An Evolutionary Algorithm and a Variable Neighborhood Descent Algorithm for the Single Vehicle Problem with Deliveries and Selective Pickups

Bruno Petrato Bruck
Departamento de Informática - Universidade Federal de Viçosa
Av. P. H. Rolfs, s/n - DPI, Campus UFV - CEP 36570-000 Viçosa, MG - Brazil
bruno.p.bruck@gmail.com

André Gustavo dos Santos
Departamento de Informática - Universidade Federal de Viçosa
Av. P. H. Rolfs, s/n - DPI, Campus UFV - CEP 36570-000 Viçosa, MG - Brazil
andre@dpi.ufv.br

ABSTRACT

This paper presents two approaches for the Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP), where a vehicle departs loaded from the depot and must visit every delivery customer serving a certain demand, but may not visit all the pickup customers, only the ones that are profitable, since for every pickup made, there is an associated revenue. The objective is to find a minimal cost and feasible route, where the cost is defined as the total travel costs minus the total revenue earned with the pickups performed. Despite the many real applications, there are not many researchs on this problem. We propose an improved Evolutionary (EA) Algorithm whose crossover and mutation operators use data mining strategies to keep trac of good features of the individuals of the population. This EA has an intensification phase, which uses a local search procedure to improve the quality of the individuals and new solutions are introduced regularly to avoid premature convergence. The second approach is a Variable Neighborhood Descent Algorithm (VND) that uses constructive algorithms to generate good solutions to start from. The algorithms were tested with a benchmark of 68 instances from the literature, and the results compared to other papers. The results show that this version of the EA is better than the previous one and finds 7 optimal solutions and some new best values. The VND approach, although is less time costly is not able to find the same or better results than the EA in most cases.

KEYWORDS. Vehicle Routing Problem, Evolutionary Algorithm, Variable Neighborhood Descent, Single Vehicle Problem with Deliveries and Selective Pickups, Combinatorial Optimization.

1 Introduction

In the Single Vehicle Routing Problem with Deliveries and Selective Pickups (SVRPDSP) there are a set of customers to be served and a depot. Each customer has a certain demand of goods to be delivered and may have a demand of goods to be picked up. While the former is mandatory, the latter is not, but if served generates a revenue. Therefore it is possible to perform only profitable pickups. The vehicle departs from the depot loaded and shall perform a route visiting the customers, serving all delivery demands, but not necessarily all the pickup demands. A route is feasible if it begins and ends at the depot and all delivery demands have been served without exceeding the vehicle capacity at any time. The travel cost of a route is the sum of the travel costs between consecutive customers (or depot) visited in the route, and the net cost of a route is its travel cost minus the total revenue of all the pickups served. The SVRPDSP problem consists in finding a feasible route with minimal net cost.

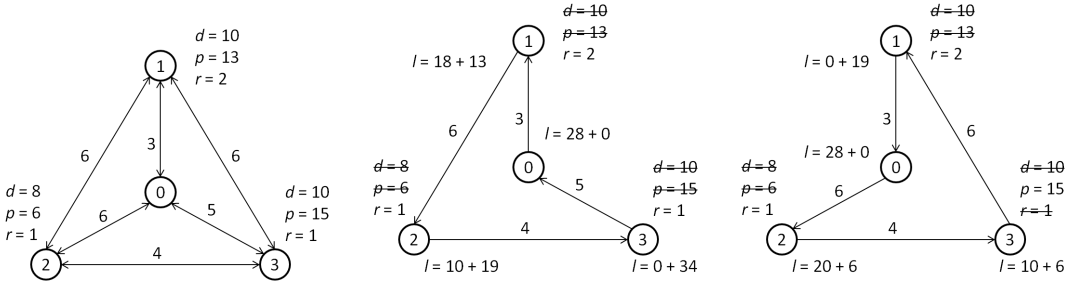
The problem may be conveniently described using a complete directed graph $G = \{V, A\}$, where $V = \{0, 1, ..., n\}$ is a vertex set representing the depot (vertex 0) and the n customers, and $\{A = (i, j) : i, j \in V, i \neq j\}$ is a set of arcs. Each customer i has a delivery demand $d_i > 0$, a pickup demand $p_i \geq 0$, and a revenue $r_i > 0$. Each arc $(i, j) \in A$ has a travel cost c_{ij}. The vehicle has a capacity Q, with $Q \geq \sum_{i \in V} d_i$. The problem is NP-hard, since TSP is a special case (when $p_i = 0, \forall i \in V$).

The SVRPDSP arises in several pratical contexts, such as in drink factories where there are delivery demands of bottles to supermarkets and also demands of empty bottles to be picked up and returned to the factory [1]. Logistic services companies are also a good example where such problem can arise, since while there are delivery demands of packages, there is also a constant demand for picking up products to be deliveried elsewhere, but first they have to go to the depot. Furthermore, another pratical application for this problem is in eletronic and batery manufactories, which are being responsible for picking up their broken and used products that otherwise would go as normal trash and pollute. One can argue that at some point all the pickups would have to be fulfilled, but not necessarily in the same route, as another vehicle can serve them in another route latter, or even a third party service can be used to collect these pickups, what could prove to be less costly for the company than send its own vehicle only to pick up the remainder goods.

Note that there is no assumption on the number of times a customer may be visited. The delivery and pickup demands may be served simultaneously in a single visit, or separately, in two different visits, in case there is not enough space to perform the pickup simultaneously. A second visit may be profitable, in case the revenue is greater than the aditional travel cost. Note that, a pickup demand cannot be partially collected, i.e., either it is fully collected or not collected at all.

Figure 1 shows a small example with 3 customers and the optimal solution for two different vehicle capacities. Note that in the second case not all pickup demands are collected. The travel cost in 1(b) is 18 and the total revenue 4, then the net cost of the solution is 18 - 4 = 14. The solution in 1(c) has a net cost of 19 - 3 = 16.

The remainder of this paper is organized as follows: section 2 extend the classification of the SVRPDSP and review the literature presenting previous research on this problem and their results; The proposed Evolutionary Algorithm (EA) and Variable Neighborhood Descent (VND) methods, along with some heuristic procedures are presented and described in section 3. Furthermore, section 4 describes the parameter calibration of the algorithms and analyses the experimental results, including new best solutions not previously found in the literature. Finally, section 5 present some conclusions regarding the results and propose some further research.



(a) Example with 3 customers (b) Optimal solution for $Q = 34$ (c) Optimal solution for $Q = 28$

Figure 1: A small example and its optimal solutions for different vehicle capacities. Demand and pickup served are strikethrough, as well as revenue not collected

2 Related Literature

The first mention of the problem was made by [2], citing it as an extension of the One-to-Many-to-One Single Vehicle Pickup and Delivery Problem that had received limited attention, despite its many practical applications and its similarity with other vehicle routing problems. To our knowledge there are only three papers and a master's thesis exploring it. They propose integer linear programming formulations able to solve small instances to optimality, and heuristics based on Tabu Search, General Variable Neighborhood Search and a Evolutionary Algorithm (EA) to solve other sets of instances. The present paper extends the approach of the EA and also proposes a Variable Neighborhood Descent (VND) to compare with the literature results.

In [3], Süral and Bookbinder, present a classification for this problem using the notation $\alpha/\beta/\gamma$, where α is the number of vehicles (1 for single and M for multiple), β denotes the pickup service options ($must$ or $free$ if the pickup is ,respectively, mandatory or optional) and γ is the precedence order for visiting two customers ($prec$ if all deliveries must precede the pickups, or any if they can be visited in any order). Therefore, by this notation the SVRPDSP can be described as the $1/free/any$ problem. Their work is the first approach of the problem, where a mixed integer linear programming formulation is proposed along with some improvements on the constraints to strength the formulation, such as constraint disaggregation, coefficient improvement, cover and logical inequalities, and lifted subtour elimination constraints. They modified 24 instances from the literature, with sizes of 10, 20 and 30 customers, to test their formulation. These instances were adapted for the SVRPDSP by setting some of the delivery demands as pickups in 3 different ways (20% of the customers reset as pickups, then 30% and 40%), generating a total of 72 instances, which were tested with some combinations of the formulation and the improvements resulting in about 75% of the instances optimally solved in a reasonable computational time for the best combination.

[4] proposed another MILP formulation and a Tabu Search metaheuristic (TS) to compare their efficiency with some classical constructive heuristics. They gathered 17 instances from the Capacitaded Vehicle Routing Problem from the VRPLIB [5]. As these instances only have delivery demands they had to generate the pickup demands and profit values. The pickup demands were generated based on the values of the delivery demands and the profit values generated based on a formula that results in values proportionals to the average cost of an instance, multiplied by a factor ω. They considered 4 values for the parameter ω, therefore resulting in a total of 68 instances. Computational tests with classical heuristics yielded gap values ranging from very small to high values, so to overcome this scaling problem they split the gap measure into two other measures, called $cgap$ and $pgap$. The former is based on the routing cost and it is defined as $cgap = 100(\bar{c} - \underline{c})/\underline{c}$ where $\bar{c}$ is the routing cost of the solution and $\underline{c}$ the routing cost lower bound.

The latter is based only in the revenue generated by the pickup demands served and it is defined as $pgap = 100(\bar{p} - \underline{p})/\bar{p}$, where $\bar{p}$ is the profit upper bound and $\underline{p}$ the profit value of the solution. The TS metaheuristic was then, tested with the same instances, yieling better gap values than some classical heuristics with the average value for $cgap = 4.03$ and $pgap = 0.36$, against $cgap = 10.57$ and $pgap = 2.01$ from the heuristics.

A hybrid metaheuristic was proposed by [6]. At first it solves the Travelling Salesman Problem (TSP) and the Knapsack problem for the instance and then uses their solutions to create an initial solution for a General Variable Neighborhood Search algorithm (GVNS). The lower bound is calculated in the same way as proposed by [4], using the values of the optimal solutions of the TSP and Knapsack formulations. The autors were able to significantly improve the solutions found by [4], including 3 optimal solutions as its values are equal to their respective lower bound.

Recently a new hybrid approach was proposed by [7], combining an Evolutionary Algorithm with a data mining strategy in order to track good patterns (sequence of customers) in solutions found along the execution, using such information to guide the crossover and mutation operators. This data mining strategy works with sequences of customers, called patterns. It also uses a Variable Neighborhood Search algorithm (VNS) in the intensification phase, in order to improve the quality of some individuals in the population. The results found are competitive with the ones in [6] and new optimal solutions were found. One of the contributions of the present work is an extention of this algorithm by: proposing a new evaluation criteria for the patterns considering in addition to the frequency, the average cost of the solutions where a given pattern appears; the addition of a repair procedure for unfeasible solutions and an improvement procedure; and another local search procedure replacing the VNS in the previous approach. All these new features are described in details in section 3.

3 Proposed Heuristics

Before going through the details of the Evolutionary Algorithm and VND, it is necessary to explain a few heuristic procedures used by these methods, such as the constructive algorithms, a repair procedure for unfeasible solutions and an optimization procedure, to increase the quality of given solutions based on the features of the problem.

3.1 Constructive algorithms

Four constructive algorithms were implemented, where the first two usually generate good quality solutions and the remaining two generate medium and poor quality solutions. The only case in this work where poor quality solutions are needed is in the EA metaheuristic, where there must be diversity among the individuals of the population.

3.1.1 tspBased

In this method, at first, the Traveling Salesman Problem (TSP) is solved for the instance, considering only the delivery customers, using the software IBM ILOG CPLEX 12.2. Then the optimal route is used as base for inserting the pickup customers in the best possible positions, generating a solution for the SVRPDSP. It differs from the method presented in [6] by the addition of a *Restricted Candidate List* (RCL) with size $rclSize$. This list works by keeping at each iteration the best candidates to insert into the current route and randomily selecting one to insert. This ensures that different solutions can be generated by this algorithm when $rclSize > 1$.

3.1.2 tspKnapsackBased

The only difference from the previous method is that it only inserts into the route of the TSP solution the pickup customers that belong to the optimal solution of the Knapsack Problem considering only the pickup customers. As the *tspBased* it usually yields good quality solutions.

3.1.3 nearestNeighbor

At first the solution contains only the depot, and one by one the customers are inserted into the route. The nearest customers are inserted first. It differs from the classical nearest neighborhood heuristic by considering a RCL with size equal to $0.1n$, where n is the number of customers of a given instance. Through tests it was observed that this algorithm yields best results the greedier it is and the value $0.1n$ has proven to balance well the quality and the generation of different solutions. Is is important to emphasise that the importance of this algorithm in this work is only generating medium and poor quality solutions, so further tests with the RCL size were not done.

3.1.4 cheapestInsertion

The solution has, initially, only the depot, and at each iteration a new customer is inserted in the route on a good position considering the criteria cost-benefit. It has the same RCL as in *nearestNeighbor* with size of $0.1n$.

3.2 Repair heuristic

Since it is common to generate unfeasible solutions during local search procedures there must be a procedure to repair solutions. The proposed method works in two phases. In the first, all the delivery customers not served are inserted in the best possible position of the current route, according to cost value, ensuring that the constraint of serving all delivery demands is satisfied. For the second phase, if the solution is still unfeasible, the route is analized to find where there are pickups overweighing the vehicle, removing them from the route and thus ensuring feasibility.

As one can notice, pickup customers are, frequently, removed from the route in the process, but the fact that the position they were was making the solution unfeasible does not mean there is not a valid and profitable position for them in the current route. Therefore, after repairing a solution, this condition is verified and the solution may be improved.

As the repair procedure is applied to every unfeasible solution generated it will not be mentioned from now on. Consider its use implicit in all solutions.

3.3 Improvement heuristic

If a pickup demand is served at the position i in a given feasible route, it means that at any position greater than i this demand can be fulfilled without making the route unfeasible, since if there was enough space to perform the pickup before, there will be, for certain, space for it at any time after, considering that the delivery load will always decrease. Therefore if a customers' pickup demand is performed before its delivery demand, this solution is not the optimal, since both demands can be fulfilled simultaneously with cost zero. This procedure takes advantage of this property to improve the cost of a given solution and it is called right after the repair procedure.

3.4 Variable Neighborhood Descent

This approach simply combines the initial generation of a solution using the constructives *tspBased* and *tspKnapsackBased* detailed in section 3.1 with a classical Variable Neighborhood

Descent Algorithm([8]), which uses as neighborhood structures: 2-opt, swap, 2-Or-Opt, 3-Or-Opt, 4-Or-Opt.

3.5 Evolutionary Algorithm

Basically this Evolutionary Algorithm is an improved version of the one presented in [7] and has five distinct phases, which are the initialization, crossover, mutation, intensification and diversification (Algorithm 1).

In the Initialization phase, the first population is generated by randomly selecting a constructive algorithm among the four types available, described in detais in section 3.1. In case the choosen algorithm is either *tspBased* or *tspKnapsackBased*, the $rclSize$ must be set. If it is the first time this type of constructive is called, the $rclSize$ is set as 1 (greed solution), otherwise it is set as 2. As equal elements are not allowed at any time in the population the constructives are called until the population has been fully generated with mutually different elements. As one can notice, it may be necessary many calls to the constructive algorithms to assure this constraint. This can decrease its performance and then, should be avoided. In order to do so, if a new element is not sucessfully inserted into the population within 3 iterations, the value of $rclSize$ is set as 3 increasing the chances of generating a different element, in case *tspbased* or *tspKnapsackBased* is called.

The second phase is the Crossover procedure, in which elements of the population are combined to generate new solutions so as to improve the quality of the individuals and promote, at some degree, diversification.

In order to produce better quality individuals the crossover procedure needs a criteria to evaluate the features of a given individual and decide which ones are the best. Therefore, before continuing explaining the crossover procedure we must go through the detais of how our EA keeps track of good attibutes .

Basically we use a Data Mining strategy, in which every new individual has its route analized to extract patterns (sequence of customers) within a given range [$minPatternSize$, $maxPattern Size$]. As in [7], each pattern found is stored in a structure called $patternsList$ along with the frequency it has appeared in the solutions already analized. In addition to these informations, we propose to keep record of the average cost of the route in which a pattern was found so as to improve the robustness of the evaluation criteria that decides how good a pattern is. Therefore, we now have two types of data to evaluate a pattern and since cost value is usually much higher than the frequency value, this data must be normalized. Lets call $nFrequency$ the normalized frequency value, $nAvgCost$ the normalized average cost and $qualityIndex = (1 - nAvgCost) + nFrequency$, which will be the value used to evaluate the patterns, since it considers both measures. The closer to 2 the better.

Having understood how the $patternsList$ works, we can now detail the crossover procedure. At first two elements of the population are randomly selected and the patterns of both parents extracted along with their respective $qualityIndex$ value. Let us refer to the parents as $P1$ and $P2$. Two patterns of $P2$ are chosen, giving more probability for the ones with higher $qualityIndex$ values, and they are forced to appear in the best possible position of $P2$'s route generating two new individuals. The same is done to the parent $P2$ and so for each recombination 4 children are generated. This process is performed $popSize/2$ times in each iteration.

After having completely generated the child population it is time to combine it with the parent population to decide which individuals will be held for the next iteration. As in [7], the new population is half composed of the best elements of both populations. A quarter is selected through tournaments, where the best of two elements (one from each population) wins and is inserted in the new population. The other quarter is composed of randomly selected individuals. This 3 stage process guarantee the diversity, which is a very important factor for the success of this kind of algorithm.

In the mutation phase, all patterns are extracted from the $patternsList$ along with the $qualityIndex$ values. Then, one pattern is selected to be forced into a randomly selected individual, giving more chances the higher the value of $qualityIndex$. In case the new individual is better than its former, it replaces the latter. It is important to emphasize here that the function which forces the patterns to appear in a given individual never generates unfeasible solutions, since it uses the repair procedure, described in section 3.2.

In the intensification procedure an individual is ramdomly selected and its route shaken $intensity$ times, using one of five different structures to shake a route, also ramdomly selected, which are 2-Opt, swap, 1-Or-Opt, 2-Or-Opt and 3-Or-Opt. Since these are classical structures we are not going into the details of them. Initially $intensity = 1$ and at each 5 iterations without improvement on the best solution, its value is increased by one. When the best solution is updated the value is reset to 1. A local search (Algorithm 2) procedure is then, applied to the shaken individual. This procedure randomly select a neighborhood structure and completely explores it, returning the local minima. It uses the same structures used by the shake method.

Finally the diversification phase is simply replacing half of the population (the worst individuals) by new ones, generated by the constructive algorithms at each 10 iterations.

Algorithm 1 Evolutionary Algorithm pseudocode

```
1:  procedure EVOLUTIONATY(popSize, numIt)
2:      //Initialization
3:      itsWithoutImpr ← 0
4:      intensity ← 1            // When a new best solution is found it is reset to 1
5:      pop ← ∅                  // population
6:      bestSol ← any with cost = ∞          // best solution
7:      for i ← 1 to popSize do
8:          rclSize ← random from {1, 2, 3, 4}
9:          constructive ← random constructive heuristic
10:         Generate individual using constructive and rclSize
11:         pop ← pop ∪ individual
12:         Update bestSol if necessary
13:     end for
14:     Compute patternsList list
15:
16:     //EA main loop
17:     for i ← 1 to numIt do
18:         //Crossover
19:         childPop ← ∅
20:         for j ← 1 to popSize/2 do
21:             parents ← random 2 individuals from pop
22:             //Do this for each of the 2 parents
23:             Select maxSons patterns from one parent
24:             child ← force patterns into the other parent
25:             if child is feasible then
26:                 childPop ← childPop ∪ child
27:                 Update patternsList list using child
28:                 Update bestSol if necessary
29:             end if
30:         end for
```

```
        newPop ← best popSize/2 from childPop ∪ pop
        Add popSize/4 from tournament(pop,childPop)
        Add popSize/4 random from childPop ∪ pop

        //Mutation
        for j ← 1 to popSize/2 do
            individual ← random from pop
            newMut ← mutate individual
            Update patternsList list using newMut
            Update bestSol if necessary
        end for

        //Intensificate
        for j ← 1 to popSize/5 do
            ind ← random individual from pop
            for i ← 0 to intensity do
                Shake ind
            end for
            ind ← LocalSearch to ind
            Update patternsList list using ind
            Update bestSol if necessary
        end for

        //Diversification
        if i mod 10 = 0 then          // Every 10 iterations
            for j ← popSize − 1 to popSize/2 do          // Replace the worst
                newInd ← random by constructive
                Update patternsList list using newInd
            end for
        end if

        if itsWithoutImpr = 5 then
            itsWithoutImpr ← 0
            intensity + +
        end if
    end for
end procedure
```

Algorithm 2 Local Search pseudocode

```
1: procedure LOCALSEARCH(individual)
2:     neighborhood ← randomly select from {2-opt, swap, 2-Or-Opt, 3-Or-Opt, 4-Or-Opt}
3:     newind ← local optima of neighborhood using individual
4:     return newind
5: end procedure
```

4 Computational results

In order to test the proposed approaches we have used the 68 instances proposed by [4] and used by [6] and [7]. These are derived from a set of instance of the Capacitaded Vehicle Routing Problem of the VRPLIB [5].

The results were obtained on Intel(R) Core(TM) i7 3.07GHz machine with 6Gb RAM, running the operating system Ubuntu 12.04. All codes were made in C++, and the MILP formulations to obtain the lower bounds were solved using IBM ILOG CPLEX 12.2 under the academic license.

The lower bound is calculated as proposed in [4]: TSP solution for delivery customers minus Knapsack solution for pickup customers. To achieve this lower bound the vehicle must visit the customers in an optimal TSP order doing delivery and pickup simultaneously (for those customers in a Knapsack optimal solution). This is a valid lower bound since there is no way to attend the delivery customers with a lower travel cost, and there is no way to get more revenue from the pickup customers. For some instances there is no feasible solution reaching this lower bound, because of the vehicle capacity. If it is not possible to do a simultaneous pickup, the vehicle must return to the customer a second time (increasing the travel cost) or do a pickup in another customer instead of the best ones (decreasing the total revenue).

To measure the quality of the solutions generated by the constructive algorithms *tspBased* and *tspKnapsackBased*, they were tested with three values for the the RCL (1, 2 and 3) and each configuration run five times, except the ones with $RCL = 1$, where the algorithm was run only once, since it is deterministic. A total of 52 out of the 68 instances were used for these tests (16 to 51 customers). The results, presented in Figure 2, show that these algorithms frequently generate very good solutions, including optimal for some instances, which is an odd feature for a construtive algorithm, but could be due to using information regarding the optimal solutions of the TSP and Knapsack problems, that are also used to determine a lower bound. To simplify the analysis, the figure only shows the *tspBased* with $RCL = 1$ and *tspKnapsackbased* with $RCL = 1, 2$, since the other configurations proved to be worse in an ANOVA (Analysis of Variance) using the software Minitab 16.

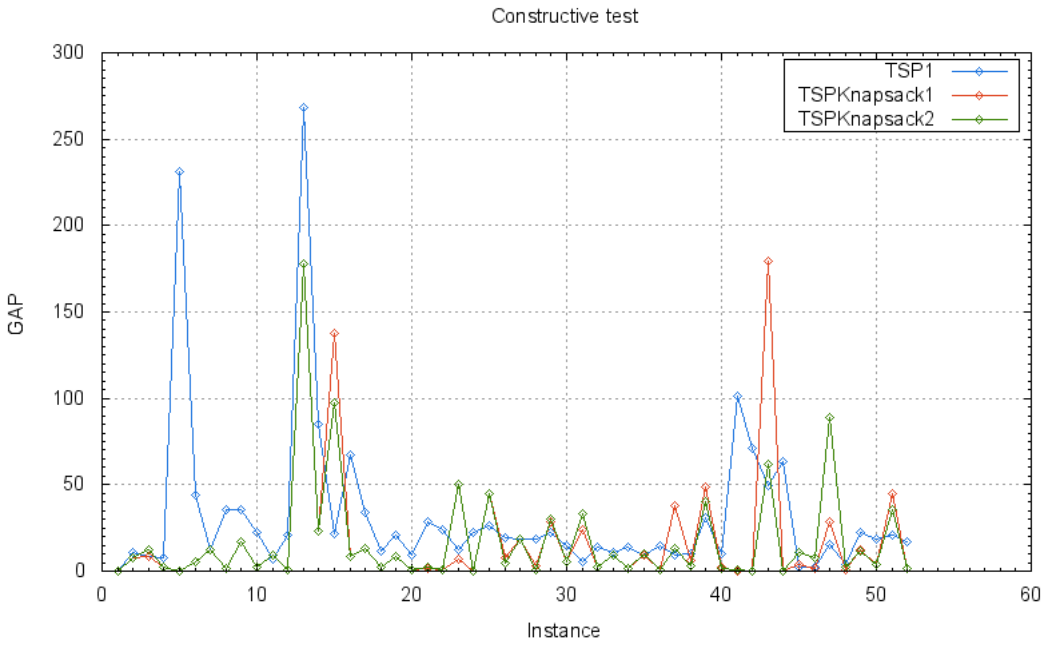


Figure 2: Best GAP values for the constructives in 5 runs

To calibrate the parameters, the algorithm was run in a subset of 16 instances with all combinations of $popSize = \{10, 20, 30\}$ and $numIt = \{10, 20, 30\}$. To know which combination yields the best results we have done an ANOVA (Analysis of Variance) on the average gap values using the software Minitab 16. The result can be seen in Figure 3, where one can notice that there is no combination that is statistically proven to be the best or better than other one with 95% of confidence. Therefore we decide to choose a combination that balance the variation, the best average gap value and the run time. Although the combination `PS_30_IT_30` has the best gap value it needs more time and has almost the same variation as the combination `PS_20_IT_20`, which was chosen to be the configuration for the EA.

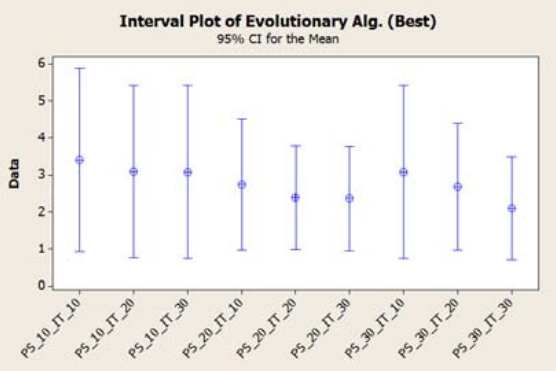


Figure 3: ANOVA for the parameter calibration

Table 1 compares the algorithms of the present work to the ones of the literature grouped by instance type. The measure used is the average gap to the theoretical lower bound, the same used in [4]. All approaches clearly outmatch the Tabu Search algorithm proposed by [4]. The VND algorithm does not outmatch neither the EA nor the GVNS, but is competitive. The results of the new EA are better than those of the previous one, therefore proving the effectiveness of the modifications proposed. Furthermore, comparing to the GVNS, the results of our EA approach are better for instances of type *half*, and close for the other types, although this is not clear for instance of type *p_two*. This is due to the very high gap value of 262.53% for instance *076_B_p_two*, which greatly increses the average gap. If we exclude this instance, the values for the best and average gaps of the EA decreases to 7.57% and 8.81%, respectively, and GNVS to 6.20% and 8.58%. A similar situation occours for the VND.

Table 1: Comparison of the Algorithms with the literature considering the gap (%)

Algorithm	`half`		`one`		`p_two`		`two`	
	best	avg	best	avg	best	avg	best	avg
TS [4]	28.12	-	5.38	-	86.78	-	2.19	-
EA [7]	4.26	6.83	1.83	2.12	19.26	25	0.66	0.8
EA	2.38	4.15	1.32	1.68	22.56	25.11	0.47	0.75
VND	4.14	4.15	2.07	2.15	28.72	28.72	0.75	0.78
GVNS [6]	3.44	6.35	1.04	1.28	9.65	13.25	0.39	0.47

Table 2 details the results of the methods proposed here and the best algorithm of the literature (GNVS) for all 68 instances. The EA and the VND metaheuristics were run ten times each with four hours of time limit. The first column lists the name of the instances. The next six columns show, respectively, the value of best solution found and the average value of the ten runs, for each one of the methods. The last column shows the theoretical lower bound. Solutions obtained by one algorithm that are better or equal to the ones obtained by the others are highlighted in bold face. Solutions marked with '*' are certainly optimals, since they reach the lower bound.

Note that in Table 2, regarding the best solutions found, no algorithm outmatches the others,

as there is at least one instance where one outperforms the others. Nevertheless, one can notice that the new EA found the same or better results than the GVNS for more than a half of the instances, including 17 new solutions. Although the VND was not as efficient as the EA, it has found about 1/3 of solutions equal or better than the GVNS. Concerning optimal solutions, EA clearly outperforms the others, as it was the one that found more optimal solutions: 7 in total, including all optimal found by the others. It is important to notice that the EA found the optimal solution in average, i.e. in all 10 runs, for 3 instances, the VND for 5 instances and the GVNS for none.

5 Conclusion

In the present paper two approaches were tested and compared against the results of the literature using 68 benchmark instances. The EA algorithm proved to be better than the VND in most of the cases, although for some instances, the VND outmatched all algorithms in the literature, including the EA.

Regarding the comparison between the EA and the GVNS, the results were balanced, but EA found more optimal solutions, and also more better solutions in average. The only 3 optimal solutions found by GVNS were also found by EA.

Acknowledgments

This project was supported by Capes - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, and partially supported by Gapso and Sydle.

References

[1] J. Privé, J. Renaud, F. Boctor and G. Laporte. "Solving a vehicle-routing problem arising in soft-drink distribution" J. of the Operational Research Society. vol 57, pp. 1045-1052, 2006

[2] G. Laporte, I. Gribkovskaia. "One-to-Many-to-One Single Vehicle Pickup and Delivery Problems", *The Vehicle Routing Problems: Latest Advances and New Challenges*, Golden, Raghavan, Wasil (editors), Springer, 2008

[3] H. Süral and J.H. Bookbinder. "The single-vehicle routing problem with unrestricted backhauls", *Networks* vol 41, pp. 127-136, 2003

[4] I. Gribkovskaia, G. Laporte and A. Shyshou. "The single vehicle routing problem with deliveries and selective pickups", *Computers & Operations Research* vol 35, pp. 2908-2924, 2008

[5] D. Vigo. "VRPLIB: A Vehicle Routing Problem LIBrary". [Online]. Avaiable: http://www.or.deis.unibo.it/research_pages/ORinstances/VRP LIB/VRPLIB.html

[6] I. M. Coelho. "Contribuições para o problema de roteamento de veículos de rota única com entrega obrigatórias e coletas seletivas", Master thesis, Programa de Pós-Graduação em Computação, Universidade Federal Fluminense, 2011

[7] B. P. Bruck, A. G. Santos and J. E. C. Arroyo. "Hybrid metaheuristic for the single vehicle routing problem with deliveries and selective pickups". Proceedings of the *2012 IEEE Congress on Evolutionary Computation*, pp. 910-917. IEEE Press, Brisbane, Australia, 2012

[8] P. Hansen and C, N. Mladenović. "Variable neighborhood search: Principles and applications", *European Journal of Operational Research* vol 130, pp. 449-467, 2001

Table 2: Results of the EA and VND algorithms

instance	EA		VND		GVNS		Lower Bound
	best	average	best	average	best	average	
016_B_half	**36.60***	**36.60***	**36.60***	**36.60***	39.86	40.61	36.60
016_B_one	**-150.73**	-148.95	-145.44	-145.44	**-150.73**	**-150.73**	-155.41
016_B_p_two	133.80	140.41	142.03	142.03	**132.28**	**133.43**	130.99
016_B_two	-536.06	-532.80	-530.84	-530.84	**-536.13**	**-536.13**	-540.81
021_B_half	**-20.16***	**-20.16***	**-20.16***	**-20.16***	-18.78	-13.96	-20.16
021_B_one	**-307.80**	-304.54	-300.00	-300.00	**-307.80**	**-307.21**	-316.09
021_B_p_two	**137.59**	144.59	147.03	147.03	**137.59**	**141.17**	132.21
021_B_two	**-901.28**	-895.39	-893.48	-893.48	**-901.28**	**-900.69**	-909.57
022_B_half	-62.40	-57.41	-56.86	-56.86	**-62.63**	**-58.59**	-64.97
022_B_one	**-421.04**	**-421.04**	**-421.04**	**-421.04**	**-421.04**	**-421.04**	-429.15
022_B_p_two	124.25	**124.29**	124.33	124.33	**123.59**	124.86	116.01
022_B_two	**-1,149.38**	-1,149.37	**-1,149.38**	**-1,149.38**	**-1,149.38**	**-1,149.38**	-1157.49
023_B_half	-80.01	-79.83	-61.67	-61.67	**-80.95**	**-80.76**	-94.06
023_B_one	-697.41	-688.26	-640.23	-640.23	**-698.35**	**-698.26**	-711.46
023_B_p_two	**269.08**	273.72	291.07	291.07	269.86	**273.23**	260.88
023_B_two	-1,932.16	-1,926.52	-1,874.98	-1,874.98	**-1,933.10**	**-1,932.91**	-1946.21
026_B_half	**-92.41**	**-92.41**	**-92.41**	**-92.41**	-87.87	-85.53	-92.47
026_B_one	**-497.17**	-495.91	-495.87	-495.87	**-497.17**	**-497.17**	-504.40
026_B_p_two	**146.11**	**146.14**	146.34	146.34	146.90	147.18	139.67
026_B_two	**-1,334.26**	-1,333.48	-1,332.96	-1,332.96	**-1,334.26**	**-1,334.26**	-1341.49
030_B_half	**-378.64**	**-378.64**	**-378.64**	**-378.64**	**-378.64**	-370.69	-382.80
030_B_one	**-1,152.07**	**-1,152.07**	**-1,152.07**	**-1,152.07**	**-1,152.07**	-1,146.64	-1156.23
030_B_p_two	**82.29**	**82.29**	85.49	85.49	**82.29**	85.69	81.33
030_B_two	**-2,699.00**	**-2,699.00**	**-2,699.00**	**-2,699.00**	**-2,699.00**	-2,697.26	-2703.16
031_B_half	**-89.15**	**-88.39**	-85.66	-85.51	-87.06	-84.90	-91.79
031_B_one	**-511.07**	**-510.65**	-507.82	-507.69	**-511.07**	-508.04	-514.05
031_B_p_two	**123.15**	124.35	131.07	131.07	**123.15**	**123.39**	115.52
031_B_two	**-1,355.59**	**-1,354.83**	-1,352.34	-1,352.21	-1,354.49	-1,352.73	-1358.57
033_B_half	**-150.73**	-146.47	**-150.73**	**-150.73**	**-150.73**	-146.01	-157.09
033_B_one	**-765.79**	**-765.10**	-753.55	-753.55	**-765.79**	-761.80	-778.21
033_B_p_two	**194.61**	**195.60**	198.42	198.42	199.17	202.08	188.44
033_B_two	-2,007.28	**-2,005.94**	-1,995.74	-1,995.74	**-2,007.98**	-2,003.99	-2020.40
036_B_half	**-128.26**	**-128.26**	**-128.26**	**-128.26**	-127.06	-124.65	-128.53
036_B_one	**-624.41**	**-624.41**	**-624.41**	**-624.41**	-616.12	-610.63	-624.67
036_B_p_two	132.60	**132.93**	133.56	133.56	**130.42**	136.41	121.94
036_B_two	**-1,616.64**	**-1,616.64**	**-1,616.64**	**-1,616.64**	-1,608.35	-1,604.65	-1616.90
041_B_half	**-185.75**	**-185.75**	-181.85	-181.85	-183.86	-179.36	-186.35
041_B_one	-758.06	-756.29	-757.26	-749.98	**-760.52**	**-758.60**	-767.97
041_B_p_two	111.69	**112.89**	131.86	131.86	**109.48**	115.16	100.89
041_B_two	-1,922.71	-1,921.02	-1,920.60	-1,913.59	**-1,923.86**	**-1,921.50**	-1931.31
045_B_half	**-491.15***	**-491.15***	**-491.15***	**-491.15***	**-491.15***	-490.52	-491.15
045_B_one	**-1,648.51***	-1,647.63	**-1,648.51***	**-1,648.51***	**-1,648.51***	-1,647.88	-1648.51
045_B_p_two	**198.04***	**198.57**	201.22	201.22	199.82	201.85	198.04
045_B_two	**-3,963.32***	-3,962.41	**-3,963.32***	**-3,963.32***	**-3,963.32***	-3,962.69	-3963.32
048_B_half	-36,786.80	-36,786.80	**-37,205.50**	**-37,205.50**	-37,200.62	-37,200.62	-37753.00
048_B_one	**-107,059.00***	**-106,796.00**	-106,423.00	-106,423.00	-107,058.78	-106,682.88	-107059.00
048_B_p_two	-3,830.83	-3,829.53	-4,140.10	**-4,140.10**	**-4,244.69**	-4,122.47	-4797.03
048_B_two	-247,865.00	-247,762.00	-247,663.00	-247,663.00	**-247,946.25**	**-247,871.39**	-248298.00
051_B_half	**-310.84**	**-308.87**	-305.94	-305.94	-310.24	-307.35	-320.61
051_B_one	**-1,088.26**	**-1,085.42**	-1,077.35	-1,077.19	-1,086.52	-1,085.09	-1098.86
051_B_p_two	**123.60**	**125.41**	131.14	131.14	126.70	127.99	116.58
051_B_two	-2,644.71	**-2,641.86**	-2,633.79	-2,633.63	**-2,645.35**	-2,640.95	-2655.30
072_B_half	**-408.09**	-406.36	-406.87	**-406.87**	-406.57	-404.96	-409.78
072_B_one	-984.20	-978.86	**-1,024.33**	-1,020.47	-1,024.04	**-1,023.07**	-1027.24
072_B_p_two	-33.43	-32.31	-30.16	-30.16	**-36.03**	**-35.85**	-39.24
072_B_two	-2,247.83	-2,197.82	**-2,259.30**	-2,255.44	-2,259.01	**-2,259.01**	-2262.21
076_B_half	-574.04	-472.20	-576.66	**-576.62**	**-579.25**	-576.19	-579.52
076_B_one	-1,731.91	-1,713.61	-1,752.71	-1,752.35	**-1,758.92**	**-1,753.27**	-1759.19
076_B_p_two	51.95	55.30	57.55	57.55	**23.62**	**26.95**	14.33
076_B_two	-4,102.38	-4,076.64	-4,112.02	-4,111.66	**-4,118.23**	**-4,113.12**	-4118.50
101_B_half	-909.32	-909.32	**-910.73**	**-910.73**	-906.79	-900.82	-922.71
101_B_one	-2,538.99	**-2,538.99**	-2,538.99	**-2,538.99**	**-2,541.35**	-2,536.94	-2552.38
101_B_p_two	-49.55	**-49.55**	-49.55	**-49.55**	**-53.89**	-47.44	-66.59
101_B_two	-5,798.30	**-5,798.30**	-5,798.30	**-5,798.30**	**-5,800.66**	-5,793.33	-5811.69
111_B_half	-1,322.56	-1,322.56	-1,380.63	**-1,380.63**	**-1,384.06**	-1,380.26	-1386.67
111_B_one	-3,309.57	-3,309.57	-3,314.79	**-3,314.79**	**-3,315.78**	-3,312.67	-3320.83
111_B_p_two	-249.66	-249.66	-249.73	-249.73	**-252.84**	**-251.35**	-257.18
111_B_two	-7,124.81	-7,124.81	-7,182.88	**-7,182.88**	**-7,186.31**	-7,181.60	-7188.92