DEPARTMENT OF ENGINEERING MANAGEMENT

An Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls

Daniel Palhazi Cuervo, Peter Goos, Kenneth Sörensen & Emely Arráiz

UNIVERSITY OF ANTWERP
Faculty of Applied Economics

Stadscampus
Prinsstraat 13, B.226
BE-2000 Antwerpen
Tel. +32 (0)3 265 40 32
Fax +32 (0)3 265 47 99
www.ua.ac.be/tew

FACULTY OF APPLIED ECONOMICS

DEPARTMENT OF ENGINEERING MANAGEMENT

An Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls

Daniel Palhazi Cuervo, Peter Goos, Kenneth Sörensen & Emely Arráiz

RESEARCH PAPER 2013-010
JUNE 2013

University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium
Research Administration – room B.226
phone: (32) 3 265 40 32
fax: (32) 3 265 47 99
e-mail: joeri.nys@ua.ac.be

The papers can be also found at our website:
www.ua.ac.be/tew (research > working papers) &
www.repec.org/ (Research papers in economics - REPEC)

D/2013/1169/010

An Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls

Daniel Palhazi Cuervo[1,2], Peter Goos[1,3], Kenneth Sörensen[1] and Emely Arráiz[2]

[1]University of Antwerp, Faculty of Applied Economics, Belgium
[2]Simón Bolívar University, Department of Computing and Information Systems, Venezuela
[3]Erasmus University of Rotterdam, Erasmus School of Economics, The Netherlands

June 5, 2013

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 wide neighborhood structure at each iteration. This is efficiently done using an additional data structure that stores information about the set of neighboring solutions. Second, the heuristic performs constant transitions between feasible and infeasible regions 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. Through these results, the paper 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 traveled distance 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 serve 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 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 et al. (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 et al., 1999), a heuristic approach is still necessary to solve larger instances. Recent metaheuristic implementations have proven to be the most successful approach to solve the VRPB in terms of computing time and solution quality (Brandão, 2006; Røpke and Pisinger, 2006; Gajpal and Abad, 2009; Zachariadis and 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 therefore, to identify its key components. In this way, the algorithms can be adapted in order to exploit the building blocks that have proven to be effective, and disregard 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" to describe the heuristic is due to the constant transitions between feasible and infeasible regions 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 and important features of its implementation are explained in Section 5. The computational experiments carried out and their results are discussed in Section 6. The performance of the ILS algorithm is compared to that shown by other metaheuristic algorithms for the VRPB in Section 7. We present our final conclusions in Section 8.

2 Problem definition

The VRPB can be defined using graph theory and second-order logic. Consider a complete undirected graph $G = (C, A)$ where $C = D \cup L \cup B$ 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 $A = \{(i,j)|(i,j) \in C \times C, i \neq j\}$ is the set of arcs that connect the customers. A nonnegative cost $c_{i,j}$ that is associated with every arc $(i,j) \in A$ represents the distance between customers i and 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 $m \geq \max(m_L, m_B)$, where m_L and m_B are the minimum numbers of vehicles needed to separately serve all linehaul and backhaul customers, respectively. A solution to the VRPB comprises a set of m routes. Each route r is defined as a chain of s_r vertices $\langle v_1^r, v_2^r, \ldots, v_{s_r}^r \rangle$, that satisfies the following constraints:

1. Every route starts and finishes at the depot.

$$\forall\, r:\ 1 \leq r \leq m \mid v_1^r = v_{s_r}^r = 0 \tag{1}$$

2. Every vehicle visits at least one linehaul customer.

$$\forall\, r:\ 1 \leq r \leq m \mid (s_r > 2)\ \wedge\ (\exists\, i:\ 1 < i < s_r \mid v_i^r \in L) \tag{2}$$

3. Every customer is visited exactly once.

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

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

$$\forall\, r:\ 1 \leq r \leq m \mid (\forall\, i:\ 1 < i < s_r \mid v_i^r \neq 0) \tag{4}$$

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.

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

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

$$\forall\, r:\ 1 \leq r \leq m \mid (\forall\, i,j:\ 1 \leq i < j \leq s_r \mid (v_j^r \in L \Rightarrow v_i^r \in L \cup D)\ \wedge\ (v_i^r \in B \Rightarrow v_j^r \in B \cup D)) \tag{6}$$

The objective of the VRPB is to find a set of routes that minimizes the sum of the distances traveled by the vehicles. The objective function can be written as:

$$\Sigma\, r:\ 1 \leq r \leq m \mid (\Sigma\, i : 1 \leq i < s_r \mid c_{v_i^r, v_{i+1}^r}) \tag{7}$$

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 and Wright, 1964) for the VRPB. Jacobs-Blecha and Goetschalckx (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 and Glover, to appear). 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 and 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 (Osman and Wassan, 2002; Brandão, 2006; 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) 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 cost of the solution previously explored. The implementation of these diversification mechanisms involve additional

[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.

building blocks and data structures that increase the complexity of the algorithms. Since the LS heuristic is common to most of the existing algorithms, it is reasonable to think that it is the key component to solve the VRPB. We strongly believe that, in order to achieve a better performance, this is the component that should receive most of the attention. By expanding the exploration area and improving the performance of the LS heuristic, it is possible to implement metaheuristics that apply simpler diversification mechanisms without compromising the quality of the solutions obtained. The development of an algorithm with these characteristics is the main goal of this paper.

4 Iterated local search algorithm

The ILS algorithm 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 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; Voudouris and Tsang, 1999; Vansteenwegen et al., 2009). A detailed explanation of the ILS metaheuristic can be found in Lourenço et al. (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.

Algorithm 1: Iterated local search pseudocode

Input: An initial solution S_0

```
1 S_best ← OLS(S_0)
2 for i ← 1 to num_it do
3 |   S_pert ← perturb(S_best)
4 |   S_pert ← OLS(S_pert)
5 |   if traveled_distance(S_pert) < traveled_distance(S_best) then
6 |   |   S_best ← S_pert
7 return S_best
```

4.1 Initial solution

We implemented two alternative procedures for the construction of the initial solution. The first procedure randomly inserts the customers in the solution without considering the capacity constraint of the problem. The resulting initial solution might or might not violate this constraint. The second procedure is a modified version of the greedy insertion heuristic discussed by Potvin and Rousseau (1993). This algorithm iteratively inserts the customer at the position that produces the smallest increase of the solution cost. In order to satisfy the constraint of non-empty routes, the algorithm inserts a randomly selected linehaul customer in

every route as an initialization step. This initial random selection allows the algorithm to generate different initial solutions for the same problem instance. 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 insertion heuristic to produce feasible solutions, or slightly infeasible ones when the order of insertions does not allow to generate a feasible solution.

The capacity constraint is the only constraint that is relaxed during the ILS algorithm. Only solutions that comply with every other VRPB restriction are considered. 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.

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 a new concept in the VRP literature (for instance, see Nagata and Bräysy (2009); Toth and Vigo (2003)). The objective of this approach is to allow temporary excursions to an infeasible region 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 and Savelsbergh, 1997): I) Intra-route and inter-route customer relocation, II) Intra-route and inter-route customers exchange, III) Inter-route crossover and IV) Intra-route 2-opt. At each iteration of the OLS, all four neighborhoods are explored and the solution with the lowest cost is selected to continue the search. The mechanism that allows the efficient generation and update of the set of neighboring solutions is outlined in Section 5. The cost function used to evaluate the quality of a solution is the one described by Brandão (2006):

$$\text{cost}(S) = \text{traveled\_distance}(S) + \alpha \sum_{1 \leq r \leq m} [\text{lh\_excess\_load}(r) + \text{bh\_excess\_load}(r)] \tag{8}$$

The first term of the expression is the total distance traveled by the vehicles. 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. When a locally optimal feasible solution is found during the execution, the algorithm verifies whether the new solution is better than the best feasible solution found so far. If so, the penalty factor is set back to α_0 and another complete cycle of the algorithm is executed. Otherwise, the best feasible solution found is returned. 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. Figure 1 shows the oscillating excess loads for each iteration of the algorithm, along with the values of the total distance traveled by the vehicles[2]. Note the start of a new oscillation every time a better feasible solution is found (when the excess load is equal to zero). Furthermore, note that, when the oscillation starts and the penalty α is low, the heuristic is able to explore solutions with a lower traveled

[2]Figure 1 was obtained by applying the OLS heuristic to solve instance N1 of the benchmark set JBG. See Section 6 for more information about the set of benchmark instances used in this paper.

distance due to the relaxation of the capacity constraint. When the penalty α is increased at the end of the oscillation, the capacity constraint is forced and the total traveled distance of the solutions tend to increase.

Algorithm 2: Oscillating local search heuristic

Input: An initial solution S_0

```
1  S_best ← S_0
2  S_act ← S_0
3  α ← α_0
4  while true do
5      while neighbors_with_better_cost(S_act, α) ≠ ∅ do
6          S_act ← best_neighbor(S_act, α)
7      if is_feasible(S_act) then
8          if traveled_distance(S_act) < traveled_distance(S_best) then
9              S_best ← S_act
10             α ← α_0
11         else
12             return S_best
13     else
14         α ← β × α
```

The initial value α_0 (of the 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 Figure 1. The larger the α_0 value, the smaller the ability of the heuristic to explore infeasible solutions. Similarly, the parameter β determines the speed of the transition from the infeasible to the feasible region of the solution space. It defines the length of the oscillations shown in Figure 1. The larger the β value, the faster the penalization to infeasible solutions is increased and the faster the search is oriented toward feasible solutions. The effect of both parameters on the performance of the ILS algorithm is studied in Section 6.1.

Other oscillating heuristics that have been previously proposed 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 is being explored (Toth and Vigo, 2003; Brandão, 2006). If the heuristic has explored a feasible region for an arbitrary number of iterations, α is decreased in order to stimulate 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. In contrast to this strategy, we propose to update the penalty factor according to the effectiveness of the exploration. The penalty α is increased only if the OLS heuristic gets trapped in a locally optimal solution. The α value is set back to α_0 when a new best feasible solution is found and another complete oscillation is 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. Similar to the generation of the initial solution, the insertions of the customers are performed without considering the capacity constraint of the problem. This leads the perturbation to produce solutions that are usually

infeasible. However, the feasibility of these 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 6.1.

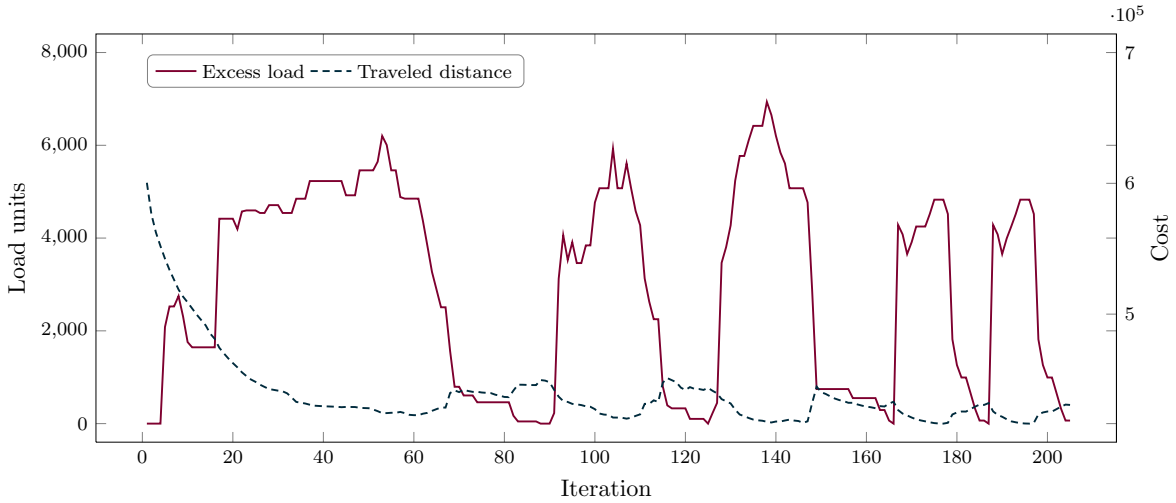

Figure 1: Search pattern of the OLS heuristic in terms of the excess load and the cost of the solutions explored at each iteration.

5 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 et al. (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 explored at each iteration. Taking into account the fact that the neighborhood operators involve one or two routes, the generation of the entire set of neighboring solutions has a quadratic time complexity $\Theta((l+b)^2)$. This complete generation of the neighborhoods is performed only at the first iteration of the OLS heuristic. For each neighboring solution, the following information is stored in an additional data structure: the operator that produces the solution, the routes and the customers involved in the application of the operator, the difference in the traveled distance and the difference in the excess load transported. In the following iterations of the OLS, only the neighboring solutions that involve the routes previously modified (by the application of a neighborhood operator) are updated. Assuming that the customers are (approximately) uniformly distributed over the different routes, the time complexity of the update procedure can be expressed as $\Theta((l+b)/m)$. This linearisation of the time complexity accomplished by the update procedure (in comparison to the full generation) substantially reduces the execution time of

the OLS heuristic.

This approach to reduce the time to compute the neighboring solutions is similar in spirit to the one proposed by Zachariadis and Kiranoudis (2010). The information stored for each neighboring solution is equivalent to the concept of "static move descriptor" that they propose. The main difference lies in the fact that they do not consider the exploration of infeasible solutions. In our implementation, since the cost of a solution involves the excess load carried by the vehicles, it is necessary to store the difference of this value. Moreover, the excess load of a vehicle is dependent on the entire set of customers that are served along the route. For that reason, it is not possible to apply the rules to update the set of neighbouring solutions that they describe. Instead, it is necessary to update all neighbouring solutions that involve the routes that have been modified in the previous iteration. The cost of a solution also depends on the value of the penalty factor α. Since this value is dynamically modified during the execution of the OLS heuristic, it is necessary to explore the entire set in order to find the neighboring solution with the lowest cost. For all these reasons, the update rule that we apply requires more calculations than that used by Zachariadis and Kiranoudis (2010); however, our approach is a generalization that is able to handle the relaxation of constraints imposed on the routes. Other constraints like tour length, time windows or driving hours, can be easily included and handled by our approach.

6 Computational experiments

A set of numerical experiments were performed using two benchmark sets of instances available in the literature. The first set (*set JBG*) was proposed by Jacobs-Blecha and Goetschalckx (1989) and consists of 62 instances where the total number of customers ranges from 25 to 150. The second 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 et al. (1971). Each of these VRP instances was used to construct three new VRPB instances by considering 33%, 50% and 66% of the total number of 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 GHz. Intel Core i7 processor. It is important to point out that the distance between each pair of customers is calculated in different ways for each benchmark set: for the set GJB, the distances 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 obtained by other algorithms.

6.1 Statistical Analysis

In this section, we describe the statistical analysis carried out to better understand the behaviour 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 generation (`random`) or the greedy insertion algorithm (`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 6.2.

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 factor needs to be increased for the first time, it 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.

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

Table 1: Parameters and levels tested.

The algorithm was executed 10 times using each combination of parameter values to solve every instance of the benchmark set JBG, resulting in $2 \times 3 \times 5 \times 6 \times 62 \times 10 = 111600$ executions. For each set of 10 executions, three performance measures were determined for each problem instance: 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 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.

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` $\times$ `pert_size`	0.9990	0.5237	0.8955
`ini_sol` $\times\alpha_0$	0.9303	0.9992	0.9604
`ini_sol` $\times\beta$	0.2811	0.7479	0.8960
`pert_size` $\times\alpha_0$	$<$ **0.0001**	$<$ **0.0001**	$<$ **0.0001**
`pert_size` $\times\beta$	0.9788	0.9799	0.6874
$\alpha_0 \times \beta$	0.9440	0.9951	0.6379

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

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 regions 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 also insensitive to the transition speed from the infeasible to the feasible region of the solution space. What it 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 parameters `pert_size` and α_0 is shown in Figure 2. Observe that the execution time of the algorithm increases proportionally with the perturbation size and the ability of the OLS algorithm 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.

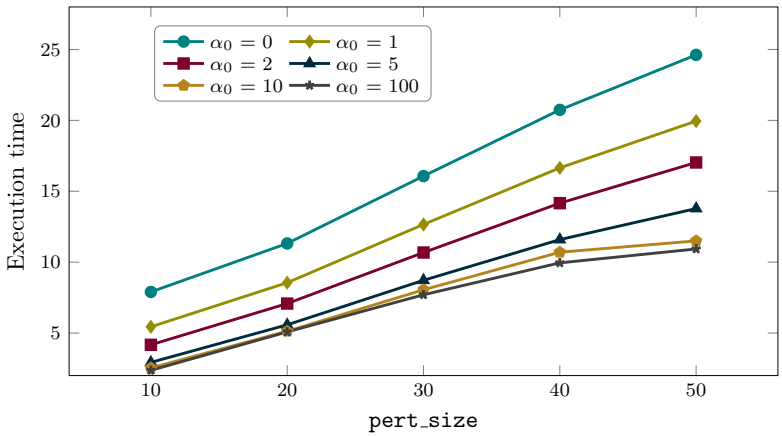

Figure 2: Influence of the perturbation size (`pert_size`) and the initial penalty (α_0) on the average execution time.

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

The large impact of the exploration of infeasible solutions on the algorithm performance deserves a more careful analysis. In Figure 3, it can be observed that the more capable the algorithm is to explore infeasible regions 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 Figure 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 low values of α_0 (between 0 and 10). In contrast, when $\alpha_0 = 100$, the quality of the best solutions found goes down to a considerable

extent. Provided the algorithm can be executed at least 10 times and the perturbation size is equal to 30%, low values of the initial penalty α_0 do not decrease the ability of the algorithm to find good solutions. Nevertheless, high values of α_0 limit the ability to explore infeasible solutions and have a considerably negative impact on the solution quality.

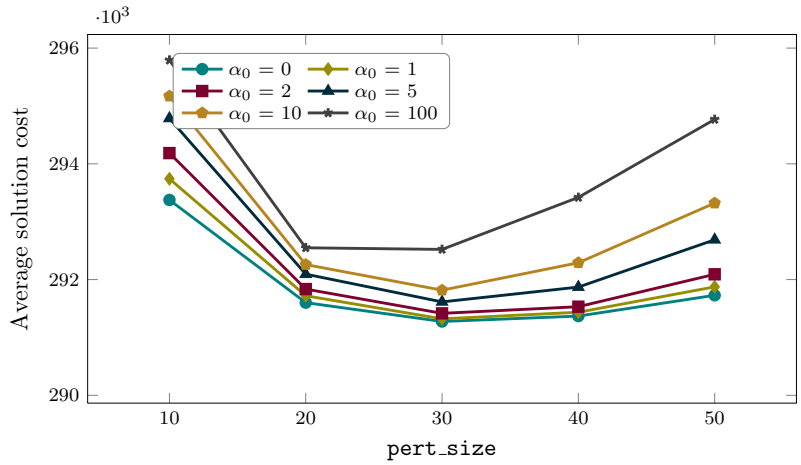

Figure 3: Influence of the perturbation size (`pert_size`) and the initial penalty (α_0) on the average cost of the solutions found.

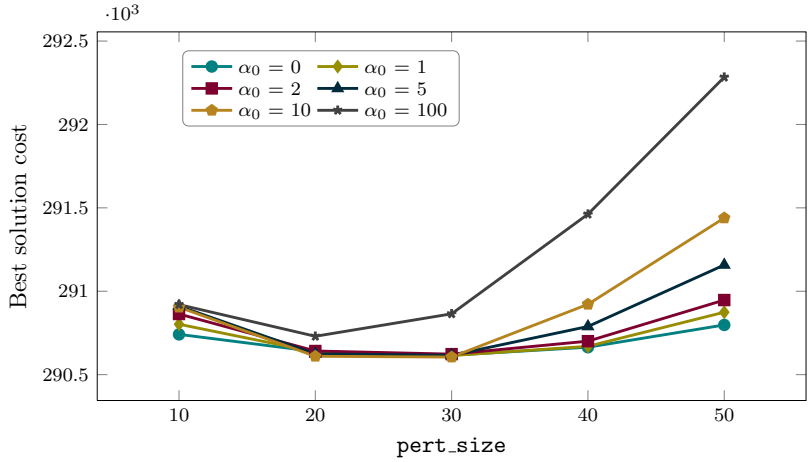

Figure 4: Influence of the perturbation size (`pert_size`) and the initial penalty (α_0) on the average cost of the best solutions found.

6.2 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 solution cost and the average ratio ((solution cost/best known solution cost) $\times$ 100) for the best solution found and the average solution, 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 number of iterations mainly increase the robustness of the algorithm by decreasing the average cost of the solutions obtained.

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

Table 3: Parameters and final values used.

		Best solution		Average solution		
Iter.	Num. best sols.	Avg. cost	Avg. ratio	Avg. cost	Avg. ratio	Avg. time (s)
Set JBG						
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

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

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. 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 A. A bold value for the cost of the solution indicates that it is lower than the cost of the previous best solution known for the instance.

7 ILS vs. State of the art metaheuristic algorithms

In this section, we compare the performance of the ILS to the performance 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 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, this results will not be considered for comparison. Instead, results obtained with experiments carried out under the conditions stated above are considered. 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 the execution of 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).

For our ILS algorithm, the best solution is reported out of 10 executions along with the average execution time of the algorithm.

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 Mflops (millions of floating-point operations per second) measured for each computer (Gajpal and Abad, 2009). Table 5 shows the specifications of the computers used in each paper along with the corresponding Mflops values reported by Dongarra (2006). Since the 2.93 GHz. Intel Core i7 is not included in Dongarra (2006), we executed the same benchmark program to estimate the number of Mflops[3]. The largest number of Mflops obtained was 1598 Mflops. Therefore, this is the value used to perform the comparison. The same situation took place for the 1.66 GHz. Intel Core 2 Duo used by Zachariadis and Kiranoudis (2012). Since we did not have access to the same computer, we executed the benchmark set on a 2.13 GHz. Intel Core 2 Duo and scaled the number of Mflops 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 Mflops reached by our computer.

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

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

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

Table 5: Processors used to execute the algorithms along with the time scaling factor according to the 1598 Mflops reached by the 2.93 GHz. Intel Core i7 processor.

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 GHz. 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 out of 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 its considerably larger than the one consumed by the ILS-400. For the set *JBG*, 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 solutions obtained and requires a considerably shorter execution time. The other algorithms here compared are considerably more complex than the ILS algorithm proposed. This is a major argument in favour of the ILS algorithm; the main concept behind it is simple and yet very effective. This suggests that the published algorithms are unnecessarily complex.

Algorithm	Num. Best sol.	Avg. best sol. cost	Avg. sol. cost	Avg. scaled time (s)
Set JBG				
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[4]	-	-	-	-
ILS-400	32/33	700.72	704.42	3.83
ILS-1000	**33/33**	**700.64**	703.52	7.35

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

[4]The performance of the RPA is not tested using the benchmark set TV.

8 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 wide 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 path 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. We have shown 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.

9 Acknowledgement

We acknowledge the financial support of the Flemish Fund for Scientific Research (FWO).

References

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

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

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

I. Deif and L. Bodin. Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with backhauling. In A. E. Kidder, editor, *Proceedings of the Babson Conference on Software Uses in Transportation and Logistics Management*, pages 75 – 96, Babson Park, MA, 1984.

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

S. Eilon, C. Watson-Gandy, and N. Christofides. Vehicle scheduling. In *Distribution Management: Mathematical Modelling and Practical Analysis*, chapter 9. Griffin, London, 1971.

B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 2009.

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

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

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

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

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

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

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

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

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

I. Osman and N. Wassan. A reactive tabu search meta-heuristic for the vehicle routing problem with backhauls. *Journal of Scheduling*, 5:263 – 285, 2002.

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

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

K. Sörensen and F. Glover. Metaheuristics. In *Encyclopedia of Operations Research and Management Science*. Springer, Berlin, to appear.

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

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

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

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

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

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

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

C. Yano, T. Chan, L. Richter, T. Cutler, K. Murty, and D. McGettigan. Vehicle routing at Quality Stores. *Interfaces*, 17:52 – 63, 1987.

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

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

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

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

Table 7: 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
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			

Table 8: Cost of the solutions obtained for instances in set TV by the ILS algorithm with 1000 iterations.