A deterministic iterated local search algorithm for the vehicle routing problem with backhauls

Article *in* Top · October 2015

DOI: 10.1007/s11750-015-0404-x

1 author:

José Brandão
University of Minho
17 PUBLICATIONS **1,597** CITATIONS

SEE PROFILE

The final version of this article has been published in:
Brandão, J. (2016). A Deterministic Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls. TOP 24: 445–465. doi: 10.1007/s11750-015-0404-x.

A Deterministic Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls

José Brandão

Departamento de Gestão, Escola de Economia e Gestão, Universidade do Minho, Largo do Paço, 4704 – 553, Braga, Portugal.
CEMAPRE, ISEG, University of Lisbon, Portugal.
E-mail: sbrandao@eeg.uminho.pt

Abstract

The vehicle routing problem with backhauls is a variant of the classical capacitated vehicle routing problem. The difference is that it contains two distinct sets of customers: those who receive goods from the depot, who are called linehauls, and those who send goods to the depot, who are referred to as backhauls. In this paper we describe a new deterministic iterated local search algorithm, which is tested using a large number of benchmark problems chosen from the literature. These computational tests have proven that this algorithm competes with best known algorithms in terms of the quality of the solutions and at the same time, it is simpler and faster.

Keywords: Backhauls; iterated local search; linehauls; logistics; vehicle routing.

1 Introduction

The classical capacitated *vehicle routing problem* (VRP) is a mathematical problem regarding the activity of distributing goods from a central depot to a set of customers, who are geographically dispersed, using a fleet of vehicles of the same capacity. Each vehicle starts off from the depot, visits a set of customers following a given sequence, and then returns to the depot. The trip performed by a vehicle is called a route. Each vehicle travels on only one route and each customer belongs to just one route. The objective is to minimise the total cost of travel.

The VRP, and its extensions, represents a very important practical distribution problem. More precisely, the economic importance of the VRP results from the fact that, according to Waters (1990), transport is the largest element in distribution costs and typically accounts for 5% of the product value. Among the many other authors who have studied and highlighted the relevance of transportation costs, Hoff et al. (2010) deserve a particular mention. The resolution of this problem presents an enormous challenge, and,

consequently, this is another reason why it has been one of most studied combinatorial optimization problems over the last fifty years.

The vehicle routing problem with backhauls (VRPB) is a variant of the VRP, whose distinguishing feature is the existence of two disjointed sets of customers: those that require the delivery of goods, who are called linehauls, and those that require the collection of goods, who are called backhauls.

In practice, there are many situations which correspond to the VRPB, although sometimes a better representation of the real problem requires the consideration of additional constraints, such as time windows, or different types of vehicles. A practical example of this problem is found in drinks distribution, where the delivery of full bottles is followed by the collection of empty ones at the end of the route. Casco et al. (1988) provide several examples of real-world applications.

Amongst the problems that make up delivery and pickup, we can distinguish the following three categories, based on the type of demand of customers and on the order in which they must be served:

i) Each customer can require delivery, or pickup, or both;

ii) Pickups can be made before deliveries;

iii) In a route, the linehaul or delivery customers are served first, followed by backhaul or pickup customers. No route can contain only backhauls, but routes can have only linehauls. This condition has been assumed by all the authors that studied this type of problem, as, in general, the total amount of deliveries is higher than the total amount of pickups.

Category *iii)* is the one studied in this paper, and, according to the literature, it is the most common practice, as it makes the tasks of unloading and loading vehicles much easier and faster, especially if they are rear-loaded.

The reader can find out more from a complete survey of all these types of problems in Parragh et al. (2008). This survey presents a classification of these problems and mentions most of the published algorithms used to solve them. Other similar surveys have been made by Toth and Vigo (2002), Berbeglia et al. (2007), Battarra et al. (2014) and Irnich et al. (2014).

The VRPB, with the same characteristics as described here, have been studied by many authors, including the following: Deif and Bodin (1984), Goetschalckx and Jacobs-Blecha (1989, 1993), Toth and Vigo (1997, 1999), Mingozzi et al. (1999), Osman and Wassan (2002), Brandão (2006), Ropke and Pisinger (2006), Wassan (2007), Gajpal and Abad (2009) and Zachariadis and Kiranoudis (2012). More recently, during the course of this research, two other algorithms have been published: one by Vidal et al. (2014) and another by Cuervo et al. (2014). Several other authors have studied the VRPB with some side constraints. This is the case, for example, of Thangiah et al. (1996), Reimann et al. (2002) and Ropke and Pisinger (2006), who investigated the VRPB with time windows, and Salhi et al. (2013), who researched the VRPB with a heterogeneous fleet.

Deif and Bodin (1984) adapted the Clarke and Wright (1964) savings algorithm, which was originally conceived for the VRP, for solving the VRPB problem. The algorithms of Goetschalckx and Jacobs-Blecha (1989, 1993) were also based on classical heuristics and they achieved a better performance than the algorithms that had been published before. The heuristic of Toth and Vigo (1999), published some years later, outperformed all the previous approximate algorithms that had been published. This heuristic was mathematically-based. It used the results from a Lagrangian relaxation for the formation of clusters of customers, which were applied using their method for exact resolution (Toth and Vigo 1997).

Toth and Vigo (1997) and Mingozzi et al. (1999) proposed two exact methods for the VRPB, which were able to solve problems exactly in cases of up to 100 customers.

All the other articles describe metaheuristics, which are among the best published algorithms for the VRPB. The algorithms of Osman and Wassan (2002), Brandão (2006) and Wassan (2007) are based on tabu search. The algorithm developed by Ropke and Pisinger (2006) applies large neighbourhood search. Gajpal and Abad (2009) used a multi-ant colony system. Zachariadis and Kiranoudis (2012) proposed an algorithm whose main characteristics are the neighbourhood which results from exchanging customer sequences of variable length and the use of the concept of promises to promote diversification. The algorithm of Vidal et al. (2014) performs a hybrid genetic search and, through a proper use of its modular components, it is capable of solving many VRP variants, including the VRPB. Cuervo et al. (2014) developed an iterated local search algorithm that allows the exploration of infeasible solutions during the search process. In terms of performance, the common features of these metaheuristics are the following: they yield much better solutions than classical heuristics, although they require substantially more computing time; and they are able to solve much larger problems than those that are possible to solve with exact methods.

The VRPB can be defined more precisely using graph theory, as follows. Let $G = (V, A)$ be a directed network where $V = \{0\} \cup L \cup B$ is a set of vertices and $A = \{(i, j): i, j \in V\}$ is the set of arcs. The subsets $L = \{1, 2, \ldots, n\}$ and $B = \{n + 1, n + 2, n + m\}$, represent the linehaul customers and the backhaul customers, respectively, and 0 represents the depot. The total number of customers is represented by N. To each arc $(i, j) \in A$ is associated a distance d_{ij}, with $d_{ij} = d_{ji}$ for each $i, j \in V$, except if $j \in B$ and $i \in L$; $d_{ji} = d_{0j} = +\infty$, for each $j \in B$ and $i \in L$. In the depot there are M identical vehicles with a capacity Q. Each customer $i \in L \cup B$ requires a given quantity q_i to be delivered ($i \in L$) or collected ($i \in B$). The number of vehicles is defined as $M \geq \max\{M_L, M_B\}$, where M_L and M_B are the minimum number of vehicles needed to serve all linehaul and backhaul customers,

respectively. The value of M_L (M_B) can be obtained by considering the demand of the linehauls (backhauls) and by solving the corresponding *bin packing problem* with a bin of capacity Q.

The solution of the VRPB consists of finding a minimum cost set of routes, based on the following:

i) In a route, due to constraints of precedence, linehauls are served before backhauls and no route can only contain backhaul customers.

ii) The total demand of all the customers on a route, considering linehauls and backhauls separately, cannot exceed the capacity of the vehicle. Each customer is visited just once, and all their distribution requirements are satisfied.

It is worth remembering that the number of routes, M, is fixed in advance, and it is not necessarily the minimum required. Therefore, it may be that an optimal solution with M routes has a higher cost than another solution with a number of routes lower (or greater) than M. The only reason for using this assumption is that it has been made by those other authors who studied this problem, namely, it is present in all the algorithms used in this paper for the sake of comparison.

The VRP is *NP-hard* in the strong sense as was proved, for example, by Lenstra and Rinnoy Kan (1981). Therefore, the VRPB is also NP-hard, since, if there are no backhaul customers, the VRPB is just a VRP. Consequently, in general, i.e., excluding very specific cases, there are no exact algorithms capable of finding the optimal solution if the instances of the problem are large. In fact, so far as we know, only VRPB examples with up to one hundred customers have been solved to optimality.

The remainder of this paper is organised as follows. In Section 2, we describe the general characteristics of the iterated local search. In Section 3, the methods for generating the initial solutions are defined. In Section 4, we present our iterated local search algorithm. In Section 5, we present the computational experiments and compare the quality

of our algorithm with the best algorithms published, and in the final section, we draw the main conclusions.

2 The iterated local search

The iterated local search (ILS) is explained in detail in Lourenço et al. (2003). The most relevant aspects of this metaheuristic are the way used to go beyond a local optimum, s^*, and the way of exploring the solution space to discover successive local optima. This dynamic search is performed through a mechanism called *perturbation*, which transforms a given solution into another one, together with a way of choosing between two solutions for continuing with the search process, which is called *acceptance criterion*. This means the following. Given s^*, a perturbation is applied, giving it a new solution, s_p. Then, s_p is improved by a given heuristic, which generates a new local optimum, s_p^*. If s_p^* passes an acceptance test, then the next perturbation is applied to s_p^*; otherwise, it is applied to s^*; and so forth, until a stopping criterion is reached. The cost of a solution s is $f(s)$, and the best solution discovered so far is denoted by s_b. Therefore, the iterated local search works as follows:

1. Generate an initial solution s_0; set $s_b = s_0$.
2. Apply local search to s_0, generating the solution s^*; if $f(s^*) < f(s_b)$, set $s_b = s^*$.
3. While the termination criterion is not satisfied, do the following:
    3.1. Apply a perturbation to s^* to obtain s_p;
    3.2. Apply local search to s_p to obtain a local optimum s_p^*;
    3.3. If s_p^* satisfies the acceptance criterion, make $s^* = s_p^*$;
    3.4. If $f(s_p^*) < f(s_b)$, set $s_b = s_p^*$.

3 Methods for generating the initial solutions

In order to enhance diversification, we applied a multi-start strategy, which was implemented by the use of four different initial solutions. This strategy revealed to be more effective in finding good solutions than increasing the number of iterations. The main goals pursued for the design of methods for determining the initial solutions were the following: *i)* to be simple and fast; *ii)* to produce solutions with *M* routes; *iii)* to generate solutions with good properties, i.e., located in promising regions of the solution space. Two of the initial solutions are obtained by parallel constructive methods, while the other two are created by sequential ones. The first two methods are more effective in finding solutions with *M* routes, and in fact, this happened with all the test problems, whilst the other two are less effective and, in a few cases, that number was greater than *M*. In all the methods, linehauls are first inserted in the routes and, then the backhauls.

3.1 Method 1 – parallel nearest neighbour

Before starting to construct routes, the customers of each set are sequenced by non-increasing order of demand, forming two ordered lists. The customers are placed in a route following their order in these lists, starting with the linehauls. Initially, *M* routes are created, each one containing one customer from the linehauls list. Then un-routed customers are inserted in one of these routes, until they are all routed. A new customer is inserted according to its nearest vertex (customer or depot), bearing in mind all the vertices that already feature in the routes, without exceeding the capacity of the vehicle.

If, owing to capacity constraints, a customer cannot be inserted in any of the existing routes, then a new route has to be started with that customer, if it is a linehaul. However, if this un-routed customer is a backhaul, the linehaul closest to the depot has to be removed first from the existing routes that contain more than one linehaul. This linehaul is used for starting a new route and then the backhaul is inserted, as no route can contain only backhauls, and these must be served after the linehauls of the route. This procedure is applied in the same manner to all the other methods. However, it is worth saying that,

although it has been programmed, this procedure was not required by Methods 1 and 2 for any of the test problems, as all the initial solutions contained *M* routes.

3.2 Method 2 – parallel insert

This method operates in exactly the same way as Method 1, except that the new customer is inserted where it has less impact on the total distance travelled.

3.3 Method 3 – sequential insert

Each route is started with the linehaul nearest to the depot. Then, vehicle capacity permitting, the un-routed linehaul which impacts the distance of the route the least is inserted, taking into consideration all the possible positions of insertion along the route. When no more linehauls can be inserted, the un-routed backhauls are inserted, following the same process. If the route is full, then a new route is constructed in the same way, and the process is repeated until all customers are routed. If the final solution contains less than *M* routes, or if there are backhauls remaining un-routed, then additional routes are created, using the procedure explained in the last paragraph of Section 3.1.

3.4 Method 4 – open VRP

In an open VRP, vehicles are not required to return to the depot at the end of the route. This means that if a VRPB route contains linehauls and backhauls, these two paths correspond to two open VRP routes. Therefore, the rationale behind this method is to solve two distinct open VRPs, one for linehauls, and the other for backhauls and, at the end, link each path of the linehaul solution with another path of the backhaul solution, until no backhaul path remains unlinked.

Each open VRP is solved using a sequential nearest neighbour heuristic. Each route is started with the un-routed customer nearest to the depot, followed by adding the nearest customer to this one, providing that its cargo does not exceed the free capacity in the vehicle, and so on. When no further customer can be added to the route under construction, a new route is started, or, if all customers are already routed, the process stops.

After solving the two open VRPs, each backhaul path is linked to the end of one of the linehaul paths. For each group of two paths (one linehaul and one backhaul) there are four different ways of connecting their extremes. All these combinations are considered and, at the end, the least cost link is chosen. The two paths linked (one linehaul and another backhaul) form a route of the solution and they are removed from the sets of paths available. This step is repeated until either the set of linehaul or the set of backhaul paths is empty. In the end, each linehaul path remaining is connected to the depot, creating a closed route. If this VRPB solution contains less than *M* routes, or routes containing only backhaul customers, the procedure defined in Section 3.1 is applied.

4 The iterated local search algorithm

This section describes our iterated local search algorithm (ILSA), following the general principles described in Section 2. First, we present the main features of the ILSA and then the whole operational structure is described. The objective is to minimise the total distance travelled, *f(s)*. Therefore, the moves are evaluated by the following equation:

$$f(\mathrm{s}) = \sum_{i=1}^{K} d(r_i). \qquad (1)$$

Where $d(r_i)$ is the distance travelled in the route r_i and K is the number of routes of *s*. K is equal to M, except if the method for creating the initial solution is unable to find a solution with M routes and this is the only constraint that might not be respected along the search. If $K > M$ an *elimination procedure* is applied at the beginning of every iteration. This procedure consists of trying to remove a customer from a route which only has one customer (if no routes exist with only one customer, then the execution of this procedure is halted) and inserting it in any other route that is feasible, independent of the cost of insertion. Although it is very simple, this procedure is usually effective. Therefore, there was no need to apply a more sophisticated procedure, as the solutions generated by

Methods 1 and 2 always have $K = M$, and the same happens with most of the solutions produced by the other two methods.

4.1 Local search

Given a solution *s*, the neighbourhood of *s*, *N(s)*, is the set of all solutions that can be reached from *s*, through a given kind of modification, which is called the *neighbourhood structure*. The purpose of the local search is to find the local optimum, $f(s^*)$, with $s^* \in N(s)$. This is equivalent of saying, in a minimisation problem, that:

$$f(s^*) \leq f(s'), \forall s' \in N(s). \quad (2)$$

Therefore, the local search consists of generating all the solutions of *N(s)* and choosing the best one. It is worth noting that many $s' \in N(s)$ solutions may exist, such as $f(s') \leq f(s)$, or it may come to pass that the local minimum is *s*, i.e., $s = s^*$. In practice, a complete exploration of *N(s)* requires a systematic approach, which may be rather time-consuming, even for a simple neighbourhood structure. In order to reduce computing time, another strategy can be used, which consists of accepting the first solution better than *s* and pursuing the search from this new solution. Many other alternatives can be chosen in-between these *overall best* and *first best* strategies.

The neighbourhood structures applied in the ILSA were the following: *i) insertion*, *ii) swap*, *iii) cross over*, *iv) interchange (2, 0)*, *v) interchange (2, 1)*, *vi) intra swap*, and *vii) shift*.

The first five kinds of structures consist of moves between the routes of *s*, whilst the other two are performed inside each route of *s*. The moves insertion, cross over and interchange (2, 0), may reduce the number of routes, but this is never permitted. The overall best strategy was applied to interchange (2, 0) and interchange (2, 1) moves, whilst the first best strategy was applied to cross over, intra swap and shift moves. On the other hand, both strategies – overall best and first best – have been used with insertion and swap moves at different steps of the ILSA.

In the *insertion* move, a customer i is removed from its current route and a trial insertion is made in any one of the other routes between the vertices where its insertion cost is lower. This operation is applied to every $i \in V \setminus \{0\}$.

A *swap* move consists of exchanging two customers of the same type (both linehauls or both backhauls) belonging to two different routes. Given a customer $i \in r_i$ (route r_i) and a customer $j \in r_j$, first i and j are removed from their routes and then they are inserted, respectively, in r_j and r_i, in the best feasible position. All the pairs of customers $(i, j) \in L$ and $(i, j) \in B$ are considered.

The *cross over* between two routes r_i and r_j consists of deleting one arc $(i, j) \in r_i$ and another arc $(k, l) \in r_j$ and replacing them by arcs (i, l) and (k, j). The customers i, j, k and l do not need to be of the same type but, if they are of a different type, only combinations that keep the solution feasible in relation to the precedence constraints are allowed. The potential moves are constituted by all the combinations obtained, taking into consideration all pairs of routes of s and all arcs (i, l) and (k, j) for each pair of routes.

An *interchange (2, 0)* move consists of removing two consecutive customers of the same type from one route and inserting them into a different route. These two customers must remain connected, but their order can be reversed, if that reduces the cost of insertion.

The *interchange (2, 1)* consists of a direct exchange of two consecutive customers, i and j, from one route, with a customer k of a different route. Customers i and j go to the position of k and this takes the position of i and j, but the order of i and j can be reversed. Customers i, j and k must be of the same type.

The *intra swap* move exchanges two customers of the same type inside a route.

The *shift* move consists of moving one customer backward or forward inside the same route. First, the backward moves are evaluated and, then, the forward ones.

4.2 Perturbation

The purpose of perturbing a local optimum *s**, i.e., making changes in *s**, is to escape from the region of *s** and to go to a different region of the solution space, in order to try to find other better local optima, using the local search. The transformation of *s** cannot be either too strong, neither too weak. If it is too strong, then the benefit of departing from *s** is lost, i.e., it will correspond to a random restart. Otherwise, if the transformation is too weak, the local search will reverse it, going back to *s**. In the VRPB, the strength of a perturbation may be measured by the number of arcs in s_p* that are different from *s**, plus the number of customers in different routes of both solutions.

Since the search will pursue from s_p*, another desirable characteristic of s_p* is that it is located in the vicinity of good solutions. We tried precisely to achieve this goal, by applying a perturbation which is based on the following idea: that the current local minimum would be better if the largest arcs could be removed and replaced by smaller ones. Therefore, the perturbation used in the ILSA consists of removing a given number of the largest arcs and replacing them by other ones (not necessarily smaller), using the procedure defined below. Afterwards, the local search performed in s_p*, will try to find a local solution with a better combination of arcs, and consequently, a better solution.

The perturbation is defined by the following procedure. Firstly, the sum of the two arcs incident to each customer is calculated. Secondly, a list is created which contains the customers placed in order by non-increasing order of that sum. Then, following the order of the list, the swaps in *s* between the members of this list are performed. A swap can be executed if the two customers are from different routes, if they are of the same type and if all the constraints are respected. Let us suppose that the list is [x_1, x_2, ..., x_i, x_j, ..., x_N]. First, x_1 and x_2 are swapped, if the three conditions defined above are satisfied. Otherwise, the swap (x_1, x_3) is tried, and so on. If, for example, the swap (x_1, x_3) is executed, then the next trial swap is (x_2, x_3). Or suppose that (x_i, x_N) were swapped, then the next trial swap is (x_j,

x_{j+1}). This means that after every swap move, the first element of the next trial swap is shifted one position to the right, and that each customer may be swapped more than once, with different customers.

Since the descent swap move is also included in the local search, there is some risk of immediately reversing the moves performed in the perturbation. In order to try to avoid this, other descent local search moves are applied immediately after the perturbation, and before the swap move is applied again, as can be observed in the algorithm described below.

4.3 Acceptance criterion

As was explained at the beginning of Section 2, the *acceptance criterion* determines the solution to which the perturbation is applied in the next iteration. There are three basic alternatives: *i)* always choose the best known solution (s_b); *ii)* always choose the current solution (s); *iii)* choose s, if $f(s) < \beta f(s_b)$ or s_b, otherwise; where $\beta > 1$ is a constant. The application of *i)* means that the search will continue in the neighbourhood of s_b, i.e., it favours intensification, whilst alternative *ii)* promotes the exploration of a different region of the solution space and therefore, increases diversification. On the other hand, the third alternative mixes the properties of the other two, i.e., if a good solution is found in the neighbourhood of the current solution, then the search goes on in this region, if not, the search returns to the neighbourhood of s_b. Therefore, it allows a trade-off between intensification and diversification. This strategy raises the problem of how to find the most appropriate value of β. The best strategy will depend on the whole algorithm.

In the ILSA, the perturbation is always applied to the current solution. This is called a random walk acceptance criterion and this choice has the goal of favouring diversification.

4.4 Pseudocode of the iterated local search algorithm

s_{bg} – global best solution found.
r(s) – number of routes of *s*.
t – counter of the total number of iterations.
t_b – counter of the number of iterations without improving the best solution.
z – limit of the total number of moves in the perturbation.

```
1:  set w = 1
2:  while w ≤ 5 do
3:      if w ≤ 4 then
4:          generate s with Method w, set s_b = s, t_b = 0, t = 1;
5:      if w = 1 then set s_bg = s;
6:      while t_b < 2000 do
7:          do {
8:              if r(s) > M then {Elimination_procedure(s);
9:              if r(s) = M then set s_b = s;}
10:             set d = f(s)
11:             Interchange_2_1(s)
12:             Interchange_2_0(s)
13:             if f(s) < f(s_b) then
14:                 Insertion_overall_best(s)
15:                 Swap_overall_best(s)
16:                 Crossover(s)
17:                 set s_b = s, t_b = 0;
18:             do {
19:                 set d_1 = f(s)
20:                 Insertion_first_best(s)
21:                 Shift(s)
22:                 Intra_swap(s) } while(d_1 > f(s))
23:             Swap_first_best(s) } while(d > f(s))
24:         set z = 0.15N + t mod 9, t_b = t_b + 1, t = t + 1
25:         Perturb(s)
26:     end while
27:     if f(s_bg) > f(s_b) and r(s_b)= M then set s_bg = s_b;
28:     if f(s_bg) > f(s_b) and r(s_b)= M then set s_bg = s_b;
29:     if w = 5 then
30:         set s = s_bg, z = N, t_b = 0, t = 1
31:         Perturb(s);
32: end while
33: return s_bg
```

4.5 Parameters setting

The ILSA contains only two parameters – the perturbation size (*z*), and the limit of the number of iterations, without improving the best solution (*T*). However, only the first

one belongs to the structure of the algorithm, while *T* is only an indirect way of controlling the computing time, but, obviously, it is has influence on the quality of the final solutions.

The perturbation size is the number of moves that are executed in the perturbation. The value of *z* should be understood as a limit because, taking into consideration the method of executing the moves, there is no guarantee *a priori* that they all can be executed, specially if a large number is defined. There is no theoretical way of establishing the best perturbation size, therefore it has to be found experimentally. In order to do this, we chose six test problems, H6, I2, L3, M3, N1 and N5, from set 1 (see Section 5), with *N* between 68 and 150. The ILSA was executed for a large number of iterations (25000), only using Method 1 to generate the initial solution. The number of moves is defined as a function of the size of the problem, plus a constant that depends on the value of the current iteration, *t*, in order to try to avoid cycling, since the perturbation procedure is deterministic. The following five different functions were tested: *i) z = 0.05N + t mod 9*; *ii) z = 0.1N + t mod 9*; *iii) z = 0.15N + t mod 9*; *iv) z = 0.2N + t mod 9* and *v) z = 0.25N + t mod 9*. Table 1 shows the total distance of the best solutions found, total computing time and the iteration in which the last improvement occurred (this is referred to one of the test problems, which is not necessarily the same for all *z* functions).

Table 1
Influence of perturbation size.

z function	***i***	***ii***	***iii***	***iv***	***v***
Total distance	2215399	2148658	2130392	2131332	2133164
CPU (s)	308	422	565	681	763
Last improvement	93	286	2186	17039	23998

The main conclusions that can be taken from Table 1 are the following: *i)* The perturbation size that yields better solutions is *z = 0.15N + t mod 9*; *ii)* Larger perturbations make the execution slower – this happens because steps 7 to 23, except steps 14 to 17, are repeated more times owing to the higher diversity of the solutions explored; *iii)* The fact

that a new best solution can be found after as many as 23,998 iterations, shows that a deterministic perturbation does not prevent diversification.

Steps 29 to 31 of the ILSA consists of starting from the best solution found with the improvement of the four initial solutions, applying a perturbation to it and then repeating the improvement process. Since the solution space around the starting solution has already been thoroughly explored, it is important to conduct the search towards a far region of the solution space. Therefore, perturbation size should be higher. In order to determine its best size, we made a computational experiment similar to the previous one, using $z = 0.5N$, $z = 0.75N$, $z = N$ and $z = 1.25N$. The results obtained have shown that $z = N$ and $z = 1.25N$ are the best, and that they give the same solutions, but with $z = N$, the execution is slightly faster.

The value of T influences the quality of the solutions obtained and computing time. Usually a value that gives a good trade-off between these two goals is established. In order to find this value, we performed a computational experiment in which we used eight test problems, H6, I2, L3, M3, N1, N5, O2 and O5, Method 1 to generate the initial solution, and several values of T. The results of this experiment are in Table 2. The main conclusions that can be drawn from this table are the following: the quality of the solution does not improve if T is greater than 3000 (in general, with different problems and different initial solutions, the quality of the solution may improve, but it is likely that the improvement will be small); from $T = 2000$ to $T = 3000$ the improvement is only 0.09%, but computing time almost doubles; if the precision desired is not very high, $T = 500$ is a good compromise, because the solution value is only 0.32% above the best one, but computing time is six times lower. Considering all this, we decided to choose $T = 2000$.

Table 2
Influence of the limit of the number of iterations without improving the best solution.

T	**100**	**500**	**1000**	**2000**	**3000**	**5000**
Total distance (D)	3,077,987	3,062,593	3,062,593	3,055,586	3,052,879	3,052,879
CPU (s)	12	42	69	137	254	371
D / D*	1.0082	1.0032	1.0032	1.0009	1	1

CPU / CPU*	1.0	3.5	5.8	11.4	21.2	30.9

5 Computational experiments

Our algorithm has been programmed in the C language and executed on a Toshiba Tecra T7300 at 2.0 GHz, 2 GB of RAM. The performance of our algorithm was tested using two sets of benchmark problems from the literature.

The first set of problems was originally proposed by Goetschalckx and Jacobs-Blecha (1989) and is constituted by sixty eight randomly generated Euclidean problems. The respective data and the solution values are available in http://www.isye.gatech.edu/people/ faculty/Marc_Goetschalckx/cali/Lineback/EvalOnly/Lhbhcase.txt (the solution costs found by these authors for the problems of group O were taken from here).The second set is made up of thirty three problems which were constructed by Toth and Vigo (1997).

In order to evaluate the performance of the ILSA, we compare it, in terms of solution cost and computing time, with state of the art algorithms from the literature, namely the following: Brandão (2006), version TS-K-tree_r (B06); Ropke and Pisinger (2006), version 6R-no learning (RP06); Gajpal and Abad (2009) (GA09); Zachariadis and Kiranoudis (2012) (ZK12); Vidal et al. (2014) (VCGP14); Cuervo et al. (2014), using 1000 iterations (CGSA14). These algorithms have been executed on computers with the following respective processors: Pentium III at 500 MHz, Pentium IV at 1.5 GHz, Intel Xeon at 2.4 GHz, Intel T5500 at 1.66 GHz, Opteron 250 at 2.4 GHz, Intel Core i7 at 2.93 GHz. For a correct comparison of computing time, the same computer should be used, but since this is not possible, we can get a very rough idea of the speeds of different computers using Dongarra's (2006, 2014) tables, which present the number (in millions) of floating-point operations per second (Mflop/s) executed by each computer. Nevertheless, we should note that even this is approximate, as these tables do not contain all computer models, as

happens with the computer used by us, but based on our experience, we know that its speed is similar to the Pentium IV at 2.4 GHz. Considering all this, we arrived at the speeds given in Table 3.

Table 3
Relative speeds of the computers used by the algorithms.

Algoritm	Processor	Mflop/s	Speed scaled
ILSA	Intel Core 2 Duo T7300, 2 GHz	884	1
B06	Pentium III, 500 MHz	72.5	0.0820
RP06	Pentium IV, 1.5 GHz	326	0.3688
GA09	Intel Xeon, 2.4 GHz	884	1
ZK12	Intel Core 2 T5500, 1.66 GHz	796	0.9005
VCGP14	Opteron 250, 2.4 GHz	1291	1.4604
CGSA14	Intel Core i7, 2.93 GHz	1598	1.8077

In Tables 4 and 5 we present the results obtained by the ILSA, with the settings specified in Section 4.4, for the first and second sets of problems, respectively. In both sets, the distances were calculated in double precision, but the way they are presented in the tables is different, as is explained below.

In the first set, we used the same assumptions taken by Toth and Vigo (1997) and Mingozzi et al. (1999) in their exact methods: the Euclidean distance is calculated in double precision, then is multiplied by 10, and afterwards rounded to the nearest integer. The value of final solution is divided by 10, and then rounded off to the nearest integer. Besides this integer value, the corresponding real value is also given in another column.

In the second set, the Euclidean distance between each pair of customers is calculated in double precision and then immediately rounded off to the nearest integer. This follows the rule used by Toth and Vigo (1997), but sometimes the difference between the integer solution value and the real one is quite large.

The optimal solution distances, marked with an asterisk, were taken from Toth and Vigo (1997) and Mingozzi et al. (1999), and the best known solution distances, when the optimal ones are not available, were taken from other articles under comparison (most of them are common to all these articles). In relation to the first set, we should point out the following details: *i)* The optimal solution distances of problems G1 and I2, given by (1997)

and (1999) are, respectively, 306,305 and 309,943, but the distances given by all the best heuristics are 306,306 and 309,944, respectively. This means that, most likely, the solutions are the same, but there are some rounding discrepancies. Therefore, we wrote 306,306 and 309,944 in Table 4. ii) Problem L1 has been solved by different authors considering different capacities of the vehicle, which is due to discrepancies among the different sources of the data. For example, Brandão (2006) used Q = 4,000, while Gajpal and Abad (2009) used Q = 4,400. In this article, we used Q = 4,400.

Table 4
Results for the first set of problems.

Nº	Name	N	n	m	Q	M	Best Published	ILSA		
								Distance		CPU (s)
1	A1	25	20	5	1550	8	229886*	229886	229885.65	1.0
2	A2	25	20	5	2550	5	180119*	180119	180119.21	1.2
3	A3	25	20	5	4050	4	163405*	163405	163405.38	1.2
4	A4	25	20	5	4050	3	155796*	155796	155796.41	0.8
5	B1	30	20	10	1600	7	239080*	239080	239080.16	1.3
6	B2	30	20	10	2600	5	198048*	198048	198047.77	1.3
7	B3	30	20	10	4000	3	169372*	169372	169372.29	0.9
8	C1	40	20	20	1800	7	249448*	250557	250556.77	2.2
9	C2	40	20	20	2600	5	215020*	215020	215020.23	2.5
10	C3	40	20	20	4150	5	199346*	199346	199345.96	2.1
11	C4	40	20	20	4150	4	195366*	195366	195366.63	1.8
12	D1	38	30	8	1700	12	322530*	322530	322530.13	2.5
13	D2	38	30	8	1700	11	316709*	316709	316708.86	2.2
14	D3	38	30	8	2750	7	239479*	239479	239478.63	2.0
15	D4	38	30	8	4075	5	205832*	205832	205831.94	2.6
16	E1	45	30	15	2650	7	238880*	238880	238879.58	3.9
17	E2	45	30	15	4300	4	212263*	212263	212263.11	2.2
18	E3	45	30	15	5225	4	206659*	207051	207050.50	2.6
19	F1	60	30	30	3000	6	263173*	263173	263173.96	8.7
20	F2	60	30	30	3000	7	265213*	265213	265214.16	4.6
21	F3	60	30	30	4400	5	241120*	241120	241120.78	7.0
22	F4	60	30	30	5500	4	233861*	233861	233861.85	5.7
23	G1	57	45	12	2700	10	306306*	306306	306305.40	9.4
24	G2	57	45	12	4300	6	245441*	245441	245440.99	4.9
25	G3	57	45	12	5300	5	229507*	229507	229507.48	6.8
26	G4	57	45	12	5300	6	232521*	232521	232521.25	7.5
27	G5	57	45	12	6400	5	221730*	221730	221730.35	6.5
28	G6	57	45	12	8000	4	213457*	213457	213457.45	6.4
29	H1	68	45	23	4000	6	268933*	268933	268933.06	10.3
30	H2	68	45	23	5100	5	253365*	253365	253365.50	9.2
31	H3	68	45	23	6100	4	247449*	247449	247449.04	8.6
32	H4	68	45	23	6100	5	250221	250221	250220.77	10.4
33	H5	68	45	23	7100	4	246121*	246121	246121.31	9.6
34	H6	68	45	23	7100	5	249135*	249135	249135.32	9.9
35	I1	90	45	45	3000	10	350246	350246	350245.28	27.3
36	I2	90	45	45	4000	7	309944*	309944	309943.84	20.4
37	I3	90	45	45	5700	5	294507	294507	294507.38	25.6
38	I4	90	45	45	5700	6	295988	297237	297236.69	21.6
39	I5	90	45	45	5700	7	301226	302380	302379.64	20.9
40	J1	94	75	19	4400	10	335007	335007	335006.68	33.3
41	J2	94	75	19	5600	8	310417	310417	310417.21	41.7
42	J3	94	75	19	8200	6	279219	279219	279219.21	30.6
43	J4	94	75	19	6600	7	296533	297615	297615.09	37.2
44	K1	113	75	38	4100	10	394071	395076	395075.67	36.9
45	K2	113	75	38	5200	8	362130	365754	365754.24	37.3
46	K3	113	75	38	5200	9	365694	369049	369049.02	47.6
47	K4	113	75	38	6200	7	348950	350304	350303.38	58.2
48	L1	150	75	75	4400	10	417897	418452	418452.10	86.8
49	L2	150	75	75	5000	8	401228	401965	401964.95	89.1
50	L3	150	75	75	5000	9	402678	403151	403150.89	95.2
51	L4	150	75	75	6000	7	384637	385552	385551.88	87.2

52	L5	150	75	75	6000	8	387565	388532	388532.47	92.0
53	M1	125	100	25	5200	11	398593	401447	401446.39	65.7
54	M2	125	100	25	5200	10	396917	399301	399300.59	53.4
55	M3	125	100	25	6200	9	375695	377979	377978.81	90.5
56	M4	125	100	25	8000	7	348140	348671	348671.62	99.5
57	N1	150	100	50	5700	11	408101	408690	408689.44	70.3
58	N2	150	100	50	5700	10	408066	409360	409359.79	90.7
59	N3	150	100	50	6600	9	394338	394547	394547.36	73.8
60	N4	150	100	50	6600	10	394788	394998	394997.85	83.3
61	N5	150	100	50	8500	7	373477	378985	378984.75	112.7
62	N6	150	100	50	8500	8	373759	376882	376881.94	101.8
63	O1	200	100	100	5700	10	526261	484000	483999.60	58.1[+]
64	O2	200	100	100	5700	11	507125	478202	478201.69	75.1
65	O3	200	100	100	6600	9	491935	459495	459494.82	65.4
66	O4	200	100	100	6600	10	489074	464797	464796.76	69.4
67	O5	200	100	100	8500	7	479438	442378	442377.56	66.8
68	O6	200	100	100	8500	8	476391	444138	444137.74	70.2
Average (first 62 problems)							290558.10	291154.18	291154.23	30.5
Number of optimal solutions							34	32		

[+] The problems of group O were solved with T = 500.

Table 5
Results for the second set of problems.

Nº	Name	N	n	m	Q	M	Best published	ILSA	
								Distance	CPU (s)
1	Eil22_50	21	11	10	6000	3	371*	371	0.5
2	Eil22_66	21	14	7	6000	3	366*	366	0.5
3	Eil22_80	21	17	4	6000	3	375*	375	0.5
4	Eil23_50	22	11	11	4500	2	682*	682	0.3
5	Eil23_66	22	15	7	4500	2	649*	656	0.3
6	Eil23_80	22	18	4	4500	2	623*	623	0.5
7	Eil30_50	29	15	14	4500	2	501*	501	0.5
8	Eil30_66	29	20	9	4500	3	537*	538	0.8
9	Eil30_80	29	24	5	4500	3	514*	519	0.9
10	Eil33_50	32	16	16	8000	3	738*	738	1.0
11	Eil33_66	32	22	10	8000	3	750*	750	0.8
12	Eil33_80	32	26	6	8000	3	736*	736	0.7
13	Eil51_50	50	25	25	8000	3	559*	560	3.3
14	Eil51_66	50	34	16	8000	4	548*	548	3.0
15	Eil51_80	50	40	10	8000	4	565*	565	5.1
16	EilA76_50	75	37	38	140	6	739*	739	16.9
17	EilA76_66	75	50	25	140	7	768*	768	22.9
18	EilA76_80	75	60	15	140	8	781	793	16.5
19	EilB76_50	75	37	38	100	8	801*	801	10.7
20	EilB76_66	75	50	25	100	10	873*	875	23.1
21	EilB76_80	75	60	15	100	12	919*	928	16.3
22	EilC76_50	75	37	38	180	5	713*	714	15.5
23	EilC76_66	75	50	25	180	6	734*	734	22.1
24	EilC76_80	75	60	15	180	7	733	737	31.5
25	EilD76_50	75	37	38	220	4	690*	690	15.9
26	EilD76_66	75	50	25	220	5	715	715	18.7
27	EilD76_80	75	60	15	220	6	694	695	34.5
28	EilA101_50	100	50	50	200	4	831	847	36.1
29	EilA101_66	100	67	33	200	6	846*	855	45.4
30	EilA101_80	100	80	20	200	6	856	891	27.3
31	EilB101_50	100	50	50	112	7	923	940	23.0
32	EilB101_66	100	67	33	112	9	983	1028	34.5
33	EilB101_80	100	80	20	112	11	1008	1016	47.5
Average							700.6	705.9	14.4
Number of optimal solutions							24	16	

The algorithms B06 and GA09 have been executed by their authors five and eight times, respectively, and the other ones ten times, because they all possess random

parameters. The distances and computing times shown in Table 6 are average values for each set of problems and the average of all executions of each algorithm. These algorithms, except CGSA14, have only solved the first sixty two problems of the first set.

Table 6
Comparison of the performance of different algorithms.

Algorithm	ILSA	B06	RP06	GA09	ZK12	VCGP14	CGSA14
Set 1							
Average distance	291154.18	291305.7	291823.35	290920.90	291927.72	290611.0+	291170.2
Average scaled CPU (s)	30.5	66.84	26.22	67.57	223.09	86.75	41.38
Difference to the best (%)	0.21	0.26	0.43	0.12	0.47	0.02	0.21
CPU time ratio	-	2.2	0.9	2.2	7.3	2.8	1.4
Set 2							
Average distance	705.88	702.50	704.54	702.35	Not solved	Not solved	703.52
Average scaled CPU (s)	14.4	24.36	15.55	25.64	-	-	13.29
Difference to the best (%)	0.75	0.27	0.56	0.24	-	-	0.41
CPU time ratio	-	1.7	1.1	1.8	-	-	0.9

+ These results are presented with a lower precision.

From Table 6 we can conclude that the seven algorithms have a similar performance, which is, in fact, very good, because more than half of the solutions are proven to be optimal. Both VCGP14 and GA09 yield a little better solutions than ILSA, but they are substantially slower. ILSA produces slightly better results than the other five for set 1, and a little worse for set 2, but this set is not so representative, as it contains about half of the problems, and, on average, they are substantially smaller.

The main advantage of ILSA over the other algorithms, is that it is much simpler, only uses standard procedures for the local search, and requires only two parameters. Furthermore, it is has two other advantages: it is deterministic, what means that the results are fully reproducible and it is substantially faster than the others except RP06, which is more or less identical, and CGSA14 which is only a bit slower. Additionally we should note that, according to Table 2, if $T = 500$ is used, then computing time can be reduced very substantially, with only a small loss in the quality of the solutions. If this is done, the results are the following: *i)* For set 1: the average distance is 291,462.52 and the average CPU is 10.2 s, i.e., the distance increases only 0.11% (although, being still better than RP06 and ZP12), but the computing time is three times less; *ii)* For set 2: the average distance is 707.88 and the average CPU is 4.1 s, i.e., the distance increases 0.28% and the computing time is 3.5 times less.

This algorithm contains an intensification cycle, defined by steps 7 to 23, that is applied while there is an improvement in the current solution. Note these steps are applied once in every iteration (excluding steps 14 to 17, which are applied only when a new best solution is found), but they are repeated, without perturbing the current solution, every time

the current solution is improved. In order to evaluate the influence of this cycle, the same sets of problems were solved without it. The results obtained were the following: i) For set 1: the average distance increased 4.1% in relation to that presented in Table 5 and the average CPU increased 1.1%; ii) For set 2: the average distance increased 6.0% and the average CPU increased 5.9%. This means that the intensification causes a large improvement in the solution quality with only a small increase in the computing time.

The analysis of the computational time complexity of the ILSA should be performed considering the trial moves that are evaluated in each iteration. Among the five types of moves used in the algorithm, the most demanding computationally and, therefore, the dominant, is the swap. The asymptotic time complexity of this kind of move is $O(N^2)$, i.e., the theoretical computational time of each iteration. In order to have a better idea of the practical computing time required by the algorithm we should determine the experimental time complexity. This has been made through the use of the information presented in Tables 4 and 5, taking into account that 2000 iterations were performed for solving each problem. Assuming that $f(N)$ represents the computing time, in seconds, required to solve a problem with N customers, executing 2000 iterations, that time can be estimated approximately by $f(N) = 0.005N^2 - 0.13N$.

6 Conclusions

This paper presents an iterated local search algorithm for the VRPB which is capable of finding very good solutions in a short time. This algorithm is simple, deterministic, almost parameter free and is fast. Therefore it can be a very good alternative to more sophisticated and more complex algorithms.

This paper successfully applies a metaheuristic that has been little explored and shows that it has much potential. The perturbation mechanism proposed has proven to be very effective. The intensification procedure which we devised plays an important role in the performance of the algorithm. This functions as follows: after finding a solution better than the current one or a new best local optimum, the intensification mechanism thoroughly explores the region of this solution, eventually finding a better local optimum; if this happens, the other improving procedures drive the search towards another region of the solution space and, if a better solution is found, the process is repeated. In fact, this cycle is frequently repeated three or four times consecutively. Another relevant feature of this algorithm is the use of a multi-start strategy.

Acknowledgements

This research was partially supported by the Fundação para a Ciência e Tecnologia, under project nº PTDC/EGE-GES/104092/2008. This support is gratefully acknowledged. We also thank the anonymous referees for their valuable comments.

References

Battarra M, Cordeau J-F, Iori M (2014) Pickup-and-Delivery Problems for Goods Transportation. In: Toth P, Vigo D (eds) Vehicle Routing: Problems, Methods, and Applications. MOS/SIAM Series on Optimization, 2nd edition, pp 161–192

Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G (2007) Static pickup and delivery problems: a classification scheme and survey. TOP 15:1-31

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

Casco D, Golden B, Wasil E (1988) Vehicle routing with backhauls: models, algorithms and case studies. In: Golden B, Assad A (eds) Vehicle routing: methods and studies, Elsevier Science Publishers, pp 127-147

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12, 568-581

Cuervo P, Goos P, Sörensen K, Arráiz E (2014) An iterated local search algorithm for the vehicle routing problem with backhauls. European Journal of Operational Research 237:454-464

Deif I, Bodin L (1984) Extension of the Clarke and Wright algorithm for solving the vehicle routing with backhauling. In: Kidder A (ed) Proceedings of the Babson Conference on Software Uses in Transportation and Logistics Management, Babson Park, pp 75-96

Dongarra J (2006) Performance of various computers using standard linear equations software". Report CS-89-85, University of Tennessee

Dongarra J (2014) Performance of various computers using standard linear equations software". Report CS-89-85, University of Tennessee

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

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

Goetschalckx, M, Jacobs-Blecha C (1993) The vehicle routing problem with backhauls: properties and solution algorithms. Technical Report MHRC-TR.88-13, Georgia Institute of Technology.

Hoff A, Andersson H, Christiansen M, Hasle G, Løkketangen A (2010) Industrial aspects and literature survey: fleet composition and routing. Computers and Operations Research 37:2041-2061

Irnich S, Schneider M, Vigo D (2014) Four Variants of the Vehicle Routing Problem. In: Toth P, Vigo D (eds) Vehicle Routing: Problems, Methods, and Applications. MOS/SIAM Series on Optimization, 2nd edition, pp 241–272

Lenstra J, Rinnoy Kan, A (1981) Complexity of vehicle routing and scheduling problems. Networks 11:221-227

Lourenço H, Martin O, Stützle T (2003) Iterated local search. In: Glover F, Kochenberger, G (eds) Handbook of Metaheuristics, Kluwer, pp 321-353

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

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

Parragh S, Doerner M, Hartl, R (2008) A survey on pickup and delivery problems - Part I: Transportation between customers and depot. Journal für Betriebswirtschaft 58:21–51
Reimann M, Doerner M, Hartl R (2002) Insertion based ants for vehicle routing problems with backhauls and time windows. In: Dorigo et al. (eds) Ant algorithms, Lecture Notes in Computer Science, Springer, Berlin/Heidelberg vol. 2463, pp 135–147
Ropke 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
Salhi, S, Wassan N, Hajarat M (2013) The fleet size and mix vehicle routing problem with backhauls: formulation and set partitioning-based heuristics. Transportation Research Part E-Logistics and Transportation Review 56:22-35
Thangiah R, Potvin J, Sun T (1996) Heuristic approaches to vehicle routing with backhauls and time windows. Computers and Operations Research 23:1043-1057
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 (2002) VRP with backhauls. In: Toth P, Vigo D (eds) The vehicle routing problem, SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA vol. 9, pp 195–224
Vidal T, Crainic T, Gendreau M, Prins C (2014) A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 234, 658-673
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
Waters C (1990) Expert systems for vehicle scheduling. Journal of the Operational Research Society 41:505-515
Zachariadis E, Kiranoudis C (2012) An effective local search approach for the vehicle routing problem with backhauls. Expert Systems with Applications 39:3174-3184