# An Iterated Variable Neighborhood Descent Algorithm applied to the Pickup and Delivery Problem with Time Windows

**Carlo S. Sartori**
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Porto Alegre - RS - Brazil
cssartori@inf.ufrgs.br

**Marcelo W. Friske**
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Porto Alegre - RS - Brazil
mwfriske@inf.ufrgs.br

**Luciana S. Buriol**
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Porto Alegre - RS - Brazil
buriol@inf.ufrgs.br

## ABSTRACT

Pickup and Delivery Problems are variations of the Vehicle Routing Problem, and arise in many real-world transportation scenarios. This work studies the Pickup and Delivery Problem with Time Windows, in which goods have to be transported from one location to another, respecting certain time and vehicle capacity constraints. It aims at minimizing the number of vehicles used, as well as the operational costs to perform all routes. To solve this problem, an algorithm is proposed by embedding a Variable Neighborhood Descent method into an Iterated Local Search metaheuristic. Experiments are carried out with standard literature instances, showing that the algorithm is able to deliver good solutions in competitive time. The student, and first author, developed and wrote the entire work, as well as interacted with a partner company, as part of his scientific initiation project and bachelor thesis in computer science, always advised by the other two authors.

**KEYWORDS. Vehicle Routing Problem. Pickup and Delivery. Iterated Local Search.**

**Paper topics: Logistic and Transport (L&T), Metaheuristics (MH)**

## 1. Introduction

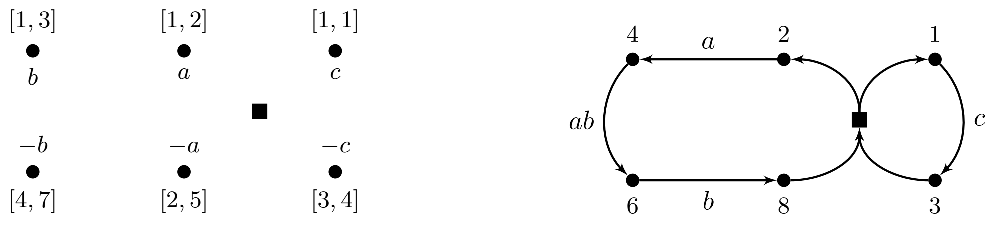
Vehicle Routing Problems (VRP) have a wide range of scientific studies due to their usefulness in several real-world cases, mainly in the transportation and logistic areas. Many variations and mathematical models exist in order to comprise all the particularities of each scenario. Among them, there is the Capacitated Vehicle Routing Problem (CVRP), Vehicle Routing Problem with Time Windows (VRPTW), and *Pickup and Delivery Problem with Time Window*s (PDPTW). The latter is the focus of this research work.

In the studied PDPTW a set of customer requests has to be serviced by a set of vehicles. Each request is a pair of pickup and delivery locations. Goods have to be transported from the pickup location to the delivery location. This creates two side constraints: i) *precedence constraint*, the pickup has to be visited before the corresponding delivery; and ii) *pairing constraint*, the pickup and delivery pair must be visited by the same vehicle. Every location has a time window, stating the minimum and maximum time a vehicle can start the service. This time window cannot be violated (*hard* time window). There is also a service time, or the amount of time a vehicle takes to complete the service, and a demand, which is the amount of goods the vehicle should pickup or deliver at the given location. In this problem, there is an one-to-one mapping between pickup and delivery locations, thus the demand of a pickup is strictly complementary to its delivery.

The fleet of vehicles is homogeneous and each vehicle has a maximum capacity, which is the amount of goods that can be carried at once. This capacity should never be violated. They are located at a single common depot, from where their routes start and end. There is a maximum number of vehicles to be used, which should not be exceeded as well.

The PDPTW is a $\mathcal{NP}$-Hard combinatorial optimization problem, Ropke and Pisinger [2006], that can be formulated on a graph $G = (V, E)$, where all locations are vertices in $V$, and paths are edges in $E$ connecting two locations, with associated cost (distance) and travel time. Figure 1 presents an example instance and a possible feasible solution using two vehicles.

Figure 1: Example of a PDPTW solution. (a) Locations with demands and time windows [min, max], square node is the depot and circles are requests locations; (b) a feasible solution example using two vehicles, where node labels are the exact time a vehicle reaches the node, and arc labels are the load carried by the vehicle.



(a) Locations in a graph (b) Feasible Solution

The objective is to first minimize the number of vehicles used, and then the total cost of all routes. This problem arises in many real-world applications, such as product delivery, dial-a-ride problems, courier services, airline scheduling, bus routing and logistics and maintenance support, Nanry and Barnes [2000]. Thus the study of efficient solution methods for this problem can bring improvements to several services of daily usage.

This work was motivated by a partnership with a software company in Porto Alegre, uMov.me (`http://www.umov.me/`), in a research project to solve several real-world routing problems. The choice of the PDPTW happened because it is a broad problem that generalizes others (such as CVRP and VRPTW) as shown by Pisinger and Ropke [2007].

## 2. Related Works

The first work to apply a metaheuristic to solve the multi-vehicle PDPTW was Nanry and Barnes [2000]. It used a *Reactive Tabu Search* containing three simple movements: i) moving a request from one route to another; ii) exchanging a request in one route with a request in another; and iii) relocating a request to another position within its route. The method was tested with instances containing up to 50 requests, which were later considered too easy and ended up unused.

Li and Lim [2003] proposed a *Tabu-embedded Simulated Annealing* approach for solving the PDPTW. The method uses the same neighborhoods of Nanry and Barnes [2000]. The authors introduced a new set of benchmark instances, which became the standard set for the PDPTW. The objective function first minimizes the number of vehicles used, then the total cost of all routes.

Bent and Van Hentenryck [2006] have applied a *Two-stage Hybrid Algorithm* to solve the PDPTW, and tested the method with the instances presented by Li and Lim [2003]. The first stage of the algorithm aims at minimizing the number of vehicles used by means of a *Simulated Annealing*. The second stage minimizes the travel distance by means of a *Large Neighborhood Search*, which uses a remove and reinsert approach.

Ropke and Pisinger [2006] developed an *Adaptive Large Neighborhood Search* for a variation of the PDPTW considered. The method uses a two-stage approach with the same heuristic in both, adjusting the parameters accordingly. In the first stage vehicle minimization is considered, while in the second stage the total distance is minimized. The method uses a remove and reinsert strategy. It was able to improve on more than half of all instances of Li and Lim [2003] data set, and can be considered the current *state-of-the-art* method for the PDPTW.

This work is a summary of the the bachelor thesis in computer science of the first author, Sartori [2016]. Further details can be obtained from the original text.

## 3. Proposed Algorithm

This work proposes an ILS algorithm, Lourenço et al. [2010], embedding a VND, Mladenović and Hansen [1997], to be used as the local search. The resulting algorithm is an *Iterated Variable Neighborhood Descent* (IVND) and it is presented in Algorithm 1. The algorithm has five parameters, which are explained along this section.

**Algorithm 1** IVND

```
Parameter: $I, K, \alpha, n_e, \mathcal{N}$
1: $s_0 = \text{ModifiedInsertionHeuristic}(\ )$
2: $s^* = s^{*\prime} = \text{VND}(s_0, K, \mathcal{N})$
3: repeat
4:     $s' = \text{Perturbation}(s^{*\prime}, n_e)$
5:     $s^{*\prime} = \text{VND}(s', K, \mathcal{N})$
6:     $s^{*\prime} = \text{AcceptanceCriterion}(s^*, s^{*\prime}, \alpha)$
7: until $i_b > I$
8: return $s^*$
```

In line 1, the initial solution $s_0$ is generated, and improved in line 2 by the VND metaheuristic. Then, the main loop (lines 3-7) is repeated until the number of *iterations without improvement*, $i_b$, reaches a maximum value $I$, the *stopping criterion*. The loop alternates between perturbations, improvements and acceptance of new solutions. The current solution $s^{*\prime}$ is perturbed in line 4, creating $s'$, which is improved in line 5 by the VND. A new local minimum $s^{*\prime}$ is generated. In line, 6 the algorithm chooses to accept the new solution or continue from the incumbent $s^*$. Solution feasibility is kept at all moments.

Unless otherwise stated, a solution is minimized by the function in Equation 1, following the lexicographic order of its terms.

$$e(s) = (\ |s|, \sum_{r \in s} d(r)\ ) \tag{1}$$

The first term is the number of routes in solution $s$, and the second term is the total distance traveled by all routes. Note that the second term is usually referred as the total cost, though, in this case, distance and cost will be used interchangeably.

### 3.1. Initial Solution

Initial solutions are generated by a modified version of the Insertion Heuristic proposed by Solomon [1987]. First, an empty route is initialized with one request, using two criteria. From the set of requests, those that have a feasible insertion and the pickup location with minimum value of starting time window are chosen. When more than one request meets the previous criterion, the one with the pickup location closest to the depot (*minimum distance*) is chosen. If there are any ties left, the pickup location with highest index is selected. A similar method was proposed by Li and Lim [2003]. Then, for each unrouted request, its best position in the route is computed as the sum of the costs to insert the pickup and the delivery location in the given position. The request and position that minimize the objective function are chosen.

This procedure is repeated until there are no more requests to be routed, or no more feasible insertions, whichever happens first. If all requests have been routed, a feasible solution is returned. Otherwise, a feasible solution could not be generated, and the algorithm halts returning no solution.

## 3.2. Variable Neighborhood Descent

The embedded VND works as the local search procedure in the ILS. The proposed method is allowed to iterate over $K$ times. This allows for a better search of solutions, avoiding certain local minima, while being reasonably fast.

Four neighborhoods are used in the proposed IVND: i) *Shift Request*, ii) *Exchange Request*, iii) *Rearrange Request*, and iv) *Unbalanced Shift Request*. The first three were used by Nanry and Barnes [2000]; Li and Lim [2003], while the last is inspired by a method of Bent and Van Hentenryck [2006]. All of them are explored in a *best improvement* manner. For all neighborhoods, described next, infeasible moves are forbidden with regards to the PDPTW constraints.

**i. Shift Request (SR)**: The *shift* neighborhood attempts to move a request from one route to another. For every pair of routes $r_1, r_2 \mid r_1 \neq r_2$, a request in route $r_1$ is removed and inserted in $r_2$. The pair and position that minimize the cost the most are chosen. The basic idea is pictured in Figure 2(b). This neighborhood and *Unbalanced Shift Request* are important because they are able to reduce the number of routes.

**ii. Exchange Request (ER)**: The *exchange* neighborhood swaps two requests between routes. For every pair of routes $r_1, r_2 \mid r_1 \neq r_2$, a request $p_1$ is removed from route $r_1$, and a request $p_2$ from route $r_2$. Then, $p_1$ is inserted in $r_2$, as well as $p_2$ is inserted in $r_1$. The pair and position that minimize the cost the most are chosen. The idea is shown in Figure 2(c).

**iii. Rearrange Request (RR)**: The *rearrange* neighborhood is the only *intra-route* neighborhood, meaning its movements only affect a single route. For every route $r$, a request $p$ is removed and reinserted in another position in $r$. The route and position that minimize the cost the most are chosen. This allows further refinements. Figure 2(d) pictures this idea.

**iv. Unbalanced Shift Request (USR)**: The *unbalanced shift* neighborhood is based on the *Shift Request*. Their difference relies in the objective function used to evaluate the movement. In the *Shift Request*, the original evaluation in Equation 1 is used, while in the *Unbalanced Shift Request* the objective function presented in Equation 2 is used.

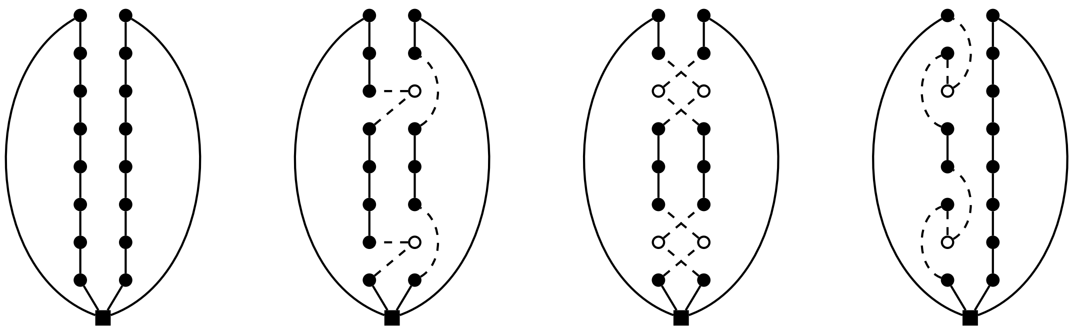
$$e(s) = (\ |s|, -\sum_{r \in s} |r|^2, \sum_{r \in s} d(r)\ ) \tag{2}$$

The terms are also minimized in lexicographical order. The first and last terms are the same as in Equation 1. The second term maximizes $\sum_{r \in s} |r|^2$, which means it favors routes with many locations and fewer locations, instead of routes with a balanced distribution of them. The same function was proposed by Bent and Van Hentenryck [2006] in the first stage of their algorithm to reduce the number of vehicles by integrating smaller routes into larger ones.

## 3.3. Perturbation

The perturbation moves the search to other areas of the solution space, avoiding local minima. In order to perform perturbation, IVND employs a series of consecutive randomly chosen *Exchange Requests*. For each exchange, two routes in solution $s$ are picked at random, say $r_1,\ r_2 \mid r_1 \neq r_2$, and two requests $p_1 \in r_1,\ p_2 \in r_2$ are removed and reinserted at a random position in the other route, i.e., $p_1$ in $r_2$, and $p_2$ in $r_1$. This is repeated until $n_e N$ exchanges ($n_e$ a parameter, $N$ the number of request in the instance), or $N$ unsuccessful tries have been performed. Infeasible movements are forbidden.

Figure 2: Neighborhood Movements of VND



(a) Original (b) *Shift* (c) *Exchange* (d) *Rearrange*

■ *depot* ○ *moved request* ● *other requests*

## 3.4. Acceptance Criterion

The acceptance criterion in ILS guides the search towards solution diversification, or intensification. The proposed IVND employs a criterion based in the number of *iterations without improvement*. The method automatically accepts the new $s^{*\prime}$ if its objective function value $f(s^{*\prime})$ is smaller than the value $f(s^{*})$ of the incumbent solution. Otherwise, it accepts $s^{*\prime}$ over $s^{*}$ with probability $\alpha(i_b/I)$. Variable $i_b$ contains the number of iterations without improvement, parameter $I$ is the maximum number of iterations without improvement, and parameter $\alpha$ is responsible for adjusting the acceptance rate.

## 4. Computational Results

In order to test the proposed algorithm, the IVND has been implemented in C++ and compiled with the GNU C++ compiler `g++ 4.8.4` using `-O3` optimization flag. Experiments were carried out in a computer with *AMD FX 8150* processor running at 3.6 GHz, 32 GB of RAM and operating system Ubuntu 16.04 LTS 64-bits. Only one core has been used during the experiments.

The standard set of PDPTW instances has been used, containing a total of 354 instances. These instances are separated according to the number of locations: 100, 200, 400, 600, 800, and 1000. Also, for each size, instances are divided according to their location distribution: clustered, random, or partially clustered and partially random. Further, instances are separated into type 1 instances, with shorter planning horizon, and type 2, with longer planning horizon. All instances as well as best known solutions (BKS) are available at SINTEF's Website [SINTEF, 2017].

Because the IVND is an stochastic method, the results for each instance are the average of 10 runs. Also, due to the fact that results for all the 354 instances would not fit in the available space, only the average of all instances for each size and planning horizon type is presented. More details can be obtained from the the *online* repository (`https://bitbucket.org/cssartori/ivnd-pdptw/`).

Experiments were also made with 40 real-world instances provided by the partner company, uMov.me, for a scenario closely related to the PDPTW, with sizes varying from 1 to 70 requests. Due to the lack of space, we could not report these results. However our analysis verified that the IVND was able to reach good solutions in small time for these instances. More detailed information can be obtained from the work of Sartori [2016] and from the repository.

## 4.1. Parameters

In Section 3 the IVND was defined with five parameters: maximum number of iterations without improvement ($I$), maximum number of VND iterations ($K$), acceptance rate ($\alpha$), perturbation size ($n_e$), and VND's neighborhood order ($\mathcal{N}$). To choose their values a tuning procedure was employed using the `irace` package, López-Ibánez et al. [2011]. The scenario defined for irace had a maximum running time of 320,000 seconds and a total of 18 instances, 3 for each size. The parameters ranges tested, as well as the best values reported by irace can be seen in Table 1.

Table 1: Parameter tuning ranges for irace

| **Parameter** | **Range** | **Best Value** |
|---|---|---|
| $I$ | {50,...,100} | 82 |
| $K$ | {50,...,100} | 98 |
| $\alpha$ | {0.0,...,1.0} | 0.288 |
| $n_e$ | {0.01,...,0.50} | 0.266 |
| $\mathcal{N}$ | {SR, ER, RR, USR} | SR, RR, ER, USR |

## 4.2. General Performance of IVND

Table 2 presents the average initial and final results of the IVND for each size and type of instance. Column *Num.* is the total number of instances tested per type and size. Columns $\#V_b$, $\#V_i$ and $\#V_f$ are the average number of vehicles of the BKS, of the initial solution of IVND, and of the final solution of IVND, respectively. Column gap$_\text{c}$ (%) refers to the percentage deviation of the total cost, or distance, of IVND solution $s$ relatively to the BKS $s^*$: it is calculated as gap$_\text{c}(s) = (s_c - s^*_c)/s^*_c$, where $s_c$ is the accumulated cost of all routes in a solution $s$. Column gap$_\text{v}$ (%) is the percentage deviation of the number of vehicles of the final solution to the BKS, given by gap$_\text{v}(s) = (s_v - s^*_v)/s^*_v$, where $s_v$ is the number of vehicles used in solution $s$. The column *t(s)* gives the average CPU time in seconds taken to reach the reported solution.

Table 2: Average Initial and Final Results of IVND for standard instances

| | **Instances** | | **BKS** | | **Initial Solution** | | | **Final Solution** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Type** | Size | Num. | $\#V_b$ | Cost | $\#V_i$ | gap$_\text{c}$(%) | t(s) | $\#V_f$ | gap$_\text{v}$(%) | gap$_\text{c}$(%) | t(s) |
| 1 | 100 | 29 | 11.10 | 1,158.50 | 14.24 | 28.37 | 0.01 | 11.52 | 3.86 | 0.35 | 0.56 |
| | 200 | 30 | 15.43 | 3,439.06 | 19.30 | 39.65 | 0.06 | 16.83 | 10.57 | -3.62 | 12.60 |
| | 400 | 30 | 29.37 | 8,175.50 | 36.80 | 42.54 | 0.31 | 32.75 | 13.92 | -1.62 | 76.80 |
| | 600 | 30 | 42.47 | 16,352.54 | 52.57 | 47.12 | 1.31 | 46.82 | 12.91 | 2.01 | 210.64 |
| | 800 | 30 | 55.13 | 28,265.12 | 67.90 | 48.68 | 2.83 | 61.42 | 14.90 | 3.95 | 385.51 |
| | 1000 | 30 | 68.33 | 43,590.16 | 83.27 | 45.84 | 14.91 | 76.10 | 15.35 | 4.94 | 693.80 |
| | Avg. | | 36.97 | 16,830.15 | 45.68 | 42.03 | 3.24 | 40.91 | 11.92 | 1.00 | 229.99 |
| 2 | 100 | 27 | 2.96 | 906.04 | 4.30 | 75.02 | 0.04 | 3.03 | 2.90 | 0.93 | 4.77 |
| | 200 | 30 | 4.57 | 2,690.69 | 6.17 | 73.73 | 0.27 | 5.07 | 14.53 | -4.23 | 48.95 |
| | 400 | 30 | 8.53 | 6,278.10 | 11.83 | 87.31 | 1.89 | 10.07 | 24.63 | -1.30 | 182.48 |
| | 600 | 30 | 12.03 | 13,422.25 | 16.63 | 82.77 | 5.59 | 14.42 | 27.59 | 0.16 | 328.12 |
| | 800 | 30 | 15.70 | 21,595.10 | 22.03 | 86.14 | 15.48 | 19.28 | 32.36 | 6.28 | 398.96 |
| | 1000 | 28 | 19.57 | 31,431.00 | 26.50 | 79.50 | 25.79 | 23.60 | 29.72 | 7.97 | 664.33 |
| | Avg. | | 10.56 | 12,720.53 | 14.58 | 80.75 | 8.18 | 12.41 | 21.96 | 1.64 | 271.27 |
| Avg. | | | 23.77 | 14,801.56 | 30.13 | 61.39 | 5.71 | 26.66 | 16.94 | 1.32 | 250.63 |

There is a major difference between solution quality of type 1 and type 2 instances for both initial and final solutions. Initially, type 1 instances have 9 more vehicles in average than the BKS, while type 2 instances have only an average of 4 more vehicles. However, the total cost of the routes has an average gap$_\text{c}$ of 40% for type 1 and 80% for type 2 instances. The main reason for these differences is that shorter routes are harder to create at first, hence more routes are needed to attend all requests. Additionally, it is more straightforward to find better insertion positions when routes are smaller, reducing the total cost.

In the final solutions, the number of additional vehicles drops to only 4 for type 1 instances and only 2 for type 2. However, the average gap$_\text{v}$ of the instances is high, reaching 15% on type 1 and 32% on type 2 instances. The final gap$_\text{c}$ decreases to 1.00% for type 1 and to 1.32% for type 2 instances. Some negative gaps are possible, even though the solution is not better than the BKS, because when the IVND can no longer reduce the number of vehicles, it reduces the total cost of the routes. A similar situation has been reported by Ropke and Pisinger [2006].

## 4.3. Comparing Results from the IVND and Literature Methods

This section compares the IVND performance to the main methods from the literature published to our knowledge. Table 3 presents the computational environment for each of the methods. Column *Reference* presents the referred work, column *Acronym* presents an abbreviation used to refer to the work, and column *Instances* gives the instances that were tested with the method. Column *Environment* gives the processor used, while column *speed(%)* presents an estimated percentage of speed of each computer relative to the computer used by this work. The estimation is highly conservative, and is based in the PassMark Tool (`http://www.cpubenchmark.net`).

Table 3: Computational Environments

| **Reference** | **Acronym** | **Instances** | **Environment** | **speed(%)** |
|---|---|---|---|---|
| Li and Lim [2003] | LL | {100} | Intel 686 Generation | 12.50 |
| Bent and Van Hentenryck [2006] | BVH | {100, 200, 600} | AMD Athlon K7 (1.2 GHz) | 16.67 |
| Ropke and Pisinger [2006] | RP | All | Pentium IV (1.5 GHz) | 20.00 |
| This Work | IVND | All | AMD FX 8150 (3.6 GHz) | 100.00 |

In Table 4 the results of IVND for each size and instance type are compared to the results of the other three methods, by means of the number of vehicles (#V), the cost gap ($\text{gap}_c(\%)$) and running time (t(s)). The BKS column has been omitted, though it is available in Table 2. The reported times are all normalized to the time of our tests, following the relation in Table 3.

Table 4: Comparison of IVND average results with the main literature methods. Sign - means there is not enough information to compute the results; sign *x* means the method was not tested on the set of instances.

| | | **LL** | | | **BVH** | | | **RP** | | | **IVND** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Type** | **Inst. Size** | #*V* | $\text{gap}_c(\%)$ | t(s) | #*V* | $\text{gap}_c(\%)$ | t(s) | #*V* | $\text{gap}_c(\%)$ | t(s) | #*V* | $\text{gap}_c(\%)$ | t(s) |
| 1 | 100 | 11.21 | -1.17 | 36.95 | 11.10 | 0.00 | 64.81 | 11.10 | 0.01 | 8.19 | 11.52 | 0.35 | 0.56 |
| | 200 | - | - | - | 15.77 | -2.26 | 172.17 | 15.64 | -1.63 | 31.67 | 16.83 | -3.62 | 12.60 |
| | 400 | - | - | - | *x* | *x* | *x* | 30.15 | -1.44 | 108.52 | 32.75 | -1.62 | 76.80 |
| | 600 | - | - | - | 43.93 | -0.88 | 602.22 | 43.72 | -1.15 | 272.31 | 46.82 | 2.01 | 210.64 |
| | 800 | - | - | - | *x* | *x* | *x* | 50.57 | -0.98 | 493.54 | 61.42 | 3.95 | 385.51 |
| | 1000 | - | - | - | *x* | *x* | *x* | 70.15 | -1.45 | 756.14 | 76.10 | 4.94 | 693.80 |
| | Avg. | - | - | - | - | - | - | 36.89 | -1.11 | 278.40 | 40.91 | 1.00 | 229.99 |
| 2 | 100 | 2.96 | 1.57 | 159.72 | 2.96 | 0.01 | 57.68 | 3.00 | 0.57 | 18.48 | 3.03 | 0.93 | 4.77 |
| | 200 | - | - | - | 4.77 | -0.23 | 194.34 | 4.63 | 0.66 | 73.75 | 5.07 | -4.23 | 48.95 |
| | 400 | - | - | - | *x* | *x* | *x* | 8.77 | -0.84 | 243.82 | 10.07 | -1.30 | 182.48 |
| | 600 | - | - | - | 13.37 | -0.89 | 621.80 | 12.5 | -3.74 | 615.91 | 14.42 | 0.16 | 328.12 |
| | 800 | - | - | - | *x* | *x* | *x* | 16.59 | -6.82 | 1,073.52 | 19.28 | 6.28 | 398.96 |
| | 1000 | - | - | - | *x* | *x* | *x* | 20.48 | -3.01 | 1,414.42 | 23.60 | 7.97 | 664.33 |
| | Avg. | - | - | - | - | - | - | 10.99 | -2.20 | 573.32 | 12.41 | 1.64 | 271.27 |
| Avg. | | - | - | - | - | - | - | 23.94 | -1.66 | 425.86 | 26.66 | 1.32 | 250.63 |

Most of the $\text{gap}_c(\%)$ are negative, which as noted before does not imply a better solution. The number of vehicles (#V) is the first objective to be minimized, thus the solution is only better if the number of vehicles is indeed smaller than of the BKS.

Those results show that the IVND, in spite of its simplicity, is able to reach good solutions in competitive time, especially for instances of size 100, 200 and 400. For these, its solution quality and running times are comparable to the ones found in the literature. However, the *state-of-the-art* method could not be outperformed by the IVND. For all of the cases, Ropke and Pisinger [2006] still holds the best results both in solution quality and time.

Note that the proposed IVND always keeps solution feasibility, which is expensive, especially in a problem with many side constraints as the PDPTW. Moreover, this limits the ability to walk in the solution space and move from one region to the other, exploring certain areas that could lead to better solutions. Even though the precedence constraint feasibility, as well as the capacity feasibility, are important in a real-world situation, the time windows could be allowed to become infeasible, and still be acceptable from the real point of view, since delays are likely to occur up to some extent in real applications.

## 5. Conclusion and Future Works

This work proposed an *Iterated Variable Neighborhood Descent* algorithm to solve the *Pickup and Delivery Problem with Time Windows*, which is a combinatorial $\mathcal{NP}$-Hard problem with practical applications. The method has been designed to be as simple as possible, and in fact the *Iterated Local Search* framework suits this purpose. Further, experiments with the standard set of instances for the problem have been carried out to check the performance of the algorithm.

The proposed IVND was able to perform well for some of the literature instances. It has been shown how an ILS based algorithm, with best improvement strategy and exploring only feasible regions was able to scale on the PDPTW. Results support the fact that it scales even for the largest instances in the standard set, and that the average running time is usually low enough for practical uses. Although, it was not able to outperform the *state-of-the-art* method.

In future works, we plan on trying new solution approaches, such as matheuristics. We also plan on exploring the infeasible regions of the solution space to analyze the impact on solution quality. At last, throughout the interaction with the partner company we plan on studying the same variation with *soft* time windows, because it is important from a real point of view, as delays are likely to occur in real-world scenarios.

## 6. Acknowledgments

The authors would like to acknowledge the scholarship support provided by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq), and the partnership of company uMov.me during this project.

## References

Bent, R. and Van Hentenryck, P. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33:875–893.

Li, H. and Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(02):173–186.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, Université Libre de Bruxelles.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2010). *Iterated Local Search: Framework and Applications*, p. 363–397. Springer US, Boston, MA.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Nanry, W. P. and Barnes, J. W. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472.

Sartori, C. S. (2016). Optimizing solutions for the pickup and delivery problem. Bachelor thesis in computer science, UFRGS. Available at: `http://hdl.handle.net/10183/150897`.

SINTEF (2017). Information on li & lim instances. Available at: `https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/`. Accessed: 2017-01-19.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265.