	20	30	35	1.1	20	40	50	1.2	20	70	120	1.7	20	120	225	2.5	20				
6	20	60	1000	1.0	20	30	1500	1.1	20	150	3000	1.4	20												
13	50	20	20	1.0	4	30	35	1.1	2	40	50	1.2	4	70	120	1.7	4	120	225	2.5	2	200	400	3.2	1
14	50	120	1000	1.0	4	160	1500	1.1	2	300	3500	1.4	1												
15	50	50	100	1.0	4	100	250	1.6	3	160	450	2.0	2												
16	50	40	100	1.0	2	80	200	1.6	4	140	400	2.1	3												
17	75	50	25	1.0	4	120	80	1.2	4	200	150	1.5	2	350	320	1.8	1								
18	75	20	10	1.0	4	50	35	1.3	4	100	100	1.9	2	150	180	2.4	2	250	400	2.9	1	400	800	3.2	1
19	100	100	500	1.0	4	200	1200	1.4	3	300	2100	1.7	3												
20	100	60	100	1.0	6	140	300	1.7	4	200	500	2.0	3												

In the tables presented hereafter, **Inst. No.** denotes the number of the test-problem, ***n*** is the number of customers, **BKS** represents the best known solution reported in the literature, **Best Sol.** and **Time** indicate, respectively, the best solution and the original computational time in seconds associated to the corresponding work, **Sol.** is the average solution obtained by ILS-RVND, **Gap** denotes either the gap between the best solution found by a given algorithm and the BKS, either the mean of the gaps between the best solutions and the BKS, **Avg. Gap** corresponds to the gap between the average solutions and the BKSs, **Avg. Time** represents the average computational time in seconds. **#Best** is the number of BKS found or improved, **#FSM Inst.** and **#HVRP Inst.** correspond, respectively, to the total number of FSM and HVRP instances. **Scaled time** indicates the scaled time in seconds of each computer, with respect to our 2.93 GHz. The best solutions are highlighted in boldface and the solutions improved by ILS-RVND are underlined.

6.1 Parameter tuning

A set of instances with varying sizes was selected for tuning the main parameters of the ILS-RVND heuristic, that is, *MaxIter* and *MaxIterILS*. It has been empirically observed that the suitable values of *MaxIterILS* depends on the size of the instances, more precisely, on the number of customers and vehicles. For the sake of simplicity, we have decided to use an intuitive and straightforward linear expression for computing the value of *MaxIterILS* according to n and v, as shown in Eq. 3.

$$MaxIterILS = n + \beta \times v \qquad (3)$$

The parameter β in Eq. 3 corresponds to a non-negative integer constant that indicates the level of influence of the number of vehicles v in the value of *MaxIterILS*.

Table 2 Average gap and time in seconds between the solutions obtained by each β with $MaxIter = 350$

β	Inst. No.								Average	
	4		13		17		20			
	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time
1	1.68%	1.55	0.63%	13.26	0.60%	27.20	0.74%	50.84	0.91%	23.21
2	0.98%	1.87	0.48%	15.98	0.59%	29.88	0.72%	55.39	0.69%	25.78
3	0.49%	2.21	0.40%	18.55	0.57%	32.58	0.74%	60.17	0.55%	28.38
4	0.72%	2.46	0.37%	20.98	0.49%	35.18	0.63%	64.48	0.55%	30.78
5	0.00%	2.82	0.43%	23.65	0.49%	37.65	0.62%	69.13	0.38%	33.31
6	0.00%	3.09	0.33%	25.84	0.47%	40.18	0.62%	73.82	0.36%	35.73
7	0.24%	3.36	0.32%	28.20	0.45%	42.57	0.59%	77.79	0.40%	37.98
8	0.00%	3.65	0.29%	30.58	0.45%	45.58	0.61%	82.23	0.34%	40.51
9	0.00%	3.91	0.30%	32.85	0.43%	47.44	0.51%	87.05	0.31%	42.81
10	0.00%	4.23	0.28%	34.89	0.40%	50.11	0.58%	90.61	0.31%	44.96

Table 3 Average gap and time in seconds between the solutions obtained by each β with $MaxIter = 400$

β	Inst. No.								Average	
	4		13		17		20			
	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time
1	2.13%	1.68	0.51%	14.43	0.64%	29.61	0.78%	55.19	1.02%	25.23
2	0.50%	2.03	0.49%	17.49	0.60%	32.56	0.67%	60.32	0.56%	28.10
3	0.28%	2.40	0.42%	20.32	0.56%	35.44	0.65%	65.43	0.48%	30.90
4	0.48%	2.71	0.33%	22.87	0.54%	38.12	0.64%	70.29	0.50%	33.49
5	0.01%	3.02	0.30%	25.42	0.51%	41.00	0.58%	75.01	0.35%	36.11
6	0.00%	3.38	0.31%	28.18	0.46%	43.83	0.58%	79.81	0.34%	38.80
7	0.23%	3.67	0.28%	30.71	0.47%	46.38	0.53%	84.48	0.38%	41.31
8	0.23%	3.94	0.26%	33.33	0.48%	49.23	0.52%	89.44	0.37%	43.98
9	0.00%	4.30	0.29%	35.94	0.48%	51.73	0.49%	94.27	0.32%	46.56
10	0.00%	4.61	0.31%	38.26	0.48%	54.22	0.51%	98.00	0.33%	48.77

Four instances with varying number of customers (20–100) and vehicles were chosen as a sample for tuning the values of the parameters. For each of these test-problems we executed ILS-RVND 10 times in all of their respective HFVRP variants. Three values of *MaxIter* were tested, specifically 350, 400 and 450. For each of these, ten values of β were evaluated.

In order to select an attractive parameters configuration we took into account the quality of the solutions obtained in each variant, measured by the average gap between the solutions obtained using a given β value and the respective best solution found in the literature, and the computational effort, measured by the average CPU time of the complete execution of the algorithm. The gap was calculated using Eq. 4.

$$gap = \frac{\text{ILS-RVND\_solution} - \text{literature\_solution}}{\text{literature\_solution}} \times 100 \tag{4}$$

Table 2 contains the results of the average gap and the average CPU time for the tests involving $MaxIter = 350$, while those obtained for $MaxIter = 400$ and $MaxIter = 450$ are presented in Tables 3 and 4, respectively.

From Tables 2–4 we can observe that the quality of the solutions and the computational time tend to increase with the value of β and *MaxIter*. This behavior was obviously expected since more trials are given to the algorithm when the values of these two parameters increase. However, we can also verify that the variation of the average gap tends to be quite small from a given value of β. For example, in the case of $MaxIter = 350$ and $MaxIter = 400$, it is possible to notice that there are no significant modifications in the average gaps from $\beta = 5$, wheres the same happens from $\beta = 4$, in the case of $MaxIter = 450$. Although these three configurations had produced similar results, we decided to choose $MaxIter = 400$ and $\beta = 5$ because it was the one that obtained the smaller average gap when compared to the other two.

Table 4 Average gap and time in seconds between the solutions obtained by each β with *MaxIter* $= 450$

β	Inst. No.								Average	
	4		13		17		20			
	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time	Avg. Gap	Avg. Time
1	1.69%	1.95	0.50%	16.51	0.55%	33.90	0.66%	62.94	0.85%	28.82
2	0.71%	2.32	0.46%	19.79	0.60%	37.05	0.62%	68.53	0.60%	31.92
3	0.50%	2.73	0.42%	22.98	0.53%	40.58	0.61%	74.60	0.51%	35.22
4	0.01%	3.06	0.35%	26.28	0.49%	43.85	0.58%	80.38	0.36%	38.39
5	0.01%	3.47	0.36%	29.08	0.50%	47.02	0.58%	86.19	0.36%	41.44
6	0.01%	3.86	0.29%	32.13	0.45%	49.82	0.61%	91.57	0.34%	44.34
7	0.01%	4.20	0.30%	35.05	0.44%	53.18	0.53%	97.23	0.32%	47.42
8	0.00%	4.54	0.29%	37.98	0.45%	56.10	0.54%	102.13	0.32%	50.19
9	0.00%	4.91	0.27%	40.44	0.39%	59.05	0.53%	107.34	0.30%	52.93
10	0.00%	5.31	0.25%	43.56	0.37%	62.07	0.45%	112.42	0.27%	55.84

6.2 Impact of the perturbation mechanisms

In this subsection we are interested in evaluating the impact of the set of perturbation mechanisms in the HVRP and FSM. We ran the ILS-RVND algorithm with each perturbation separately and also with more than one perturbation included and the results for FSM and HVRP are presented in Tables 5 and 6, respectively. All instances were considered and we executed the ILS-RVND 30 times for each of the five HFVRP variants using $\beta = 5$ and *MaxIter* $= 400$. It is noteworthy to remember that perturbation *Split* ($P^{(3)}$) is only applied for the FSM.

According to Table 5 it can be verified that ILS-RVND clearly had a better performance, in terms of solution quality, when all perturbations were included, as can be seen by the values of the average gaps and the number of best solutions found. From Table 6 it can be seen that the three configurations had a similar performance in terms of average solutions, but the version that includes $P^{(1)} + P^{(2)}$ outperformed the other two both in terms of best solutions and computational time.

6.3 Deterministic Ordering versus Random Ordering of the Variable Neighborhood Descent

In order to illustrate the impact of the RVND in the performance of the proposed solution approach we ran two versions of our algorithm 30 times in all instances of each of the five HFVRP variants. The first version employs the random neighborhood ordering (RVND) in the local search phase and it makes use of the values of *MaxIter* and *MaxIterILS* specified in Sect. 6.1. The second one employs the traditional VND as a local search procedure with the following deterministic order $N^{(1)}, N^{(2)}, \ldots, N^{(7)}$. However, a different stopping criterion was adopted with a view of performing a fair comparison between both versions. The value of *MaxIterILS* is the same, but instead of *MaxIter*, we took the average time obtained for each instance when running the first version and set as an execution time limit for the second version.

Table 5 Impact of perturbation mechanism on FSM instances

Inst.	n	$P^{(1)}$			$P^{(2)}$			$P^{(3)}$			$P^{(1)}+P^{(3)}$			$P^{(2)}+P^{(3)}$			$P^{(1)}+P^{(2)}+P^{(3)}$		
No.		Gap	Avg. Gap	Avg. Time	Gap	Avg. Gap	Avg. Time	Gap	Avg. Gap	Avg. Time	Gap	Avg. Gap	Avg. Time	Gap	Avg. Gap	Avg. Time	Gap	Avg. Gap	Avg. Time
3	20	0.11%	0.52%	4.49	0.00%	0.15%	4.94	0.00%	0.02%	4.82	0.00%	0.00%	4.85	0.00%	0.00%	4.85	0.00%	0.00%	4.51
4	20	9.39%	9.39%	2.92	6.90%	9.22%	2.93	0.00%	0.62%	1.69	0.00%	0.17%	2.78	0.00%	0.16%	2.95	0.00%	0.00%	3.01
5	20	0.02%	0.33%	4.76	0.02%	0.31%	4.87	0.00%	0.02%	5.72	0.00%	0.00%	5.77	0.00%	0.00%	5.77	0.00%	0.00%	5.42
6	20	8.68%	8.68%	3.15	8.68%	8.68%	3.13	0.00%	0.00%	1.96	0.00%	0.00%	3.02	0.00%	0.00%	3.17	0.00%	0.00%	3.15
13	50	2.07%	2.75%	21.58	1.85%	2.67%	24.99	0.03%	0.42%	36.12	0.03%	0.42%	36.12	0.03%	0.42%	36.12	0.03%	0.34%	29.78
14	50	6.95%	7.00%	16.77	6.94%	6.98%	16.42	0.02%	0.26%	4.56	0.00%	0.00%	11.42	0.00%	0.00%	11.81	0.00%	0.00%	12.74
15	50	3.37%	4.43%	17.78	3.33%	4.33%	17.35	0.19%	1.87%	6.02	0.00%	0.24%	14.47	0.00%	0.19%	15.09	0.00%	0.07%	16.47
16	50	1.21%	1.58%	18.06	0.55%	1.47%	17.64	0.50%	1.77%	11.09	0.21%	0.50%	19.02	0.26%	0.43%	19.98	0.00%	0.33%	18.88
17	75	2.06%	2.56%	48.54	1.95%	2.49%	47.80	1.52%	3.13%	36.50	0.30%	0.71%	56.79	0.15%	0.62%	58.10	0.00%	0.47%	48.45
18	75	11.17%	11.87%	53.51	11.15%	11.74%	53.29	0.93%	3.23%	43.57	0.22%	0.49%	64.03	0.18%	0.43%	65.16	0.04%	0.33%	52.34
19	100	6.22%	7.92%	100.58	5.59%	7.82%	95.99	0.20%	1.93%	11.63	0.01%	0.30%	52.93	0.03%	0.29%	52.33	0.00%	0.11%	66.96
20	100	4.17%	5.28%	93.00	3.58%	5.23%	89.31	1.20%	3.84%	24.40	0.15%	0.68%	75.22	0.31%	0.66%	72.31	0.00%	0.34%	80.32
Average		4.62%	5.19%	32.09	4.21%	5.09%	31.55	0.38%	1.43%	15.67	0.08%	0.29%	28.87	0.08%	0.27%	28.97	0.01%	0.17%	28.50
#Best/#FSM Inst.			13/36			14/36			18/36			28/36			26/36			32/36	

Table 6 Impact of perturbation mechanism on HVRP instances

Inst. No.	n	$P^{(1)}$ Gap	Avg. Gap	Avg. Time	$P^{(2)}$ Gap	Avg. Gap	Avg. Time	$P^{(1)}+P^{(2)}$ Gap	Avg. Gap	Avg. Time
13	50	0.00%	0.17%	19.55	0.00%	0.06%	21.50	0.00%	0.09%	19.16
14	50	0.00%	0.01%	12.81	0.00%	0.01%	13.09	0.00%	0.01%	11.24
15	50	0.00%	0.00%	14.63	0.00%	0.00%	14.34	0.00%	0.00%	12.52
16	50	0.00%	0.55%	14.04	0.00%	0.43%	13.72	0.00%	0.20%	12.26
17	75	0.05%	0.31%	34.52	0.07%	0.26%	34.19	0.00%	0.31%	29.75
18	75	0.00%	0.55%	40.26	0.00%	0.34%	42.72	0.00%	0.45%	37.36
19	100	0.11%	0.12%	89.96	0.11%	0.11%	83.86	0.11%	0.12%	70.69
20	100	0.38%	0.72%	80.75	0.32%	0.67%	75.71	−0.19%	0.66%	66.11
Average		0.07%	0.30%	38.32	0.06%	0.24%	37.39	−0.01%	0.23%	32.39
#Best/#HVRP Inst.			12/16			12/16			15/16	

Table 7 Deterministic Ordering versus Random Ordering of the Variable Neighborhood Descent

Variant	ILS-VND Gap	Avg. Gap	#Best/ #Inst.	ILS-RVND Gap	Avg. Gap	#Best/ #Inst.
HVRPFD	0.29%	0.67%	2/8	−0.05%	0.24%	8/8
HVRPD	0.59%	1.23%	2/8	0.03%	0.22%	7/8
FSMFD	0.07%	0.19%	6/12	0.00%	0.09%	11/12
FSMF	0.29%	0.43%	6/12	0.01%	0.23%	9/12
FSMD	0.43%	0.92%	6/12	0.00%	0.17%	11/12
Average	0.33%	0.69%		0.00%	0.19%	

Table 7 presents the results obtained using the deterministic neighborhood ordering (ILS-VND) and those found using the random neighborhood ordering (ILS-RVND). It can be observed that the ILS-RVND clearly outperformed the ILS-VND in all variants. A possible explanation to this fact is that ILS-VND converges prematurely to poor local optima and therefore it tends to generate, on average, low quality solutions when compared to ILS-RVND, which in turn is less-likely to produce solutions that get easily trapped in local optima.

6.4 Comparison with the literature

In this subsection we compare the results obtained by ILS-RVND with those found in the literature. It is important to mention that some authors, like Tarantilis et al. (2004), Li et al. (2007) and Brandão (2009), reported the results of single-runs. Moreover, others, like (Prins 2009a), reported the average gap between the average solutions and the previous BKS, but not the average solutions themselves.

Table 8 Results for HVRPFD instances

Inst. No.	n	BKS	ILS-RVND Best Sol.	Time	Gap	Sol.[b]	Time[b]	Gap[b]
13	50	3185.09[a]	**3185.09**	18.87	0.00%	3189.17	19.04	0.13%
14	50	10107.53[a]	**10107.53**	10.58	0.00%	10107.94	11.28	0.00%
15	50	3065.29[a]	**3065.29**	11.78	0.00%	3065.34	12.48	0.00%
16	50	3265.41[a]	**3265.41**	11.87	0.00%	3278.06	12.22	0.39%
17	75	2076.96[a]	**2076.96**	29.44	0.00%	2083.19	29.59	0.30%
18	75	3743.58[a]	**3743.58**	35.75	0.00%	3758.84	36.38	0.41%
19	100	10423.32	**<u>10420.34</u>**	70.55	−0.03%	10421.39	73.66	−0.02%
20	100	4806.69	**<u>4788.49</u>**	66.88	−0.38%	4839.53	68.46	0.68%
Avg. time/Avg. gap				31.97	−0.05%		32.89	0.24%

[a]Optimality proved

[b]Average of 30 runs

6.4.1 HVRPFD

To our knowledge the HVRPFD was only examined by Baldacci and Mingozzi (2009). From Table 8 it can be verified that the ILS-RVND found all proven optimal solutions and improved the results of the two instances in which the optimal solution was not proved by Baldacci and Mingozzi (2009).

6.4.2 HVRPD

Tables 9 and 10 present a comparison between the results found by ILS-RVND and the best heuristics proposed in the literature, namely those of Taillard (1999), Tarantilis et al. (2004) and Prins (2009a). Although ILS-RVND was capable of finding 7 of the 8 BKSs in a competitive computational time, the average gap show that our algorithm is not as effective for the HFVRPD as those developed by Li et al. (2007) and Prins (2009b) that managed to obtain better gaps in a single-run.

6.4.3 FSMFD

In Tables 11 and 12 a comparison is performed between the results found by ILS-RVND and the best heuristics available in the literature, particularly the ones of Choi and Tcha (2007), Prins (2009a) and Imran et al. (2009). The ILS-RVND failed to equal the results of two instances but it was capable to improve the results of another one. When individually comparing the ILS-RVND with each one of these algorithms, one can verify that the ILS-RVND produced, on average, highly competitive solutions. However, in terms of computational time, our algorithm is still slower than the SMA-U1 of Prins (2009a).

Table 9 Results for HVRPD instances

Inst. No.	n	BKS	HCG Taillard		BATA Tarantilis et al.		HRTR Li et al.		SMA-D2 Prins		ILS-RVND					
			Best Sol.	Time[b]	Best Sol.	Time	Best Sol.	Time	Best Sol.	Time	Best Sol.	Time	Gap	Sol.[c]	Time[c]	Gap[c]
13	50	1517.84[a]	1518.05	473	1519.96	843	**1517.84**	358	**1517.84**	33.2	**1517.84**	18.46	0.00%	1518.58	19.29	0.05%
14	50	607.53[a]	615.64	575	611.39	387	**607.53**	141	**607.53**	37.6	**607.53**	10.74	0.00%	607.64	11.20	0.02%
15	50	1015.29[a]	1016.86	335	**1015.29**	368	**1015.29**	166	**1015.29**	6.6	**1015.29**	11.99	0.00%	1015.33	12.56	0.00%
16	50	1144.94[a]	1154.05	350	1145.52	341	**1144.94**	188	**1144.94**	7.5	**1144.94**	11.57	0.00%	1145.04	12.29	0.01%
17	75	1061.96[a]	1071.79	2245	1071.01	363	**1061.96**	216	1065.85	81.5	**1061.96**	29.80	0.00%	1065.27	29.92	0.31%
18	75	1823.58[a]	1870.16	2876	1846.35	971	**1823.58**	366	**1823.58**	190.6	**1823.58**	36.09	0.00%	1832.52	38.34	0.49%
19	100	1117.51	**1117.51**	5833	1123.83	428	1120.34	404	1120.34	177.8	1120.34	63.14	0.25%	1120.34	67.72	0.25%
20	100	1534.17[a]	1559.77	3402	1556.35	1156	**1534.17**	447	**1534.17**	223.3	**1534.17**	61.94	0.00%	1544.08	63.77	0.65%

[a]Optimality proved

[b]Average time of 5 runs

[c]Average of 30 runs

Taillard (1999) CPU: Sun Sparc 10 workstation with 50 MHz; Tarantilis et al. (2004) CPU: Pentium II 400 MHz; Li et al. (2007) CPU: AMD Athlon 1 GHz; Prins (2009a) CPU: Pentium IV M 1.8 GHz

Table 10 Summary of results for HVRPD

Algorithm	Best Run				Average[1]	
	Gap	Scaled Time	BKS Found	BKS Improved	Gap	Scaled Time
HCG (Taillard 1999)	0.93%	–	1	0	2.50%	9.30
BATA (Tarantilis et al. 2004)	0.62%	27.24	1	0	–	–
HRTR (Li et al. 2007)	0.03%	57.16	7	0	–	–
SMA-D2 (Prins 2009a)	0.08%	25.38	6	0	–	–
ILS-RVND	0.03%	30.47	7	0	0.22%	31.89

[1] Average of 5 runs for Taillard (1999) and of 30 runs for ILS-RVND

6.4.4 *FSMF*

Tables 13 and 14 illustrate the results obtained by the ILS-RVND for the FSMF. These results are compared with those of Choi and Tcha (2007), Brandão (2009), Prins (2009b), Imran et al. (2009), and Liu et al. (2009). It can be seen that the proposed algorithm found one new best solution, equaled the results of 8 instances, but it failed to obtain the best known solutions in another 3. The average results are quite competitive in terms of solution quality when compared to the other approaches. In addition, except for the algorithm of Prins (2009a), ILS-RVND outperformed all others in terms of computational time.

6.4.5 *FSMD*

The best results obtained in the literature for the FSMD using heuristic approaches were reported by Choi and Tcha (2007), Brandão (2009), Prins (2009a), Imran et al. (2009) and Liu et al. (2009). These results along with those found by ILS-RVND are presented in Tables 15 and 16. In this variant the optimal solutions were determined by Baldacci and Mingozzi (2009) for all instances. From Table 15 it can be observed that, except for one instance, ILS-RVND was capable of finding all optimal solutions. Even though ILS-RVND performed, on average, slower than the algorithms of Choi and Tcha (2007) and Prins (2009b), the results obtained are quite satisfactory, especially in terms of solution quality.

7 Concluding remarks

This article dealt with Heterogeneous Fleet Vehicle Routing Problem (HVRFP). This kind of problem often arises in practical applications and one can affirm that this model is more realistic than the classical homogeneous Vehicle Routing Problem. Five different HFVRP variants involving limited and unlimited fleets were considered. These variants were solved by an Iterated Local Search algorithm that uses Variable Neighborhood Descent with random neighborhood ordering (RVND) in the local search phase.

Table 11 Results for FSMFD instances

Inst. No.	n	BKS	CG Choi and Tcha		SMA-U1 Prins		VNS1 Imran et al.		ILS-RVND					
			Best Sol.	Time	Best Sol.	Time	Best Sol.	Time[b]	Best Sol.	Time	Gap	Sol.[c]	Time[c]	Gap[c]
3	20	1144.22[a]	**1144.22**	0.25	**1144.22**	0.01	**1144.22**	19	**1144.22**	3.87	0.00%	1144.22	4.05	0.00%
4	20	6437.33[a]	**6437.33**	0.45	**6437.33**	0.07	**6437.33**	17	**6437.33**	2.77	0.00%	6437.66	3.03	0.01%
5	20	1322.26[a]	**1322.26**	0.19	**1322.26**	0.02	**1322.26**	24	**1322.26**	4.57	0.00%	1322.26	4.85	0.00%
6	20	6516.47[a]	**6516.47**	0.41	**6516.47**	0.07	**6516.47**	21	**6516.47**	2.8	0.00%	6516.47	3.01	0.00%
13	50	2964.65[a]	**2964.65**	3.95	**2964.65**	0.32	**2964.65**	328	**2964.65**	27.67	0.00%	2971.32	27.44	0.22%
14	50	9126.90[a]	**9126.90**	51.70	**9126.90**	8.90	**9126.90**	250	**9126.90**	11.27	0.00%	9126.91	11.66	0.00%
15	50	2634.96[a]	**2634.96**	4.36	2635.21	1.04	**2634.96**	275	**2634.96**	13.47	0.00%	2635.02	13.83	0.00%
16	50	3168.92[a]	**3168.92**	5.98	3169.14	13.05	3168.95	313	**3168.92**	17.55	0.00%	3170.81	18.20	0.06%
17	75	2004.48[a]	2023.61	68.11	**2004.48**	23.92	**2004.48**	641	**2004.48**	43.33	0.00%	2012.23	43.68	0.39%
18	75	3147.99[a]	**3147.99**	18.78	3153.16	24.85	3153.67	835	3149.63	47.39	0.05%	3158.24	47.80	0.33%
19	100	8661.81[a]	8664.29	905.20	8664.67	163.25	8666.57	1411	**8661.81**	60.33	0.00%	8664.81	59.13	0.03%
20	100	4153.11[d]	4154.49	53.02	4154.49	41.25	4164.85	1460	**4153.02**	58.97	0.00%	4155.90	59.07	0.07%

[a]Optimality proved

[b]Total time of 10 runs

[c]Average of 30 runs

[d]First found by Prins (2009b)

Choi and Tcha (2007) CPU: Pentium IV 2.6 GHz; Prins (2009b) CPU: Pentium IV M 1.8 GHz; Imran et al. (2009) CPU: Pentium M 1.7 GHz

Table 12 Summary of results for FSMFD

Algorithm	Best Run				Average[1]	
	Gap	Scaled Time	BKS Found	BKS Improved	Gap	Scaled Time
CG (Choi and Tcha 2007)	0.08%	35.98	9	0	0.11%	42.82
SMA-U1 (Prins 2009b)	0.02%	6.18	7	0	–	6.86
VNS1 (Imran et al. 2009)	0.04%	117.92	8	0	–	–
ILS-RVND	0.00%	24.50	10	1	0.09%	24.64

[1] Average of 5 runs for both Choi and Tcha (2007) and Prins (2009b) and of 30 runs for ILS-RVND

The proposed algorithm (ILS-RVND) was tested on 52 well-known benchmark instances with up to 100 customers. The ILS-RVND was found capable to improve the results of 4 instances and to equal the results of another 42. Furthermore, we believe that the developed solution approach has a quite simple structure and it also relies on very few parameters. This not only facilitates the reproduction of the procedure, but it also reduces the tunning efforts. In addition, we can verify that the algorithm has proven to be flexible, as can be observed by the robust and competitive results obtained in each of the five HFVRP variants considered in this work. According to Cordeau et al. (2002), when it comes to VRP heuristics, simplicity and flexibility are just important as solution quality and computational time.

As for future work, we intend to improve our algorithm to solve large-sized instances, as well as other HFVRP variants that include additional features such as time windows, pickup and delivery services, multiple depots and so forth.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments that considerably improved the quality of this paper. This research was partially supported by CNPq, CAPES and FAPERJ.

Appendix: New best solutions

A.1 HVRPFD

Instance 19: 8 routes, cost 10420.34
(vehicle type): list of customers.

```
(A): 0 87 42 14 38 43 15 57 2 0
(A): 0 60 83 8 46 45 17 84 5 99 96 6 0
(A): 0 12 80 68 29 24 25 55 54 0
(B): 0 52 7 19 11 64 49 36 47 48 82 18 89 0
(B): 0 27 69 1 70 30 20 66 32 90 63 10 62 88 31 0
(B): 0 76 77 3 79 78 34 35 65 71 9 51 81 33 50 0
(C): 0 94 95 59 93 85 61 16 86 44 91 100 98 37 92 97 13 0
(C): 0 53 58 40 21 73 72 74 22 41 75 56 23 67 39 4 26 28 0
```

Instance 20: 12 routes, cost 4788.49
(vehicle type): list of customers.

Table 13 Results for FSMF instances

Inst. No.	n	BKS	CG Choi and Tcha		TSA1 Brandão		SMA-D1 Prins		VNS1 Imran et al.		GA Liu et al.		ILS-RVND					
			Best Sol.	Time	Best Sol.	Time	Best Sol.	Time	Best Sol.	Time[b]	Best Sol.	Time[c]	Best Sol.	Time	Gap	Sol.[d]	Time[d]	Gap[d]
3	20	961.03[a]	**961.03**	0	**961.03**	21	**961.03**	0.04	**961.03**	21	**961.03**	0	**961.03**	4.60	0.00%	961.10	4.91	0.01%
4	20	6437.33[a]	**6437.33**	1	**6437.33**	22	**6437.33**	0.03	**6437.33**	18	**6437.33**	0	**6437.33**	3.00	0.00%	6437.63	3.16	0.00%
5	20	1007.05[a]	**1007.05**	1	**1007.05**	20	**1007.05**	0.09	**1007.05**	13	**1007.05**	2	**1007.05**	5.53	0.00%	1007.05	5.88	0.00%
6	20	6516.47[a]	**6516.47**	0	**6516.47**	25	**6516.47**	0.08	**6516.47**	22	**6516.47**	0	**6516.47**	2.91	0.00%	6516.47	3.07	0.00%
13	50	2406.36[a]	**2406.36**	10	**2406.36**	145	**2406.36**	17.12	**2406.36**	252	**2406.36**	91	2408.41	30.37	0.09%	2419.38	30.29	0.54%
14	50	9119.03[a]	**9119.03**	51	**9119.03**	220	**9119.03**	19.66	**9119.03**	274	**9119.03**	42	**9119.03**	11.45	0.00%	9119.03	11.89	0.00%
15	50	2586.37[a]	**2586.37**	10	2586.84	110	**2586.37**	25.1	**2586.37**	303	**2586.37**	48	**2586.37**	19.29	0.00%	2586.80	20.24	0.02%
16	50	2720.43[a]	**2720.43**	11	2728.14	111	2729.08	16.37	**2720.43**	253	2724.22	107	**2720.43**	19.98	0.00%	2737.59	20.67	0.63%
17	75	1734.53[a]	1744.83	207	1736.09	322	1746.09	52.22	1741.95	745	**1734.53**	109	**1734.53**	53.70	0.00%	1748.06	52.49	0.78%
18	75	2369.65[a]	2371.49	70	2376.89	267	**2369.65**	36.92	**2369.65**	897	**2369.65**	197	2371.48	54.22	0.08%	2380.98	55.35	0.48%
19	100	8661.81[a]	8664.29	1179	8667.26	438	8665.12	169.93	8665.05	1613	8662.94	778	8662.86	64.90	0.01%	8665.31	63.92	0.04%
20	100	4038.46	4039.49	264	4048.09	601	4044.78	172.73	4044.68	1595	**4038.46**	1004	<u>**4037.90**</u>	94.22	−0.01%	4051.11	93.88	0.31%

[a]Optimality proved

[b]Total time of 10 runs

[c]Average time of 10 runs

[d]Average of 30 runs

Choi and Tcha (2007) CPU: Pentium IV 2.6 GHz; Brandão (2009) CPU: Pentium M 1.4 GHz; Prins (2009b) CPU: Pentium IV M 1.8 GHz; Imran et al. (2009) CPU: Pentium M 1.7 GHz; Liu et al. (2009) CPU: Pentium IV 3.0 GHz

Table 14 Summary of results for FSMF

Algorithm	Best Run				Average[1]	
	Gap	Scaled Time	BKS Found	BKS Improved	Gap	Scaled Time
CG (Choi and Tcha 2007)	0.06%	58.34	8	0	0.17%	58.36
TSA1 (Brandão 2009)	0.08%	39.95	6	0	–	–
SMA-D1 (Prins 2009b)	0.10%	11.39	8	0	–	10.92
VNS1 (Imran et al. 2009)	0.05%	126.60	9	0	–	–
GA (Liu et al. 2009)	0.01%	–	10	0	0.19%	107.96
ILS-RVND	0.01%	30.35	8	1	0.23%	30.48

[1] Average of 5 runs for both Choi and Tcha (2007) and Prins (2009b), of 10 runs for Liu et al. (2009) and of 30 runs for ILS-RVND

```
(A): 0 73 74 56 22 41 57 2 0
(A): 0 42 15 43 38 14 91 6 0
(A): 0 55 25 24 29 34 78 79 0
(A): 0 28 76 77 26 0
(A): 0 60 84 17 45 8 83 18 0
(B): 0 27 50 33 81 3 68 80 12 0
(B): 0 46 47 36 49 64 63 90 32 70 31 0
(B): 0 52 7 82 48 19 11 62 10 88 0
(B): 0 1 51 9 35 71 65 66 20 30 69 0
(C): 0 54 4 39 67 23 75 72 21 40 53 0
(C): 0 58 13 87 97 98 85 93 59 99 96 0
(C): 0 89 5 61 16 86 44 100 37 92 95 94 0
```

A.2 FSMFD

Instance 20: 25 routes, cost 4153.02
(vehicle type): list of customers.

```
(A): 0 16 86 17 60 0
(A): 0 70 90 63 11 62 88 0
(A): 0 4 39 25 55 0
(A): 0 13 58 53 0
(A): 0 52 48 18 0
(A): 0 93 61 100 37 0
(A): 0 83 45 84 5 0
(A): 0 51 9 66 20 0
(A): 0 69 1 50 76 28 0
(A): 0 68 80 54 0
(A): 0 31 27 0
(A): 0 89 6 96 59 0
(A): 0 77 3 29 24 12 0
(A): 0 91 44 38 14 92 0
(A): 0 21 74 75 22 41 0
```

Table 15 Results for FSMD instances

Inst. No.	n	BKS	CG Choi and Tcha		TSA1 Brandão		SMA-D1 Prins		VNS1 Imran et al.		GA Liu et al.		ILS-RVND					
			Best Sol.	Time	Best Sol.	Time	Best Sol.	Time	Best Sol.	Time[b]	Best Sol.	Time	Best Sol.	Time	Gap	Sol.[c]	Time[c]	Gap
3	20	623.22[a]	**623.22**	0.19	–	–	–	–	–	–	–	–	**623.22**	4.31	0.00%	623.22	4.58	0.00%
4	20	387.18[a]	**387.18**	0.44	–	–	–	–	–	–	–	–	**387.18**	2.59	0.00%	387.18	2.85	0.00%
5	20	742.87[a]	**742.87**	0.23	–	–	–	–	–	–	–	–	**742.87**	5.23	0.00%	742.87	5.53	0.00%
6	20	415.03[a]	**415.03**	0.92	–	–	–	–	–	–	–	–	**415.03**	3.18	0.00%	415.03	3.37	0.00%
13	50	1491.86[a]	**1491.86**	4.11	**1491.86**	101	**1491.86**	3.45	**1491.86**	310	**1491.86**	117	**1491.86**	30.68	0.00%	1495.61	31.62	0.25%
14	50	603.21[a]	**603.21**	20.41	**603.21**	135	**603.21**	0.86	**603.21**	161	**603.21**	26	**603.21**	13.92	0.00%	603.21	14.66	0.00%
15	50	999.82[a]	**999.82**	4.61	**999.82**	137	**999.82**	9.14	**999.82**	218	**999.82**	37	**999.82**	14.70	0.00%	1001.70	15.33	0.19%
16	50	1131.00[a]	**1131.00**	3.36	**1131.00**	95	**1131.00**	13.00	**1131.00**	239	**1131.00**	54	**1131.00**	17.25	0.00%	1134.52	17.77	0.31%
17	75	1038.60[a]	**1038.60**	69.38	**1038.60**	312	**1038.60**	9.53	**1038.60**	509	**1038.60**	153	**1038.60**	48.15	0.00%	1041.12	49.18	0.24%
18	75	1800.80[a]	1801.40	48.06	1801.40	269	**1800.80**	18.92	**1800.80**	606	1801.40	394	**1800.80**	52.66	0.00%	1804.07	53.88	0.18%
19	100	1105.44[a]	**1105.44**	182.86	**1105.44**	839	**1105.44**	52.31	**1105.44**	1058	**1105.44**	479	**1105.44**	77.89	0.00%	1108.21	77.84	0.25%
20	100	1530.43[a]	**1530.43**	98.14	1531.83	469	1535.12	104.41	1533.24	1147	1534.37	826	1530.52	86.66	0.01%	1540.32	88.02	0.65%

[a] Optimality proved

[b] Total time of 10 runs

[c] Average of 30 runs

Choi and Tcha (2007) CPU: Pentium IV 2.6 GHz; Brandão (2009) CPU: Pentium M 1.4 GHz; Prins (2009b) CPU: Pentium IV M 1.8 GHz; Imran et al. (2009) CPU: Pentium M 1.7 GHz; Liu et al. (2009) CPU: Pentium IV 3.0 GHz

Table 16 Summary of results for FSMD

Algorithm	Best Run				Average[1]	
	Gap	Scaled Time	BKS Found	BKS Improved	Gap	Scaled Time
CG (Choi and Tcha 2007)	0.00%	13.99	11	0	0.12%	21.05
TSA1 (Brandão 2009)[2]	0.02%	61.36	6	0	–	–
SMA-D1 (Prins 2009b)[2]	0.04%	7.09	7	0	–	8.46
VNS1 (Imran et al. 2009)[2]	0.02%	134.32	7	0	–	–
GA (Liu et al. 2009)[2]	0.04%	–	6	0	0.56%	142.05
ILS-RVND	0.00% (0.00%)[2]	29.77 (42.74)[2]	11 (7)[2]	0	0.17% (0.26%)[2]	30.38 (43.54)[2]

[1] Average of 5 runs for both Choi and Tcha (2007) and Prins (2009b), of 10 runs Liu et al. (2009) and of 30 runs for ILS-RVND

[2] Results for instances 13–20

```
(A): 0 8 46 36 49 64 7 0
(A): 0 40 73 72 26 0
(A): 0 19 47 82 0
(A): 0 2 57 15 43 42 87 0
(A): 0 98 85 99 0
(A): 0 30 32 10 0
(A): 0 56 23 67 0
(A): 0 97 95 94 0
(A): 0 33 81 79 0
(A): 0 71 65 35 34 78 0
```

A.3 FSMF

Instance 20: 18 routes, cost 4037.90
(vehicle type): list of customers.

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

References

Baldacci, R., Mingozzi, A.: A unified exact method for solving different classes of vehicle routing problems. Math. Program. **120**, 347–380 (2009)

Baldacci, R., Battarra, M., Vigo, D.: Routing a heterogeneous fleet of vehicles. In: The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 11–35. Springer, Berlin (2008)

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., Schiavinotto, T.: Hybrid metaheuristics for the vehicle routing problem with stochastic demands. J. Math. Model. Algorithms **5**(1), 91–110 (2006)

Brandão, J.: A deterministic tabu search algorithm for the fleet size and mix vehicle routing problem. Eur. J. Oper. Res. **195**, 716–728 (2009)

Chen, P., Huang, H.K., Dong, X.Y.: Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. Expert Syst. Appl. **37**(2), 1620–1627 (2010)

Cheung, R., Hang, D.: Multi-attribute label matching algorithms for vehicle routing problems with time windows and backhauls. IIE Trans. **35**, 191–205 (2003)

Choi, E., Tcha, D.W.: A column generation approach to the heterogeneous fleet vehicle routing problem. Comput. Oper. Res. **34**, 2080–2095 (2007)

Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. **12**, 568–581 (1964)

Cordeau, J.F., Gendreau, M., Laporte, G., Potvin, J.Y., Semet, F.: A guide to vehicle routing problem. J. Oper. Res. Soc. **53**, 512–522 (2002)

Dongarra, J.J.: Performance of various computers using standard linear equations software. Tech. Rep. CS-89-85. Computer Science Department, University of Tennessee (2010)

Gendreau, M., Laporte, G., Musaraganyi, C., Taillard, E.D.: A tabu search heuristic for the heterogeneous fleet vehicle routing problem. Comput. Oper. Res. **26**, 1153–1173 (1999)

Glover, F.: Future paths in integer programming and links to Artificial Intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986)

Glover, F., Laguna, M., Marti, R.: Scatter search and path relinking: advances and appl. In: Handbook of Metaheuristics, pp. 1–36. Kluwer Academic, Dordrecht (2003)

Golden, B.L., Assad, A.A., Levy, L., Gheysens, F.G.: The feet size and mix vehicle routing problem. Comput. Oper. Res. **11**, 49–66 (1984)

Hoff, A., Andersson, H., Christiansen, M., Hasle, G., Løkketangen, A.: Industrial aspects and literature survey: Fleet composition and routing. Comput. Oper. Res. **37**, 1515–1536 (2010)

Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)

Ibaraki, T., Imahori, S., Nonobe, K., Sobue, K., Uno, T., Yagiura, M.: An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. Discrete Appl. Math. **156**(11), 2050–2069 (2008)

Imran, A., Salhi, S., Wassan, N.A.: A variable neighborhood-based heuristic for the heterogeneous fleet vehicle routing problem. Eur. J. Oper. Res. **197**, 509–518 (2009)

Lee, Y., Kim, J., Kang, K., Kim, K.: A heuristic for vehicle fleet mix problem using tabu search and set partitioning. J. Oper. Res. Soc.. **59**, 833–841 (2008)

Li, F., Golden, B., Wasil, E.: A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. Comput. Oper. Res. **34**, 2734–2742 (2007)

Lima, C.M.R.R., Goldbarg, M.C., Goldbarg, E.F.G.: A memetic algorithm for the heterogeneous fleet vehicle routing problem. Electron. Notes Discrete Math. **18**, 171–176 (2004)

Liu, S., Huang, W., Ma, H.: An effective genetic algorithm for the fleet size and mix vehicle routing problems. Transp. Res., Part B, Methodol. **45**, 434–445 (2009)

Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Handbook of Metaheuristics, pp. 321–353. Kluwer Academic, Dordrecht (2003)

Mladenovic, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)

Moscato, P., Cotta, C.: A gentle introduction to memetic algorithm. In: Handbook of Metaheuristics, pp. 105–144. Kluwer Academic, Dordrecht (2003)

Ochi, L., Vianna, D., Drummond, L.M.A., Victor, A.: An evolutionary hybrid metaheuristic for solving the vehicle routing problem with heterogeneous fleet. Lect. Notes Comput. Sci. **1391**, 187–195 (1998a)

Ochi, L., Vianna, D., Drummond, L.M.A., Victor, A.: A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. Future Gener. Comput. Syst. **14**, 285–292 (1998b)

Or, I.: Traveling salesman-type combinational problems and their relation to the logistics of blood banking. PhD thesis, Northwestern University, USA (1976)

Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Ann. Oper. Res. **41**(1–4), 421–451 (1993)

Pessoa, A., Uchoa, E., de Aragão, M.P.: Robust branch-and-cut-and-price algorithms for vehicle routing problems. In: The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 297–325. Springer, Berlin (2008)

Pessoa, A., Uchoa, E., de Aragão, M.P.: A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. Network **54**(4), 167–177 (2009)

Prins, C.: Efficient heuristics for the heterogeneous fleet multitrip vrp with application to a large-scale real case. J. Math. Model. Algorithms **1**, 135–150 (2002)

Prins, C.: A GRASP $\times$ evolutionary local search hybrid for the Vehicle Routing Problem. In: Bio-inspired Algorithms for the Vehicle Routing Problem, Studies in Computational Intelligence, vol. 161, pp. 35–53. Springer, Berlin (2009a)

Prins, C.: Two memetic algorithms for heterogeneous fleet vehicle routing problems. Eng. Appl. Artif. Intell. **22**(6), 916–928 (2009b)

Renaud, J., Boctor, F.: A sweep-based algorithm for the fleet size and mix vehicle routing problem. Eur. J. Oper. Res. **140**, 618–628 (2002)

Subramanian, A., Drummond, L., Bentes, C., Ochi, L., Farias, R.: A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Comput. Oper. Res. **37**(11), 1899–1911 (2010)

Taillard, E.D.: A heuristic column generation method for heterogeneous fleet. RAIRO. Rech. Opér. **33**, 1–14 (1999)

Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Jy, P.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transp. Sci. **31**, 170–186 (1997)

Tarantilis, C.D., Kiranoudis, C.T.: A meta-heuristic algorithm for the efficient distribution of perishable foods. J. Food Eng. **50**, 1–9 (2001)

Tarantilis, C.D., Kiranoudis, C.T.: A flexible adaptive memory-based algorithm for real-life transportation operations: Two case studies from dairy and construction sector. Eur. J. Oper. Res. **179**, 806–822 (2007)

Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S.: A list based threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. J. Oper. Res. Soc. **54**, 65–71 (2003)

Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S.: A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. Eur. J. Oper. Res. **152**, 148–158 (2004)

Yaman, H.: Formulations and valid inequalities for the heterogeneous vehicle routing problem. Math. Program. **106**, 365–390 (2006)