Univariate Marginal Distribution Algorithm and Random Variable Neighbourhood Descent Applied to the Vehicle Routing Problem with Private Fleet and Common Carrier

William Higino
williamhigino@gmail.com

Antonio Augusto Chaves
antonio.chaves@unifesp.br

Vinícius Veloso de Melo
vinicius.melo@unifesp.br

UNIFESP
São José dos Campos

ABSTRACT

Among the different classes of Vehicle Routing Problems are the Vehicle Routing Problems with Profits (VRPPs), where it is not mandatory to service all the customers. A relatively new VRPP is the VRPPFCC (Vehicle Routing Problem with Private Fleet and Common Carrier). In this problem, it is sometimes useful to directly serve only part of the shipping demand, outsourcing the rest of it to other companies. This paper presents the combination between the Univariate Marginal Distribution Algorithm (UMDA) and Random Variable Neighbourhood Descent (RVND), a local search procedure, in the solution of the VRPPFCC. The implementation uses a vector of random keys as solution representation; thus a decoding heuristic is also developed, converting random keys to actual solutions for the VRPPFCC. Computational tests and conclusions focus on the comparison of the effectiveness of the methods, comparing the obtained solutions to the best known solutions for the problem.

KEYWORDS. Univariate Marginal Distribution Algorithm. Random Variable Neighbourhood Descent. Vehicle Routing Problem with Private Fleet and Common Carrier.

Logistics and Transport; Metaheuristics; Probabilistic Models

Introduction

According to [Toth e Vigo, 2014], Vehicle Routing Problems (VRPs) consist in, for an existing set of transport requesting customers and a given vehicle fleet, determining a set of routes to serve the transport requests with minimal costs. In a practical way, it means deciding which vehicles should serve which customer and in which sequence, so that all the routes are concluded with minimal financial cost or execution time.

A particular class of VRPs is the Vehicle Routing Problems with Profits (VRPPs). They are characterized by the lack of obligatoriness in the service of all customers. Instead, a profit or loss rate to the service of each customer is defined.

Considering the use of third-party shipping companies, outsourcing costs are usually higher than costs of the direct service. However, as stated by [Toth e Vigo, 2014], using the proprietary fleet has higher costs in some specific cases, with customers located in difficult access areas, as occurs in the Small Package Shipping industry. Thus, the Vehicle Routing Problem with Private Fleet and Common Carrier (VRPPFCC) is concerned both with the selection of outsourced customers and with the routing of vehicles in the private fleet.

Formally, as indicated by [Chu, 2005] and [Bolduc et al., 2008], the VRPPFCC consists in the service of a set of customers in a way that each customer is served only once, all the routes associated with the proprietary fleet start and finish in the deposit, each vehicle of the private fleet executes only one route, the demand served by each route is within the designated vehicle capacity, and the total cost is minimized.

Essentially, the total cost minimization consists in two simultaneous elements. First, the identification of which customers are more profitable if directly served and which have a better profitability if outsourcing is required. The second element consists in the routing process using the private fleet vehicles to directly serve the selected customers [Toth e Vigo, 2014].

The VRPPFCC has a few previous applications since its introduction. In [Chu, 2005], the first application of the problem, a simple heuristic was applied to five local instances. Essentially, the customers with lowest outsourcing costs are outsourced until the left customers can be directly served by the proprietary fleet. An initial solution is then constructed with the unserved customers, using [Clarke e Wright, 1964] savings heuristic, before going through intra-route and inter-routes local search procedures.

[Bolduc et al., 2008] modelled the problem as an heterogeneous VRP. The approach also included an initial solution based on [Clarke e Wright, 1964] savings heuristic. Different local search procedures were applied afterwards, including 4-opt*, 2*-interchange, 2-add-drop, and swap. In the study, new homogeneous and heterogeneous instances were adapted, including up to 483 customers.

[Côté e Potvin, 2009] and [Potvin e Naud, 2011] proposed tabu searches to the problem. The first was successful to most homogeneous instances. The second approach, using ejection chains neighbourhoods, achieved good results for both homogeneous and heterogeneous instances. However, both tabu search applications had a considerably higher execution time than the previous algorithms.

[Stenger et al., 2013] implemented a VNS (Variable Neighbourhood Search) algorithm with cyclic improvements, also considering variations of the problem with multiple depots and non-linear outsourcing costs.

The last approach, in [Vidal et al., 2015], used an exhaustive solution representation with implicit customer selection. A set of customers in a given order was set to each vehicle. Then, a local search determined which customers should be directly served, and which should be outsourced. In the local search, 2-opt, 2-opt*, path relocation, swap, and cross exchange neighbourhood structures were used. Three heuristic frameworks were used to evaluate the solution representation: A multi-start local-improvement search (MS-LS), an adaptation of [Prins, 2009] ILS (Iterated Local Search), and UHGS (Unified Hybrid Genetic Search), an advanced population-based method with

diversity management adapted from [Vidal et al., 2014]. The solution representation presented a good performance, specially with the UHGS approach, where most homogeneous instances had the best solution found or improved.

This paper details an application of an Estimation Distribution Algorithm, the Univariate Marginal Distribution Algorithm (UMDA), in the solution of the VRPPFCC. The method is implemented using an array of real values $\in [0 : 1]$ to represent solutions. The UMDA iteratively generates statistic models based on the population, and uses the models to sample new solutions for the problem.

A heuristic to decode the arrays into VRPPFCC solutions was developed and used in both methods, evaluating how close the solutions generated by the method are to the best known solutions. Using domain knowledge about the problem, the heuristic aims to convert any array of real values into a reasonable solution.

Aligned to the UMDA, there is an implementation of the Random Variable Neighbourhood Descent (RVND), a local search procedure aiming to find better solutions in the proximity of the search space, using a set of different inter and intra-route neighbourhood structures.

The remaining sections of this paper are organized as follows. Section 2 summarizes the problem mathematical model. In section 3, UMDA is explained, as well as the solution representations and decoding algorithms. In section 4, the RVND local search procedure is detailed. Section 5 details the conducted tests, using instances obtained from the literature, to compare the methods, and explains the results. Section 6 presents the conclusions from this study and mentions future research directions.

Mathematical Model for VRPPFCC

In [Bolduc et al., 2008], the VRPPFCC was formulated as a heterogeneous VRP, as described below.

N The quantity of served customers.

V The set of nodes $V = \{0, 1, 2, ..., n\}$, where 0 indicates the depot, and $\{1, 2, ..., n\}$ represents each of the N customers.

C Set $C = (c_{ijk})$, where c_{ijk} represents the travel costs between customers i and j in vehicle k.

q The set with each customer's demand, $q = \{q_1, ..., q_n\}$, where q_i denotes the demand of customer i.

m The maximum number of vehicles available.

Q The maximum capacity of each vehicle, being Q_k the capacity of vehicle k.

f The set of fixed costs of each vehicle, $f = \{f_1, ..., f_m\}$, where f_i denotes the fixed cost for each vehicle, if used.

e The set of outsourcing costs, $e = \{e_1, ..., e_n\}$, where e_i denotes the cost to outsource the service of customer i.

$$x_{ijk} = \begin{cases} 1 \text{ if the vehicle } k \text{ visits customer } j \text{ immediately after customer } i \\ 0 \text{ otherwise} \end{cases}$$
$$i, j \in N, \quad i \neq j, \quad k = 1, 2, ..., m$$

$$y_{ik} = \begin{cases} 1 \text{ if customer } i \text{ is served by vehicle } k \\ 0 \text{ otherwise} \end{cases}$$

$$z_i = \begin{cases} 1 \text{ if customer } i \text{ is outsourced} \\ 0 \text{ otherwise} \end{cases}$$

$u_{ik} =$ An upper bound for the cargo in vehicle k right after leaving customer i.

$$\text{Minimize} \sum_{k=1}^{m} f_k y_{0k} + \sum_{i=0}^{n} \sum_{\substack{j=0 \\ j \neq i}}^{n} \sum_{k=1}^{m} c_{ijk} x_{ijk} + \sum_{i=1}^{n} e_i z_i \tag{1}$$

subject to

$$\sum_{j=1}^{n} \sum_{k=1}^{m} x_{0jk} = \sum_{i=1}^{n} \sum_{k=1}^{m} x_{i0k} \leq m \tag{2}$$

$$\sum_{\substack{j=0 \\ j \neq h}}^{n} x_{hjk} = \sum_{\substack{i=0 \\ i \neq h}}^{n} x_{ihk} = y_{hk} \tag{3}$$

$$h \in \{0, ..., n\}; \quad k \in \{1, ..., m\}$$

$$z_i + \sum_{k=1}^{m} y_{ik} = 1 \quad i \in \{1, ..., n\} \tag{4}$$

$$\sum_{i=1}^{n} q_i y_{ik} \leq Q_k \quad k \in \{1, ..., m\} \tag{5}$$

$$u_{ik} - u_{jk} + Q_k x_{ijk} \leq Q_k - q_j \tag{6}$$

$$i, j \in \{1, ..., n\}; \quad i \neq j; \quad k \in \{1, ..., m\}$$

$$x_{ijk} \in \{0, 1\} \tag{7}$$

$$i, j \in \{0, ..., n\}; \quad i \neq j; \quad k \in \{1, ..., m\}$$

$$y_{ik} \in \{0, 1\} \quad i \in \{0, ..., n\}; \quad k \in \{1, ..., m\} \tag{8}$$

$$z_i \in \{0, 1\} \quad i \in \{0, ..., n\} \tag{9}$$

$$u_{ik} \geq 0 \quad i \in \{1, ..., n\}; \quad k \in \{1, ..., m\} \tag{10}$$

Each term in Equation 1 represents part of the total cost. The first consists in the fixed costs incurred in the used vehicles. In the second, the travel costs from the depot to the served customers, and back to the depot are calculated for each vehicle. The third term consists in the sum of outsourcing costs, incurred on the outsourced customers. Constraints 2 limit the number of used vehicles as m. The constraints in Eq. 3 assert the same vehicle k arrives and leaves customer h. In Eq. 4 it is assured that each customer is served only by the private fleet, or by the outsourced company's fleet. Constraints in Eq. 5 limit the demand services by each vehicle as, at most, its capacity. The constraints in Eq. 6 eliminate sub-routes. Constraints in Eq. 7, 8, and 9 define the decision variables as binary. Finally, Eq. 10 states that the cargo of the vehicle after leaving each customer will never be negative.

UMDA

The Univariate Marginal Distribution Algorithm (UMDA), proposed in [Pelikan e Mühlenbein, 1998], is an Estimation of Distribution Algorithm (EDA, [Larranaga, 2002]).

EDAs are evolutionary algorithms based on the original GAs. Traditional GAs use crossover and mutation genetic operators to generate solutions. However, those operators do not guarantee that good schemas [Holland, 1975] will be kept, occasionally violating the Building Blocks Hypothesis (as noted in [Grefenstette, 1993]). This motivated the creation of a new type of algorithms, named Probabilistic Model Building Genetic Algorithms (PMBGAs), later renamed as EDAs.

In summary, EDAs are population-based algorithms which generate probabilistic models based on promising solutions to guide their search, while looking for improved solutions. At each iteration of the algorithm, the probabilistic model is updated, based on the current promising solutions. The population is then renewed by sampling the last generated model. Since the generation of new individuals is based on the probabilistic model, no crossover or mutation operators are required.

Figure 1 shows the basic workflow of EDAs. First, a probability model is built, based on N individuals from the population. The model is then used to generate the new population. The process is repeated, generating a new model and new samples at each iteration.

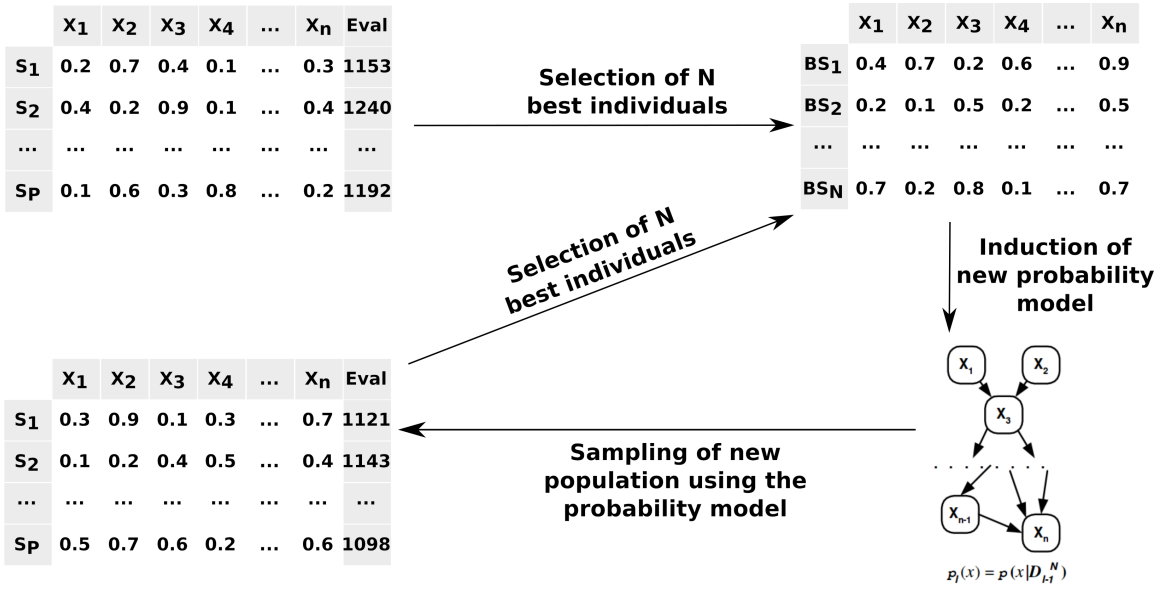

Figura 1: EDAs workflow (adapted from [Bengoetxea, 2002]).

The UMDA is an EDA that assumes each variable to be independent. For each position $i \in \{0, ..., n-1\}$ from an individual with length n, a univariate marginal frequency $p_i(x_i)$ is defined. $p_i(x_i)$ indicates the number of strings that have x_i on the i^{th} position in the population P. The distribution of the parents is then estimated as stated in Equation 11:

$$p(X) = \prod_{i=0}^{n-1} p_i(x_i) \tag{11}$$

New individuals are created by sampling the generated distribution. For each position i, a value is sampled to the i^{th} position. It is set to a with the probability of $p_i(a)$. Once all new individuals have been created, the population is evaluated, and promising individuals are used to update the probabilistic model.

In this work, UMDA was used to determine real values $\in [0 : 1]$ for each position in the solution array. We assumed that each independent variable follows a normal distribution. Therefore,

our model has a pair of parameters for each position i, named mean (μ) and standard deviation (σ), estimated from the selected (promising) solutions. Using the mean and standard deviation, a normal distribution was used to sample a new population at each generation.

Solution Decoding

In the experiments, solutions were represented as arrays of real numbers between 0 and 1. We created a procedure to decode such arrays as solutions for VRPPFCC, defining routes for the served customers, and which customers should be outsourced.

The decoding procedure, detailed in Algorithm 1, uses the sorted Random Keys array to indicate the order in which the customers should be evaluated and included in the routes. Each customer has its cost of being directly served compared to the outsourcing cost, and in case outsourcing indicates a lower cost, the customer is not included in the route of any of the vehicles in the proprietary fleet.

Algorithm 1 Solution decoder

```
1:  function DECODESOLUTION(randomKeysSolution)
2:      routes ← list(vehicles)
3:      for each vehicle in vehicles do
4:          routes(vehicle) ← list()
5:      end for
6:      outsourcedCustomers ← list()
7:      customersOrder ← order(randomKeysSolution)
8:      customers ← customers[customersOrder]
9:      for each customer in customers do
10:         bestVehicle, bestPosition ← getLowestCostVehicleAndPosition(routes, customer)
11:         if bestVehicle >= 1 then
12:             routes(bestVehicle).insert(customer, bestPosition)
13:         else
14:             outsourcedCustomers.push(customer)
15:         end if
16:     end for
17:     return routes
18: end function
```

In the evaluation of direct service costs, the vehicle setup costs are not considered, even if the customer will be the first one serviced by the vehicle. Since the setup cost of vehicles is relatively high, considering it in the comparison with outsourcing costs would, in most cases, force all customers to be outsourced. Thus, we consider only the travel costs. Each customer has its travel costs calculated, considering every possible position it could be inserted in the given moment. Regarding the customer's demands, the capacity of each vehicle was also considered to assure that the routes respect the vehicles capacities. The lowest cost vehicle and position evaluation is explained in Algorithm 2.

The decoded routes were represented using matrices. Each vehicle route was represented by a row, with the order of visited customers for the corresponding vehicle. The outsourced customers were not indicated in any of the routes, being deduced to be outsourced during the evaluation function calculation.

Algorithm 2 Lowest cost vehicle and position evaluation

```
1: function GETLOWESTCOSTVEHICLEANDPOSITION(routes, customer)
2:     customerDemand ← demands[customer]
3:     lowestCost, outsourcingCost ← externalCosts[customer]
4:     lowestCostVehicle, lowestCostPosition ← 0
5:     for each vehicle in vehicles do
6:         vehicleCustomers ← routes(vehicle)
7:         usedCapacity ← sum(demands[vehicleCustomers])
8:         exceedsCapacity ← (customerDemand + usedCapacity) > capacities[vehicle]
9:         if !exceedsCapacity then
10:            for each position in length(vehicleCustomers) do
11:                prevCustomer ← vehicleCustomers[position − 1]
12:                nextCustomer ← vehicleCustomers[position]
13:                costInVehicleInPosition ← distances[prevCustomer][customer] +
   distances[customer][nextCustomer] − distances[prevCustomer][nextCustomer]
14:                if costInVehicleInPosition < lowestCost then
15:                    lowestCost ← costInVehicleInPosition
16:                    lowestCostVehicle ← vehicle
17:                    lowestCostPosition ← position
18:                end if
19:            end for
20:        end if
21:    end for
22:    return lowestCostVehicle, lowestCostPosition
23: end function
```

RVND

The RVND (Random Variable Neighbourhood Descent) is a local search procedure derived from the Variable Neighbourhood Descent ([Mladenović e Hansen, 1997]).

VND was proposed on the idea that the global optimum is the optimal solution considering every possible neighbourhood structure. It also accounts that a local optimum for a given neighbourhood structure might be different from the local optimum from another neighbourhood structure. Therefore, VND explores the search space though systematic neighbourhood changes, and only moves the solution when an improvement is achieved. Thus, VND allows an intensive search on the considered search space region, providing different ways to look for better solutions.

RVND differs in the order neighbourhood structures are explored. While VND has a predefined order for the neighbourhoods structures to be applied, RVND randomly selects the neighbourhood structures order to be applied at each execution.

The RVND procedure is presented in Algorithm 3. First, a neighbourhood structures list (NSL) containing a predefined number of neighbourhood moves to be applied to the solution is initialized. The procedure then enters a loop, where a neighbourhood structure $N^n \in NSL$ is chosen at random at each iteration. The solution is then updated, in cases where the applied neighbourhood structure presents an improvement to the previous solution. Finally, after every movement has been applied to the solution, the best found solution is returned.

Algorithm 3 RVND Source: Adapted from [Penna et al., 2013]

```
1:  procedure RVND(sol)
2:      Initialize neighbourhood structures list NSL
3:      while NSL ≠ ∅ do
4:          Randomly choose a neighbourhood structure N^(n) ∈ NSL
5:          Find the best neighbour sol' of sol ∈ N^(n)
6:          if f(sol') < f(sol) then
7:              sol ← sol'
8:              f(sol) ← f(sol')
9:              Update NSL
10:         else
11:             Remove N^(n) from NSL
12:         end if
13:     end while
14:     return sol
15: end procedure
```

VRP Neighborhoud structures

[Subramanian et al., 2013] present an ILS heuristic using RVND as a local search method. The proposed algorithm is applied to Vehicle Routing Problems, thus several neighbourhood structures considering intra-routes and inter-routes movements are defined.

In this work, the neighbourhood structures presented in [Subramanian et al., 2013] are used by the RVND to improve the solutions found by UMDA. The VRP movements are categorized in intra-routes, when they change arches or customers inside a specific route, or inter-routes movements, when more than one route is affected.

Inter-route movements

Some of the inter-routes movements are based on λ-interchanges [Osman, 1993], consisting on the change of up to λ consecutive customers between two routes. Others are based on cross-exchange [Taillard et al., 1997], which exchanges two segments between different routes. The inter-routes structures used in RVND are detailed below.

Shift(1,0) - A customer i is transferred from a route r_1 to a different route r_2.

Swap(1,1) - A customer i from a route r_1 is swapped with a customer j from a route r_2.

Shift(2,0) - Two consecutive customers i and j are transferred from a route r_1 to a different route r_2.

Swap(2,1) - Two consecutive customers i and j, belonging to a route r_1, are swapped with a customer i' from a different route r_2.

Swap(2,2) - Two consecutive customers i and j, belonging to a route r_1, are swapped with two different customers i' and j' from a different route r_2.

Cross - The arch between adjacent customers i and $i+1$ in a route r_1, and the arch between adjacent customers j and $j+1$ in a route r_2 are removed. New arches are then created, connecting customers i and $j+1$, as well as customers j and $i+1$.

t-Shift - A set of t consecutive customers is transferred from a route r_1 to a route with lower costs r_2. t is incremented from 3 to the total length of r_1, according to the capacity of r_2. This movement intends to empty routes with higher costs, using vehicles with lower fixed or variable costs to serve the customers.

Intra-route movements

The proposed intra-route movements were based on reinsertion, Or-opt [Or, 1976], 2-opt and swap. Each of the used intra-route neighbourhoods is explained below.

Reinsertion - A customer is removed from the current position in the route, and reinserted in a different position of the route.

Or-opt2 - Two consecutive customers are removed from their current positions, and reinserted in different positions in the same route, but keeping the consecutiveness.

Or-opt3 - Three consecutive customers are removed from their current positions, and reinserted in different positions in the same route, but keeping the consecutiveness.

2-opt - Two non-adjacent arches $(i, i+1)$ and $(j, j+1)$ are removed. The loose sub-route is then reversed, and new arches (i, j) and $(i+1, j+1)$ are created, reconnecting the entire route.

Swap - Two customers in the same route are swapped, changing their servicing order. The sub-route between the swapped customers is also reversed.

Selected customers movements

A third category of movements is proposed, in order to alternate the selection of outsourced customers during the local search procedure. The proposed movements were called Customer-Insertion, Customer-Removal, and Customer-Swap.

Customer-Insertion - A customer, previously outsourced, is inserted in a position p from a route i.

Customer-Swap - A customer, previously outsourced, is Swapped with a different customer, currently allocated in a route i.

Customer-Removal - A customer, previously allocated to a route i, is removed from the route and outsourced.

Integration with UMDA

To integrate UMDA with RVND, we defined a stagnation point to the UMDA (no improvement in the best found solution after 250 generations), and included the best obtained solution in a local repository every time stagnation was reached. Once the UMDA execution finished, each of the solutions stored in the repository was sent to RVND, looking for improved solutions using the neighbourhood structures. Also, RVND was run 30 independent times to use different neighbourhood orders for a same solution.

Computational Tests and Results

To compare the performance of UMDA alone, and its combination with RVND under similar circumstances, we performed a set of tests using some of the instances available for the VRPPFCC. UMDA was implemented in R, with the most costly functions (solution decoding and evaluation) implemented in C++, while RVND was implemented entirely in C++. Also, all tests were performed on a computer with Intel i7-3610QM @ 2.30GHz CPU and 8GB of RAM running Ubuntu 15.10.

The instances used in the tests were adapted from [Christofides, 1976] in [Bolduc et al., 2008]. Instances with different sizes, ranging from 50 to 199 customers, allow a comparison between the methods behaviour for different individuals' lengths. The instances are also divided between homogeneous and heterogeneous fleet, depending on the variety of vehicles available. Results were measured in terms of distance from the best known solution found in the literature.

Initially, UMDA alone was tested to provide a performance baseline for the method. The execution time was limited according to each instance size, after different tests determined the minimum time for stagnation of the method. Stagnation, in this case, was defined as no improvement in the best found solution after 250 generations. Every time stagnation was reached, the UMDA population was restarted, starting a new search until reaching the time limit. The population size was defined as 20 * N, after initial tests with different population sizes were conducted. After the execution of UMDA alone, we performed a second set of tests with UMDA integrated with RVND, as explained in subsection 4.2.

Table 1 shows the results obtained by the UMDA metaheuristic alone, and the improvements obtained by executing the method aligned to the RVND local search procedure. The first three columns summarize the results obtained by the UMDA alone, using only the decoding heuristic

Tabela 1: Results Summary

Instance	N	m	UMDA			UMDA + RVND			Time	BKS
			Min	Mean	Diff (%)	Min	Mean	Diff (%)		
CE-01	50	4	1129.82	1138.19	0.92	1120.07	1125.50	0.05	30	1119.47
CE-02	75	9	1879.43	1884.76	3.58	1842.09	1858.06	1.52	125	1814.52
CE-03	100	6	1959.30	1974.57	2.10	1945.25	1960.39	1.37	200	1919.05
CE-04	150	9	2575.58	2599.96	2.80	2542.97	2574.16	1.50	500	2505.39
CE-05	199	13	3161.43	3181.02	2.59	3139.43	3159.48	1.88	1200	3081.59
CE-06	50	4	1216.82	1225.36	0.77	1207.47	1214.41	0.00	30	1207.47
CE-07	75	9	2064.19	2077.35	2.98	2037.65	2050.50	1.65	125	2004.53
CE-08	100	6	2094.51	2107.73	2.07	2090.03	2099.82	1.85	200	2052.05
CE-09	150	10	2509.00	2520.17	3.56	2483.70	2496.25	2.52	500	2422.74
CE-10	199	13	3466.40	3483.58	2.51	3433.62	3450.85	1.54	1200	3381.67
CE-11	120	6	2420.51	2435.36	3.84	2332.24	2336.45	0.06	300	2330.94
CE-12	100	8	1980.34	1991.79	1.41	1955.17	1972.27	0.12	200	1952.86
CE-13	120	6	2944.76	2960.35	3.01	2864.56	2871.74	0.20	300	2858.83
CE-14	100	7	2237.02	2244.78	1.08	2224.37	2226.26	0.51	200	2213.02
Average					2.37			1.05		
CE-H-01	50	4	1206.54	1218.65	1.25	1201.27	1205.66	0.80	30	1191.7
CE-H-02	75	9	1840.76	1856.68	2.77	1820.82	1839.34	1.65	125	1791.21
CE-H-03	100	6	1942.18	1959.21	1.26	1922.38	1939.20	0.23	200	1917.96
CE-H-04	150	9	2522.29	2558.59	1.64	2512.19	2532.38	1.23	500	2481.64
CE-H-05	199	13	3205.97	3225.19	2.00	3157.30	3188.18	0.45	1200	3143.01
CE-H-06	50	4	1216.84	1227.52	0.83	1210.00	1218.46	0.26	30	1206.82
CE-H-07	75	9	2074.12	2088.66	2.08	2044.50	2065.66	0.62	125	2031.85
CE-H-08	100	6	2009.03	2028.08	1.13	1991.78	2012.73	0.27	200	1986.51
CE-H-09	150	10	2514.86	2534.15	2.84	2483.82	2500.72	1.57	500	2445.49
CE-H-10	199	13	3326.62	3340.77	1.68	3298.76	3311.40	0.83	1200	3271.7
CE-H-11	120	6	2394.56	2414.87	3.72	2331.30	2340.88	0.98	300	2308.76
CE-H-12	100	8	1948.96	1953.69	2.11	1919.58	1934.05	0.57	200	1908.74
CE-H-13	120	6	2929.50	2950.89	3.07	2854.06	2873.90	0.42	300	2842.18
CE-H-14	100	7	1940.15	1953.39	1.70	1917.00	1932.28	0.48	200	1907.75
Average					2.00			0.74		

vehicle and position searching procedures to determine the customers distribution on the vehicles, service orders, as well as the relation of outsourced customers. The next three columns, indicated under UMDA + RVND, details the results for the combination of UMDA and the local search procedure, performed on the best solutions found after each UMDA execution.

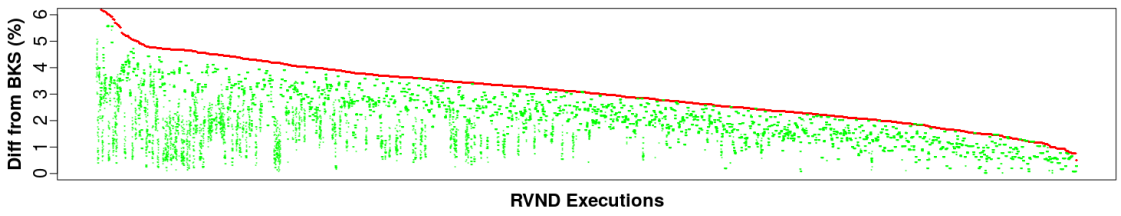

Figura 2: RVND Solutions Improvement.

By the results, it's clear the RVND had a considerable impact on the solutions obtained, due to the usage of the neighbourhood structures to explore the search space. Figure 2 details the

improvements in the solutions after the RVND executions. The upper line indicates the difference from the starting solutions provided to the method by UMDA, while the points beneath indicate the difference in the final solutions obtained by the procedure. As noted by the points, the most intense improvements occurred with initial solutions with a difference around 4%.

Although the RVND application did improve the best results obtained, the method used to integrate UMDA and RVND has alternatives. Applying the local search procedure at different moments, such as periodically during the execution of the UMDA, or through hybrid methods that evaluate when it seems promising to apply local search procedures. This change should likely improve the obtained results. Also, changes in the behaviour inside the RVND can be studied through the usage of different neighbourhood structures, as well as variations in the implementation of the RVND.

Conclusions and Future Directions

This paper evaluated the application of UMDA and RVND in the solution of the VRPPFCC, a routing vehicle problem with outsourcing costs.

UMDA used a random keys vector to represent solutions, and a decoding algorithm was implemented using the random keys order. A heuristic consisting of evaluations on the capacity of customers previously delegated to each vehicle, as well as the cost impact on the solution for each new inserted customer, was used to determine whether each customer should be directly served, and at which position of which vehicle.

The performed tests allowed us to verify the performance of the UMDA, as well as the impact of the RVND on the solutions. Though the UMDA alone obtained solutions with an average difference from the BKS around 2%, the results from UMDA aligned with RVND were much closer to the BKS, finding the best known solution for an instance, and having an average difference of less than half the one of UMDA alone.

As future directions, we intend to evaluate the performance of the RVND working with other metaheuristics in the solution of the VRPPFCC. Also, the combination of the metaheuristic and local search procedures offer a good starting point for the application of hybrid methods, as the Clustering Search [Oliveira et al., 2013], which manages the local search application on promising areas of the search space throughout the metaheuristic execution.

Referências

Bengoetxea, E. (2002). *On processes and threads: synchronization and communication in parallel programs*. PhD thesis, PhD Thesis. University of the Basque Country.

Bolduc, M.-C., Renaud, J., Boctor, F., e Laporte, G. (2008). A perturbation metaheuristic for the vehicle routing problem with private fleet and common carriers. *Journal of the Operational Research Society*, 59(6):776–787.

Christofides, N. (1976). The vehicle routing problem. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 10(1):55–70.

Chu, C.-W. (2005). A heuristic algorithm for the truckload and less-than-truckload problem. *European Journal of Operational Research*, 165(3):657–667.

Clarke, G. e Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Côté, J.-F. e Potvin, J.-Y. (2009). A tabu search heuristic for the vehicle routing problem with private fleet and common carrier. *European Journal of Operational Research*, 198(2):464–469.

Grefenstette, J. J. (1993). Deception considered harmful sk. *Foundations of Genetic Algorithms 1993 (FOGA 2)*, 2:75.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Larranaga, P. (2002). A review on estimation of distribution algorithms. In *Estimation of distribution algorithms*, p. 57–100. Springer.

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

Oliveira, A. C. M. d., Chaves, A. A., e Lorena, L. A. N. (2013). Clustering search. *Pesquisa operacional*, 33(1):105–121.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*. Xerox University Microfilms.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451.

Pelikan, M. e Mühlenbein, H. (1998). Marginal distributions in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel*, volume 98, p. 90–95. Citeseer.

Penna, P. H. V., Subramanian, A., e Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19(2):201–232.

Potvin, J.-Y. e Naud, M.-A. (2011). Tabu search with ejection chains for the vehicle routing problem with private fleet and common carrier. *Journal of the Operational Research Society*, 62(2):326–336.

Prins, C. (2009). A grasp× evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem*, p. 35–53. Springer.

Stenger, A., Vigo, D., Enz, S., e Schwind, M. (2013). An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. *Transportation Science*, 47(1):64–80.

Subramanian, A., Penna, P. H. V., Ochi, L. S., e Souza, M. J. F. (2013). *Meta-Heurísticas em Pesquisa Operacional*, chapter Um Algoritmo Heurístico Baseado em Iterated Local Search para Problemas de Roteamento de Veículos. Omnipax.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., e Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2):170–186.

Toth, P. e Vigo, D. (2014). *Vehicle Routing Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics and the Mathematical Optimization Society, second edition.

Vidal, T., Crainic, T. G., Gendreau, M., e Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673.

Vidal, T., Maculan, N., Ochi, L. S., e Penna, P. H. V. (2015). Large neighborhoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science, Articles in Advance*.