European Journal of Operational Research 237 (2014) 454–464

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

An iterated local search algorithm for the vehicle routing problem with backhauls

Daniel Palhazi Cuervo [a,b,*], Peter Goos [a,c,d], Kenneth Sörensen [a], Emely Arráiz [b]

[a] University of Antwerp, Faculty of Applied Economics, Belgium
[b] Universidad Simón Bolívar, Departamento de Computación, Venezuela
[c] Erasmus University of Rotterdam, Erasmus School of Economics, The Netherlands
[d] University of Leuven, Faculty of Bioscience Engineering, Belgium

ARTICLE INFO

Article history:
Received 7 June 2013
Accepted 4 February 2014
Available online 14 February 2014

Keywords:
Vehicle routing problem
Metaheuristic
Iterated local search
Oscillating local search

ABSTRACT

The Vehicle Routing Problem with Backhauls (VRPB) is an extension of the VRP that deals with two types of customers: the consumers (linehaul) that request goods from the depot and the suppliers (backhaul) that send goods to the depot. In this paper, we propose a simple yet effective iterated local search algorithm for the VRPB. Its main component is an oscillating local search heuristic that has two main features. First, it explores a broad neighborhood structure at each iteration. This is efficiently done using a data structure that stores information about the set of neighboring solutions. Second, the heuristic performs constant transitions between feasible and infeasible portions of the solution space. These transitions are regulated by a dynamic adjustment of the penalty applied to infeasible solutions. An extensive statistical analysis was carried out in order to identify the most important components of the algorithm and to properly tune the values of their parameters. The results of the computational experiments carried out show that this algorithm is very competitive in comparison to the best metaheuristic algorithms for the VRPB. Additionally, new best solutions have been found for two instances in one of the benchmark sets. These results show that the performance of existing metaheuristic algorithms can be considerably improved by carrying out a thorough statistical analysis of their components. In particular, it shows that by expanding the exploration area and improving the efficiency of the local search heuristic, it is possible to develop simpler and faster metaheuristic algorithms without compromising the quality of the solutions obtained.

1. Introduction

The Vehicle Routing Problem (VRP) was originally described by Dantzig and Ramser (1959) and considers the delivery of goods from a central depot to a set of customers. The basic VRP involves determining a set of routes for a fleet of vehicles, each one starting and ending at the depot, such that the demand of every customer is satisfied and the total cost is minimized. A solution to the VRP must satisfy the following constraints: (I) every customer must be visited exactly once and (II) the amount of goods delivered by each vehicle must not exceed the vehicle capacity. The VRP is one of the most frequently studied combinatorial optimization problems in the literature. Its practical importance has encouraged the operations research community to develop more complex variants that better model the scenarios found in the real world. The Vehicle Routing Problem with Backhauls (VRPB) is an extension of the VRP that deals with two types of customers: the consumers (*linehaul*) that request goods from the depot and the suppliers (*backhaul*) that send goods to the depot. The VRPB was formulated to model the distribution process of companies that transport their final products to retailer stores, and supply their production center with commodities from providers in the same area. However, applications of the VRPB can be found in many scenarios that involve the return of goods to the distribution center (e.g. reverse logistics). A solution to the VRPB must satisfy the following additional constraints: in each route, (I) the load of goods sent to the *consumers* and the load of goods received from the *suppliers* must not exceed the vehicle capacity, (II) every vehicle must visit at least one *consumer* and (III) must visit the *consumers* before the *suppliers*. The second constraint is due to the assumption that the delivery of goods to the consumers is the main profitable activity. In such a scenario, a vehicle should only be used provided there is at least one consumer that needs to be served. The third constraint is

* Corresponding author at: University of Antwerp, Faculty of Applied Economics, ANT/OR Operations Research Group, Prinsstraat 13, 2000 Antwerp, Belgium. Tel.: +32 3 265 41 33.
E-mail address: daniel.palhazicuervo@uantwerpen.be (D. Palhazi Cuervo).

http://dx.doi.org/10.1016/j.ejor.2014.02.011
0377-2217/

due to the assumption that the load cannot be rearranged at the delivery or pick-up points. For an extensive classification of the VRP variants and the related literature, see Eksioglu, Vural, and Reisman (2009).

The VRPB is a generalization of the original VRP. Therefore, it is an NP-hard problem in the strong sense. Even though there are exact algorithms that are able to find optimal solutions for instances with around a hundred customers (Mingozzi, Giorgi, & Baldacci, 1999), a heuristic approach is still necessary to solve larger instances. Recent metaheuristic implementations have proven to be the most successful approaches to solve the VRPB in terms of computing time and solution quality (Brandão, 2006; Gajpal & Abad, 2009; Røpke & Pisinger, 2006; Zachariadis & Kiranoudis, 2012). These algorithms usually have multiple components and parameters that need to be tuned. Nevertheless, it is very unusual to find solid (statistical) studies about the efficacy of each building block and the effect of the parameter values on the algorithm performance. Such studies are of paramount importance in order to identify the reasons why an algorithm is effective and to identify its key components. This allows an algorithm to be adapted in order to exploit the building blocks that have proven to be effective, and to remove the unnecessary components. It is our view that the use of more complex components or strategies should be justified by the value they add to the algorithm performance, either in terms of solution quality or execution time.

In this paper, we propose a simple iterated local search (ILS) algorithm to solve the VRPB. The main component of this algorithm is an oscillating local search (OLS) heuristic that allows the consideration of infeasible solutions. The term "oscillating" refers to the constant transitions between feasible and infeasible portions of the solution space. These transitions are regulated by a dynamic adjustment of the penalty applied to infeasible solutions. Additionally, the OLS heuristic explores a rich neighborhood structure (produced by four different neighborhood operators) at each iteration. This is efficiently done by implementing an additional data structure that stores information about the set of neighboring solutions. The execution time of the heuristic is considerably reduced by an appropriate update of this data structure at each iteration. We discuss the results of an extensive computational experiment carried out with two purposes: (I) to identify the key features of the algorithm and (II) to determine the set of parameter values that produce the best algorithm performance. Finally, we compare the ILS algorithm to the best-performing metaheuristic algorithms available in the literature.

The structure of the paper is as follows. A mathematical definition of the VRPB based on graph theory and second-order logic is presented in Section 2. A brief overview of the literature is presented in Section 3. The ILS algorithm is described in Section 4 along with some important features of its implementation. The computational experiments carried out and their results are discussed in Section 5. The performance of the ILS algorithm is compared to that shown by other metaheuristic algorithms for the VRPB in Section 6. We present our final conclusions in Section 7.

2. Problem definition

The VRPB can be defined using graph theory and second-order logic. Consider a complete undirected graph $G = (V, E)$ where $V = D \cup L \cup B$ is a set of $1 + l + b$ vertices composed of the disjoint subsets $D = \{0\}, L = \{1, \ldots, l\}$ and $B = \{l+1, \ldots, l+b\}$ that represent the depot, the linehaul customers and the backhaul customers, respectively. The set $E = \{(i,j) | (i,j) \in V \times V, i \neq j\}$ is the set of edges that connect the customers with each other and with the depot. A nonnegative value $c_{i,j}$ is associated with every edge $(i,j) \in E$, representing the cost to travel from customer i to customer j. An amount of goods q_i is associated with every customer $i \in L \cup B$ that represents the amount requested from or delivered to the depot, depending on whether the customer is linehaul ($i \in L$) or backhaul ($i \in B$). There are m identical vehicles in the depot, each with capacity Q. It is assumed that $m \geqslant \max(m_L, m_B)$, where m_L and m_B are the minimum numbers of vehicles needed to separately serve all linehaul and backhaul customers, respectively. Each of these parameters can be determined by solving a bin packing problem. A solution to the VRPB comprises a set of m routes. Each route $r \in R = \{1, \ldots, m\}$ is defined as a chain of s_r vertices $\langle v^r_1, v^r_2, \ldots, v^r_{s_r} \rangle$, that satisfies the following constraints:

1. Every route starts and finishes at the depot.

$$\forall r : 1 \leqslant r \leqslant m \mid v^r_1 = v^r_{s_r} = 0 \tag{1}$$

2. Every vehicle visits at least one linehaul customer.

$$\forall r : 1 \leqslant r \leqslant m \mid (s_r > 2) \wedge (\exists i : 1 < i < s_r \mid v^r_i \in L) \tag{2}$$

3. Every customer is visited exactly once.

$$\begin{aligned} &\forall i : i \in L \cup B \mid (\exists r, k : 1 \leqslant r \leqslant m \wedge 1 < k < s_r \mid v^r_k = i \\ &\wedge \neg(\exists r', k' : 1 \leqslant r' \leqslant m \wedge 1 < k' < s_{r'} \mid v^{r'}_{k'} = i \\ &\wedge \neg(r' = r \wedge k' = k))) \end{aligned} \tag{3}$$

4. None of the vehicles performs an intermediate stop at the depot.

$$\forall r : 1 \leqslant r \leqslant m \mid (\forall i : 1 < i < s_r \mid v^r_i \neq 0) \tag{4}$$

5. In every route, neither the load of goods sent to the linehaul customers nor the load of goods received from the backhaul customers exceeds the vehicle capacity.

$$\begin{aligned} &\forall r : 1 \leqslant r \leqslant m \mid ((\Sigma i : 1 < i < s_r \wedge i \in L \mid q_{v^r_i}) \leqslant Q) \\ &\wedge ((\Sigma i : 1 < i < s_r \wedge i \in B \mid q_{v^r_i}) \leqslant Q) \end{aligned} \tag{5}$$

6. In every route, the linehaul customers are served before the backhaul customers.

$$\begin{aligned} &\forall r : 1 \leqslant r \leqslant m \mid (\forall i, j : 1 \leqslant i < j \leqslant s_r \mid (v^r_j \in L \Rightarrow v^r_i \in L \cup D) \\ &\wedge (v^r_i \in B \Rightarrow v^r_j \in B \cup D)) \end{aligned} \tag{6}$$

The objective of the VRPB is to find a set of routes that minimize the total cost. The objective function can be written as:

$$\text{minimize} \quad \Sigma r : 1 \leqslant r \leqslant m \mid (\Sigma i : 1 \leqslant i < s_r \mid c_{v^r_i, v^r_{i+1}}) \tag{7}$$

3. Literature review

Several exact and heuristic algorithms are available in the literature to solve the VRPB. Exact algorithms perform a systematic search over the solution space and guarantee to find the best possible solution. However, since the execution time of these algorithms increases at an exponential rate, they are only useful to solve small instances. The first exact algorithm for the VRPB was proposed by Yano et al. (1987). They develop a branch and bound framework based on a set covering approach for a retail chain. Toth and Vigo (1997) propose a branch and bound algorithm in which a lower bound is obtained from the Lagrangian relaxation of some constraints of the underlying linear programming model. The lower bound is then progressively strengthened in a cutting plane fashion. Mingozzi et al. (1999) propose a new VRPB linear programming model and develop a branch and bound algorithm. This algorithm computes a lower bound by combining different heuristic methods for solving the dual LP-relaxation of the exact formulation.

Heuristic algorithms have been shown to be able to overcome the limitations of exact approaches. These algorithms are generally able to find solutions within an acceptable computing time for instances that are not tractable by exact algorithms. However, the solutions obtained cannot be guaranteed to be optimal. The first heuristic algorithm for the VRPB was proposed by Deif and Bodin (1984). They develop an extension of the Clarke and Wright heuristic (Clarke & Wright, 1964) for the VRP. Goetschalckx and Jacobs-Blecha (1989) develop a two-phase heuristic based on a space-filling curves approach. Jacobs-Blecha and Goetschalckx (1992) later introduce an extension of the generalized assignment heuristic for the VRP proposed by Fisher and Jaikumar (1981). Toth and Vigo (1999) propose a "cluster first, route second" algorithm. It uses a clustering method that exploits the information contained within a solution obtained from a Lagrangian relaxation of the VRPB.

More recently, the most successful strategies that have been applied in heuristic algorithms have converged to more general frameworks called metaheuristics. A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sörensen & Glover, 2013). There exist several metaheuristic implementations for the VRPB. Osman and Wassan (2002) propose a reactive tabu search in which the tabu tenure is dynamically modified during the search process. Brandão (2006) develops a new multi-phase tabu search procedure in which the initial solution is obtained from the formulation of the VRPB as a minimum cost K-tree problem. Røpke and Pisinger (2006) propose an unified model that can handle many VRP variants and is solved by a large neighborhood search. Wassan (2007) develops a hybrid algorithm combining a reactive tabu search with adaptive memory programming. Gajpal and Abad (2009) propose an ant colony system with two types of ants: the first type is used to assign customers to vehicles and the second type is used to construct the routes. Zachariadis and Kiranoudis (2012) propose a local search heuristic that diversifies the exploration by incorporating the concept of promises when evaluating a tentative move. This diversification strategy is inspired by the memory structure characteristic of the tabu search metaheuristic. At each iteration, sequences of customers that appear in the solution explored are tagged with a promise value that is equal to the cost of the solution. These promise values are used to discard lower-quality moves that involve the same sequences of customers.

Most of the metaheuristic algorithms for the VRPB include, or are based upon, a local search (LS) heuristic.[1] However, the algorithms differ in the diversification mechanism they use. For instance, the ant colony system (Gajpal & Abad, 2009) uses a random-proportional rule at each iteration of the construction phase. This rule is influenced by the solutions previously explored via the update of the pheromone trail. The algorithms based upon the tabu search metaheuristic (Brandão, 2006; Osman & Wassan, 2002; Wassan, 2007) prevent the return to solutions already explored by using a memory structure. This structure is updated using a strategy that is particular to each algorithm. Brandão (2006) uses a static strategy, while Osman and Wassan (2002) and Wassan (2007) and Wassan (2007) use a strategy that is adapted according to the effectiveness of the exploration. The local search proposed by Zachariadis and Kiranoudis (2012) follows a similar strategy to prevent the cycling phenomenon, although its implementation requires more memory. The algorithm stores sequences of consecutive customers that have appeared in solutions already explored. Local search moves that involve the stored sequences are only performed if the solution cost of the new solution is lower than the costs of the solutions previously explored. The implementation of these diversification mechanisms involve additional building blocks and data structures that increase the complexity of the algorithms.

The common component of most existing algorithms is the LS heuristic. This suggests that the LS is the key requirement for effectively solving the VRPB. Therefore, we strongly believe that, in order to achieve a better performance, the LS component should receive most of the attention. More specifically, we believe it is possible to implement metaheuristics that apply simpler diversification mechanisms (without compromising the quality of the solutions obtained) by expanding the exploration area and improving the performance of the LS heuristic. The development of an algorithm with these characteristics is the main goal of this paper.

4. Iterated local search algorithm

The ILS is a metaheuristic that involves the iterative application of a LS heuristic and the use of a perturbation as a diversification mechanism. At each iteration, a new initial solution is generated and used by the LS heuristic as a new starting point for the search. This initial solution is generated by randomly performing a small modification, called perturbation, to a good locally optimal solution previously found. Instead of generating a new initial solution from scratch, the perturbation mechanism generates a promising initial solution by retaining part of the structure that made the original solution a good solution. Despite its simplicity, the ILS algorithm has proven to be a very successful approach to solve combinatorial optimization problems (Stützle, 2006; Vansteenwegen, Souffriau, Vanden Berghe, & Van Oudheusden, 2009; Voudouris & Tsang, 1999). A detailed explanation of the ILS metaheuristic can be found in Lourenço, Martin, and Stützle (2003).

The pseudocode of the ILS algorithm that we implemented is shown in Algorithm 1. At each iteration, the perturbation is applied to the best feasible solution found so far. The execution of the algorithm is stopped when a maximum number of iterations is reached. In the following sections, we describe the main components of the algorithm: the procedure used to generate the first initial solution, the OLS heuristic and the perturbation. Additionally, we discuss important features of the implementation in order to improve the performance of the algorithm.

Algorithm 1. Iterated local search (ILS)

Input: An initial solution S_0

$S_{\text{best}} \leftarrow \text{OLS}(S_0)$
for $i \leftarrow 1$ **to** num_it **do**
  $S_{\text{pert}} \leftarrow \text{perturb}(S_{\text{best}})$
  $S_{\text{pert}} \leftarrow \text{OLS}(S_{\text{pert}})$
  if $\text{cost}(S_{\text{pert}}) < \text{cost}(S_{\text{best}})$ **then**
    $S_{\text{best}} \leftarrow S_{\text{pert}}$
return S_{best}

4.1. Initial solution

We implemented two insertion heuristics for the construction of the initial solution. Both heuristics start with an empty solution (composed of m empty routes) and iteratively insert one customer at a time. Since the definition of the VRPB requires that every vehicle visits at least one linehaul customer (see constraint (2) in Section 2), both heuristics insert a randomly selected linehaul customer at the beginning of each route as an initialization step. At each following iteration, one of the remaining customers is inserted in the solution. Each heuristic applies a different approach

[1] It is our opinion that the large neighborhood search proposed by Røpke and Pisinger (2006) cannot be considered as a traditional LS heuristic. Instead, it is an algorithm that combines several insertion and removal heuristics to explore a larger portion of the solution space. This strategy is different from the conventional LS, which uses simpler neighborhood operators.

in order to choose the customer to be inserted and to determine the route and the position to insert it in.

The first heuristic applies a completely random approach. At each iteration, one customer is randomly selected and inserted in a randomly chosen route. If the selected customer is a linehaul customer, it is inserted at the beginning of the route. If it is a backhaul customer, it is inserted at the end of the route. The customers are inserted in the solution without considering the capacity constraint of the problem. This relaxation leads to initial solutions that might or might not violate the vehicle capacity constraint.

The second heuristic applies a greedy approach similar to the insertion heuristic described by Potvin and Rousseau (1993) and discussed in depth by Campbell and Savelsbergh (2004). At each iteration, the heuristic evaluates the insertion of each remaining customer in every route at every possible position. The heuristic performs the insertion (by selecting the customer, the route and the position in the route) that produces the smallest increase in the solution cost. Initially, the capacity constraint is verified every time a customer is considered for insertion. However, this verification is stopped if no feasible insertion can be found for any of the remaining customers to be inserted. This leads the greedy heuristic to produce feasible solutions, or solutions with a relatively small excess load when the order of insertions does not allow a feasible solution to be generated. The random initialization step allows the greedy heuristic to generate different initial solutions for the same problem instance.

The capacity constraint is the only constraint that is relaxed during the ILS algorithm. This relaxation is particularly helpful when generating initial solutions for instances that are very restricted, and for which finding a feasible initial solution is very difficult. In that case, the responsibility of finding a feasible solution is delegated to the OLS heuristic. Only solutions that comply with every other VRPB restriction are considered during the execution of the ILS algorithm. Consequently, the number of routes is always equal to the number of vehicles and the service order is always respected in every route.

4.2. Oscillating local search heuristic

We developed a new OLS heuristic for the VRPB. This heuristic allows the consideration of solutions that violate the capacity constraint of the problem. The idea of considering infeasible solutions during the search process is not new in the VRP literature (for instance, see Nagata & Bräysy (2009) and Toth & Vigo (2003) and Toth & Vigo (2003)). The objective of this approach is to allow temporary excursions to an infeasible portion of the solution space and help the algorithm to reach new promising feasible regions. The mechanism that guides these excursions is a dynamic adjustment of the penalty applied to the cost of infeasible solutions.

The pseudocode of the OLS heuristic is shown in Algorithm 2. The neighborhood structure considered is generated by the well known operators (Kinderwater & Savelsbergh, 1997): (I) Intra-route and inter-route customer relocation, (II) Intra-route and inter-route customer exchange, (III) Inter-route crossover and (IV) Intra-route 2-opt. All four neighborhoods are simultaneously explored and a new solution is selected to continue the search using an overall best improvement strategy. The solution with the lowest cost is selected from the entire set of neighboring solutions produced by the different operators. The mechanism that allows the efficient generation and update of the set of neighboring solutions is outlined in Section 4.4. The cost function used to evaluate the quality of a solution is the one described by Brandão (2006):

$$\text{cost}(S,\alpha) = \text{cost}(S) + \alpha \sum_{1 \leqslant r \leqslant m} [\text{lh\_excess}(r) + \text{bh\_excess}(r)] \qquad (8)$$

The first term of the expression is the total cost of the routes. The second term is the sum of the excess load (both the load requested by the linehaul customers and the load supplied by the backhaul customers) transported by each vehicle, multiplied by a penalty α. If every vehicle of the solution satisfies the capacity constraint, this term is equal to zero. The penalty α defines the influence of the capacity constraint violation on the overall solution cost. This penalty is initialized to a value α_0 and multiplied by a factor $\beta > 1$ when the exploration process cannot find a better solution.

An initial solution S_0, the initial penalty α_0 and the penalty increase factor β are input parameters of the heuristic. The solution S_0 can be infeasible and have a total excess load greater than zero. In fact, in the context of the ILS algorithm, this is usually the case. The initial solution S_0 is produced by either one of the insertion heuristics (described in Section 4.1) or the perturbation operator (explained in Section 4.3). Neither of these guarantees solutions that comply with the capacity constraint. It is the OLS heuristic's responsibility to find a feasible solution. In order to obtain a feasible solution for all instances, the OLS heuristic must involve a broad enough neighborhood structure. The four neighborhood operators mentioned above lead to such neighborhood structure. If the set of neighboring solutions is not large enough, the transition of the OLS heuristic to a feasible portion of the solution space may be problematic.

Initially, the OLS heuristic explores the set of neighboring solutions and selects the solution with the lowest cost considering the value of the penalty α (lines 10 and 11 of the pseudocode in Algorithm 2). This improvement procedure is repeated until a locally optimal solution is found (i.e. until the set of neighboring solutions does not contain a solution with a lower cost). If the locally optimal solution found is infeasible, the penalty is increased in order to encourage the exploration of solutions with lower excess loads (lines 19 and 20 of the pseudocode). If the locally optimal solution found is feasible, the algorithm verifies whether the new solution is better than the best feasible solution found so far. If so, the new locally optimal solution is stored and the penalty factor is set back to α_0 (lines 13–16 of the pseudocode). If the current solution is worse than or has the same cost as the previous best feasible solution found, the heuristic stops its execution (lines 17 and 18 of the pseudocode). This is because this condition suggests that the exploration is cycling around the same feasible locally optimal solution, or that the exploration has been oriented to an unpromising portion of the solution space (that contains worse feasible solutions). The application of this strategy gives an oscillating pattern to the search performed by the OLS heuristic. This pattern can be observed in the values of the excess load transported by the vehicles in the solutions explored during the execution of the algorithm. Fig. 1 shows the oscillating excess loads for each iteration of the algorithm, along with the total routing costs.[2] Note the start of a new oscillation every time a better feasible solution (characterized by zero excess load) is found. Furthermore, note that, when the oscillation starts and the penalty α is low, the heuristic is able to explore solutions with lower routing costs due to the relaxation of the capacity constraint. When the penalty α is increased at the end of the oscillation, the capacity constraint is imposed again and the total routing costs of the solutions tend to increase.

The initial value α_0 (of the penalty parameter α) and the parameter β define the execution of the OLS heuristic. The value α_0 determines the ability of the heuristic to explore infeasible regions of the solution space. In other words, it determines the maximum excess load in the oscillations shown in Fig. 1. The larger the α_0 value,

[2] Fig. 1 was obtained by applying the OLS heuristic to solve instance N1 of the benchmark set GJB. See Section 5 for more information about the set of benchmark instances used in this paper.

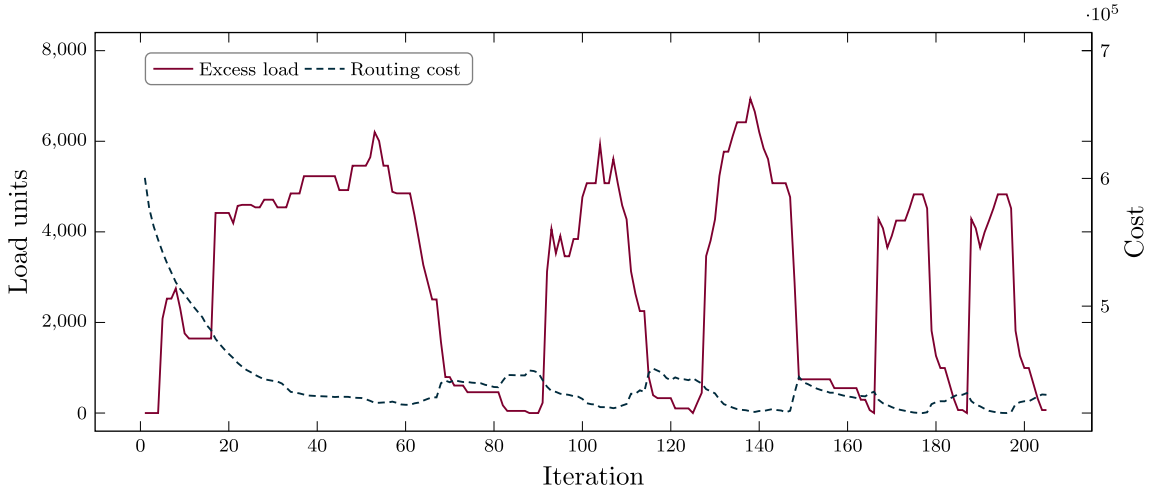

Fig. 1. Search pattern of the OLS heuristic in terms of the excess load and the routing cost of the solutions explored at each iteration.

the smaller the ability of the heuristic to explore infeasible solutions. The parameter β determines the speed of the transition from infeasible to feasible portions of the solution space. It defines the length of the oscillations shown in Fig. 1. The larger the β value, the faster the penalization of infeasible solutions is increased and the faster the search is oriented towards feasible solutions. The effect of both parameters on the performance of the ILS algorithm is studied in Section 5.1.

Algorithm 2. Oscillating local search (OLS) heuristic

Input: An initial solution S_0, an initial penalty value α_0 and a penalty increase factor β

```
if is_feasible(S_0) then
    S_best_feas ← S_0
    cost_best_feas ← cost(S_0)
else
    cost_best_feas ← ∞
α ← α_0
S_act ← S_0
stop_criterion ← false
while ¬stop_criterion do
    while neighbors_with_lower_costs(S_act, α) ≠ ∅ do
        S_act ← neighbor_with_lowest_cost(S_act, α)
    if is_feasible(S_act) then
        if cost(S_act) < cost_best_feas then
            cost_best_feas ← cost(S_act)
            S_best_feas ← S_act
            α ← α_0
        else
            stop_criterion ← true;
    else
        α ← β × α
return S_best_feas
```

Oscillating heuristics that have been proposed previously apply a different strategy to update the penalty factor α. The usual approach is to update α according to the region of the solution space that has been explored (Brandão, 2006; Toth & Vigo, 2003). If the heuristic has explored a feasible region for an arbitrary number of iterations, α is decreased in order to encourage the exploration of infeasible solutions. Similarly, if the heuristic has explored an infeasible region for a given number of iterations, α is increased in order to guide the exploration to a feasible region. Unlike in this strategy, we update the penalty factor according to the effectiveness of the exploration. The penalty α is increased only if the OLS heuristic gets trapped in an infeasible locally optimal solution. The α value is set back to α_0 when a new best feasible solution is found and another complete oscillation has been performed.

4.3. Perturbation

The perturbation mechanism iteratively relocates customers in the solution. At each iteration, the operator removes a randomly selected customer, and inserts it back in a different randomly selected position. As in the generation of the initial solution, the insertions of the customers are performed without considering the capacity constraint. This usually leads the perturbation to produce solutions that are infeasible. However, the feasibility of the solutions is restored by the application of the OLS heuristic. The number of customers relocated is the parameter that defines the size of the perturbation. This is a key factor of the ILS algorithm because it determines the portion of the locally optimal solution that is modified. If the perturbation is too small, the ILS algorithm will tend to get trapped in a locally optimal solution. On the other hand, if the perturbation is too large, the information contained within the good locally optimal solution is lost and the ILS algorithm behaves as if a random initial solution were used at each iteration. The effect of the size of the perturbation on the performance of the ILS algorithm is also studied in Section 5.1.

4.4. Implementation details

In this section, we describe two important features of the implementation of the ILS algorithm: the data structure used to represent a VRPB solution, and the calculation process of the neighboring solutions generated during the OLS heuristic. This is because a wise choice of these features results in a substantial reduction of the computation time.

A solution to the VRPB is represented using a modified version of the data structure proposed by Kytöjoki, Nuortio, Bräysy, and Gendreau (2007). This structure combines the versatility of a linked list and the fast manipulation of static arrays. The calculation of the neighboring solutions is the most time-consuming step of the OLS heuristic, since all four neighborhoods are simultaneously explored at each iteration. In order to reduce the execution time of this step, we implemented a strategy very similar to that proposed by Zachariadis and Kiranoudis (2010). The main principle behind this strategy is simple: when a local search operator is applied to a solution, only a limited portion of that solution is

modified. As a result, the set of neighboring solutions is also only modified to a limited extent. Only the neighboring solutions that involve the modified portion of the solution need to be recalculated.

Each neighboring solution is represented using a data structure that Zachariadis and Kiranoudis (2010) call *static move descriptor*. This structure stores the operator that produces the solution, the portion of the solution the operator is applied to (the routes and the customers involved) and the difference in cost. We extended this structure by additionally storing the difference in the excess load transported by the vehicles. This value is required for the calculation of the cost function used by the OLS heuristic. The entire set of neighboring solutions is generated only at the first iteration of the OLS heuristic. Taking into account the fact that the neighborhood operators involve one or two routes, this complete generation has a quadratic time complexity $\Theta((l+b)^2)$. In the following iterations, only a small subset of neighboring solutions are recalculated. Zachariadis and Kiranoudis (2010) describe with great detail the set of rules to be used in order to determine the neighboring solutions to be recalculated. However, that set of rules is not applicable to our implementation because it does not consider the update of the difference in the excess load transported. This is because the excess load of a vehicle is dependent on the entire set of customers served by that vehicle. For that reason, when the OLS heuristic selects a new solution to continue the search, it is necessary to update all neighboring solutions that involve the routes that have been modified (by applying the relevant operator). Assuming that the customers are (approximately) uniformly distributed over the different routes, the time complexity of the update procedure can be expressed as $\Theta((l+b)/m)$. The application of this simple rule leads to the update of a larger number of neighboring solutions in comparison to the set of rules proposed by Zachariadis and Kiranoudis (2010). Nevertheless, the reduction of the execution time is still very substantial in comparison to the full generation of all neighboring solutions. On the other hand, this simple rule is a more flexible approach from an implementation point of view, as it can be used for updating the neighborhoods produced by more complex operators. Specific set of rules similar to those described by Zachariadis and Kiranoudis (2010) can be difficult to define for the Ejection Chain (Glover, 1996) or the K-opt (Lin, 1965), for example. Secondly, our approach can be extended to handle additional constraints on the routes (i.e. tour length, time windows or limited driving hours) without requiring major modifications. The exploration of these ideas is a promising avenue for future research.

5. Computational experiments

A set of numerical experiments was performed using two benchmark sets of instances available in the literature. The first set (*set GJB*) was proposed by Goetschalckx and Jacobs-Blecha (1989) and consists of 68 instances where the total number of customers ranges from 25 to 200.[3] This set was originally composed of 15 groups of benchmark instances (named after the letters A to O). However, most of the literature only considers the first 14 groups and disregards group O. For this reason, and in order to obtain results that are comparable to those of other algorithms, we also exclude group O from the numerical experiments. However, the cost of the solutions obtained by our algorithm for the instances within this group are included in the summary table shown in Appendix B. The second benchmark set (*set TV*) was proposed by Toth and Vigo (1997) and consists of 33 instances where the total number of customers ranges from 21 to 100. This set was generated based on 11 VRP instances proposed by Eilon, Watson-Gandy, and Christofides (1971, chap. 9). Each of these VRP instances was used to construct three new VRPB instances by considering 33%, 50% and 66% of the customers as linehaul customers. The algorithms described in the previous sections were coded in C++ and all the experiments were executed on a 2.93 gigahertz Intel Core i7 processor. It is important to point out that the cost to travel between each pair of customers is calculated in different ways for each benchmark set: for the set GJB, the costs are calculated using double precision, while for the set TV, they are rounded to the nearest integer. This is a convention used by most of the algorithms available in the literature. We stick to it in order to obtain solution costs that can be compared with those produced by other algorithms.

5.1. Statistical analysis

In this section, we describe the statistical analysis carried out to better understand the behavior of the ILS algorithm. The main purpose of this analysis is to identify the key components of the algorithm and to determine the set of parameter values that yields the best performance. The following parameters were studied using a full factorial experiment:

- The procedure to generate the first initial solution (`ini_sol`), i.e. the random heuristic (`random`) or the greedy heuristic (`greedy`).
- The size of the perturbation (`pert_size`).
- The initial penalty (α_0).
- The multiplicative factor used to increase the penalty (β).

Note that the number of iterations executed by the algorithm is not in the list of parameters studied. This is because it is reasonable to expect that the larger the number of iterations executed, the larger the execution time of the algorithm and the better the quality of the solutions obtained. For that reason and in order to reduce the size of the experiment, we fixed the number of iterations to 400. Still, the impact of this parameter on the performance of the algorithm is analyzed separately in Section 5.2.

The different values for the parameters studied are shown in Table 1. Note that the values of the perturbation size (`pert_size`) are expressed as a percentage of the total number of customers $(l+b)$ in an instance. Also, note that the first value tested for the initial penalty (α_0) is zero. When $\alpha_0 = 0$, the OLS heuristic is able to explore infeasible regions of the solution space without any restriction. In this case, when the penalty needs to be increased for the first time, α is assigned the value one. Additionally, note that when $\alpha_0 = 100$, the capacity of the OLS heuristic to explore infeasible regions is very limited.

The algorithm was executed 10 times using each combination of parameter values to solve every instance in groups A–N of the benchmark set GJB, resulting in $2 \times 3 \times 5 \times 6 \times 62 \times 10 = 111{,}600$ executions. For each set of 10 executions for a given problem instance, three performance measures were determined: the cost of the best solution found (out of the 10 executions), the average solution cost and the average execution time. For each of these measures, a mixed-effects analysis of variance (ANOVA) model was estimated using the statistical package JMP. Each model uses

Table 1
Parameters and levels tested.

Parameter	Levels
`ini_sol`	`random`, `greedy`
`pert_size`	10%, 20%, 30%, 40%, 50%
α_0	0, 1, 2, 5, 10, 100
β	1.10, 2, 5

[3] There are some discrepancies between different papers regarding some instances in this benchmark set; particularly for the group of instances F and the instance L1 (see Zachariadis & Kiranoudis (2012) for further discussion). Therefore, it is important to clarify that, in this paper, all the instances in group F consider the demand of the customer located at position (5103, 11065) to be equal to 101. Additionally, the vehicle capacity used for the instance L1 is equal to 4400.

Table 2
p-Values of the *F*-tests to determine the significance of each term in the ANOVA models for the best solution cost, the average solution cost and the average execution time.

Factor	Best solution cost	Avg. solution cost	Execution time
ini_sol	0.3701	0.1799	0.7466
pert_size	**0.0006**	**<0.0001**	**<0.0001**
α_0	**0.0007**	**<0.0001**	**<0.0001**
β	0.6327	0.6830	0.1115
ini_sol × pert_size	0.9990	0.5237	0.8955
ini_sol × α_0	0.9303	0.9992	0.9604
ini_sol × β	0.2811	0.7479	0.8960
pert_size × α_0	**<0.0001**	**<0.0001**	**<0.0001**
pert_size × β	0.9788	0.9799	0.6874
$\alpha_0 \times \beta$	0.9440	0.9951	0.6379

a random effect for the instance of the benchmark set in order to indicate that all the measurements for the same instance are correlated. Table 2 shows the *p*-values of the *F*-tests that indicate the significance of each parameter or interaction in the ANOVA models. Bold *p*-values indicate parameters or interactions that have a significant impact on the corresponding measure.

The size of the perturbation (pert_size) and the initial penalty (α_0) are the important parameters of the algorithm. Observe that both parameters, as well as their interaction, are statistically significant for all the performance measures. This shows that the ability of the OLS heuristic to oscillate between feasible and infeasible portions of the solution space, and the degree of diversification provided by the perturbation, are the key aspects of the algorithm. Unexpectedly, neither the initial solution (ini_sol) nor the multiplicative factor (β) have a significant impact on any of the performance measures. This shows that the search performed by the algorithm is robust enough to be insensitive to the initial solution. Additionally, the OLS heuristic is insensitive to the transition speed from infeasible to feasible portions of the solution space. What is important is the ability to explore infeasible solutions, not the way in which the exploration is oriented to a feasible region.

The average execution time for each combination of the parameters pert_size and α_0 is shown in Fig. 2. Observe that the execution time of the algorithm increases with the perturbation size and the ability of the OLS heuristic to explore infeasible solutions (hence, the execution time is inversely proportional to α_0). In other words, the larger the portion of the solution space that can be explored by the OLS heuristic, the longer the time consumed by the search process. Similarly, the larger the portion of the solution that is modified by the perturbation, the longer the time that is needed by the OLS heuristic to find a locally optimal solution.

The average solution cost and the average cost of the best solutions found for each combination of the parameters pert_size and α_0 are shown in Figs. 3 and 4, respectively. Observe from Fig. 3 that the perturbation size that produces the best algorithm performance in terms of average solution cost is 30%. Additionally, Fig. 4 shows that the average cost of the best solutions found is minimized when the perturbation is between 20% and 30%. Smaller perturbations generally do not allow the algorithm to escape from poor locally optimal solutions. Larger perturbations lead to a loss of information about the good locally optimal solutions which prevents the algorithm from properly exploring their surrounding solution space.

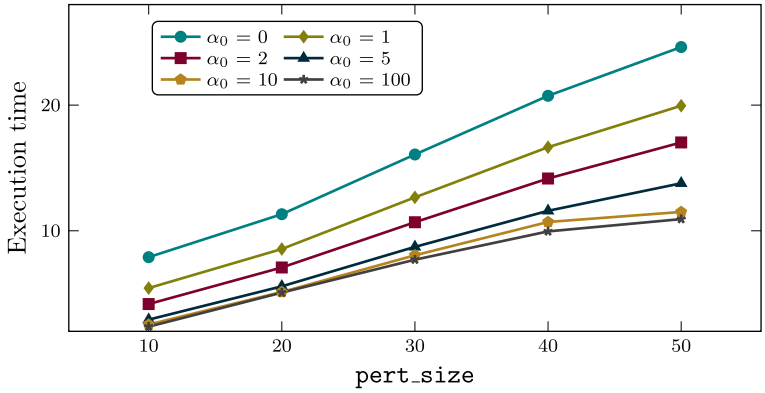

Fig. 2. Influence of the perturbation size (pert_size) and the initial penalty (α_0) on the average execution time.

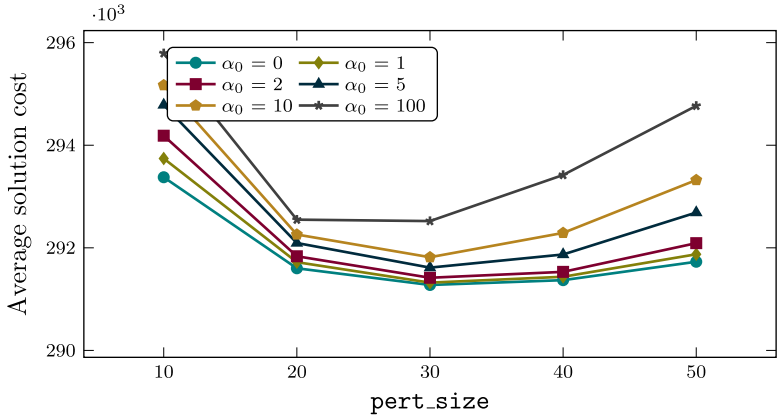

Fig. 3. Influence of the perturbation size (pert_size) and the initial penalty (α_0) on the average cost of the solutions found.

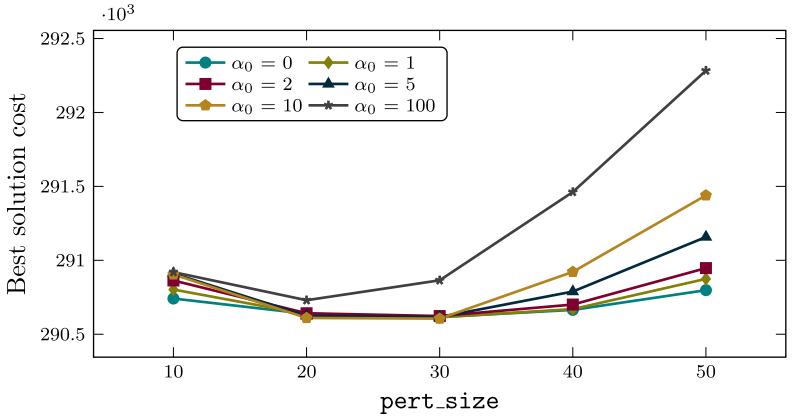

Fig. 4. Influence of the perturbation size (`pert_size`) and the initial penalty (α_0) on the average cost of the best solutions found.

The large impact of the exploration of infeasible solutions on the algorithm performance deserves a more careful analysis. In Fig. 3, it can be observed that the more capable the algorithm is to explore infeasible portions of the solution space (the lower the initial penalty α_0), the lower the average cost of the solutions found. This shows that the robustness of the algorithm increases as a result of this ability. A similar trend can be observed in Fig. 4 for the average cost of the best solutions found. However, when the perturbation size is between 20% and 30%, this performance measure seems to be insensitive to the exact value of α_0, provided it is between 0 and 10. In contrast, when $\alpha_0 = 100$, the quality of the best solutions found goes down to a considerable extent. As a result, any small value of α_0 leads to excellent solutions provided the algorithm can be executed at least 10 times and the perturbation size is equal to 30%. On the contrary, high values of α_0 limit the ability to explore infeasible solutions and have a considerably negative impact on the solution quality.

5.2. Effect of the number of iterations on the algorithm performance

In this section, we analyze the effect of the number of iterations on the performance of the ILS algorithm. We study the trade-off between the execution time of the algorithm and the quality of the solutions obtained. To this end, we executed the algorithm 10 times using five different numbers of iterations to solve the instances in both benchmark sets. The values used for the other parameters are shown in Table 3. The results obtained for both benchmark sets are shown in Table 4. The second column shows the number of best known solutions found by the ILS algorithm. The next pairs of columns show the average cost and the average ratio ((solution cost/best known solution cost) $\times$ 100) for the best solution cost and the average solution cost, respectively. The final column shows the average execution time of the algorithm. Observe that the algorithm finds very good solutions, even within a very short execution time. A number of iterations equal to 400 seems to suffice to find the best known solutions for the vast majority of the instances in the benchmark sets. Larger numbers of iterations mainly increase the robustness of the algorithm by decreasing the average cost of the solutions obtained. Additionally, observe that the rate at which the execution time grows decreases with the number of iterations. The execution time increases by a factor of 2 when the number of iterations is increased from 100 to 200. However, the execution time only increases by a factor of around 1.5 when the number of iterations is increased from 400 to 800. This phenomenon is due to the stopping criterion used by the OLS heuristic. The OLS heuristic stops either when the exploration cycles or when it has been oriented to an unpromising portion of the solution space (that contains worse feasible solutions). Therefore, the better the quality of the initial solution, the shorter the number of oscillations generally performed by the OLS. As a result, the execution time of the OLS tends to be shorter when the ILS algorithm explores portions of the solution space that are close to a very good solution. Naturally, the probability of this happening increases with the number of iterations already executed.

Table 3
Parameters and final values used.

Parameter	Level
`ini_sol`	`random`
`pert_size`	30%
α_0	0
β	5

Table 4
Effect of the number of iterations on the solution quality and the average execution time.

Iter.	Num. best sols.	Best solution cost		Average solution cost		Avg. time (seconds)
		Avg. cost	Avg. ratio	Avg. cost	Avg. ratio	
Set GJB						
100	50/62	290683.81	100.03	292520.09	100.61	4.25
200	56/62	290623.19	100.01	291748.59	100.36	8.23
400	58/62	290593.84	100.00	291332.41	100.23	14.31
800	61/62	290582.86	100.00	291229.25	100.19	20.12
1000	62/62	290576.21	100.00	291170.16	100.17	22.89
Set TV						
100	28/33	701.19	100.07	706.91	100.80	1.43
200	29/33	701.06	100.05	704.87	100.52	2.68
400	32/33	700.72	100.02	704.42	100.46	3.76
800	33/33	700.64	100.00	703.85	100.38	6.21
1000	33/33	700.64	100.00	703.52	100.33	7.35

The ILS algorithm found new best solutions for two instances in benchmark set TV (instance Eil_B101_66 and Eil_A101_80) when executing 400 iterations or more. These two new solutions are shown in Appendix A.[4] When executing 1000 iterations, the ILS algorithm is able to find the best known solutions to all the instances in both benchmark sets. The solutions obtained by the ILS algorithm (when executing 1000 iterations) are shown in Appendix B. A bold value for the cost of the solution indicates that it is lower than the

[4] The sequence of visited customers is shown for every route that composes the solution, along with the routing costs and the loads transported by each vehicle. The backhaul customers are indicated using bold values.

cost of the previous best solution known for the instance.

6. Comparison of the ILS with other metaheuristic algorithms

In this section, we compare the performance of the ILS to that of other state-of-the-art metaheuristics for solving the VRPB. The algorithms compared are **BTS**: Brandão (2006) tabu search, **LNS**: Røpke and Pisinger (2006) large neighborhood search, **RTS-AMP**: Wassan (2007) reactive adaptative memory programming search, **MACS**: Gajpal and Abad (2009) multi-ant colony system, **RPA**: Zachariadis and Kiranoudis (2012) route promise algorithm and **ILS**: the proposed iterated local search.

For a fair comparison, we only take into account results reported for experiments carried out under the following conditions:

- A limited number of runs are executed by the algorithm for each instance in the benchmark sets.
- The execution time is bounded.
- The parameters of the algorithm are fixed or the way they are calculated remains the same for every execution.

The paper corresponding to the MACS algorithm (Gajpal & Abad, 2009) additionally reports the best solutions (for the instances in the benchmark sets) found during the overall research activity. However, since neither the conditions under which the algorithm was tested nor the values of the parameters are specified, these results are not considered for comparison. Other papers report results corresponding to several versions of the algorithm. In this sense, the BTS results compared here are the ones reported for the *K_tree_r* version of the algorithm. Similarly, the LNS results are the ones reported for the *6R-no learning* configuration and the ILS results are those obtained by executing 400 iterations (ILS-400) and 1000 iterations (ILS-1000) with the parameter values shown in Table 3. Additionally, each paper reports the solutions obtained using different conditions for their experiments. The conditions for each algorithm are the following:

- RTS-AMP: reports the best solution out of 5 runs and the corresponding execution time.
- LNS: reports the best solution out of 10 runs and the average execution time.
- BTS: reports the best solution out of 5 runs and the corresponding execution time.
- MACS: reports the best solution out of 8 runs and the average execution time.
- RPA: reports the best solution out of 10 runs and the average execution time (until the best known solution is found by the algorithm).
- ILS: reports the best solution out of 10 executions and the average execution time.

The computing time comparison is not straightforward due to the different computers used to execute the different algorithms. A rough performance comparison can be done using the megaflops (millions of floating-point operations per second) measured by executing a benchmark program for each computer (Gajpal & Abad, 2009). Table 5 shows the specifications of the computers used in each paper along with the corresponding megaflops values reported by Dongarra (2006). Since the 2.93 gigahertz Intel Core i7 is not included in Dongarra (2006), we executed the same benchmark program to estimate the number of megaflops.[5] The largest number of megaflops obtained was 1598. Therefore, this is the value used to perform the comparison. The same problem occurred for the 1.66 gigahertz Intel Core 2 Duo used by Zachariadis and Kiranoudis (2012). Since we did not have access to the same kind of computer, we executed the benchmark set on a 2.13 gigahertz Intel Core 2 Duo and scaled the number of megaflops relative to the speed of both processors. The last column in Table 5 contains the time scaling factors for all the computers relative to the 1598 megaflops reached by our computer.

Table 5
Processors used to execute the algorithms along with the time scaling factor relative to the 1598 megaflops reached by the 2.93 gigahertz Intel Core i7 processor.

Algorithm	Processor	Megaflops	Time factor
RTS-AMP	50 megahertz Sun Sparc1000	10	0.006
BTS	500 megahertz Pentium III	72.5	0.045
LNS	1.5 gigahertz Pentium IV	326	0.204
MACS	2.4 gigahertz Intel Xeon	884	0.553
RPA	1.66 gigahertz Intel Core 2 Duo	857	0.536
ILS	2.93 gigahertz Intel Core i7	1598	1

Table 6
Performance comparison of the algorithms and their scaled execution times.

Algorithm	Num. best sols.	Avg. best sol. cost	Avg. sol. cost	Avg. scaled time (seconds)
Set GJB				
RTS-AMP	40/62	290981.80	–	11.01
BTS	39/62	291160.00	291305.70	36.67
LNS	50/62	291014.70	291823.34	14.48
MACS	46/62	290655.29	**290920.90**	37.35
RPA	**62/62**	**290576.06**	291927.72	35.08
ILS-400	58/62	290593.84	291332.41	14.31
ILS-1000	**62/62**	**290576.21**	291170.16	22.89
Set TV				
RTS-AMP	21/33	706.40	–	4.33
BTS	25/33	702.20	702.50	13.36
LNS	26/33	701.18	704.50	8.54
MACS	27/33	701.48	**702.30**	14.17
RPA[a]	–	–	–	–
ILS-400	32/33	700.72	704.42	3.83
ILS-1000	**33/33**	**700.64**	703.52	7.35

[a] The performance of the RPA has not been tested using the benchmark set TV.

A performance comparison of the algorithms is shown in Table 6. The metrics used for comparison are the number of best known solutions found, the average cost of the best solutions found and the average cost of the solutions. The scaled execution time is also shown for each algorithm; this scaled value represents the approximate time that would have been consumed if the 2.93 gigahertz Intel Core i7 had executed the algorithm. The bold values correspond to the best value for each performance metric. It can be observed that the proposed ILS algorithm is very competitive in comparison to the other algorithms. The ILS-400 shows a better performance (with respect to two of the three performance metrics) than that shown by most of the algorithms. It finds a larger number of best known solutions and has a lower average cost of the best solutions found than the other algorithms, except for the RPA. The MACS algorithm shows a better performance in terms of the average cost of all the solutions obtained. However, its execution time is considerably larger than the one consumed by the ILS-400. For the set *GJB*, both ILS-1000 and RPA are able to find the best known solutions to all the instances in the benchmark set. However, the ILS-1000 shows a better performance in terms of the average cost of all the

[5] The program executed was the Netlib C version of the LINPACK benchmark program that estimates the number of megaflops by solving a 100×100 system of equations. This code is available at http://www.netlibe.org/benchmark.

solutions obtained and requires a considerably shorter execution time.

It is important to point out that the RTS-AMP, LNS, BTS MACS and RPA are all considerably more complex than the ILS algorithm proposed. Together with the excellent performance of the ILS algorithm, these are major arguments in favor of our approach. The main concept behind the ILS is simple and yet very effective. This suggests that the published algorithms are unnecessarily complex.

7. Conclusions

In this paper, we introduce a simple iterated local search algorithm to solve the VRPB. The main component of this algorithm is an oscillating local search heuristic that has two important features. The first feature is the ability to explore a broad neighborhood structure (composed of four different neighborhoods) at each iteration. We implement a mechanism for an efficient update of the set of neighboring solutions that substantially reduces the execution time of the algorithm. The second feature is the ability to explore solutions that violate the capacity constraint of the problem. The OLS heuristic embedded in the ILS algorithm performs constant transitions between feasible and infeasible portions of the solution space. These transitions are regulated by a dynamic adjustment of a penalty applied to the cost of infeasible solutions. The development of efficient mechanisms to handle the violation of other constraints through the penalization of the cost function would be a useful topic for future research.

We carried out an extensive statistical study of the ILS algorithm. The results obtained show that there are two important components: the ability of the OLS heuristic to oscillate between feasible and infeasible portions of the solution space and the size of the perturbation. The results obtained also allowed us to identify the values of the parameters that yield the best algorithm performance. We compare the ILS algorithm to the best performing algorithms in the literature using two benchmark sets of instances. Despite the fact that the ILS is considerably simpler than the other algorithms compared, it shows a very competitive performance. The ILS algorithm is able to find the best known solutions to all the instances in both benchmark sets in a considerably shorter execution time. Additionally, new best solutions have been found for two instances in one of the benchmark sets. A key result of our work is that, by improving the performance of the LS heuristic, it is possible to develop faster algorithms with simpler components without compromising the quality of the solutions obtained.

Acknowledgments

We acknowledge the financial support of the Flemish Fund for Scientific Research (FWO). We also thank the reviewers for their valuable comments that allowed us to improve the quality of this paper.

Appendix A. New best solutions

See Tables 7 and 8.

Appendix B. Cost of the solutions obtained by the ILS for each benchmark set

See Tables 9 and 10.

Table 7
New best solution for the instance Eil_A101_80.

Route	Cost	Load (linehaul)	Load (backhaul)
0 43 17 59 58 60 18 33 19 54 32 4 45 **95 85 91 96** 0	149	200	32
0 47 2 46 70 78 74 48 75 80 77 5 76 11 0	70	195	0
0 23 21 10 44 20 24 55 61 62 3 64 63 28 **87 93 84 86 98 82 94 90** 0	192	198	95
0 42 6 66 39 38 16 9 50 71 25 56 22 0	92	199	0
0 1 41 27 65 8 57 53 26 51 52 40 29 37 7 **89 92 81 97 100 99** 0	198	200	123
0 72 15 67 68 14 49 13 69 31 12 36 73 79 30 34 35 **83 88** 0	155	199	17
$l = 80, b = 20, m = 6, Q = 200$	856		

Table 8
New best solution for the instance Eil_B101_66.

Route	Cost	Load (linehaul)	Load (backhaul)
0 19 18 51 52 46 54 20 **75 85 71** 0	88	111	40
0 63 64 2 28 15 50 49 39 0	71	112	0
0 1 34 53 23 24 44 48 14 **89 70 84 78 94 93 68** 0	162	112	104
0 47 22 7 8 13 33 43 **88 97 77 90 76** 0	141	112	56
0 36 27 3 37 17 45 16 38 **80 92 91 74** 0	119	110	85
0 62 66 67 61 11 30 26 10 29 **72 86 81 96** 0	118	110	46
0 9 65 40 25 57 **98 100 99 69** 0	53	112	45
0 35 21 59 42 5 55 32 **79 83 73** 0	109	112	53
0 60 4 41 58 12 56 6 31 **82 95 87** 0	122	112	26
$l = 67, b = 33, m = 9, Q = 112$	983		

Table 9
Cost of the solutions obtained for instances in set GJB by the ILS algorithm with 1000 iterations.

Instance	Best known sol. cost	Obtained sol. cost	Instance	Best known sol. cost	Obtained sol. cost
A1	229885.65	229885.65	I1	350245.28	350245.28
A2	180119.21	180119.21	I2	309943.84	309943.84
A3	163405.38	163405.38	I3	294507.38	294507.38
A4	155796.41	155796.41	I4	295988.45	295988.45
B1	239080.16	239080.16	I5	301236.01	301236.01
B2	198047.77	198047.77	J1	335006.68	335006.68
B3	169372.29	169372.29	J2	310417.21	310417.21
C1	250556.77	250556.77	J3	279219.21	279219.21
C2	215020.23	215020.23	J4	296533.16	296533.16
C3	199345.96	199345.96	K1	394071.16	394071.17
C4	195366.63	195366.63	K2	362130.00	362130.00
D1	322530.13	322530.13	K3	365694.08	365694.08
D2	316708.86	316708.86	K4	348949.39	348949.39
D3	239478.63	239478.63	L1	417896.72	417896.72
D4	205831.94	205831.94	L2	401228.81	401228.80
E1	238879.58	238879.58	L3	402677.72	402677.72
E2	212263.11	212263.11	L4	384636.33	384636.33
E3	206659.17	206659.17	L5	387564.55	387564.55
F1	263173.96	263173.96	M1	398593.19	398593.19
F2	265214.16	265214.16	M2	396916.97	396916.97
F3	241120.78	241120.78	M3	375695.41	375695.42
F4	233861.85	233861.85	M4	348140.16	348140.16
G1	306305.40	306305.40	N1	408100.62	408100.62
G2	245440.99	245440.99	N2	408065.44	408065.44
G3	229507.48	229507.48	N3	394337.86	394337.86
G4	232521.25	232521.25	N4	394788.37	394788.36
G5	221730.35	221730.35	N5	373476.31	373476.30
G6	213457.45	213457.45	N6	373758.65	373758.65
H1	268933.06	268933.06	O1	–	478347.72
H2	253365.50	253365.50	O2	–	477256.15
H3	247449.04	247449.04	O3	–	457294.48
H4	250220.77	250220.77	O4	–	458874.87
H5	246121.31	246121.31	O5	–	436974.20
H6	249135.32	249135.32	O6	–	438004.69

Table 10
Cost of the solutions obtained for instances in set TV by the ILS algorithm with 1000 iterations.

Instance	Best known sol. cost	Obtained sol. cost	Instance	Best known sol. cost	Obtained sol. cost
Eil_22_50	371	371	Eil_A76_80	781	781
Eil_22_66	366	366	Eil_B76_50	801	801
Eil_22_80	375	375	Eil_B76_66	873	873
Eil_23_50	682	682	Eil_B76_80	919	919
Eil_23_66	649	649	Eil_C76_50	713	713
Eil_23_80	623	623	Eil_C76_66	734	734
Eil_30_50	501	501	Eil_C76_80	733	733
Eil_30_66	537	537	Eil_D76_50	690	690
Eil_30_80	514	514	Eil_D76_66	715	715
Eil_33_50	738	738	Eil_D76_80	694	694
Eil_33_66	750	750	Eil_A101_50	831	831
Eil_33_80	736	736	Eil_A101_66	846	846
Eil_51_50	559	559	Eil_A101_80	857	**856**
Eil_51_66	548	548	Eil_B101_50	923	923
Eil_51_80	565	565	Eil_B101_66	988	**983**
Eil_A76_50	739	739	Eil_B101_80	1008	1008
Eil_A76_66	768	768			

References

Brandão, J. (2006). A new tabu search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research, 173*, 540–555.

Campbell, A. M., & Savelsbergh, M. (2004). Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science, 38*(3), 369–378.

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

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science, 6*, 80–91.

Deif, I., & Bodin, L. (1984). Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with backhauling. In A. E. Kidder (Ed.), *Proceedings of the Babson conference on software uses in transportation and logistics management* (pp. 75–96). Babson Park, MA: Babson College.

Dongarra, J. (2006). *Performance of various computers using standard linear equation software*. Technical report CS-89-85, University of Tennessee.

Eilon, S., Watson-Gandy, C., & Christofides, N. (1971). Vehicle scheduling. In *Distribution management: Mathematical modelling and practical analysis*. London: Griffin.

Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering, 57*(4), 1472–1483.

Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks, 42*, 109–124.

Gajpal, Y., & Abad, P. L. (2009). Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. *European Journal of Operational Research, 196*, 102–107.

Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics, 65*(1), 223–253.

Goetschalckx, M., & Jacobs-Blecha, C. (1989). The vehicle routing problem with backhauls. *European Journal of Operational Research, 42*, 39–51.

Jacobs-Blecha, C., & Goetschalckx, M. (1992). *The vehicle routing problem with backhauls: Properties and solution algorithms*. Technical report MHRC-TR-88-13, Material Handling Research Center, Georgia Institute of Technology.

Kinderwater, G. A. P., & Savelsbergh, M. W. P. (1997). Vehicle routing: Handling edge exchanges. In E. H. L. Aarts & J. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 337–360). Wiley.

Kytöjoki, J., Nuortio, T., Bräysy, O., & Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers and Operations Research, 34*, 2743–2757.

Lin, S. et al. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal, 44*(10), 2245–2269.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 320–353). Kluwer.

Mingozzi, A., Giorgi, S., & Baldacci, R. (1999). An exact method for the vehicle routing problem with backhauls. *Transportation Science, 33*, 315–329.

Nagata, Y., & Bräysy, O. (2009). Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. *Networks, 54*(4), 205–215.

Osman, I., & Wassan, N. (2002). A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. *Journal of Scheduling, 5*, 263–285.

Potvin, J., & Rousseau, J. (1993). Parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research, 66*, 331–340.

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

Sörensen, K., & Glover, F. (2013). Metaheuristics. In S. Gass & M. Fu (Eds.), *Encyclopedia of operations research and management science* (3rd ed.). London: Springer.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research, 174*(3), 1519–1539.

Toth, P., & Vigo, D. (1997). An exact algorithm for the vehicle routing problem with backhauls. *Transportation Science, 31*, 372–385.

Toth, P., & Vigo, D. (1999). A heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls. *European Journal of Operational Research, 113*, 528–543.

Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing, 15*(4), 333–346.

Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., & Van Oudheusden, D. (2009). Iterated local search for the team orienteering problem with time windows. *Computers and Operations Research, 36*(12), 3281–3290.

Voudouris, C., & Tsang, E. (1999). Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research, 113*(2), 469–499.

Wassan, N. (2007). Reactive tabu adaptive memory programming search for the vehicle routing problem with backhauls. *Journal of the Operational Research Society, 58*, 1630–1641.

Yano, C., Chan, T., Richter, L., Cutler, T., Murty, K., & McGettigan, D. (1987). Vehicle routing at quality stores. *Interfaces, 17*, 52–63.

Zachariadis, E., & Kiranoudis, C. (2010). A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers and Operations Research, 37*(12), 2089–2105.

Zachariadis, E., & Kiranoudis, C. (2012). An effective local search approach for the vehicle routing problem with backhauls. *Expert Systems with Applications, 39*(3), 3174–3184.