REM WORKING PAPER SERIES

Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls and Soft Time Windows

José Brandão

REM Working Paper 010-2017

October 2017

REM – Research in Economics and Mathematics
Rua Miguel Lúpi 20,
1249-078 Lisboa,
Portugal

ISSN 2184-108X

U LISBOA | UNIVERSIDADE DE LISBOA

ISEG LISBON SCHOOL OF ECONOMICS & MANAGEMENT UNIVERSIDADE DE LISBOA

REM RESEARCH IN ECONOMICS AND MATHEMATICS

Iterated Local Search Algorithm for the Vehicle Routing Problem with Backhauls and Soft Time Windows

José Brandão

Departamento de Gestão, Escola de Economia e Gestão, Universidade do Minho, Largo do Paço, 4704 – 553, Braga, Portugal.

CEMAPRE and REM, ISEG, University of Lisbon, Portugal.

E-mail: sbrandao@eeg.uminho.pt

Abstract

The vehicle routing problem with backhauls and soft time windows contains two disjoint sets of customers: those that receive goods from the depot, who are called linehauls, and those that send goods to the depot, named backhauls. To each customer is associated an interval of time (time window), during which each one should be served. If a time window can be violated it is called soft, but this violation implies an additional cost. In this paper, only the upper limit of the interval can be exceeded. For solving this problem we created deterministic iterated local search algorithm, which was tested using a large set of benchmark problems taken from the literature. These computational tests have proven that this algorithm competes with best known algorithms in terms of the quality of the solutions andcomputing time. So far as we know, there is no published paper for this problem dealing with soft time windows, and, therefore, this comparison is only with the algorithms that do not allow time windows violation.

Keywords: Backhauls; iterated local search; linehauls; logistics; soft time windows, vehicle routing.

1. Introduction

The *vehicle routing problem with backhauls and time windows* (VRPBTW) consists in finding a set routes, in order to serve a given number of dispersed customers, whose geographical location, demand, and time window for the service are known. Each route is

travelled by one vehicle assigned to it, which starts the trip at the depot, visits each customer of the route according to a given schedule, and, in the end returns to the depot.

In this problem there are two distinct sets of customers – those that require the delivery of goods, who are called linehauls, and those that require the collection of goods, named backhauls or pickup customers. In a route, the linehaul customers must be served first, followed by the backhauls (precedence constraint). Contrary to what happens in the VRPB, a route can contain only linehauls or only backhauls. Amongst the problems that make up delivery and pickup, there is another type in which no precedence constraints are imposed and each customer can require delivery, or pickup, or both. However, according to the literature, the type studied in this paper is the most common in practice. In the literature several different objective functions have been used by different authors. In this paper, the objective is to minimise the number of routes and, for the same number of routes, to minimise the total distance travelled by the vehicles. If the time window of a vertex is allowed to be violated it is called *soft* and the acronym for the problem is VRPBSTW.

The VRPBTW 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 (or time) d_{ij}, with $d_{ij} = d_{ji}$ for each $i, j \in V$, except if $j \in B$ and $i \in L$; $d_{ji} = +\infty$, for each $j \in B$ and $i \in L$. In the depot there are an unlimited number of 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 time window during which the service of vertex i must start is $[e_i, l_i]$, $i \in V$. (the time window of the depot corresponds to the driver's working day). The time of unloading at vertex i is u_i, $i \in V$, $u_0 = 0$. The travel time between i and j, plus u_i, $(i, j) \in A$ is

t_{ij}, being assumed that $d_{ij} = t_{ij} - u_i$, $(i, j) \in A$. Two customers i and j cannot be served on the same route if $e_i + t_{ij} > l_j$ and $e_j + t_{ji} > l_i$, and it is not possible to travel from i to j if $i \in B$ and $j \in L$. Furthermore, a customer j cannot be served after customer i if $e_i + t_{ij} > l_j$, $\forall i, j \in L \cup B$. It is well established that the VRPBTW is *NP-hard*.

There are plenty of articles and books that describe real-world applications comprising linehaul and backhaul customers, and surveys that classify the different types of problems of this kind and describe several algorithms for solving them. Some examples of these are the following: Casco et al. (1988), Toth and Vigo (2002), Berbeglia et al. (2007), Battarra et al. (2014) and Irnich et al. (2014).

Among the algorithms published in the literature for solving the VRPBTW,we can mention the following: Gélinas et al. (1995) – an exact algorithm; Duhamel et al. (1997) – a tabu search heuristic; Küçükoğlu and Öztürk (2015) – hybrid meta-heuristic that combines simulated annealing and tabu search; Thangiah et al. (1996 – the initial solution obtained by an insertion heuristic is then improved λ-interchanges and 2-opt* exchanges; Reimann et al. (2002) – insertion based ants; Ropke and Pisinger (2006) – large neighbourhood search heuristic; Vidal et al. (2014) – hybrid genetic search. Only the last four papers assumed an objective function identical to the one used here.

As far as we know, no article has been published for the VRPBSTW. Fu et al. (2008) identified six types of soft time windows for the VRPTW. This classification is based on the kind and amount of violation of the time window that is allowed. In this paper we allowed the time window violation that seems to be the most relevant in practice: the latest start time of the delivery can be exceeded by a given percentage, i.e., $l_i' = \alpha l_i$, where $\alpha \geq 1$, $i \in L \cup B$. The amount of the violation is penalized with a constant β.

The remainder of this paper is organised as follows. In Section 2, the method for generating the initial solution is defined. In Section 3, we present our iterated local search

algorithm. In Section 4, we present the computational experiments for both the VRPBSTW and the VRPBTW and, for this one, we compare the quality of our algorithm with the best algorithms published, and in the final section, we draw the main conclusions.

2. Method for generating the initial solution

The initial solution was generated by a sequential insertion method based on time window wideness. First, are routed all the linehaul customers and then the backhauls. Each route is started with the unrouted customer with the narrowest time window (in the case of tie, the nearest to the depot is chosen). Then, the feasible customer i, not yet routed, with the lowest insertion "cost" (c_i) is selected. This "cost" is given by the distance increase (d_i), due to its insertion in the route, plus the wideness of its time window (w_i) weighted by a parameter (p), i.e., $c_i = d_i + pw_i$. This parameter takes values between 0 and 2 with an increment of 0.1. For each value of p a different solution is obtained, being selected the best. If no customer can be inserted in the current route, then a new route is constructed in the same way, and the process is repeated until all customers are routed.

3. The iterated local search algorithm

The reader can find a detailed description of the iterated local search (ILS) in Lourenço et al. (2003). Very succinctly, this metaheuristic consists of applying iteratively and sequentially *local search* and *perturbation*.

Given a solution s, the local search finds a local optimum, s^*, by exploring the neighbourhood of s and performing a set of descent moves. The perturbation consists of applying a set of non-descent moves to given local optimum, s^*, generating s_p^*, in order to escape from this optimum, orienting the search towards a new region of the solution space. The strength of a perturbation procedure can be measured by the number of elements of s^* that is different s_p^*. Strong perturbations increase the diversity of regions explored, which

may contribute for finding better solutions. However, if these regions are far apart from each other, many good solutions close to the local optima found, may be missed.

The solution perturbed in each iteration is chosen according to the *acceptance criterion*.

In the following, first we describe the components of our iterated local search algorithm (ILSA) and then we present its general framework.

3.1. Local search

The ILSA comprises the following neighbourhood structures for improving a given solution: *i) cross over*, *ii) swap*, *iii) insertion*, *iv) interchange of chains; v) intra swap*, *vi) shift* and *vii) 2-opt*. The first four kinds of structures consist of moves between the routes of *s*, whilst the other three are performed inside each route of *s*. All the solutions generated along the search process must be feasible. Therefore, only the moves that keep the feasibility are allowed. With all these types of moves a *first best* strategy is applied, i.e., if a solution better than *s* is found, it is immediately accepted and the search pursues from this new solution.

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 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.

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.

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.

The *interchange of chains* comprises three independent cases: *interchange (2, 0), interchange (2, 1)* and *interchange (2, 2)*. The move consists of an exchange of two consecutive customers from r_i and inserting them into r_j in the same position occupied by the customers that are removed from r_j (0, 1 or 2, depending on the case) and go to r_i, taking the place of the two customers removed. The customers moved must be of the same type and remain connected, but their order can be reversed, if that is feasible and reduces the cost of insertion.

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. The *2-opt* is very well known and consists of deleting two arcs and adding two new ones. This move is applied to those routes containing only linehauls or only backhauls.

3.2. Perturbation

As explained before, the role of a perturbation is to drive the search towards a region of the solution space not yet explored. In the ILSA, the perturbation is always applied to the current solution, except when an elite solution is used, as will be described later.

In the ILSA, the following five types of perturbation procedures were used: *P1 - ejection chain, P2 - swap of linehauls with backhauls and interchange of chains, P3 - direct swap, P4 - insertion,* and *P5 - filling*. Note that some of these procedures are similar to those already defined to perform the local search, but in the perturbation are executed non improving moves. The ejection chain is applied in every iteration before one of the other four, which are applied in this sequence alternately. All the operators used in the ILSA are

deterministic. Therefore, in order to reduce the likelihood of cycling a large number of operators have been used and the way of applying some of them depends on the iteration.

The *ejection chain* as its name suggests, comprises a chain of moves. The process starts with the ejection (removal) of a customer from a route, which is then inserted in a different route after removing another customer from this route; after, this free customer is inserted in another route, etc. The chain ends when a free customer can be inserted without removing any other or if the free customer cannot be inserted in any route. This perturbation has two purposes: to eliminate routes and to perturb the solution. Because of the first aim, in phase 1 (the meaning of the phases is explained below), the search starts with the route with the least number of customers (least route, for short), and the route containing the customer that starts the ejection chain cannot receive any customer. Besides, in phase 1, the least route cannot receive customers ever, i.e., even when the chain starts in another route. In phase 2, there is no specific order for starting the search and any route can receive customers.

The *direct swap* consists in the direct exchange of two customers belonging to two different routes, being both linehauls or both backhauls. Any feasible move is accepted either if it is improving or non improving. Each time that this perturbation is applied, half of the customers are candidates for being swapped, and these candidate customers alternate at each application.

In the *swap of a linehaul with a backhaul* two customers of different type belonging to different routes are exchanged. They are removed and then inserted in the new route in any position where they are feasible. After the application of this operator, the *interchange of chains* is applied, considering the three cases mentioned before.

The *insertion* works as in local search, but the set of candidates at each application is formed by half of the customers, alternating at each application.

Filling consists of inserting in the least route as many customers from the other routes as possible. In phase 1, this perturbation is only applied if *P1* is not succeeded in removing

one or more customers from the least route. In phase 2, the filling alternates between the least and the second least route.

3.3. Elite solutions

These solutions are a set of solutions selected during the execution, after finishing the local search cycle. The composition of this set may change at each iteration, depending on the quality of the solution just found. This quality is evaluated according to the following hierarchical criteria: *i) Phase 1* - number of routes, number of customers of the least route, distance of the least route; *ii) Phase 2* - number of routes, total distance of the solution. After *T* iterations without improving the best known solution, the route chosen to apply the perturbation is the best solution of this elite set not yet explored. The value of T and the cardinality of the elite set, *E*, are defined below.

3.4. Mechanisms for minimising the number of routes

The mechanisms used for minimising the number of routes are the following:

1 – In any iteration, if the trial move generates a feasible solution with one route less than the best known feasible solution, this move is immediately accepted, independently of its cost. The operators that may eliminate one route of the current solution are cross over, interchange (2, 0), ejection chain, filling (not the route that is being filled, but another route of the solution from which the customers transferred are taken) and insertion.

2 – In order to reinforce the natural ability of the insertion move to eliminate a route, the following is done in phase 1: the trial insertion moves always start with the least route, then if a customer is removed from this route the (trial) insertion cost is reduced by a fixed cost (F_c), but if a customer inserted in this route the insertion cost is increased by F_c. This value has been defined as $F_c = RD/N$. Where R is the total number of routes of the initial solution and D is the total distance of this solution.

3.5. Phases of the algorithm and parameters setting

The ILSA comprises phase 1 and phase 2. These two phases are executed in this sequence twice. The only difference between them is that phase 1 tries directly to minimise the number of routes, while phase 2 tries to minimise the distance. This is why mechanism 2 is not applied in phase 2. The execution of each phase stops after T iterations without improving the best known solution. After repeating this cycle twice, phase 2 is executed again during T iterations without improving the best known solution. After some experiments, T was set equal 350 during the first two cycles (phase 1 followed by phase 2) and 1000 in the last application of phase 2.

The values of the parameters have been defined based on some experiments with the group of test problems that are faster to solve, i.e., the problems with $N = 100$. Besides, because most of the parameters can take an unlimited number of values, many of them were also defined by intuition. In general, if an algorithm allows sufficient diversification, the quality of the solutions increases when the number of iterations grows. However, it is also important to keep the computing time under reasonable limits. Two parameters that have a significant influence on the results are E and K. The values tried were 5 and 10 for E and 100 and 150 for K, i.e., 4 different combinations. The conclusions are that 5 and 100 generate slightly better solutions, but the computing time is a little lower when are used the values of 5 and 150 for E and K, respectively. Therefore, the best combination is 5 and 100. The most relevant for the evaluation of the final performance of the algorithm is not to try to find the best combination of parameter values, but to solve all the test problems with the same values. This is what happens with results presented in the tables of Section 4 and used in the comparison with the other algorithms.

3.6. Framework of the algorithm

s_b – best solution found.
t – counter of the number of iterations without improving the best solution. It restarts from zero in the beginning of each phase and when a new best solution is found.

1. Generate the initial solution, s; $s_b = s$.

2. Apply *phase 1* or *phase 2* while $t < T$ // phase 1 and 2 are repeated twice sequentially.
3. Repeat while there is an improvement of the current solution, s:
   - ♦ Apply cross over;
   - ♦ Apply swap;
   - ♦ Apply interchange (2, 0) and (2, 1);
   - ♦ Repeat while there is an improvement of s:
     - • Apply insertion and then intra swap to each route $r \in s$.
   - ♦ Apply interchange (2, 2);
   - ♦ Apply intra shift to each $r \in s$ and then 2-opt if r contains one type of customers only.
4. Insert s in the elite if it is better than any those already in it.
5. Perturb s or the best solution of the elite set not yet explored if t mod 100 = 0, set $t = t + 1$ and go to step 2.
6. Update T, set $s = s_b$ and execute phase 2 again, going to step 2.

4. Computational experiments

The benchmark problems were created by Gélinas et al. (1995) and Thangiah et al. (1996 based on Solomon (1987) data for the VRPTW. There are three sets of problems, containing 100, 250 and 500 customers, where the first set contains 15 problems and the other two sets contain 12 problems each. The number of backhauls is 10%, 30% or 50% of the total number of customers, as will clear in the tables below.

Our algorithm was programmed in the C language, and was executed on a desktop computer with an Intel i7-3820 processor at 3.6 GHz, and 32 GB of RAM. In order to evaluate the performance of the ILSA, we compare it, in terms of solution cost and computing time, with the best algorithms found in the literature that assume the same objective function, namely the following: Thangiah et al. (1996) – Thangiah, for short, Reimann et al. (2002) – Reimann, Ropke and Pisinger (2006) – Ropke, and Vidal et al. (2014) – Vidal. These algorithms were executed on the following computers, respectively: NeXT, Pentium III at 900 MHz, Pentium IV at 1.5 GHz, and Opteron 250 at 2.4 GHz. Following the literature, their speeds, measured in millions of floating-point operations per second (Mflop/s), can roughly be estimated as presented in Table 1.

Table 1 - Relative speeds of the computers used by the algorithms.

Algoritm	Processor	Mflop/s	Speed scaled
ILSA	Intel i7-3820, 3.6 GHz	1960	1

Thangiah	NeXT	1	0.0005
Reimann	Pentium III, 900 MHz	234	0.1194
Ropke	Pentium IV, 1.5 GHz	326	0.1663
Vidal	Opteron 250, 2.4 GHz	1385	0.7066

The ILSA is deterministic and, therefore, the results presented in the tables were obtained with one execution of the algorithm with the set of parameters defined in Section 3.5. Since in these articles there are several versions of the algorithm, we compare with the version that produces better results, except in the case of Thangiah where we use the best overall. The algorithms of Reimann, Ropke and Vidal contain several stochastic parameters. Therefore, the authors executed them 10 times, but they presented the results in different ways, as follows: Reimann – best results of the 10 executions; Ropke – average number of routes (but not the average distance), best solutions (routes and distance) and average computing time; Vidal – average solutions (routes and distance), best solutions and average computing time. The average results are the most relevant for the sake of comparison with our algorithm, but if they are not provided, we present the best solutions and the computing time required by the 10 executions. Note also that Reimann and Vidal only solved the first set of problems, and Ropke only present the global results for the other two sets. In the tables, *R* is the number of routes of the solution, *CPU* is the total computing time, in seconds. The computing times presented in Table 5 results from scaling the original times given by the authors, according to the relative speeds of the computers defined by Table 1.

Table 2 shows that the ILSA produces very slightly worse solutions than the algorithm of Vidal, a little bit better than the algorithm of Ropke (note that the distance presented for this algorithm is the best from 10 executions and, consequently, it is not directly comparable with the distance yield by the ILSA), better than Reimann, and substantially better than Thangiah. In terms of computing time, Table 5 shows that Thangiah is much faster than the ILSA, and this is faster than any of the others, being much faster than the algorithms of Vidal and of Reimann (about ten times).

Table 2 - Results for the first set of problems.

N = 100	#B	Thangiah		Reimann		Ropke		Vidal		ILSA		
		R	Distance	R	Distance	R	Distance	R	Distance	R	Distance	CPU
R101A	10%	24	1842.3	22	1831.7	22	1818.9	22	1818.9	22	1827.5	7.5
R101B	30%	24	1928.6	23	1999.2	23	1959.6	23	1959.5	23	1968.9	6.1
R101C	50%	25	1937.6	24	1945.3	24	1939.1	24	1939.1	24	1943.4	6.0
R102A	10%	20	1654.1	19	1677.6	19	1653.2	19	1653.2	19	1656.7	15.3
R102B	30%	21	1764.3	22	1754.4	22	1750.7	22	1750.7	22	1760.2	15.2
R102C	50%	21	1745.7	22	1782.2	22	1775.8	22	1775.8	22	1790.9	10.9
R103A	10%	15	1371.6	16	1348.4	15	1387.6	15	1385.4	15	1402.8	17.8
R103B	30%	16	1477.6	16	1395.9	15	1390.3	15	1390.3	15	1415.6	12.4
R103C	50%	17	1543.2	17	1467.7	17	1456.5	17	1456.5	17	1489.0	14.1
R104A	10%	13	1220.3	11	1205.8	11	1084.2	10.4	1157.9	11	1121.9	28.3
R104B	30%	12	1302.5	12	1128.3	11	1154.8	11	1155.7	11	1196.6	24.3
R104C	50%	13	1346.6	12	1208.5	11	1191.4	11	1190.9	11	1223.9	20.7
R105A	10%	17	1553.4	16	1544.8	15.4	1561.3	15	1561.6	15	1579.3	10.2
R105B	30%	18	1706.7	16	1592.2	16	1583.3	16	1583.3	16	1585.7	11.0
R105C	50%	18	1657.4	17	1633.0	16.5	1710.2	16	1709.9	16	1710.2	8.0
Total		**274**	**24051.9**	**265[b]**	**23514.9[b]**	**259.9[a]**	**23416.7[b]**	**258.4[a]**	**23488.7[a]**	**259**	**23672.6**	**207.8**
Difference (%)		**5.8**	**1.6**	**2.3**	**-0.7**	**0.3**	**-1.1**	**-0.2**	**-0.8**	**-**	**-**	**-**

a) This is the average of 10 runs. b) This the best of 10 runs.

Table 3 - Results for the second set of problems.

N = 250	#B	Thangiah		Ropke		ILSA		
		R	Distance	R	Distance	R	Distance	CPU
R1Do250A	10%	49	5085.0			46	4900.0	229.3
R1Do250B	30%	48	5423.0			45	5120.6	155.3
R1Do250C	50%	52	5403.1			49	5215.4	173.4
R1Up250A	10%	39	4278.6			32	4020.2	98.3
R1Up250B	30%	41	4715.2			34	4598.0	112.7
R1Up250C	50%	43	4921.4			36	4567.2	128.4
RC1Do250A	10%	39	4613.4			33	4200.5	94.2
RC1Do250B	30%	41	4852.2			34	4570.3	99.9
RC1Do250C	50%	41	4329.4			34	4531.6	117.5
RC1Up250A	10%	40	4445.8			33	4165.1	114.7
RC1Up250B	30%	43	4722.4			35	4568.3	90.0
RC1Up250C	50%	41	4899.4			35	4604.9	89.9
Total		**517**	**57688.9**	**449[a]**	**54499[b]**	**446**	**55062.0**	**1553.6**
Difference (%)		**15.9**	**4.8**	**0.7**	**-1.0**	**-**	**-**	**-**

a) This is the average of 10 runs. b) This the best of 10 runs.

The results of Tables 3 and 4 follow the same trend of those of Table 2. Additionally, they show that the quality of the solutions given by the ILSA becomes increasingly better than the yield by the other two algorithms (Thangiah and Ropke) as long as the number of customers increases. For example, the total number of routes given by Thangiah is 5.8%, 15.9% and 20.5% more than the yield by the ILSA, for the problems with 100, 250 and 500

customers, respectively, and a similar trend is observed with Ropke. In what concerns to the computing time, the behaviour in relation to the algorithm of Thangiah is just the opposite, i.e., the percentage difference decreases when the dimension of the problem increases, but it is still much faster than the ILSA. On the contrary, the ILSA becomes increasingly slower than the algorithm of Ropke.

Table 4 - Results for the third set of problems.

N = 500	#B	Thangiah		Ropke		ILSA		
		R	Distance	R	Distance	R	Distance	CPU
R1Do250A	10%	67	7620,4			58	6925,5	798,1
R1Do250B	30%	69	8020,2			58	7386,6	821,3
R1Do250C	50%	76	8388,3			60	7526,6	786,6
R1Up250A	10%	64	7267,2			53	6757,3	816,4
R1Up250B	30%	73	7926,6			56	7255,2	643,8
R1Up250C	50%	68	8043,7			58	7222,6	684,9
RC1Do250A	10%	61	7099,4			50	6474,1	774,5
RC1Do250B	30%	63	7707,1			53	6933,2	503,4
RC1Do250C	50%	65	7771,6			53	7107,1	482,1
RC1Up250A	10%	63	7209,4			53	6718,7	405,2
RC1Up250B	30%	63	7967,1			56	7087,5	335,1
RC1Up250C	50%	67	8135,1			55	7052,3	329,8
Total		**799**	**93156,1**	**680[a]**	**82796[b]**	**663**	**84446,7**	**7381,2**
Difference (%)		**20.5**	**10.3**	**2.6**	**-2.0**	**-**	**-**	**-**

a) This is the average of 10 runs. b) This the best of 10 runs.

Table 5 – Computing times for each set of problems, in seconds.

Problem set	Thangiah[a]	Reimann[b]	Ropke[c]	Vidal[c]	ILSA
N = 100	0.4	2686	283	2607	206
N = 250	6.0	-	1006	-	1504
N = 500	63.4	-	3492	-	7381

a) Time to execute all the versions. b) Time to execute 10 runs. c) Average time of 10 runs.

As final conclusion, we can say that, in general, the ILSA produces better solutions than the other algorithms and it is rather fast. Note that the algorithm of Thangiah is much faster, but this is not sufficient to compensate the difference in the quality of the solutions. For example, for the set of problems with 500 customers the initial solution of the ILSA is better (34 routes less) and takes less than one tenth of computing time.

The experiments concerning the VRPBSTW were performed with $\alpha = 1.03$ and $\beta = 100$, and the results obtained are presented in Table 6. In this case the cost of a solution, s, is given by the following equation:

$$c(s) = d(s) + \beta L(s).$$

Where $d(s)$ is the total distance travelled and $L(s)$ is the total lateness of s.

Table 6 – Global results for the three sets of problems when the time windows are soft.

Problem set	Hard time window		Soft time window			Difference (%)	
	R	D	R	D	Av. V. (%)	R	D
N = 100	259	23672.6	238	24042.2	3.4	-8.1	1.6
N = 250	446	55062.0	417	56541.4	3.0	-6.5	2.7
N = 500	663	8446.8	632	88331.3	3.7	-4.7	4.6
Total	**1368**	**163181.4**	**1287**	**168914.9**	**3.3**	**-5.9**	**3.5**

Av. V. – Average number of customers whose time window is violated.

We can conclude from Table 6 that allowing a lateness of 3% at each customer and using a penalty of 100 for this lateness, the number of routes was reduced on average 5.9% and the number customers with non violated time windows was 96.7%, i.e., only 3.3% of the customers have the latest start time exceeded, on average.

5. Conclusions

This paper presents an iterated local search algorithm for the VRPBSTW, which shows that allowing the violation of the time windows by a small amount the fleet of vehicles can be reduced substantially. On the other hand, when the same algorithm is used to solve the VRPBTW it is very competitive with the existing algorithms, both in terms of quality of the solutions and computing time. Furthermore, our algorithm is deterministic, what means that the results are fully reproducible. The performance of our algorithm is mainly due to the use of ejection chains, elite solutions and effective perturbation procedures.

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.
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.
Duhamel, C, Potvin, J-Y, Rousseau, J-M (1997). A tabu search heuristic for the vehicle routing problem with back-hauls and time windows, Transportation Science, 31, 49–59.
Fu Z, Eglese R, Li, L (2008). A unified tabu search algorithm for vehicle routing problems with soft time windows, Journal of the Operational Research Society, 59, 663-673.
Gélinas, S, Desrochers, M, Desrosiers, J, Solomon, M (1995). A new branching strategy for time constrained routing problems with application to backhauling, Annals of Operations Research, 61, 91–109.
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.
Küçükoğlu, I and Öztürk, N (2015). An advanced hybrid meta-heuristic algorithm for the vehicle routing problem with backhauls and time windows. Computers and Industrial Engineering, 86, 60-68.
Lourenço H, Martin O, Stützle T (2003). Iterated local search. In: Glover F, Kochenberger, G (eds) Handbook of Metaheuristics, Kluwer, pp 321-353.
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.
Solomon, M. M., (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Operations Research, 35, 2, 254-265.
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 (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.