Metaheuristics Network

Report for Task 5

—

Research on the Vehicle Routing Problem with Stochastic Demand

Leonora Bianchi and Monaldo Mastrolilli
IDSIA, USI-SUPSI

Mauro Birattari and Max Manfrin
IRIDIA, Université Libre de Bruxelles

Marco Chiarandini, Luis Paquete
Intellektik, Technische Universität Darmstadt

Olivia Rossi-Doria
ECRG, Napier University

Abstract

This task report concerns the research carried out in the Metaheuristics Network for Task 5, which focuses on Vehicle Routing Problems. The research concentrated on the Vehicle Routing Problem with Stochastic Demand, a variation of deterministic classical routing problems, where each customer demand is assumed to follow a given probability distribution, instead of having a single known value.

The first chapter of this report defines the Vehicle Routing Problem with Stochastic Demand, and resumes the related literature on stochastic routing. Chapter 2 describes the first phase of research that has been carried out by the Metaheuristics Network, and reports the discussions and possible research directions that emerged from the Second Young Researches' Network Meeting, held in June 2003. The last chapter of the report (Chapter 3) describes the second experimental phase, that was done in order to address the questions that arose after the first experimental phase. The second experimental phase was designed in particular to test the metaheuristics on a fair basis and to see how much of the exploitation of the problem similarities with the traveling salesman problem can improve the algorithms performance.

Contents

List of Algorithms

1 The Vehicle Routing Problem with Stochastic Demand

1.1 Introduction to stochastic routing problems

The deterministic vehicle routing problem (VRP) is well known in the operations research literature (see [9, 11, 15, 18, 6] for reviews). Task 5 of the Metaheuristics Network deals with a variation where each customer demand is assumed to follow a given probability distribution, instead of having a single known value. The actual customer demand is known only upon arrival at the customer's location. Such a problem is known in the literature as vehicle routing problem with stochastic demand (VRPSD). The VRPSD arises in practice whenever a company faces the problem of deliveries to (or collections from) a set of customers, whose demands are uncertain.

In a deterministic VRP routes are planned in such a way that vehicles always have enough capacity to satisfy all customers' demands on their pre-planned routes. When demands are stochastic, this is possible only by loosing the concept of 'pre-planned routes', and by adding a set of decisions rules (a *routing strategy* or *routing policy*). In fact, suppose a vehicle visits customers in a certain pre-planned order. It is possible that at a some customer location the vehicle capacity becomes attained or exceeded, and a route failure is said to occur. In such a situation, the following decision (or 'recurse action') may be taken: the vehicle returns to the depot, replenishes, and resumes its tour, or goes back to the customer where demand has not been fully satisfied. Therefore, the vehicle will always be able to serve all demands of customers on its pre-planned tour, but the distance traveled is a random quantity, and the locations where recurse actions will be taken are not known in advance. Indeed, the actual tour followed by the vehicle might be in general different by the pre-planned one. The goal, in the VRPSD, is to determine a routing policy such that demand at each customer is met, and the expected distance traveled is minimized.

We can think about the following types of routing policies:

- off-line (or a priori, static, open-loop)
- on-line (or real-time, dynamic, closed-loop)
- mixed

Off-line policies prescribe for a vehicle a sequence τ of customers to be visited in that order. Such policy is then executed in real-time, supplemented by simple recourse actions to accommodate failures that may occur when one actually observes the realizations of the random variables involved in the problem. On the other extreme are on-line policies. An on-line policy π tells, for each possible state of the system, which location to visit next. In practice, the application of an on-line policy consists of optimally re-sequencing the unvisited customers whenever a vehicle arrives at a new customer location. Mixed policies combine elements of both; for instance, the vehicle follows the prescribed tour, but it is enabled with state dependent rules that allow for preventive returns to the depot, before a route failure actually occurs.

In our study of the VRPSD we focus on the mixed policy of *preventive restocking*, which works in the following way. At each stop a decision is taken on whether returning to the depot for replenishment after serving each stop, in anticipation of possible stock-outs, or before serving, if a stockout occurred. The decision is taken in order to trade off the extra cost of returning to the depot after a stockout with the cost of returning to the depot for reloading before an out-of-stock actually occurs at a stop. Thus, the point at which the vehicle returns to the depot may be before a stockout actually takes place. Preventive restocking is done if the residual capacity of the vehicle after serving a stop is below a certain threshold, which is calculated in advance, at the moment of the routes planning.

Motivation of our approach (mixed policy) For a given VRPSD instance, and for a given set of recourse actions, the set of off-line policies is a subset of the set of mixed policies, that, in turn, is included in that of on-line ones. Since all of them aim at the minimization of the same quantity (i.e., the expected distance traveled), in principle better solution quality is achieved by on-line policies. Secomandi [32] provided an estimation of the gap between the performance of optimal on-line policies and optimal off-line/mixed policies. This gap is below 3% for optimal on-line policies, and below 1% for optimal mixed policies, in small instances with up to 8 customers. Moreover, there is theoretical evidence [32, 6] that the size of this gap tends to zero, with infinitely many customers, uniformly and independently distributed on the unit square. Therefore, off-line, mixed and on-line policies are asymptotically equivalent. Even if slightly better performing, the on-line approach has several difficulties [3]: the dispatch company might not have the resources for dynamically re-sequencing unvisited customers; the redesign of tours might be not sufficiently good to justify the required effort and cost; the operator might have other priorities, such as regularity and personalization of service. Instead, off-line and mixed policies are preferable from a computational point of view since they entail solving only one instance of an NP-hard problem. They also produce a more stable and practically predictable solution, since the customer sequence of each route is predetermined. The implication of these observations is that algorithmic effort should be directed on computing close to optimal mixed policies, as also noted by Secomandi [32].

1.2 Literature review

One of the first approaches to stochastic routing problems are the works of Jaillet [21, 22] on the traveling salesman problem with random customers, and recently addressed by Bianchi, Gambardella and Dorigo [7, 8]. The capacitated case is first analyzed by Jaillet and Odoni [23]. The works [21, 22, 23] show how to efficiently compute the expected length of a solution. Bertsimas [3] and Bertsimas, Chervi and Peterson [4] further analyze the VRPSD with one vehicle. Bertsimas [3] proposed the cyclic heuristic, by adapting to a stochastic framework one of the heuristics presented by Haimovitch and Rinnooy Kan [19] in a deterministic context. The algorithm proceeds in two phases. In the first phase a TSP is solved, without considering customers' demands. This phase provides a tour $\tau = (0, 1, ..., n, 0)$ starting and ending at the depot. The second phase considers the n tours $\tau_i = (0, i, ..., n, 1, ..., i-1, 0)$, for $i = 1, ..., n$, and computes their expected length, $E[L_{\tau_i}]$. Out of these n tours, the best tour τ^* is chosen:

$$\tau^* = \arg \min_{i=1,...,n} E[L_{\tau_i}]. \tag{1}$$

Bertsimas, Chervi and Peterson [4] improve the cyclic heuristic by applying dynamic programming, to supplement the a priori tour with rules for selecting returns trips to the depot, thus obtaining a mixed policy. Their computational experience suggests that the two versions of the cyclic heuristic provide good quality solutions. This heuristic is also shown to be asymptotically optimal under a random Euclidean model. Bertsimas [3] has also proposed a heuristic for the vehicle routing problem with stochastic customers and demand (VRPSCD).

This problem is very similar to the VRPSD, but here one supposes to know whether the next customer will require zero demand in advance, before traveling to that customer. All these approaches fall within the a priori (and mixed) optimization framework, discussed in [5]. Secomandi [32, 33] focuses on computing on-line policies for the VRPSD, by applying a rollout dynamic programming algorithm to sequentially improving a given a priori solution.

In case of a fleet of m vehicles, if balanced tours are not needed, one can arrange to have each of the $m-1$ customers that are closer to the depot served by one of the $m-1$ available vehicles. Then the remaining customers form a single vehicle instance that can be solved by the cyclic heuristic [15]. This adaptation of the cyclic heuristic has also shown to be asymptotically optimal in a random Euclidean model (see [15] and references cited therein). Nevertheless it does not seem to be a practical approach in a real situation, because one obtains $m-1$ single customer routes, and one route with $n-m+1$ customers. Yang, Mathur and Ballou [38] investigate more sophisticated approaches to solve the multi vehicle problem, with a constraint on the expected distance traveled by each vehicle. They test two heuristic algorithms, the route-first-cluster-next and the cluster-first-route-next, which separately solve the problem of clustering customers which must be served by different vehicles and the problem of finding the best route for each cluster. Both algorithms seem to be efficient and robust.

Other authors have proposed different approaches to compute a priori policies for VRPSD with a fleet of vehicles. Gendreau, Laporte and Séguin [16] present a stochastic integer programming method for VRPSD (the same method can be applied to VRPSD as well). This is the first exact method for this class of policies presented in the literature. By means of the integer L-shaped method [27] they solve instances with up to 46 and 70 customers and 2 vehicles, for the VRPSCD and VRPSD, respectively. The same authors also develop a tabu search heuristic algorithm called TABUSTOCH, for the same problem [17]. This algorithm is to be employed when instances become too large to be solved exactly by the L-shaped method. They report obtaining optimal solutions in 89.45% of the instances. The average deviation from optimality is only 0.38%, and in 97.8% is smaller than 5%. Computational times are considered reasonable.

1.3 Description of the problem

The VRPSD routing problem is defined on a complete graph $G = (V, A, C)$, where:

$$\begin{aligned} V &= \{0, 1, ..., n\}, \text{ a set of nodes with node 0 denoting the depot.} \\ A &= \{(i, j) : i, j \in V, i \neq j\}, \text{ the set of arcs joining the nodes.} \\ C &= (c_{ij}), \text{denoting the travel cost (distance) between nodes } i \text{ and } j. \end{aligned}$$

The cost matrix C is symmetric and satisfies the triangular inequality. Also, all customers have stochastic demands ξ_i, $i = 1, ..., n$, which are independently distributed with known distributions and are known only upon arriving at the customer location. For implementation porposes it is assumed that ξ_i does not exceed the vehicles capacity Q, and follows a discrete probability distribution $p_{ik} = \text{Prob}(\xi_i = k)$, $k = 0, 1, 2, ..., K \leq Q$. We assume a single vehicle. This is equivalent to assume that a set of customers has been assigned to receive service by a given vehicle. The vehicle is initially located at the depot.

We adopt the mixed routing policy of preventive restocking, described in section 1.1. The goal is finding a vehicle route and a restocking policy at each node (a threshold), to minimize total expected cost. The costs under consideration are:

- Cost of traveling from one customer to another as planned.
- Restocking cost - the cost of traveling back to the depot for restocking, before going to the next planned customer.
- Route failure cost - the cost of returning to depot for restocking caused by remaining stock in the vehicle being insufficient to satisfy demand upon arrival at a customer location. This cost is a fixed nonnegative cost b plus a cost of traveling to the depot and back to the route.

Let $0 \to 1 \to 2 \cdots \to n$ be a particular planned vehicle route. After the service completion at customer j, suppose the vehicle has a remaining load q, and let $f_j(q)$ denote the total expected cost from node j onward. The expected cost of the planned route (that is, the objective function value) is then $f_0(Q)$, and the problem consists in finding a route with the least expected total cost. If S_j represents the set of all possible loads that a vehicle can have after service completion at customer j, then, $f_j(q)$ for $q \in S_j$ satisfies

$$f_j(q) = \text{Minimum} \begin{cases} f_j^p(q) \\ f_j^r(q) \end{cases}, \tag{2}$$

where

$$f_j^p(q) = c_{j,j+1} + \sum_{k:k \le q} f_{j+1}(q-k)p_{j+1,k} + \sum_{k:k>q} [b + 2c_{j+1,0} + f_{j+1}(q+Q-k)]p_{j+1,k}, \tag{3}$$

and

$$f_j^r(q) = c_{j,0} + c_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q-k)p_{j+1,k}, \tag{4}$$

with the boundary condition

$$f_n(q) = c_{n,0}, \; q \in S_n. \tag{5}$$

In equations (2-4), $f_j^p(q)$ represents the expected cost of proceeding directly to the next node, and $f_j^r(q)$ represents the expected cost of the restocking action. These equations are used to recursively determine the objective value of the planned vehicle route and the optimal sequence of decisions after customers are served.

1.3.1 Threshold and expected cost evaluation

The expected cost-to-go in case of restocking, $f_j^r(q)$, is constant in q, since in case of restocking the vehicle will have full capacity Q before serving the next customer, whatever the current capacity q is. On the other hand, $f_j^p(q)$ is a monotonically non-increasing function in q (proof in [38]), for every fixed customer j. Therefore there is a capacity threshold value h_j such that, if the vehicle has more than this value of residual goods, then the best policy is to proceed to the next planned customer, otherwise it is better to go back to the depot for restocking (see Figure (1)).

The following algorithm is an implementation of the dynamic programming recursion (2-5) for the calculation of $f_0(Q)$ and of the thresholds. Let us assume the customers demands may take values $\xi \in \{0, 1, ..., K\}$. Then, the algorithm runs in $O(nKQ)$ time; the memory required is $O(nQ)$, if one is interested in memorizing all intermediate values $f_j(q)$, for $j = 1, 2, ..., n$ and $q = 0, 1, ..., Q$, and $O(Q)$ otherwise.

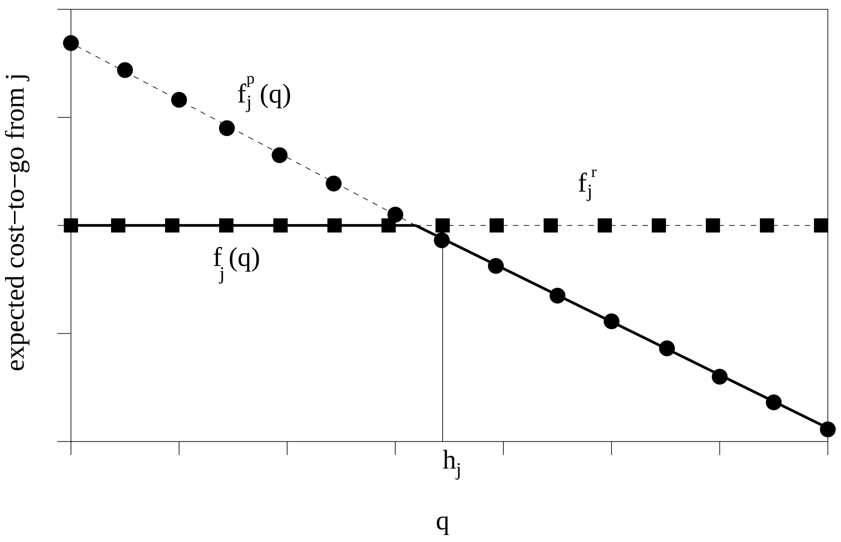


Figure 1: The bold line represents the cost function $f_j(q)$, that is, the expected cost-to-go of the vehicle from customer j on, under the restocking strategy. The quantity q is the residual capacity of the vehicle just after having serviced customer j. The function $f_j^p(q)$ would be the expected cost-to-go in case the vehicle always proceeded directly to customer $j+1$ without first going to de depot for replenishment. The function (constant in q) $f_j^r(q)$ would be the expected cost-to-go in case the vehicle always went to the depot for restocking, before going to the customer $j+1$. The figure shows that if the residual capacity q is under the threshold h_j, then it is more convenient to restock, otherwise it is more convenient to proceed to the next customer.

Procedure 1 Computation of the VRPSD objective function $\mathbf{f_0(Q)}$

```
for ($q = Q, Q-1, ..., 0$) do
  $f_n(q) = c_{n,0}$
  for ($j = n-1, n-3, ..., 1$) do
    compute $f_j^r$ using $f_{j+1}(\cdot)$
    for ($q = Q, Q-1, ..., 0$) do
      compute $f_j^p(q)$
      compare $f_j^r$ and $f_j^p(q)$ for finding the threshold $h_j$
      compute $f_j(q)$ using $f_{j+1}(\cdot)$
    end for
  end for
end for
compute $f_0(Q)$
return $f_0(Q)$
```

2 First experimental phase: from the Proceedings of the second Young Researchers' Network Meeting

The aim of the Second Young Researches' Network Meeting (Moltrasio, June 30- July 5, 2003), was to discuss the first experimental results about metaheuristics comparisons, and to decide which aspects of the problem and of the algorithms to investigate in a second experimental phase. This chapter reports results and issues that emerged from the meeting.

2.1 The Algorithms

In this section we give a brief description of the algorithms that where implemented during the first phase of the research activity of the Metaheuristic Network on the vehicle routing problem with stochastic demand.

2.1.1 The OrOpt local search for the VRPSD

The Or-opt algorithm [30], has been quite successfully applied to the VRPSD by Yang et al. [38]. Given a starting tour, the basic operation of the Or-opt algorithm consists of moving a string of size 3, 2, or 1 from one position to another in the tour. Checking if an Or-opt move leads to a better tour may be done in two stages. First compute the saving from extracting the string from the tour, and second compute the cost of inserting it back somewhere else in the tour, after the extraction point.

Computing these costs and savings in the deterministic case is quite simple, since they can usually be computed in constant time. In the stochastic case, however, it is computationally demanding, because it requires the dynamic programming recursion of equations (12-14). This leads to an $O(nKQ)$ time for the computation of the cost and saving of just one Or-opt move. In order to reduce the computational time, the following approximation scheme suggested by Yang et al. [38] have been used.

Given a string S of consecutive nodes in the a-priori tour, the approximate saving of extracting it from the tour is computed as follows. Let l and t be the nodes immediately preceding, resp. following, S in the tour, and let $f_l^{beforeExt}(q)$ and $f_t(q)$ be the corresponding cost vectors before the extraction of S. Apply one dynamic programming recursion step starting with cost vector $f_t(q)$ at node t back to node l, without considering the string S. Let $f_l^{afterExt}(q)$ be the resulting cost vector at node l, that is, after extracting S from the tour. Then, define the approximate extraction saving as a simple average over q of $f_l^{afterExt}(q) - f_l^{beforeExt}(q)$, that is,

$$\text{Approximate Extraction Saving} = \frac{\sum_{q=0}^{Q}(f_l^{afterExt}(q) - f_l^{beforeExt}(q))}{Q+1}, \tag{6}$$

The computation of the insertion cost of S between nodes i and j in the tour, is done analogously, if we assume that the insertion point (node i) is after the extraction point (node l). Let $f_i^{beforeIns}(q)$ be the cost vector at node i for the current tour, that is, before inserting S in the tour. Apply dynamic programming recursion starting with cost vector $f_j(q)$ at node j through nodes of the inserted string, and back to node i. Let $f_i^{afterIns}(q)$ be the resulting cost vector at node i, that is, after inserting the sting in the tour. Then, define the approximate insertion cost as a simple average over q of $f_i^{afterIns}(q) - f_j^{beforeIns}(q)$. That is,

$$\text{Approximate Insertion Cost} = \frac{\sum_{q=0}^{Q}(f_i^{afterIns}(q) - f_j^{beforeIns}(q))}{|S_i|}. \tag{7}$$

The total approximate cost of an Or-opt move is computed by subtracting the Approximate Extraction Saving from the Approximate Insertion Cost:

$$\text{Approximate Or-opt Move Cost} = \text{equation (7) - equation (6)}. \tag{8}$$

Note that the cost vectors are assumed to be already available from the computation of the expected cost for the starting tour, thus, they do not need to be computed when evaluating the Approximate Insertion Cost. The only computations that must be done here are the evaluation of cost vectors $f_l^{afterExt}(q)$ and $f_i^{afterIns}(q)$, and the averages in equations (6) and (7). The dynamic programming recursion for the computation of the cost vectors requires $O(KQ)$, while equations (6) and (7) require $O(Q)$ time. Therefore, with the proposed approximation, the cost of an Or-opt move can be computed in $O(KQ)$ time.

The proposed approximation scheme has some potential drawbacks though. It neglects the influence of the inserted string (or deleted string) on the nodes before node i. In principle it is thus possible that a certain Or-opt move seems good, when evaluated by the approximation scheme, but it is actually a worsening move, when evaluated by the exact objective function computation. Therefore if the approximation scheme is used to evaluate moves in the Or-opt algorithm (or in any other local search), there is no guarantee that a better tour than the starting one will be found. In practice this approximation should behave quite well since, as reported in [38], it should find the same route as the one obtained if the exact costs were computed, with significantly less computational effort (less than 10%). In the following we outline the Or-opt algorithm, implementing the proposed approximation scheme.

2.1.2 Basic tabu search

The tabu search implemented by IDSIA node in the first test phase is very basic. Like all tabu search algorithms, it and is composed by three main elements: the starting solution, the neighborhood structure and the strategy of selection of new solutions. The starting solution was generated by the nearest neighbor traveling salesman problem heuristic. The neighborhood structure is derived from the OrOpt algorithm described in section 2.1.1. The strategy of selection of new solutions is a descent-ascent method with the use of tabu list. The following is a high level pseudocode description of our tabu search, where we use the same notation as in the description of the OrOpt (section 2.1.1) **procedure** *Tabu Search for the Vehicle Routing Problem with Stochastic Demand*

```
k = 3
while( time is not over ){
    if($k < 1$) $k = 3$
    length_of_tabu_list $\in [0.8 \cdot (n-k-1), n-k-1]$
    for each potential move among those with string length $k$ {
        compute_proxy_neighbour_cost()
        check_if_move_is_tabu()
```

Procedure 2 OrOpt local search for the VRPSD

```
1: Let $k = 3$ and $T$ be an initial a-priori tour.
2: Compute the expected cost of tour $T$ by using the dynamic programming recursion of
   equations (12)-(14).
3: for all $S^k$, a set of successive nodes from $T$ do
4:    select at random a node along the a-priori tour after the node immediately preceding
      $S^k$ and compute the Approximate Or-opt Move Cost by using equation (8).
5: end for
6: if none of the sets in Step 3 and 4 results in a negative Approximate Or-opt Move Cost
   then
7:    go to Step 11.
8: else
9:    Select the set $S^k$ that results in the most negative Approximate Or-opt Move Cost in
      Step 3, and perform the corresponding Or-opt move. Than, go to Step 3.
10: end if
11: if  $k = 1$ then
12:    stop
13: else
14:    decrease $k$ by 1, and go to Step 3.
15: end if
```

```
            update_neighbour_best()
        }
        if( there is a neighbour_best ){
            currSol.shift( neighbour_best_move)
            update_true_global_best(start_solution)
            if( not a new_true_global_best ) decrease $k$ by 1
            set_tabu_move()
        }
        else tabu_move = neighbour_best_move
    tabu_list.push_back(tabu_move)
    prune_tabu_list()
}
```

Now some observations in order to explain the tabu search pseudocode.

The neighborhood is explored in such a way that at the beginning of the search, only moves with string length equal to 3 are tried. The string length is decreased each time that no best move can be found. Each time the string length has reached the minimum length (that is, 1), it is re-initialized to the maximum value (that is, 3).

In our tabu search it is important the notion of neighbour_best_move, because the neighbour_best_move is the move that is actually performed and leads to a new solution. Each time a new potential move is considered, the procedure update_neighbour_best() evaluates it and decides if the move is a new neighbour_best_move or not. The decision criterion differs in the case of tabu or non-tabu moves. If the move is tabu, than, in order to be promoted to new neighbour_best_move, the move should lead to a new global best solution (according to the approximated evaluation done with compute_proxy_neighbour_cost()). If the move is not tabu, than, it is enough that it leads to a new best solution with respect to the the current neighborhood. Not all moves are evaluated, in fact, tabu moves are evaluated only with a probability of the 30%, and non-tabu moves with a probability of the 80%. Also for this reason, sometimes no move is selected as new neighbour_best_move.

Each time a new neighbour_best_move is selected, it is put in the tabu list, in the following way: tabuMove[0] = currSol[i+1], tabuMove[1] = k, tabuMove[2] = currSol[j]. Moreover, the move is performed, and the exact objective value of the new solution is computed. In case there is no new neighbour_best_move, the last selected neighbour_best_move is put again in the tabu list. After, the oldest tabu move is removed from the list.

2.1.3 A simple memetic algorithm for the VRPSD

The Vehicle Routing Problem with Stochastic Demand (VRPSD) with the a priori strategy seems to be very similar to a TSP, at least in that the same space of a-priori tours between customers has to be searched. The simple memetic algorithm implemented is therefore based on work on the TSP, and crossover and mutation operators were chosen from permutation based operators which work well for the TSP.

A small population size of 10 individuals is used. Initial solutions are built with the randomized farthest insertion heuristic, which builds a solution starting from a random customer and then shifts the tour to start at the depot, as required by the problem. OrOpt local search is than applied to each member of the initial population. In the Steady State evolution process only one couple of parents reproduces at each generation. Tournament selection of size 2 is used to select which parents are going to be given the chance to reproduce. The crossover used in the final implementation is the Edge Recombination crossover (ER) taken from Michalewicz, and readapted to always start at the depot. OX crossover (see Michalewicz, page 217) readapted to always start from the depot, and PMX crossover (implemented as described in Michalewicz, page 216) were also implemented and tried, but ER crossover seems to work better. It tries to build an offspring exclusively from the edges present in both parents, as follows:

1. Create an edge list from both parents, that provides for each customer all the other customers connected to it in at least one of the parent;
2. start from the depot as current customer;
3. select the customer in the edge list of the current customer, with the smallest number of customers left in its edge list;
4. if there is no customer left in the edge list of the current customer, select a non yet visited customer;
5. the selected customer becomes the current customer;
6. update the edge list by removing the current customer from each adjacent customer list;
7. if the tour is complete, stop; otherwise go to step 3.

Swap based mutation swaps two adjacent alleles, never the depot. It is applied with an adaptive mutation rate with a maximum probability of 0.5. Order based mutation that picks 2 loci at random and exchanges their alleles, and inversion were also tried. OrOpt local search is again applied at each generation to improve the quality of the offspring. The improved offspring then replaces the worst member of the population.

2.1.4 Simulated Annealing

Simulated Annealing is a metaheuristics inspired by the process of annealing in physics [37, 26]. It is widely used to solve combinatorial optimization problems, especially to avoid getting trapped in local optima [1]. This is done as follows: an improving local search move is always accepted while a worsening local search move is accepted according to a probability which depends on the respective deterioration in the evaluation function value, such that the worse a move is, the less likely it is to accept it. Formally a move is accepted according to the following probability distribution, known as the Metropolis distribution:

$$p_{accept}(T, s, s') = \begin{cases} 1 & \text{if} \quad f(s') \leq f(s) \\ exp\left(-\frac{f(s')-f(s)}{T}\right) & \text{otherwise} \end{cases} \tag{9}$$

where s is the current solution, s' is a neighbor solution and $f(s)$ is the evaluation function. The parameter T, which controls the acceptance probability, is allowed to vary over the course of the search process.

Procedure 3 schematically depicts how Simulated Annealing works. The time available to solve the instance is, in our study, the termination condition while the acceptance criterion is the one of Equation 9. In the next sections we give a more detailed description of how we implemented the components, GenerateInitialSolution, SelectMove, UpdateTemperature, for the VRPSD.

Procedure 3 Simulated Annealing

```
$s \leftarrow$ GenerateInitialSolution()
$T \leftarrow T_0$
while termination condition not met do
  $s' \leftarrow$ SelectMove($\mathcal{N}(s)$)
  if $f(s') \leq f(s)$ then
    $s \leftarrow s'$
  else
    $s \leftarrow$AcceptanceCriterion(T,$s$,$s'$)
  end if
  UpdateTemperature($T$)
end while
```

Initial Solution We considered the following alternatives:

- a random sequence of customers;
- the sequence of customers generated by a constructive heuristic for Vehicle Routing Problem;
- the sequence of customers generated by constructive heuristics for Traveling Salesman Problem (nearest neighborhood heuristics, farthest insertion heuristic) [2];

- a sequence of customers obtained by solving the Traveling Salesman Problem by Iterated Local Search.

From preliminary results we noticed that the better the initial solution, the better will be the final solution for the VRPSD. Since good solutions for the TSP result to be good also for the VRPSD, we decided to adopt the fourth alternative. The ILS algorithm used to find a tour that visit all the customers exactly once and that minimize the total length without caring of any capacity constraints, is one of the highest performing metaheuristics for TSP and is described in [34]. Although it is not optimal it does provide good quality solutions to the TSP.

Move Selection The or-opt neighborhood appears particularly appealing for the VRPSD mainly because an or-opt move does not reverse any path in the route. Therefore its cost evaluation is relatively fast. We confined ourself to consider or-opt of length 1, 2 and 3. As done in the common local search provided by IDSIA, we do not consider backwards insertions but only forward. See Figure 2(a).

For the neighborhood examination strategy we considered three alternatives. Let us call k the length of the sequence to move (k in $\{1, 2, 3\}$) and i and j respectively the index of starting position and index of the ending position in the sequence. We select k, i, j in the following way:

1. k looping in 3,2,1; i in label order; j randomly;

2. i in label order; j in label order; for each pair i and j $k = \{1, 2, 3\}$;

3. k randomly; i random; j random;

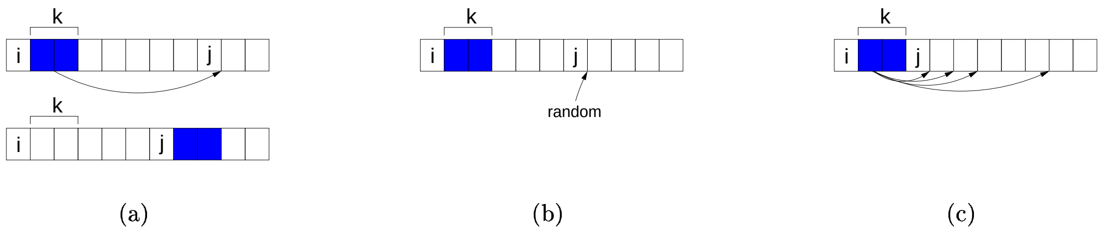


Figure 2: (a) An or-opt move of length, $k = 2$, two. Given a position i in the sequence, the k elements after it are shifted after position j. In our case j has to be bigger than i (forward insertion). (b) (c) Two examination strategies. In (b), given k and i, j is selected randomly; next k and i are updated (examination alternative 1). In (c), j is let varying in label order until the end of the sequence is reached; next k and i are updated (examination alternative 2).

The evaluation of a move can use the exact method or the approximated one. We tested both.

Temperature updating The temperature profile is determined by three main features: the initial temperature, the cooling schedule and the the re-heating schedule.

Initial Temperature. A sample of 100 elements in the neighborhood of the initial solution is used to compute the average variation in the evaluation function. The value found is then multiplied by a given factor, f, which is a parameter of the algorithm.

Cooling schedule. We used a non monotonic temperature schedule realized by the interaction of two strategies: a standard geometric cooling and a temperature re-heating. In the standard geometric cooling the temperature T_{n+1} is computed from T_n as $T_{n+1} = \alpha \times T_n$ where α is a parameter called the "cooling rate" $(0 < \alpha < 1)$.

The number of iterations at which the temperature remains constant is kept proportional to the size of the neighborhood as suggested in Johnson et al. [24]. Hence, $TL = q \cdot |\mathcal{N}|$, where q is a parameter and $|\mathcal{N}|$ is the size of the neighborhood, which depends on the examination strategy selected.

Re-heating. When the search appears to stagnate, the temperature is increased by adding T_i to the current temperature. This is done when no improvement is found for a number of steps given by $r \cdot TL$ where TL is the temperature length and r is a parameter. The solution that we consider for testing improvements is the best since the last re-heating occurred.

Tuning For selecting which neighborhood examination strategy and which move evaluation method are better and for tuning the parameters f, α, q and r, on the classes of instances that will then be used for the comparison of metaheuristics, we set up an experimental phase. This involved 74 candidate versions of Simulated Annealing, obtained by combination of the elements described above, and 24 heterogeneous instances. Instances were selected from: (i) the Christofides modified instances, (ii) the random uniform instances, (v) the random clustered instances, (vi) the asymmetric. Since we had to submit only one configuration we did not distinguished between classes. Termination times for each run of algorithm were determined by 30 random restarts of the common local search on the instance under examination. The evaluation methodology we used is the following: we run each algorithm once on each instance, we ranked the algorithms on the instances and we averaged the ranks over the instances. The algorithm with the best average rank was selected as representative of Simulated Annealing applied to VRPSD and submitted to compete against the other metaheuristics.

The algorithm selected uses the second neighborhood examination scheme of Section 2.1.4 above and the approximated method to evaluate a move. But, surprisingly, it has temperature null. This means, that the acceptance criterion of Simulated Annealing is merely reduced to accepting all non worsening moves. Hence, Simulated Annealing does not provide any improvement to the common local search provided by IDSIA and the simple algorithm which repeat the local search for all the time available accepting also side walk moves works better.

Previous works and comments Beside applications to the Traveling Salesman Problem, Simulated Annealing has been applied also to the Vehicle Routing Problem. Osman [31] implementation of SA has also the feature of starting from an improved solution. It, then, uses a 1-interchange neighborhood and cools the temperature every time a move is accepted. Results are promising even if not the state of the art for VRP [36]. Dueck [14, 13], uses always on the VRP, a modified accepting criterion. In his "Deterministic" Annealing a new solution is accepted if and only if $f(s') < f(s) + \theta$ where θ is a parameter to control the process. In the specific case of Vehicle Routing with Stochastic Demand Simulated Annealing was firstly applied by Teodorović and Pavkoviȁ [35]. They use Simulated Annealing in a construction phase, to accept or reject randomly generated routes of a fixed length determined by an analysis on the structure of the problem (the assumption on the length of routes is, in their case, possible since all customers have the same uniform distribution with the same average demand, which, instead, is not our case). In a second phase they use Simulated Annealing to improve each single routes by exchanging randomly the order of offering services in two nodes of the route. In both cases temperature is updated a fixed number of times when a thermal equilibrium is reached. Results are given on a single run on a single instance and are, therefore, hardly comparable.

However, in our attempt to use Simulated Annealing to solve the VRPSD with re-stocking policy, we faced the major problem that using the approximated method to evaluate the contribution of a move introduces already randomization on the search and the possibility of accepting worsening moves. In this case it becomes really hard to control effectively the probabilistic acceptance criterion of SA by means of temperature schedule because it overlaps with the probabilistic criterion implicitly embedded in the evaluation function. Indeed, the observation of the profile of evaluation function in some runs with different temperature, revealed almost no correlation with the behavior of temperature and, therefore, of SA acceptance criterion. This was the main reason that pushed us to test also the exact evaluation strategy. In this case it seems, however, that the loose in time produced by this strategy is so big that in it can no way perform better than the approximated method. Another consideration is related with the initial solution. Solving a TSP produces already good solutions that can be only little improved by a successive local search. In two of the instances used for the tuning we even observed that the solution found on the TSP remained the best solution found. It is crucial, we believe, to understand why this happens and if it always happens or if, instead, as we believe, the structure of an instance has an influence on the choice of the solving methodology.

2.1.5 Ant Colony Optimization

The Ant Colony Optimization (ACO) metaheuristic implemented by IRIDIA node in the first test phase is a very basic Ant Colony System (ACS) [12].
In the ACS, a set of agents, called ants, build solutions to the vehicle routing problem with stochastic demand cooperating through pheromone-mediated indirect and global communication. Here we give only a brief description of the basic principles that lie beneath the ACS. At each iteration of the algorithm, each of the m ants constructs an a-priori tour one customer after the other so that starting with the depot all customers are visited one and only one time, ending again with the depot. The ants choose the next customer to be visited probabilistically, guided by stigmergic information. No heuristic information is used in this first implementation.
The stigmergic information is in the form of a matrix of 'pheromone' values $\tau : C \times C \to \Re_{\geq 0}$, where C is the set of customers, which are an estimate of the utility of going from one customer i to a second one j in the a-priori tour, as judged by previous iterations of the algorithm. In ACS at the beginning of the algorithm the values in the matrix are initialized to a parameter τ_0, execpt for those elements that belong to the starting solution, generated by

the farthest insertion heuristic, who recieve a "reinforcement" equal to r iterations of global update rule. After each construction step a local update rule is applied to the element of the matrix corresponding to the choosen customers pairs (i,j):

$$\tau(i,j) \leftarrow (1-\psi)\cdot\tau(i,j) + \psi\cdot\tau_0 \tag{10}$$

The parameter $\psi \in [0,1]$ is the pheromone decay parameter, which controls the diversification of the construction process. The aim of the local update rule is to permit to the m ants to choose different customers j for the same given customer i.
After all the customers have been visited, the OrOpt local search routine described in section 2.1.1 is applied to the candidate solution s. At the end of the iteration the global update rule is applied to all the entries in the pheromone matrix:

$$\tau(i,j) \leftarrow \begin{cases} (1-\rho)\cdot\tau(i,j) + \rho\cdot\frac{Q}{q(s_{best})} & \text{if } (i,j) \text{ is in } s_{\text{best}} \\ (1-\rho)\cdot\tau(i,j) & \text{otherwise} \end{cases} \tag{11}$$

where Q is a parameter controlling the amount of pheromone laid down by the update rule, and the function q measures the expected cost of a candidate solution s.
The parameters we have used for the algorithm are the following:

m	τ_0	α	ψ	ρ	Q	r
5	0.5	1	0.3	0.1	10^7	100

2.1.6 Iterated Local Search

ILS is based on the simple yet powerful idea of improving a local search procedure by providing new starting solutions obtained from perturbations of the current solution, often leading to far better solutions than if using random restart. The local search is applied to the perturbed solution and a locally optimal solution is reached. If it passes an acceptance criterion, it becomes the new current solution; otherwise, one returns to the previous current solution. The perturbation must be sufficiently strong to allow the local search to explore new solutions, but also weak enough so that not all the good information gained in the previous search is lost. A detailed description of ILS algorithms can be found in [28].

To apply the ILS algorithm, four components have to be specified. These are a GenerateInitialSolution procedure that generates an initial solution s_0, a Perturbation procedure, that modifies the current solution s leading to some intermediate solution s', a LocalSearch procedure that returns an improved solution s'', and a procedure AcceptanceCriterion that decides to which solution the next perturbation is applied. A scheme for ILS is given below.

```
procedure Iterated Local Search
    $s_0$ = GenerateInitialSolution()
    $s$ = LocalSearch($s_0$)
    repeat
        $s'$ = Perturbation($s$, history)
        $s''$ = LocalSearch($s'$)
        $s$ = AcceptanceCriterion($s, s''$, history)
    until termination condition met
end
```

Since we observed that very good TSP solutions were correlated with good solutions to this problem, the GenerateInitialSolution procedure consisted in solving the problem as a TSP, *i.e.* , simply minimizing the distances. The algorithm used was an ILS algorithm which is described in [34] and is one of the most high performance metaheuristics for the TSP. The initial solution was the best over $\frac{n}{log(n)}$ random restarts of the latter algorithm, where n is the number of customers. In our implementation the LocalSearch procedure was fixed to be the common local search provided by IDSIA node. Since only a random sample of the or-opt neighborhood was examined, we let the LocalSearch run for n times. The AcceptanceCriterion consisted in accepting s'' if it is better than s, *i.e.* the best solution found so far. The Perturbation consisted in a n random moves of a 2-exchange neighborhood, *i.e.* subtour inversion between two randomly chosen customers. This operator is broke if, within the n moves, a solution is found having an objective function smaller than the objective function of the best solution plus a certain value ε. Given the instances tackled, $\varepsilon = \frac{n}{10}$ was empirically the best value found on some preliminary runs.

2.1.7 Flim-Flam

In addition to the described metaheuristics, a sixth method was submitted. It's main principles are similar to ones from ILS. Therefore, the same name given to the ILS procedures are used here. However, this approach didn't follow the same rules as in the others metaheuristics. Its purpose was to go farther by using strong algorithmic strategies for solving the VRPSD as a TSP. In fact, it consists in a very straightforward adaptation of the ILS described in [34].

In this approach, the GenerateInitialSolution consisted in generating an initial solution by means of a Nearest Neighbor heuristic. The LocalSearch procedure was a typical local search 3-opt first improvement. In addition, it applied two additional speed-up techniques, which are a fixed radius nearest neighbor within candidate lists of 40 nearest neighbors for each city [25] and *don't look bits* [2]. The AcceptanceCriterion, as in the ILS version, consisted in accepting s'' if it is better than the best solution found so far. Finally, the Perturbation consisted in applying a double-bridge move [29] that cuts the current tour at four appropriately chosen edges into four sub-tours and reconnects these in a different order to yield a new starting tour for the local search.

2.2 Experimental Results of the First Phase

The algorithms compared in the first phase are: a basic random restart local search (rrls), flim-flam (ff), iterated local search (ils), evolutionary algorithm (ea), simulated annealing (sa), ant colony system (acs), and tabu search (ts). A description of the implementations is given in Section 2.1.

Four classes of instances where considered:

rand_unif_n: 45 instances,

rand_unif_r: 50 instances.

rand_clust_n: 45 instances,

rand_clust_r: 50 instances,

Each algorithm was tested once on each instance for a time equal to the time needed by the random restart local search to complete 30 iterations. So the execution time varied on an instance-by-instance basis.

The experiments were performed on the new cluster of PCs available at IRIDIA (polyphemus). For these experiments, 6 CPUs Athlon 1400 were used.

Figure 3 illustrates the results aggregated over the four classes of instances. Figures 4, 5, 6, and 7 illustrate the results obtained by the algorithms under analysis on the classes of instances rand_unif_n, rand_unif_r, rand_clust_n, and rand_clust_r, in this order.

As it can be observed from the plots, flim-flam is by far the best algorithms. This shows that, at least on the instances considered in this research, a TSP approximation of the cost function is particularly effective. A tentative analysis of the reasons of the success of such approach is given in Section 2.3. On the other hand, Section 2.4, discusses the limitations and possible risks of such approach.

Another element that is worth noticing in the results is that the relative performance of the algorithms under analysis is similar across the different classes of instances: iterated local search is always the best performing metaheuristic followed by evolutionary algorithms and simulated annealing and then by ant colony system and tabu search. In any case, all the metaheuristics considered here obtain significantly better results than the plain random restart local search.

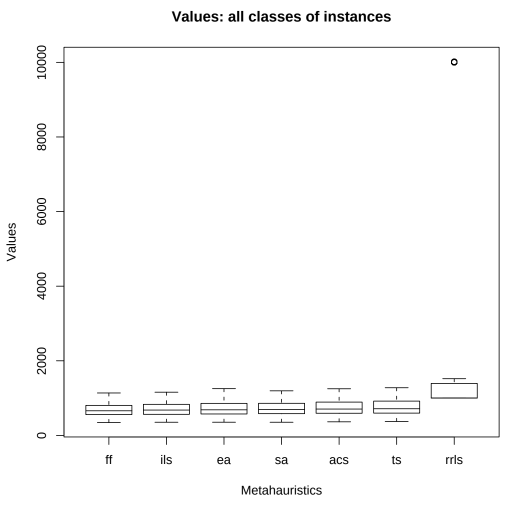
2.2.1 How to read the tables and the graphics

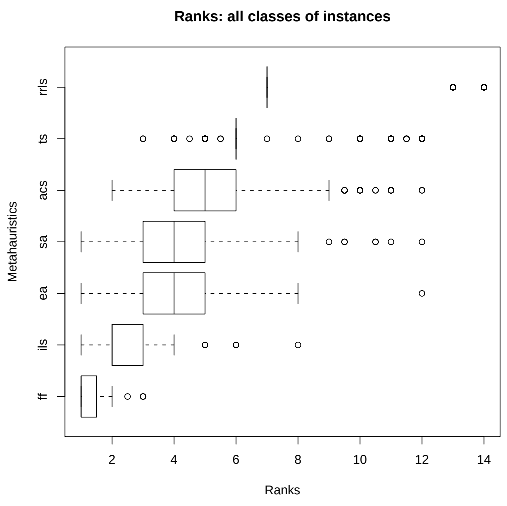
For every class of instances and then for the entire set of instances we verify if differences in solutions found by the algorithms are statistical significant. We use the *Pairwise Wilcoxon rank sum* test [10] with *p-values*[1] adjustment method by Holm [20]. In the tables associated to the graphics we report for every pair of algorithms (A,B) the *p-values* for the null hypothesis "The distributions of the solutions generated by A and by B are the same".

The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

The diagrams show the distribution of the solutions found by the algorithms during independent executions. The results of all executions on the same instance are ordered by quality of the solution (expected cost) to determine the rank. Solutions are grouped by algorithm. In the *boxplot* the median is represented by the bar, the *box* represent the interval included in the quantile 25% e 75%; the whiskers extend to the more extreme points that are no more than 1,5 times the interquartile; the circles represent points who are outside this interval.

[1]The *p-value* of a test is the probability that a value more extreme than the one that is obtained, in the direction of the alternate hypothesis, would have been obtained if the null hypothesis were true.





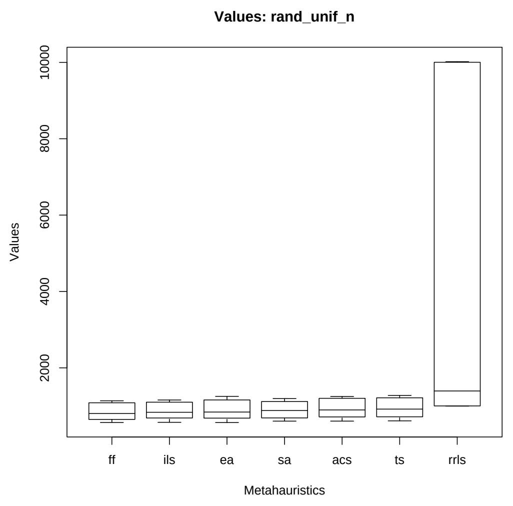
Pairwise comparisons using Wilcoxon rank sum test

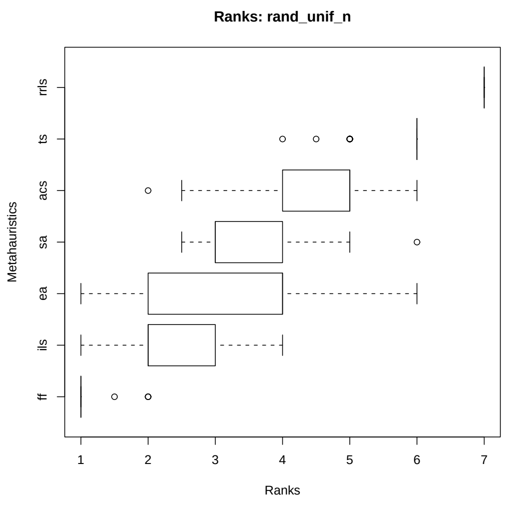
all classes of instances

	ff	ils	ea	sa	acs	ts
ils	$< 2e-16$	—	—	—	—	—
ea	$< 2e-16$	$< 2e-16$	—	—	—	—
sa	$< 2e-16$	$< 2e-16$	0.45	—	—	—
acs	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	—	—
ts	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	—
rrls	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$

P value adjustment method: holm

Figure 3: Results of the metaheuristics implemented on the all the 4 classes of instances.





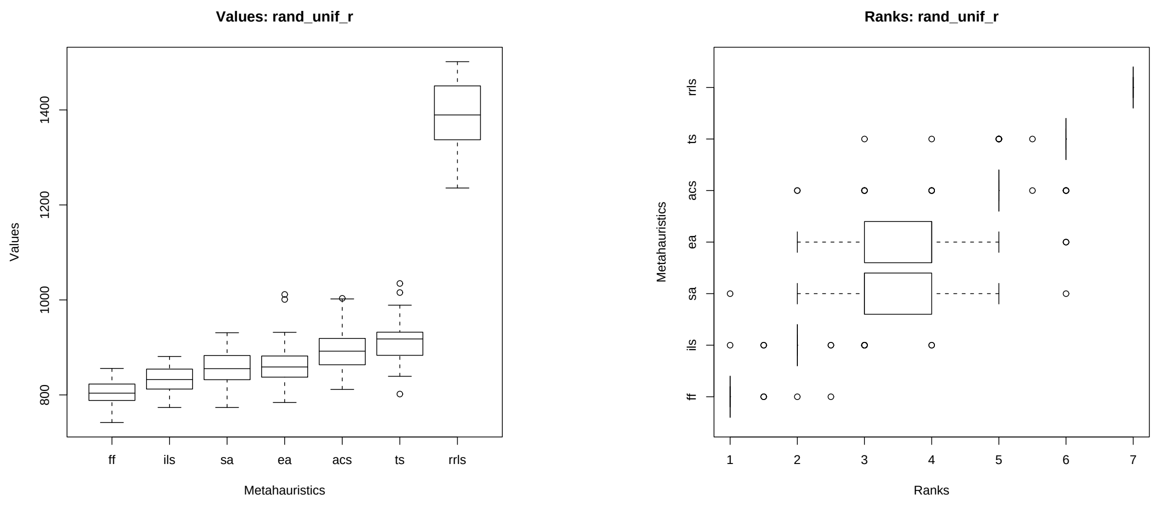
Pairwise comparisons using Wilcoxon rank sum test

rand_unif_n

	ff	ils	ea	sa	acs	ts
ils	$8.9e-08$	—	—	—	—	—
ea	$1.3e-12$	$2.2e-05$	—	—	—	—
sa	$1.2e-12$	$3.8e-07$	0.49	—	—	—
acs	$1.2e-12$	$1.4e-07$	$8.4e-06$	$2.1e-06$	—	—
ts	$1.2e-12$	$1.2e-12$	$4.6e-10$	$4.0e-11$	$8.4e-06$	—
rrls	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$

P value adjustment method: holm

Figure 4: Results of the metaheuristics implemented on the **rand_unif_n** class of instances.



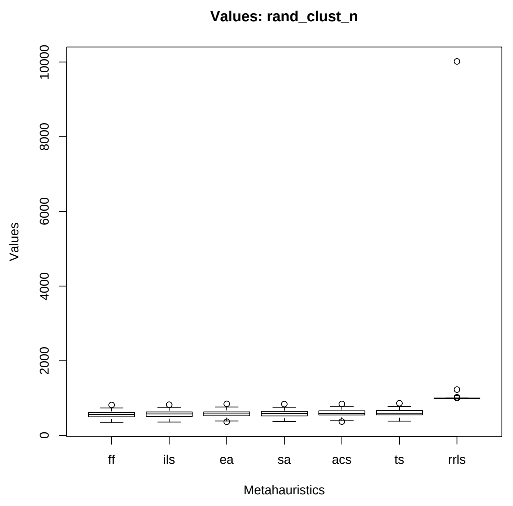
Pairwise comparisons using Wilcoxon rank sum test

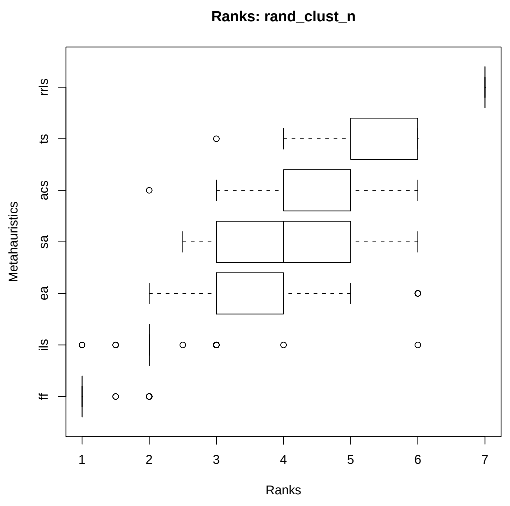
rand_unif_r

	ff	ils	sa	ea	acs	ts
ils	$2.7e-08$	—	—	—	—	—
sa	$1.6e-08$	$2.6e-08$	—	—	—	—
ea	$1.6e-08$	$7.5e-07$	0.46	—	—	—
acs	$1.6e-08$	$1.6e-08$	$4.1e-07$	$1.4e-05$	—	—
ts	$1.6e-08$	$1.6e-08$	$1.8e-08$	$7.5e-07$	$6.6e-06$	—
rrls	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$

P value adjustment method: holm

Figure 5: Results of the metaheuristics implemented on the **rand_unif_r** class of instances.





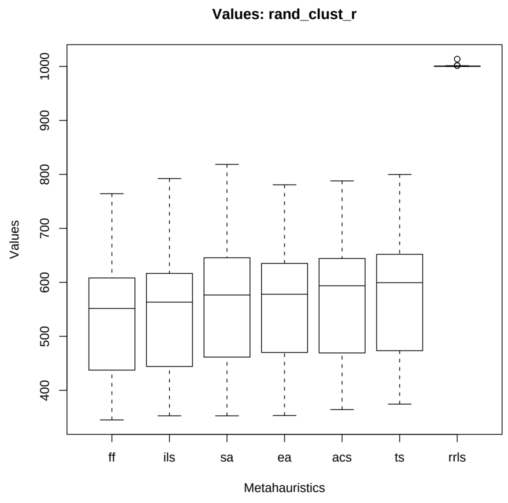
Pairwise comparisons using Wilcoxon rank sum test

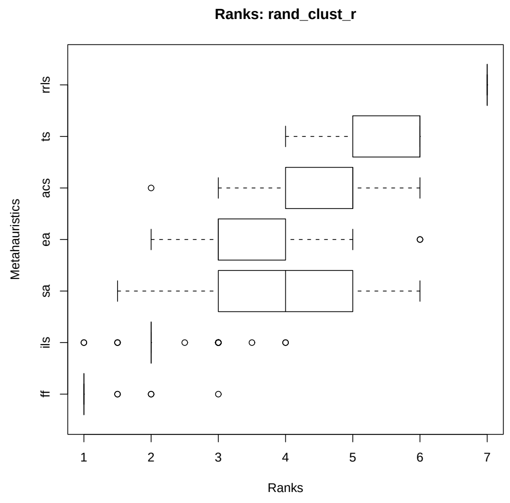
rand_clust_n

	ff	ils	ea	sa	acs	ts
ils	$6.0e-07$	—	—	—	—	—
ea	$1.2e-12$	$1.7e-06$	—	—	—	—
sa	$1.2e-12$	$2.4e-07$	0.14	—	—	—
acs	$1.2e-12$	$3.9e-10$	$2.4e-07$	$9.7e-05$	—	—
ts	$1.2e-12$	$3.4e-11$	$6.3e-11$	$2.3e-08$	$1.1e-05$	—
rrls	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$

P value adjustment method: holm

Figure 6: Results of the metaheuristics implemented on the **rand_clust_n** class of instances.





Pairwise comparisons using Wilcoxon rank sum test

rand_clust_r

	ff	ils	sa	ea	acs	ts
ils	$2.9e-08$	—	—	—	—	—
sa	$1.6e-08$	$5.1e-08$	—	—	—	—
ea	$1.6e-08$	$5.6e-06$	0.06245	—	—	—
acs	$1.6e-08$	$2.3e-08$	0.00036	$8.6e-07$	—	—
ts	$1.6e-08$	$1.6e-08$	$2.3e-06$	$2.9e-08$	$1.6e-06$	—
rrls	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$

P value adjustment method: holm

Figure 7: Results of the metaheuristics implemented on the **rand_clust_r** class of instances.

2.3 Relation between TSP and VRPSD

By definition, the Traveling Salesman Problem (TSP) and the VRP are closely related, since both are concerned with determination of routes in a graph such that a certain cost associated between nodes is minimized. In fact, if there is only one vehicle with infinite capacity, the consequent VRP can be seen as a *simple* TSP. in addition, if several vehicles of infinite capacity are considered, one can even formalize it as a Multiple TSP with several copies of the depot as different nodes.

Given the fact that very fast algorithms for solving the TSP to optimality are known, one should expect that they can also be applied in the same manner to solve the VRP. However, the addition of a constraint on the vehicle's capacity avoids its direct usage. Therefore, when the vehicle's capacity is too low, an optimal TSP tour can not be anymore optimal for this problem. Anyway, if capacity is not so extremely low when compared with the existing demand, an optimal TSP tour can be a good starting solution for applying a heuristic-based algorithm for solving the VRP.

In the case of the VRPSD, we observed a higher performance of the metaheuristics using search strategies typical for solving the TSP, i.e, minimizing total traveling distance. This could mean that, due to the stochasticity of the demand, the minimization of constraint violations considering the capacity of the vehicle becomes less important than minimizing its total traveling distance. Hence, intuitively, it would be preferable to reduce the cost of traveling between nodes, since one is not sure when has to go back to the depot. Another simpler hypothesis is concerned with the extremely fast evaluation for TSP tours which allows far more extensive exploration of solutions than performed by the approximation to the VRPSD objective function.

We conjecture that TSP-like strategies are the main key for tackling the VRPSD instances that were generated. However, we also believe that not all VRPSD instances could be tackled by a TSP approach. For instance, an instance which has a depot far away from all customers would be hardly solved by using such approach. More research with different kind of instances has to be done to verify our conjecture.

2.4 Pathological cases

Given the similarities between the TSP and the VRPSD with the a-priori strategy, and the fact that the algorithms relying on TSP strategies seem to work quite well on the VRPSD one could wonder if an optimal solution for the TSP can be arbitrarily far from optimal for the VRPSD. Indeed some pathological instances can be built for which the difference between an optimal solution for the TSP, regarded as an a-priori tour for the VRPSD, and an optimal solution for the VRSPD can be arbitrarily large.

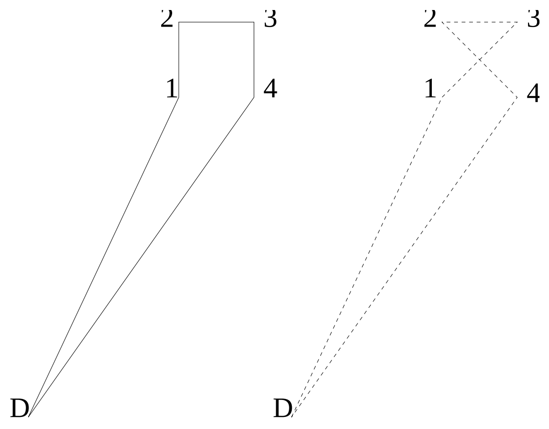


Figure 8: In this VRPSD instance the capacity Q of the vehicle is a multiple of 6, the demand of the customers is $2/3Q$, $Q/2$, $1/3Q$ and $Q/2$ respectively for customers 1, 2, 3 and 4, with spread 0 for each of them. The distance between the depot D and the other customers must be much bigger than the distance between any two customers. On the left the optimal tour for the TSP is shown, but under the above assumptions it is far from optimal for the VRPSD because the vehicle has to restock twice with this a-priori tour. On the right is shown a tour where the vehicle has to restock only once, making its expected cost as far from the one of the tour on the right as twice the distance between the depot and the customers.

A very simple example is shown in Figure 8 where we can move the depot D far enough from the customers 1, 2, 3 and 4 as to meet any arbitrary difference between the expected cost of the a-priori tour given by an optimal solution for the TSP represented by the continous line, and the expected cost of an optimal solution for the VRPSD not based on TSP, where the a-priori tour is given by the dotted line. Other more complex examples, involving a higher number of customers and/or larger demand spread, can be thought of.

On the other hand one could question the a-priori strategy for the VRPSD and ask if the expected cost of an a-priori tour is a good estimate for the cost of an actual tour realization. Here again pathological cases can be found where the difference between the expected cost of an a-priori tour can be arbitrarily far from the actual cost of the tour realization once the customers' demands are known. An idea is that of designing an instance where the probability of restocking at a given customer suffieciently far from the depot is very small, and therefore the cost of restocking gives a very small contribution to the expected cost formula, but indeed restocking can occur and when it does the cost of the actul tour is far from the expected cost. Of course in average the expected cost formula still gives a good estimate in this case.

2.5 Threshold investigation

In the VRPSD problem with the restocking strategy the *thresholds* are important quantities that are associated to each particular a priori solution. The threshold have an important practical meaning, they are used by the vehicle driver to decide whether, on the base of the remaining load, it is better to proceed to the next scheduled customer, or first to go for restocking to the depot.

The thresholds are evaluated together with the objective value relative to a specific a priori solution. In order to evaluate the quality of a solution, and/or to guide the search, VRPSD algorithms usually do not take into account the value of the thresholds, but only the value of the objective function. It is possible that, by better investigating the 'behavior' of the thresholds, one could exploit some of their properties, in order to better guide the search.

Let us recall more precisely the exact definition of threshold. Let $0 \to 1 \to 2 \cdot \cdot\cdot \to n$ be a particular planned vehicle route. After the service completion at customer j, suppose the vehicle has a remaining load q, and let $f_j(q)$ denote the total expected cost from node j onward. The expected cost of the planned route (that is, the objective function value) is then $f_0(Q)$, and the problem consists in finding a route with the least expected total cost. If S_j represents the set of all possible loads that a vehicle can have after service completion at customer j, then, $f_j(q)$ for $q \in S_j$ satisfies

$$f_j(q) = \text{Minimum} \begin{cases} f_j^p(q) \\ f_j^r(q) \end{cases}, \tag{12}$$

where

$$f_j^p(q) = c_{j,j+1} + \sum_{k:k \leq q} f_{j+1}(q-k)p_{j+1,k} + \sum_{k:k>q} [b + 2c_{j+1,0} + f_{j+1}(q+Q-k)]p_{j+1,k}, \tag{13}$$

and

$$f_j^r(q) = c_{j,0} + c_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q-k)p_{j+1,k}, \tag{14}$$

with the boundary condition

$$f_n(q) = c_{n,0}, \ q \in S_n. \tag{15}$$

In equations (12-14), $f_j^p(q)$ represents the expected cost of proceeding directly to the next node, and $f_j^r(q)$ represents the expected cost of the restocking action. Given a certain a priori solution, for each customer, the smallest value of q such that $f_j^p(q) \leq f_j^r(q)$ is called the *threshold* of customer j (see Figure 1).

A limitation of dealing with a priori solutions is that one does not know in advance at which customer the restocking will be done. This makes it impossible to use more than one vehicle for visiting the customers in the order established by the a priori solution. Here is a possible use of thresholds. If, for some customer j, the threshold h_j exceeds the total vehicle capacity Q, it means that after serving j the action of restocking must be done, and that could be considered as the end of service of the first vehicle. In this case, two vehicles could be used to shorten the total time required of visiting all customers according to the a priori solution. (But it is not clear if such a situation ever occurs in practice).

2.6 Homogeneous Failure and Restocking

The first round of experiments on the vehicle routing problem with stochastic demand has shown that the evaluation of the cost function is particularly expensive. Among the metaheuristics implemented, those that adopted an approximation of the cost function, rather than the full evaluation, obtained significantly better results.

So far, all metaheuristics implementation that have resorted to an approximation of the VRPSD cost function where TSP-based: the cost of an *a priori* tour, rather than being evaluated exactly, was approximated by its cost in the TSP sense. The rationale behind such approximation is related to the apparent similarities between VRPSD and TSP. It could be further noticed that the approximation becomes exact in the degenerate case in which the truck capacity exceeds the sum of the maximum demand of all customers.

Another problem that presents similarities with VRPSD is the capacitated vehicle routing problem (CVRP). Indeed, VRPSD degenerates into CVRP if the spread in the demand of the customers is null—in other words, if the demand is deterministic. So far none of the implementations proposed within the Metaheuristics networks exploit this similarity. A possible reason is that it seems quite complicated to predict with sufficient accuracy the load of the truck along the *a priori* tour in order to define locations at which a re-stocking will be needed.

The aim of this section is to describe a possible way to tackle this problem: The homogeneous failure and restocking approximation of the VRPSD cost function.

2.6.1 The homogeneity hypothesis

The hypothesis underlying the discussion we propose is the following:

> Hp_{hfr}: The probability of failure and restocking is uniform along the *a priori* tour.[2]

This hypothesis can be seen as a way of representing the total ignorance on the positions along the *a priori* tour at which failures and restockings will occur.

Under hypothesis Hp_{hfr}, and assuming that the probabilities of failure and restocking are known,[3] say f and r respectively, the cost of a given *a priori* tour can be expressed as the

[2]Some *border effect* exists. For instance, for what concerns the first and last link in the tour, i.e. depot-first_customer and last_customer-depot, the probability of failure and restocking are both null. This border effect will be considered in the following.

[3]This second assumption will be removed in next section.

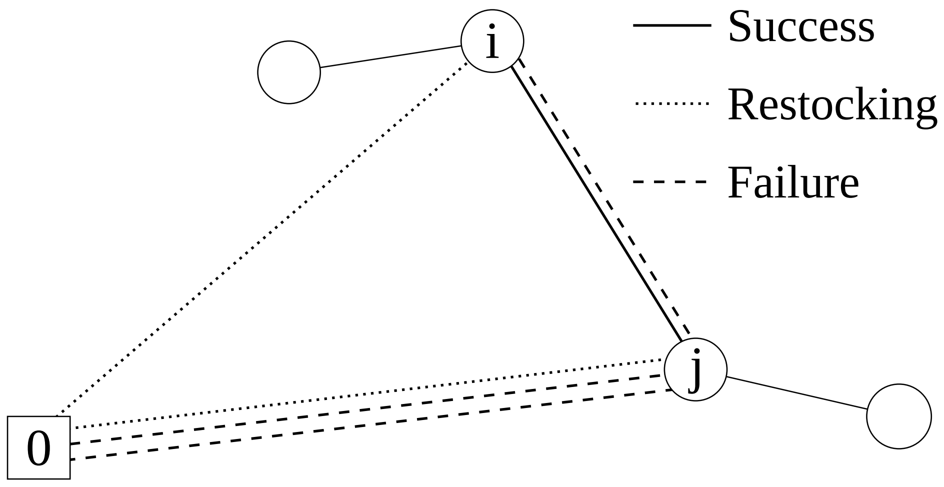


Figure 9: Inclusion on link $\langle i,j \rangle$ in the solution: The cost of including link $\langle i,j \rangle$ is a weighted average of the distance $\langle i,j \rangle$ (Success), the distance $\langle i,0,j \rangle$ (Restocking), and the distance $\langle i,j,0,j \rangle$ (Failure).

sum of individual contributions, one per each arc composing the *a priori* tour itself. Namely, say that the arc $\langle i,j \rangle$ is included in the *a priori* tour. Its contribution to the cost is:

$$C_{i,j} = (1 - f - r)D_{i,j} + r(D_{i,0} + D_{0,j}) + f(D_{i,j} + D_{j,0} + D_{0,j}) \tag{16}$$

where $D_{i,j}$ is the distance between i and j, $D_{i,0}$ is the distance between i and the depot, and $D_{j,0} = D_{0,j}$ is the distance between node j and the depot. See Figure 9 for a graphical representation of the quantities involved in Equation 16.

2.6.2 Possible search schemes

In this section we discuss possible search schemes based on the Hp_{hfr} hypothesis. To this end, let us first notice that the restocking and failure probabilities are clearly solution-dependent: In the general case, we should expect that two different *a priori* tours, say T_1 and T_2, are characterized by two different sets of restocking and failure probabilities, say r_1 and f_1 for the former, and r_2 and f_2 for the latter.

Moreover, it should be clear from Section 2.6.1 that if the restocking and failure probabilities associated with the optimal solution where known in advance, the optimal solution itself could be found by solving a TSP a graph obtained from the original one where the cost of the generic arc $\langle i,j \rangle$ is set to C_i, j as defined in Equation 16. On the other hand, when a solution is given, possibly the optimal one, the corrisponding restocking and failure probabilities can be obtained either analytically or empirically through Monte Carlo sampling.

In the typical case, however, neither the optimal solution is given (clearly...), nor the associated restocking and failure probabilities. To tackle such "chicken and egg" situations in which if one of two elements where known, the other could be easily obtained, and viceversa, but none is available, the typical approach is based on an iterative procedure. In the context of our VRPSD, an iterative solution scheme would be implemented as follows:

1. set current solution to some initial solution,
2. evaluate failure and restocking probabilities for current solution,
3. solve problem for current probabilities, and update current solution,
4. stop upon convergence; otherwise, go back to 2.

2.6.3 Discussion

We expect this approach to be effective if the uncertainty is high on the positions along an *a priori* tour at which restocking and failures will occur. In such a case, the Hp_{hfr} hypothesis is not restrictive and indeed models correctly the information available.

On the contrary, we do not expect this approach to be extremelly effective in the case in which the uncertainty is relatively low as for example if the capacity of the truck is very large compared to the average request of the customers, and if the spread of the request is relatively small. In such a case, since the number of customers served with one single load of the truck is high, the actual realization of the sum of the requests of the customers served between two consecutive stops at the depot will tend to converge to its expected value (see law of large numbers) washing out, *de facto*, most of the uncertainty. In such a case, the Hp_{hfr} hypothesis is possibly too restrictive and the solution schemes proposed in this document disregards some usefull information that could possibly be profitably employed.

2.7 New Experiments

The results of the first phase showed a certain hetereogenity in the basic components of the algorithms. Some of the implementation(s) exploited the similarities of the problem with the TSP than just tackling it as a VRPSD.

On one side this allowed to have a better insight on the problem, raising also questions on the similarities of the problem with the TSP. On the other we did not find any way to directly compare the metaheuristics. The only significative result was that those with better performances where the implementations that more deeply exploited the similarities with TSP, and with a better starting solution.

The goal, starting from this situation, was to design an experimental framework that allows us to answer the following research questions:

- how the metaheuristics compare each other on a fair basis?
- how much (of) the exploitation of the TSP similarities can improve the algorithms performance?
- how much the single metaheuristics can exploit these similarities?
- how TSP and CVRP solutions are related to VRPSD ones?

The framework that has been chosen fixes several implementation points:

Initial solution: randomized *furthest insertion*;

Local search: sampled OrOpt;

Selection: the selection critieria on the different metaheuristics will be based on the exact objective function for the VRPSD.

Furthermore every metaheuristic will be implemented using two different approximated objective functions during the local search:

- VRPSD proxy objective function;
- TSP objective function.

We will refer the metaheuristic of these groups as respectively *fair* and *flim-flamized*. The flim-flamized version has a TSP objective function which is a very coarse grained approximation to the VRPSD one but with a lower computation time. The proxy objective function is a finer approximation but more time consuming than the previous one. The higher number of solutions visited by the first version due to its fast objective function evaluation seems to overcome the lost on the approximation quality.

Within each group the metaheuristic will be fairly comparable. Furthermore the differences in performance between the two groups for the same metaheuristic will give information about the exploitation of TSP similarities.

In addition, two state of the art algorithms to solve TSP and CVRP will be also executed. This should give some information on how much is the stochasticity important in the problem .

The stopping criteria for all the algorithms will be determined by a time limit. This is fixed as the time needed to run 15 times a random restart with OrOpt local search with the VRPSD proxy objective function, and the same starting solution that will be used by the other algorithms (randomized farthest insertion heuristic).

2.8 Real-World Applications

The Vehicle Routing Problem with stochastic demand finds application in all those cases where the orders of the costumers are modified upon arrival or where it is impossible to predict the demand of the costumers. In the meeting we pointed out two specific cases for which an optimization that keeps into account the stochasticity of the problem could help in obtaining improved routes and to reduce costs. The pick up of garbage is one of them: it is, indeed, impossible to know a priori how much garbage have to be collected at each place. The delivery of petrol to petrol stations is another case subject to stochasticity of demand. When a customer issues the order it is still unknown how much he will sell in the time between the order and the new delivery. Other real applications can certainly be found. Nevertheless it appears that companies are not used to consider demands expressed by probability distributions.

Another aspect that was discussed is the convenience between mixed and online strategy. We believe that the computational effort needed to re-plan the tour every time the demand become certain is nowadays affordable. Therefore, the reasons for the choice of the policy to adopt are shifted to the agreements with the drivers, who prefer to know a priori the tour they have to follow and to other minor constraints imposed by the customers (like precedence constraints).

2.8.1 Examples of real world problems

Here is an example of a real world vehicle routing problem. Every day a company has to deliver goods from a central depot to a number of customers. In order to accomplish the whole goods distribution, the company can make use of an infinite non-homogeneous fleet of vehicles. Each type of vehicle has a specific capacity as well as specific characteristics (dimensions, trailer, embedded lift for loading/unloading goods...). Each customer i has a request qi, a service time si, a service time window [bi, ei] and a vehicle accessibility list that specifies which vehicles are allowed to service the customer. The problem is to perform the distribution while minimizing the number of tours, the total traveled distance and the total waiting and delay time.

Another real world problem, which may be regarded as an extension of the previous one, is a pick-up and delivery problem. In this case, there is no central depot: the goods have to be delivered from a set of pick-up points (each one having its service time, time window and vehicle accessibility list), to a set of delivery points. Here the goal is the minimization of the number of tours and the maximization of their efficiency (that is, the filling percentage of the vehicle, weighted on the travel distances along the tour).

It is important to mention that in any real life vehicle routing problem, besides the explicit constraints of the problem itself, there always is a set of implicit (and sometimes "fuzzy") constraints, whose handling may not be trivial. An example is the shape of the tours in the solution: some people want them to be circle-like, others want them to be more radial, or they don't want to have tours including customers of two distinct regions (even if those regions are adjacent one another...). Another example of fuzzy constraint is to have, "when this is possible", certain customers in the last position of their tour. Furthermore, while in academic problems the vehicle capacity constraint is mono-dimensional, in real problems this is a multi-dimensional constraint in the sense that each vehicle has a maximum capacity in pallets, a maximum capacity in kg and a maximum capacity in cubic meters. So, when building solutions, one should check that none of the capacities are exceeded.

2.9 Possible Extensions of the Problem

One of the aims of the Metaheuristics Network is to analyze the behavior of metaheuristics on possible reusable test cases problem. Therefore, in the same way as done for the timetabling problem, we investigated other possible formulations of Vehicle Routing, mainly with deterministic demand, in order to distinguish common features. On a side we faced the complexity of real world on the other we were constrained to simplify the problem in order to make analysis computationally feasible. In this discussion the experience in real cases provided by Ant Optima was illuminating. We came out with the following formulation that tries to approximate several over-constrained real cases still maintaining a certain generality.

A non-homogeneous fleet of vehicles has to be routed to deliver or pickup goods. Vehicles can differ for fixed costs, variable cost and capacity. Accessibility constraints restrict the use of vehicles to specific customers and to specific paths while time constraints impose that a vehicle visit a customer not later than its due date. Time constraints can be multiple, that is, composed by several intervals. The number of vehicles to use is biased by a maximum make-span per vehicle which could be expressed in time or in number of customers to visit and by the objective function to minimize. The objective function is the weighted sum of fixed costs per vehicle, cost per vehicle according to its use and total traveled length. However, the definition of weights and fixed costs should be carefully analyzed because the property to determine the number of vehicles depends from their interaction. Resuming, we can identify the following hard and soft constraints.

Hard Constraints:

- maximum make-span per vehicle
- accessibility
- Non homogeneous fleet (different fixed cost, different cost per kilometer, different capacity)
- multiple intervals time windows.

Soft Constraints (to minimize):

- fixed cost per vehicle
- variable cost per vehicle
- expected length

The problem thus defined is at the core of several real cases. Stochastic demands could be included and the work done for the VRPSD simply reused to optimize each single route. Nevertheless, we conjectured that time windows, meant as hard constraints, would require to consider only worst cases behavior, that is, the maximum possible demand for each customers. Indeed, the mixed strategy does not guarantee against failures, and failures drastically modify timetable making it easily infeasible in the case of strict time windows rules.

2.10 Meta-Problem for Constructive Optimization Algorithms

In general[4] we have an optimization problem $\mathcal{P}$ and an instance x of $\mathcal{P}$.

Typically, the set S_x of possible solutions is exponentially large in the size of the input x. The goal is to obtain an optimal (or good) solution for x belonging to some set $S_x^* \subseteq S_x$ of optimal (or good) solutions.

Constructive algorithms work as follows. Assume the elements of S_x^* can be viewed as compositions of $l_x \in \mathbf{N}$ elements of a smaller set Σ. For example, S_x^* might be some set of strings of length l_x over an alphabet Σ. Any element of S_x^* can be constructed by concatenating l_x elements of Σ.

The following method for constructing elements of S_x^* (even though unlikely to succeed in general) is instructive: Let Y be the empty string and proceed in l_x steps. In each step, select a random element of Σ and append it to Y. If it becomes clear that Y cannot be extended into any element of S_x^* (i.e. that no element of S_x^* has prefix Y), abort.

An algorithms of this kind can be described equivalently as a walk from the root of a tree to a node at depth l_x. The tree has nodes for $y \in S_x^*$ and for all prefixes of elements of S_x^*. The root of the tree is the empty string. There is an edge from node w to node v if v can be obtained by appending an element of Σ to w (i.e. if $\exists a \in \Sigma : v = wa$). Thus, the nodes

[4]The topic covered in Section 2.10 is of more general character and is not directly related to the VRPSD. Nevertheless, in Moltrasio we though it was particularly interesting; we have therefore decided to include an extract of the discussion in this report.

at depth i correspond to strings of length i. The set of nodes at depth l_x is S_x^*. If v is a node corresponding to a string of length $l > 0$ then the length $l - 1$ prefix w of v is also a node. Thus, for every $y \in S_x^*$, there is a path of length l_x from the root to y. In addition to S_x^* and all its prefixes, the tree may contain nodes at depths less than l_x which do not have descendants at depth l_x.

It is not necessary to store the entire tree. An efficiently computable predicate P on Σ^*, such that $P(y)$ is true if and only if string y is a node of the tree, is sufficient. P can be viewed as a pruning procedure. The children of any given node w can be computed by evaluating $P(wa)$ for all $a \in \Sigma$. This is sufficient for our purposes.

Whether search trees encountered in practice correspond more closely to balanced or unbalanced trees is determined by the combination of the characteristics of the underlying problem instance and the search heuristics, pruning, neighborhood functions, propagation methods employed, etc. Balanced trees occur when such techniques are relatively ineffective in the problem domain under consideration. On such problem instances any search algorithm tends to degrade to a form of exhaustive search. Fortunately, several problems from real-world applications have much more structure, and local search heuristics, pruning techniques can be quite effective.

3 Second experimental phase

3.1 The Algorithms

The results of the first experimental phase showed a certain heterogeneity in the basic components of the algorithms. Some of the implementation(s) exploited the similarities of the problem with the TSP than just tackling it as a VRPSD.

On one side this allowed to have a better insight on the problem, raising also questions on the similarities of the problem with the TSP. On the other we did not find any way to directly compare the metaheuristics. The only significant result was that those with better performances where the implementations that more deeply exploited the similarities with TSP, and with a better starting solution.

The goal, starting from this situation, was to design an experimental framework that allows us to answer the following research questions:

- how the metaheuristics compare each other on a fair basis?
- how much (of) the exploitation of the TSP similarities can improve the algorithms performance?
- how much the single metaheuristics can exploit these similarities?
- how TSP and CVRP solutions are related to VRPSD ones?

The framework that has been chosen fixes several implementation points:

Initial solution: randomized *furthest insertion* (FR in the following);

Local search: OrOpt local search;

Selection: the selection criteria on the different metaheuristics will be based on the exact objective function for the VRPSD.

Furthermore every metaheuristic will be implemented using two different approximated objective functions during the local search:

- VRPSD proxy objective function;
- TSP objective function.

We will refer the metaheuristic of these groups as respectively *-0* and *-tsp*. The -0 version has a TSP objective function which is a very coarse grained approximation to the VRPSD one but with a lower computation time. The proxy objective function is a finer approximation but more time consuming than the previous one. The higher number of solutions visited by the first version due to its fast objective function evaluation seems to overcome the lost on the approximation quality.

Within each group the metaheuristics are fairly comparable. Furthermore the differences in performance between the two groups for the same metaheuristic give information about the exploitation of TSP similarities.

In addition, two state of the art algorithms to solve TSP and CVRP have also been executed. This should give some information on how much is the stochasticity important in the problem.

	FR (-0 and -tsp)	*OrOpt (-0 and -tsp)*	*-tsp*
TS	starting solution	does not use OrOpt as black-box, but only neighbourhood structure and move cost evaluation	uses tour length for choosing best-move-of-the-neighbourhood, but not for accepting criterion
SA	“ ”	“ ”	uses tour length for choosing move and also for accepting criterion
ILS	“ ”	applied after each perturbation	uses tour length only in OrOpt (OrOpt-tsp)
GA	starting population (10 solutions)	applied to starting population and to each offspring, that will replace worst pop. member	“ ”
ACO	reinforcement of initial pheromone	applied to each ant's solution	“ ”

Table 1: Components common to all metaheuristics analyzed, and use of TSP objective function (that is, tour length instead of expected tour length).

A detailed description of the algorithms used is done in the remainder of this section, and a schematic view of how the common components FR and OrOpt and how the analogy with the TSP are used in the various metaheuristics is shown in table 1.

3.1.1 The OrOpt local search

The OrOpt algorithm has been originally proposed by Or [30], for the TSP. Given a starting tour, the basic operation of the OrOpt algorithm consists of moving a string of size 3, 2, or 1 from one position to another in the tour. Here, we propose two versions of the algorithm: OrOpt-0 and OrOpt-tsp, which differ in the way the neighboring solutions are evaluated. The first version exploits the VRPSD objective function (that is, the expected cost of the a-priori tour), while the second version exploits the TSP objective function, that is, the length of the a-priori tour. OrOpt-0 and OrOpt-tsp are described in detail in the following paragraphs.

The OrOpt-0 local search Checking if an OrOpt move leads to a better tour may be done in two stages. First compute the saving from extracting the string from the tour, and second compute the cost of inserting it back somewhere else in the tour, after the extraction point. Computing these costs and savings in the deterministic case is quite simple, since they can usually be computed in constant time. In the stochastic case, however, it is computationally demanding, because it requires the dynamic programming recursion of equations (2-4). This leads to an $O(nKQ)$ time for the computation of the cost and saving of just one OrOpt move. In order to reduce the computational time, the following approximation scheme suggested by Yang et al. [38] have been used.

Given a string S of consecutive nodes in the a-priori tour, the approximate saving of extracting it from the tour is computed as follows. Let l and t be the nodes immediately preceding, resp. following, S in the tour, and let $f_l^{beforeExt}(q)$ and $f_t(q)$ be the corresponding cost vectors before the extraction of S. Apply one dynamic programming recursion step starting with cost vector $f_t(q)$ at node t back to node l, without considering the string S. Let $f_l^{afterExt}(q)$ be the resulting cost vector at node l, that is, after extracting S from the tour. Then, define the approximate extraction saving as a simple average over q of $f_l^{afterExt}(q) - f_l^{beforeExt}(q)$, that is,

$$\text{Approximate Extraction Saving} = \frac{\sum_{q=0}^{Q}(f_l^{afterExt}(q) - f_l^{beforeExt}(q))}{Q+1}, \tag{17}$$

The computation of the insertion cost of S between nodes i and j in the tour, is done analogously, if we assume that the insertion point (node i) is after the extraction point (node l). Let $f_i^{beforeIns}(q)$ be the cost vector at node i for the current tour, that is, before inserting S in the tour. Apply dynamic programming recursion starting with cost vector $f_j(q)$ at node j through nodes of the inserted string, and back to node i. Let $f_i^{afterIns}(q)$ be the resulting cost vector at node i, that is, after inserting the sting in the tour. Then, define the approximate insertion cost as a simple average over q of $f_i^{afterIns}(q) - f_j^{beforeIns}(q)$. That is,

$$\text{Approximate Insertion Cost} = \frac{\sum_{q=0}^{Q}(f_i^{afterIns}(q) - f_j^{beforeIns}(q))}{|S_i|}. \tag{18}$$

The total approximate cost of an OrOpt move is computed by subtracting the Approximate Extraction Saving from the Approximate Insertion Cost:

$$\text{Approximate OrOpt Move Cost} = \text{equation (18) - equation (17)}. \tag{19}$$

Note that the cost vectors are assumed to be already available from the computation of the expected cost for the starting tour, thus, they do not need to be computed when evaluating the Approximate Insertion Cost. The only computations that must be done here are the evaluation of cost vectors $f_l^{afterExt}(q)$ and $f_i^{afterIns}(q)$, and the averages in equations (17) and (18). The dynamic programming recursion for the computation of the cost vectors requires $O(KQ)$, while equations (17) and (18) require $O(Q)$ time. Therefore, with the proposed approximation, the cost of an OrOpt move can be computed in $O(KQ)$ time.

The proposed approximation scheme has some potential drawbacks though. It neglects the influence of the inserted string (or deleted string) on the nodes before node i. In principle it is thus possible that a certain OrOpt move seems good, when evaluated by the approximation scheme, but it is actually a worsening move, when evaluated by the exact objective function computation. Therefore if the approximation scheme is used to evaluate moves in the OrOpt algorithm (or in any other local search), there is no guarantee that a better tour than the starting one will be found. In practice this approximation should behave quite well since, as reported in [38], it should find the same tout as the one obtained if the exact costs were computed, with significantly less computational effort (less than 10%). The OrOpt-0 algorithm, implementing the proposed approximation scheme, is outlined in procedure 4.

The OrOpt-tsp local search This version of the OrOpt local search computes the cost of an OrOpt move by just making the difference between the length of the tour after the move has been applied and the length of the original tour. Such difference may be computed in constant time, by considering only the arcs involved in the move. For instance, suppose we want to move a string S of consecutive nodes to another position in the tour, like in figure 3.1.1. Let l and t be the nodes immediately preceding, resp. following, S in the original tour,

Procedure 4 OrOpt-0 local search for the VRPSD

```
1: Let $k = 3$ and $T$ be an initial a-priori tour.
2: Compute the expected cost of tour $T$ by using the dynamic programming recursion of
   equations (2)-(4).
3: for all $S^k$, a set of $k$ successive nodes from $T$ do
4:     select at random a node along the a-priori tour after the node immediately preceding
       $S^k$ and compute the Approximate OrOpt Move Cost by using equation (19).
5: end for
6: if none of the sets in Step 3 and 4 results in a negative Approximate OrOpt Move Cost
   then
7:     go to Step 11.
8: else
9:     Select the set $S^k$ that results in the most negative Approximate OrOpt Move Cost in
       Step 3, and perform the corresponding OrOpt move. Then, go to Step 3.
10: end if
11: if $k = 1$ then
12:     stop
13: else
14:     decrease $k$ by 1, and go to Step 3.
15: end if
```

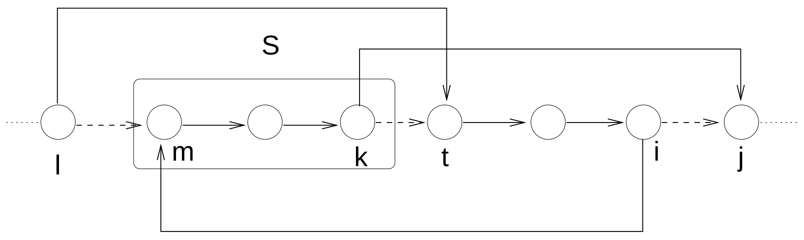


Figure 10: Arcs involved in the computation of the cost move in the OrOpt-tsp local search.

let i and j be the nodes between whom S is to be inserted, and let m and k be the first and the last nodes of S. Then, the move cost in the OrOpt-tsp sense may be computed as follows:

$$\text{OrOpt-tsp Move Cost} = d(l,t) + d(k,j) + d(i,m) - d(l,m) - d(k,t) - d(i,j). \tag{20}$$

Apart from the way the move cost is evaluated, the OrOpt-tsp explores the neighbourhood of a solution in the same way as the OrOpt-0, as outlined in procedure 5.

Procedure 5 OrOpt-tsp local search for the VRPSD

```
1: Let $k = 3$ and $T$ be an initial a-priori tour.
2: Compute the expected cost of tour $T$ by using the dynamic programming recursion of equations (2)-(4).
3: for all $S^k$, a set of $k$ successive nodes from $T$ do
4:     select at random a node along the a-priori tour after the node immediately preceding $S^k$ and compute the OrOpt-tsp Move Cost by using equation (20).
5: end for
6: if none of the sets in Step 3 and 4 results in a negative OrOpt-tsp Move Cost then
7:     go to Step 11.
8: else
9:     Select the set $S^k$ that results in the most negative OrOpt-tsp Move Cost in Step 3, and perform the corresponding OrOpt move. Then, go to Step 3.
10: end if
11: if $k = 1$ then
12:     stop
13: else
14:     decrease $k$ by 1, and go to Step 3.
15: end if
```

3.1.2 The randomized farthest insertion algorithm

The farthest insertion heuristic is a very simple heuristic for the TSP problem [2]. It builds a tour by choosing as next customer the not-yet-visited customer which is farthest, that is, the feasible customer that corresponds to the biggest distance from the current customer.

The randomized farthest insertion heuristic for the VRPSD builds a solution starting from a customer chosen at random. After the tour has been completed, it shifts the tour to start at the depot, as required by the problem.

3.1.3 Tabu Search

A tabu search (TS) algorithm is composed by three main elements: the starting solution, the neighborhood structure and the strategy of selection of new solutions. In our TS, the starting solution was generated by the randomized farthest insertion heuristic. The neighborhood structure is derived from the OrOpt local search, described in section 2.1.1. The strategy of selection of new solutions is a descent-ascent method with the use of tabu list. Procedures 6 and 7 outline the two versions of TS that we have developed, TS-0 based exclusively on the VRPSD objective function, and TS-tsp also exploiting the TSP objective for speeding up the neighborhood exploration. The emphasized lines of the two procedures correspond to instructions that are different in TS-0 and in TS-tsp.

Procedure 6 TS-0

```
set the length $k$ of the strings to be moved as $k = 3$;
while time is not over do
  if $(k < 1)$ then
    set $k = 3$
  end if
  set the length of tabu list as a random number in the interval $[0.8 \cdot (n-k-1), n-k-1]$,
  where $n$ is the number of customers;
  for all potential moves among those with string length $k$ do
    compute the VRPSD-like approximate move cost by equation (19);
    update the best-move-of-the-neighbourhood;
    check if the move is tabu;
    update the best-move-of-the-neighbourhood;
  end for
  if a best-move-of-the-neighbourhood exists then
    modify the current solution by performing the best-move-of-the-neighbourhood;
    update the best-so-far solution since the start of the TS;
    if the new current solution is not a new best-so-far solution then
      decrease the string length $k$ by 1;
    else
      set $k = 3$;
    end if
    set the tabu-move;
  else
    tabu-move = best-move-of-the-neighbourhood;
  end if
  add the tabu-move at the end of the tabu list;
  prune the tabu list from the beginning, so that its length is equal to the tabu length set
  at the beginning;
end while
```

Procedure 7 TS-tsp

```
set the length $k$ of the strings to be moved as $k = 3$;
while time is not over do
  if $(k < 1)$ then
    set $k = 3$
  end if
  set the length of tabu list as a random number in the interval $[0.8 \cdot (n-k-1), n-k-1]$,
  where $n$ is the number of customers;
  for all potential moves among those with string length $k$ do
    compute a tsp-like move cost by equation (20);
    check if the move is tabu;
    update the best-move-of-the-neighbourhood;
  end for
  if a best-move-of-the-neighbourhood exists and performing the move would lead to a better
  solution in terms of the VRPSD objective function then
    modify the current solution by performing the best-move-of-the-neighbourhood;
    update the best-so-far solution since the start of the TS;
    if the new current solution is not a new best-so-far solution then
      decrease the string length $k$ by 1;
    else
      set $k = 3$;
    end if
    set the tabu-move;
  else
    tabu-move = best-move-of-the-neighbourhood;
  end if
  add the tabu-move at the end of the tabu list;
  prune the tabu list from the beginning, so that its length is equal to the tabu length set
  at the beginning;
end while
```

Now some observations in order to explain the tabu search pseudocode.

Neighbourhood exploration The neighborhood is explored in such a way that at the beginning of the search, only moves with string length equal to 3 are tried. The string length is decreased each time that no best move can be found, and reset equal to 3 each time a new best-so-far solution is found. Each time the string length has reached the minimum length (that is, 1), it is re-initialized to the maximum value (that is, 3).

Selection of the move to perform In our tabu search it is important the notion of best-move-of-the-neighbourhood because this is the move that is actually performed and leads to a new solution. Each time a new potential move is considered, it is evaluated and a decision is taken as if is a new best-move-of-the-neighbourhood or not. The decision criterion differs in the case of tabu or non-tabu moves. If the move is tabu, then, in order to be promoted to be new best-move-of-the-neighbourhood, the move should lead to a new best-so-far solution (according to the approximated move cost in TS-0 and according to the tour-length difference in TS-tsp). If the move is not tabu, then, it is enough that it leads to a new best solution with respect to the the current neighborhood. Not all moves are evaluated, in fact, tabu moves are evaluated only with a probability of the 30%, and non-tabu moves with a probability of the 80%. Also for this reason, sometimes no move is selected as new best-move-of-the-neighbourhood. Once a new best-move-of-the-neighbourhood is selected, it may be performed, but with a different mechanism in TS-0 and in TS-tsp. in TS-0, the move is performed, and the exact objective value of the new solution is computed. In TS-tsp, the move is performed only if it really leads to an improving solution with respect to the VRPSD objective function.

Tabu moves In general, there are several possibilities for defining what is a tabu move. The aim is always to avoid going back to a solution that has been visited in the last few iterations of the local search. In our TS, the concept of tabu move is simple. After a string S has been extracted from position i and re-inserted at position j, there are two moves that become tabu: those that would re-establish the relative position of the customer immediately preceding, resp. following S in the original tour. The two tabu moves are exemplified in fig. 3.1.3.

Tabu list Each time a new best-move-of-the-neighbourhood is selected, it is put in the tabu list. In case there is no best-move-of-the-neighbourhood, the last selected best-move-of-the-neighbourhood is put again in the tabu list. After, the oldest tabu move is removed from the list.

3.1.4 Simulated Annealing

Simulated Annealing (SA) is a metaheuristics inspired by the process of annealing in physics [26, 37]. It is widely used to solve combinatorial optimization problems, especially to avoid getting trapped in local optima when using simpler local search methods [1]. This is done as follows: an improving local search move is always accepted while a worsening local search move is accepted according to a probability which depends on the respective deterioration in the evaluation function value, such that the worse a move is, the less likely it is to accept it.

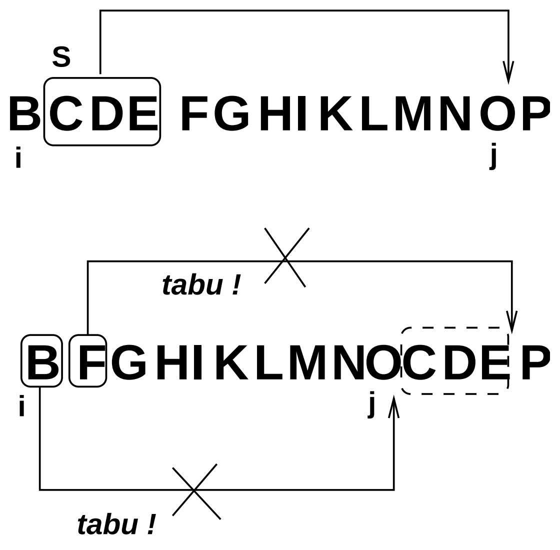


Figure 11: Example of tabu moves.

Formally a move is accepted according to the following probability distribution, known as the Metropolis distribution:

$$p_{accept}(T, s, s') = \begin{cases} 1 & \text{if} \quad f(s') \leq f(s) \\ exp\left(-\frac{f(s')-f(s)}{T}\right) & \text{otherwise} \end{cases} \tag{21}$$

where s is the current solution, s' is a neighbor solution and T is a parameter that controls the acceptance probability; T is allowed to vary over the course of the search process.

The acceptance criterion of equation (21) is the element that differentiates the two version of simulated annealing that we considered. In SA-0, $f(s') - f(s)$ is the value given by the the Approximate OrOpt Move Cost as computed in equation (19); while in SA-tsp $f(s') - f(s)$ is simply the difference between the length of tour s' and the length of tour s.

Procedure 8 schematically depicts how Simulated Annealing works. In our study the termination condition is fixed by the time available to solve the instance. In the following, we give a more detailed description of how we implemented the components, GenerateInitialSolution, SelectMove, UpdateTemperature, for the VRPSD distinguishing, where needed, between the SA-0 and SA-tsp.

Initial Solution. It is produced by the randomized farthest insertion heuristic (as described in section 3.1.2).

Move Selection. The examination of the neighborhood uses the strategy adopted in the common OrOpt local search.

Temperature updating. The temperature profile is determined by three main features: the initial temperature, the cooling schedule and the re-heating schedule.

Initial Temperature. A sample in the neighborhood of the initial solution is used to compute the average of the evaluation function. A number of 100 neighbors is

Procedure 8 Simulated Annealing (SA-0 and SA-tsp)

```
$s \leftarrow$ GenerateInitialSolution()
$T \leftarrow T_0$
while termination condition not met do
    $s' \leftarrow$ SelectMove($\mathcal{N}(s)$)
    if $f(s') \leq f(s)$ then
        $s \leftarrow s'$
    else
        $s \leftarrow$AcceptanceCriterion(T,$s$,$s'$)
    end if
    UpdateTemperature($T$)
end while
```

used. The value found is then multiplied by a given factor f which is a parameter of the algorithm. The evaluation function considered is the VRPSD expected cost of the a-priori tour in SA-0 and the length of the a-priori tour in SA-tsp.

Cooling schedule. We used a non monotonic temperature schedule realized by the interaction of two strategies: a standard geometric cooling and a temperature re-heating. In the standard geometric cooling the temperature T_{n+1} is computed from T_n as $T_{n+1} = \alpha \times T_n$ where α is a parameter called the cooling rate $(0 < \alpha < 1)$. The number of iterations at which the temperature remains constant is kept proportional to the size of the neighborhood as suggested in Johnson et al. [24]. Hence, $TL = q \cdot |\mathcal{N}|$, where q is a parameter and $|\mathcal{N}|$ is the size of the neighborhood. In the case of the common local search examination strategy it is of order $\mathcal{O}(n^3)$ with n being the number of customers.

Re-heating. When the search appears to stagnate, the temperature is increased by adding T_i to the current temperature. This is done when no improvement is found for a number of steps given by $r \cdot TL$ where TL is the temperature length and r is a parameter. The solution that we consider for testing improvements is the best since the last re-heating occurred.

For tuning the parameters f, α, q and r, on the classes of instances that will then be used for the comparison of metaheuristics, we set up an experimental phase. This involved 74 candidate versions of Simulated Annealing, obtained by a combination of the elements described above, and 19 heterogeneous instances. Instances were selected from the random uniform and the random clustered instances. Since we had to submit only one configuration we did not distinguished between classes. Termination times for each run of algorithm were determined by 15 random restarts of the common local search on the instance under examination. The evaluation methodology we used is the following: we run each algorithm once on each instance, we ranked the algorithms on the instances and we averaged the ranks over the instances. The SA-0 and the SA-tsp algorithms with the best average ranks were selected as representative of Simulated Annealing applied to VRPSD and submitted to compete against the other metaheuristics.

For SA-tsp the tuning procedure indicated the following values: $f = 0.05$,$\alpha = 0.98$,$q = 1$ and $r = 20$.

For SA-0, instead, we had a surprising result. The best configuration has temperature null. This means, that the acceptance criterion of Simulated Annealing is merely reduced to accepting all non worsening moves. Hence, Simulated Annealing does not provide any improvement to the common local search and the simple algorithm which repeats the local search for all the time available accepting also side walk moves works better.

Previous works and comments Beside applications to the Traveling Salesman Problem, Simulated Annealing has been applied also to the Vehicle Routing Problem. Osman [31] implementation of SA has also the feature of starting from an improved solution. It, then, uses a 1-interchange neighborhood and cools the temperature every time a move is accepted. Results are promising even if not the state of the art for VRP [36]. Dueck [14, 13], uses always on the VRP, a modified accepting criterion. In his "Deterministic" Annealing a new solution is accepted if and only if $f(s') < f(s) + \theta$ where θ is a parameter to control the process. In the specific case of Vehicle Routing with Stochastic Demand Simulated Annealing was firstly applied by Teodorović and Pavkovič [35]. They use Simulated Annealing in a construction phase, to accept or reject randomly generated routes of a fixed length determined by an analysis on the structure of the problem (the assumption on the length of routes is, in their case, possible since all customers have the same uniform distribution with the same average demand, which, instead, is not our case). In a second phase they use Simulated Annealing to improve each single routes by exchanging randomly the order of offering services in two nodes of the route. In both cases temperature is updated a fixed number of times when a thermal equilibrium is reached. Results are given on a single run on a single instance and are, therefore, hardly comparable.

However, in our attempt to use Simulated Annealing in the SA-0 version, the one that does solve the VRPSD with re-stocking policy, we faced the major problem that using the approximated method to evaluate the contribution of a move introduces already randomization on the search and the eventuality of accepting worsening moves. In this case it becomes really hard to control effectively the probabilistic acceptance criterion of SA by means of temperature schedule because it overlaps with the probabilistic criterion implicitly embedded in the evaluation function. Indeed, the observation of the profile of evaluation function in some runs with different temperature revealed almost no correlation with the behavior of temperature and, therefore, of the SA acceptance criterion.

3.1.5 Iterated Local Search

ILS is based on the simple yet powerful idea of improving a local search procedure by providing new starting solutions obtained from perturbations of the current solution, often leading to far better solutions than if using random restart. The local search is applied to the perturbed solution and a locally optimal solution is reached. If it passes an acceptance criterion, it becomes the new current solution; otherwise, one returns to the previous current solution. The perturbation must be sufficiently strong to allow the local search to explore new solutions, but also weak enough so that not all the good information gained in the previous search is lost. A detailed description of ILS algorithms can be found in [28].

To apply the ILS algorithm, four components have to be specified. These are a GenerateInitialSolution procedure that generates an initial solution s_0, a Perturbation procedure, that modifies the current solution s leading to some intermediate solution s', a LocalSearch procedure that returns an improved solution s'', and a procedure AcceptanceCriterion that decides to which solution the next perturbation is applied. A scheme for ILS is given in procedure 9.

Procedure 9 Itarated Local Search
```
$s_0 = \mathsf{GenerateInitialSolution}()$
$s = \mathsf{LocalSearch}(s_0)$
repeat
  $s' = \mathsf{Perturbation}(s, \mathit{history})$
  $s'' = \mathsf{LocalSearch}(s')$
  $s = \mathsf{AcceptanceCriterion}(s, s'', \mathit{history})$
until termination condition met
```

GenerateInitialSolution procedure consisted in applying the farthest insertion heuristic. The LocalSearch procedure was fixed to be the common OrOpt local search (OrOpt-0 for ILS-0 and OrOpt-tsp for ILS-tsp). The AcceptanceCriterion consisted in accepting s'' if it is better than s, *i.e.* the best solution found so far, according to the exact VRPSD objective function. The Perturbation consisted in a loop of n random moves of a 2-exchange neighborhood, *i.e.* subtour inversion between two randomly chosen customers. The loop is broken if, within n moves, a solution is found having an objective value smaller than the objective value of the best solution according to the exact VRPSD objective function, plus a certain value ε. Given the instances tackled, $\varepsilon = \frac{n}{10}$ was empirically the best value found on some preliminary runs. Otherwise, the best perturbed solution found is returned.

3.1.6 Ant Colony Optimization

For the second test phase the IRIDIA node implemented the Ant Colony Optimization (ACO) metaheuristic as an Ant Colony System (ACS) [12]. Two algorithms were implemented: ACS-0 where after the a-priori solution is constructed, the OrOpt-0 local search is applied, and ACS-tsp where the OrOpt-tsp local search is used instead of OrOpt-0.

In the ACS, a set of agents, called ants, build solutions for the VRPSD cooperating through pheromone-mediated indirect and global communication. Here we give only a brief description of the basic principles that lie beneath the ACS.

In the initialization phase each of the m ants recieve a starting solution that is generated by the randomized farthest insertion heuristic of section 3.1.2. The OrOpt local search is then applied to this starting solution to refine it. The best solution found by the colony becomes the initial starting solution that is used to "bias" the pheromone matrix reinforcing all the elements that belong to this initial starting solution by applying the global update rule for r times. At each iteration of the algorithm, each of the m ants constructs an a-priori tour one customer after the other so that starting with the depot all customers are visited one and only one time, ending again with the depot. The ants choose the next customer to be visited probabilistically, guided by stigmergic information. No heuristic information is used in those implementations. The stigmergic information is in the form of a matrix of 'pheromone' values $\tau : C \times C \rightarrow \Re_{\geq 0}$, where C is the set of customers, which are an estimate of the utility o f going from one customer i to a second one j in the a-priori tour, as judged by previous iterations of the algorithm. In ACS at the beginning of the algorithm the values in the matrix are initialized to a parameter τ_0, execpt for those elements that belong to the best starting solution, generated by the randomized farthest insertion heuristic, who recieve a "reinforcement" equal to r iterations of global update rule. After each construction step a local update rule is applied to the element of the matrix corresponding to the choosen customers pairs (i,j):

$$\tau(i,j) \leftarrow (1-\psi)\cdot\tau(i,j)+\psi\cdot\tau_0 \tag{22}$$

The parameter $\psi \in [0,1]$ is the pheromone decay parameter, which controls the diversification of the construction process. The aim of the local update rule is to permit to the m ants to choose different customers j for the same given customer i. After all the customers have been visited, the OrOpt local search routine is applied to the candidate solution s. At the end of the iteration the global update rule is applied to all the entries in the pheromone matrix:

$$\tau(i,j) \leftarrow \begin{cases} (1-\rho)\cdot\tau(i,j)+\rho\cdot\frac{Q}{q(s_{best})} & \text{if } (i,j) \text{ is in } s_{\text{best}} \\ (1-\rho)\cdot\tau(i,j) & \text{otherwise} \end{cases} \tag{23}$$

where Q is a parameter controlling the amount of pheromone laid down by the update rule, and the function q measures the expected cost of a candidate solution s. The parameters we have used for the algorithm are the following:

m	τ_0	α	ψ	ρ	Q	r
5	0.5	1	0.3	0.1	10^7	100

Algorithm 10 High level description of the ACS

```
input: An instance $x \in I$ of the VRPSD
/* Initialization phase */
Create the pheromone matrix $\tau$ and initialize every element to $\tau_0$
Each of the $m$ ants builds the initial solution using the fathest insertion heuristic
Apply to each initial solution the OrOpt Local Search
Bias the pheromone matrix toward the best initial solution applying global update for $r$ iterations
/* Building solution phase */
while time limit is not reached do
  Each of the $m$ ants builds an a-priori tour applying a probabilistic construction rule where choices are a function of the pheromone matrix $\tau$.
  /* Local Update phase */
  Applies the local update rule for encouraging diversification
  Apply to each initial solution the OrOpt Local Search
  /* Global Update phase */
  Applies the global update rule for reinforcing the best solution so far
end while
$s_{best} \leftarrow$ best solution found so far
output: $s_{best}$, "candidate" to optimal solution for $x \in I$
```

3.1.7 Genetic Algorithm

The VRPSD with the a priori strategy seems to be very similar to a TSP, at least in that the same space of a-priori tours between customers has to be searched. The simple memetic algorithm presented is therefore based on work on the TSP, and crossover and mutation operators were chosen from permutation based operators which work well for the TSP. Two different versions of the algorithm were implemented, the only difference between them being the local search they use, one using the OrOpt-0 with approximated cost function and the other the OrOpt-tsp with TSP cost function.

A small population size of 10 individuals is used. Initial solutions are built with the randomized farthest insertion heuristic. Local search is than applied to each member of the initial population. In the Steady State evolution process only one couple of parents reproduces at each generation. Tournament selection of size 2 is used to select which parents are going to be given the chance to reproduce. The crossover used in the final implementation is the Edge Recombination crossover (ER) of Whithley et al., readapted to always start at the depot. OX crossover readapted to always start from the depot, and PMX crossover were also implemented and tried, but ER crossover seems to work better. It tries to build an offspring exclusively from the edges present in both parents, as outlined in procedure 11. Swap based

Procedure 11 Simple Memetic Algorithm for the VRPSD

```
1: Create an edge list from both parents, that provides for each customer all the other
   customers connected to it in at least one of the parent;
2: start from the depot as current customer;
3: select the customer in the edge list of the current customer, with the smallest number of
   customers left in its edge list;
4: if there is no customer left in the edge list of the current customer then
5:   select a non yet visited customer;
6: end if
7: the selected customer becomes the current customer;
8: update the edge list by removing the current customer from each adjacent customer list;
9: if the tour is complete then
10:   stop.
11: else
12:   go to step 3.
13: end if
```

mutation swaps two adjacent alleles, never the depot. It is applied with an adaptive mutation rate with a maximum probability of 0.5. Order based mutation that picks 2 loci at random and exchanges their alleles, and inversion were also tried. Local search is again applied at each generation to improve the quality of the offspring. The improved offspring then replaces the worst member of the population.

3.2 Experimental Results of the Second Phase

The algorithms compared in the third phase are: a randomized farthest insertion random restart local search with VRP OrOpt (fr-0), a randomized farthest insertion random restart local search with TSP OrOpt (fr-tsp), ant colony system with VRP OrOpt (acs-0), ant colony system with TSP OrOpt (acs-tsp), evolutionary algorithm with VRP OrOpt (ea- 0), evolutionary algorithm with TSP OrOpt (ea-tsp), iterated local search with VRP OrOpt (ils-0), iterated local search with TSP OrOpt (ils-tsp), simulated annealing with VRP OrOpt (sa-0), simulated annealing with TSP OrOpt (sa-tsp), tabu search with VRP OrOpt (ts-0), tabu search with TSP OrOpt (ts-tsp), a state-of-the-art TSP algorithm implemented by Stützle (run.py), and a state-of-the-art VRP algorithm implemented by Gambardella (run.cvrp.py).

A description of the implementations is given in Section 2.1.

Four classes of instances[5]where considered:

rand_unif_n: 45 instances,

rand_unif_r: 50 instances.

rand_clust_n: 45 instances,

rand_clust_r: 49 instances,

Each algorithm was tested once on each instance for a time equal to the time needed by the algorithm fr-0 to complete 250, 500 or 1000 iterations, depending on the number of customers respectively 50, 100 or 200. So the execution time varied on an instance-by-instance basis.

The experiments were performed on the new cluster of PCs available at IRIDIA (polyphemus). For these experiments, 6 CPUs Athlon 1400 were used.

Figure 12 illustrates the results aggregated over the four classes of instances. Figures 13, 14, 15, and 16 illustrate the results obtained by the algorithms under analysis on the classes of instances rand_unif_n, rand_unif_r, rand_clust_n, and rand_clust_r, in this order.

3.3 How to read the tables and the graphics

For every class of instances and then for the entire set of instances we verify if differences in solutions found by the algorithms are statistical significant. We use the *Pairwise Wilcoxon rank sum* test [10] with *p-values*[6] adjustment method by Holm [20]. In the tables associated to the graphics we report for every pair of algorithms (A,B) the *p-values* for the null hypothesis "The distributions of the solutions generated by A and by B are the same".

The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

The diagrams show the distribution of the solutions found by the algorithms during independent executions. The results of all executions on the same instance are ordered by quality of the solution (expected cost) to determine the rank . Solutions are grouped by algorithm. In the *boxplot* the median is represented by the bar, the *box* represent the interval included in the quantile 25% e 75%; the whiskers extend to the more extreme points that are no more than 1,5 times the interquartile; the circles represent points who are outside this interval.

3.4 Discussion of the results

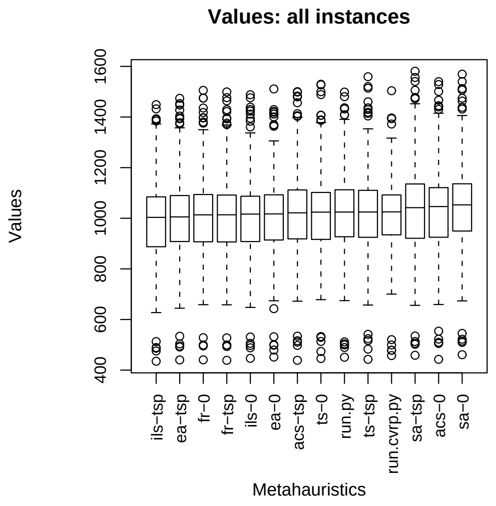
From the experimental results it is not possible to draw any strong conclusion about the relative performance of the metaheuristics, but the following observations can be done. The ILS metaheuristic is the only one which is always among the best, either in the -0 version or in the -tsp version. At the opposite extreme there is the SA metaheuristic, which is always

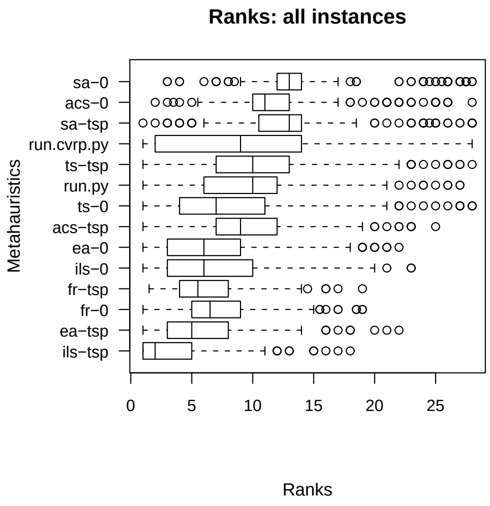
[5]The instances are such that customers coordinates are all different. Clustered instances have points normally distributed around two centers, whose coordinates belong to the $[0, 99]^2$ square.

[6]The *p-value* of a test is the probability that a value more extreme than the one that is obtained, in the direction of the alternate hypothesis, would have been obtained if the null hypothesis were true.

among the worst. The performance of the other metaheuristics seem to be not significantly different from the one of the fr-0 and fr-tsp algorithm. It has been hipothesized that this is due to the fact that all algorithms find solutions which are very near to the optimal, and thus the solution values found are not very different from each other. This would also mean that the tested instances can be solved to near optimality by the very simple FR heuristic. The testbest consisted of instances where the average number of restocks was fixed and equal to 4. This means that the number of customers before restocking was of the order of 12 to 50 customers. It has been observed that this number is not typical for real world instances, where the number of customers before restock is much smaller.

A question to which experimental data answer is whether VRPSD may be treated as a TSP. The answer is no, since the TSPsoa is never among the best algorithms. Moreover, there is no evidence that in general the -tsp version of a metaheuristic performs better that the -0 version.





Pairwise comparisons using Wilcoxon rank sum test

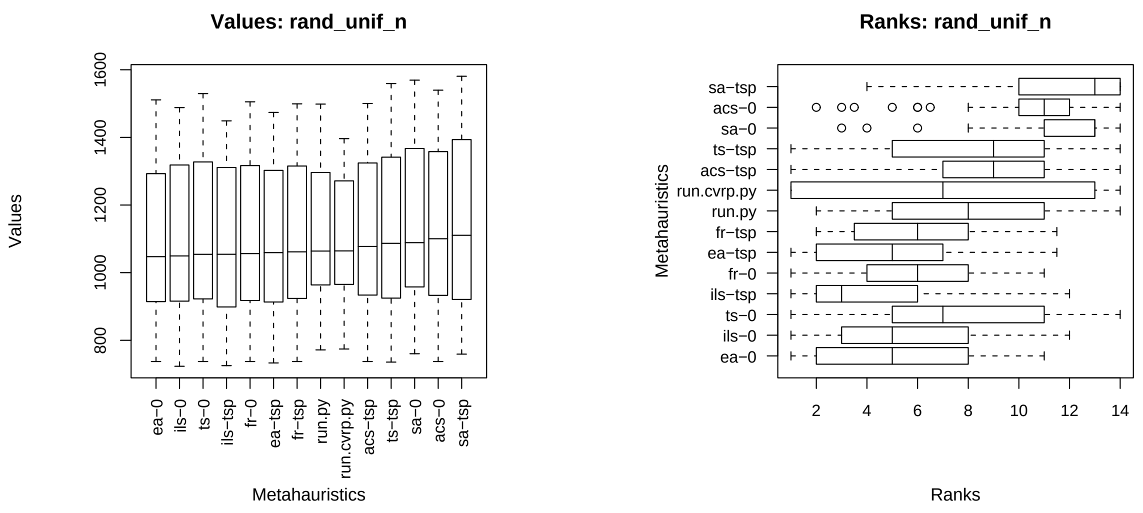
all classes of instances

	ils-tsp	ea-tsp	fr-0	fr-tsp	ils-0	ea-0	acs-tsp	ts-0
ea-tsp	2.7e-09	—	—	—	—	—	—	—
fr-0	<2e-16	0.00167	—	—	—	—	—	—
fr-tsp	1.5e-13	0.19816	1.7e-08	—	—	—	—	—
ils-0	2.1e-11	1	0.26262	1	—	—	—	—
ea-0	8.8e-10	1	1	1	1	—	—	—
acs-tsp	<2e-16	<2e-16	4.4e-13	<2e-16	1.7e-10	5.9e-10	—	—
ts-0	<2e-16	0.00016	0.19816	0.01271	0.01340	0.04461	0.05590	—
run.py	<2e-16	1.3e-14	7.5e-11	1.1e-13	5.9e-10	4.0e-11	1	0.00084
ts-tsp	<2e-16	<2e-16	1.1e-10	3.3e-13	1.2e-09	2.1e-11	1	0.00021
run.cvrp.py	4.0e-09	0.03678	0.76766	0.26560	0.04748	0.19523	1	1
sa-tsp	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	1.3e-14	<2e-16
acs-0	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	6.9e-09	2.6e-14
sa-0	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16

	run.py	ts-tsp	run.cvrp.py	sa-tsp	acs-0
ea-tsp	—	—	—	—	—
fr-0	—	—	—	—	—
fr-tsp	—	—	—	—	—
ils-0	—	—	—	—	—
ea-0	—	—	—	—	—
acs-tsp	—	—	—	—	—
ts-0	—	—	—	—	—
run.py	—	—	—	—	—
ts-tsp	1	—	—	—	—
run.cvrp.py	1	1	—	—	—
sa-tsp	1.1e-08	1.9e-09	0.00025	—	—
acs-0	0.00028	0.00010	0.02918	0.00068	—
sa-0	<2e-16	<2e-16	1.8e-08	0.19816	5.7e-12

P value adjustment method: holm

Figure 12: Results of the metaheuristics implemented on the all the 4 classes of instances.



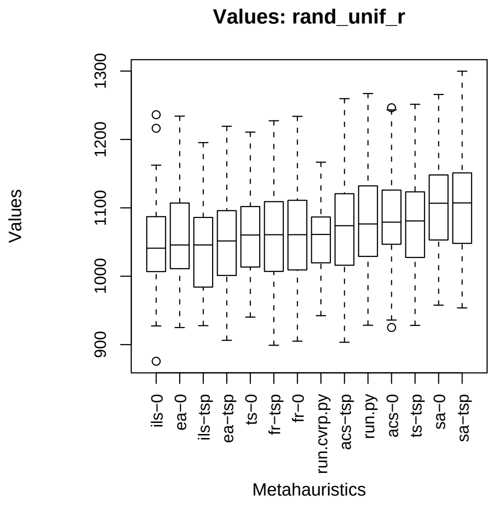
Pairwise comparisons using Wilcoxon rank sum test

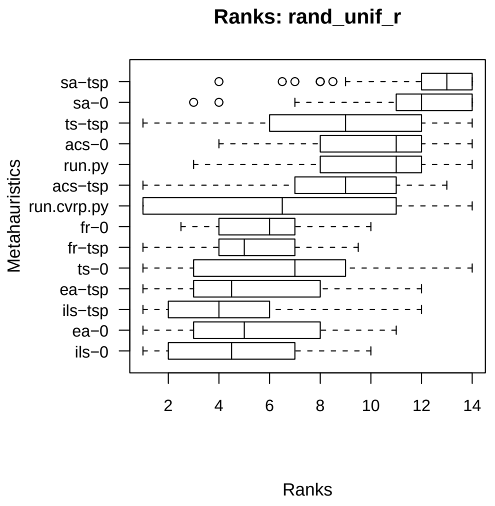
rand_unif_n

	ea-0	ils-0	ts-0	ils-tsp	fr-0	ea-tsp	fr-tsp	run.py	run.cvrp.py	acs-tsp	ts-tsp	sa-0	acs-0
ils-0	1	—	—	—	—	—	—	—	—	—	—	—	—
ts-0	0.162	0.182	—	—	—	—	—	—	—	—	—	—	—
ils-tsp	1	0.07338	0.00076	—	—	—	—	—	—	—	—	—	—
fr-0	1	1	0.87379	0.02448	—	—	—	—	—	—	—	—	—
ea-tsp	1	1	0.06882	0.87379	0.87379	—	—	—	—	—	—	—	—
fr-tsp	1	1	0.40609	0.08867	0.44083	1	—	—	—	—	—	—	—
run.py	0.02489	0.18461	1	3.8e-05	0.75495	0.04931	0.34458	—	—	—	—	—	—
run.cvrp.py	1	1	1	1	1	1	1	1	—	—	—	—	—
acs-tsp	0.00983	0.00211	1	3.8e-05	0.00398	0.00042	0.00172	1	1	—	—	—	—
ts-tsp	0.04393	0.18461	1	0.00015	0.13271	0.01582	0.04090	1	1	1	—	—	—
sa-0	9.9e-10	1.0e-11	0.00044	7.1e-11	7.1e-11	2.1e-10	2.6e-11	0.00073	0.00618	2.4e-06	0.00073	—	—
acs-0	6.7e-06	5.7e-06	0.44083	1.2e-08	3.4e-06	2.7e-06	3.2e-06	0.33779	0.34263	0.33893	0.11242	0.05522	—
sa-tsp	3.0e-09	3.4e-10	0.00263	1.6e-10	3.0e-09	2.5e-09	9.3e-09	0.00884	0.06171	0.00039	0.00308	1	0.16264

P value adjustment method: holm

Figure 13: Results of the metaheuristics implemented on the **rand_unif_n** class of instances.





Pairwise comparisons using Wilcoxon rank sum test

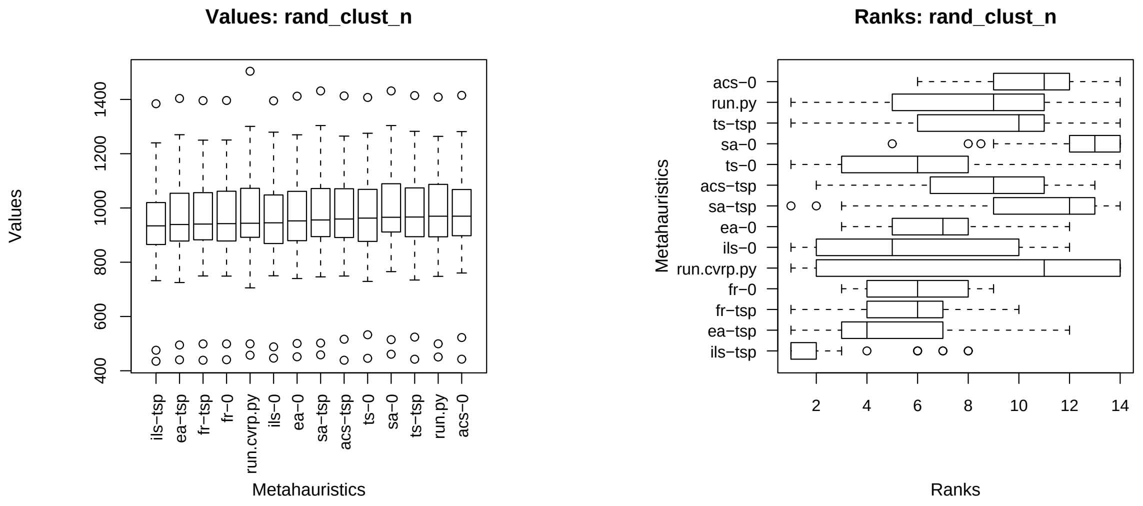
rand_unif_r

	ils-0	ea-0	ils-tsp	ea-tsp	ts-0	fr-tsp	fr-0	run.cvrp.py
ea-0	1	—	—	—	—	—	—	—
ils-tsp	1	1	—	—	—	—	—	—
ea-tsp	1	1	1	—	—	—	—	—
ts-0	0.20104	1	0.11924	1	—	—	—	—
fr-tsp	0.79205	1	1	1	1	—	—	—
fr-0	0.39858	1	0.49005	1	1	0.02629	—	—
run.cvrp.py	1	1	1	1	1	1	1	—
acs-tsp	0.00053	0.00085	0.00026	0.00027	0.33843	0.00033	0.00052	1
run.py	9.5e-06	3.1e-06	6.3e-06	5.4e-06	0.00496	1.9e-06	2.5e-06	0.26219
acs-0	2.2e-07	3.1e-06	4.6e-07	2.2e-06	4.4e-05	1.4e-07	1.4e-07	0.04121
ts-tsp	0.00043	0.00060	0.00023	0.00038	0.09810	0.00034	0.00180	0.47967
sa-0	7.1e-08	2.4e-07	7.1e-08	1.5e-07	2.0e-06	1.0e-07	1.1e-07	0.00043
sa-tsp	7.4e-08	1.3e-07	9.3e-08	2.5e-07	2.8e-07	9.7e-08	1.3e-07	0.00022

	acs-tsp	run.py	acs-0	ts-tsp	sa-0
ea-0	—	—	—	—	—
ils-tsp	—	—	—	—	—
ea-tsp	—	—	—	—	—
ts-0	—	—	—	—	—
fr-tsp	—	—	—	—	—
fr-0	—	—	—	—	—
run.cvrp.py	—	—	—	—	—
acs-tsp	—	—	—	—	—
run.py	1	—	—	—	—
acs-0	0.26219	1	—	—	—
ts-tsp	1	1	1	—	—
sa-0	3.0e-05	0.00217	0.02195	0.00060	—
sa-tsp	1.3e-05	0.00060	0.02304	0.00060	1

P value adjustment method: holm

Figure 14: Results of the metaheuristics implemented on the **rand_unif_r** class of instances.



Pairwise comparisons using Wilcoxon rank sum test

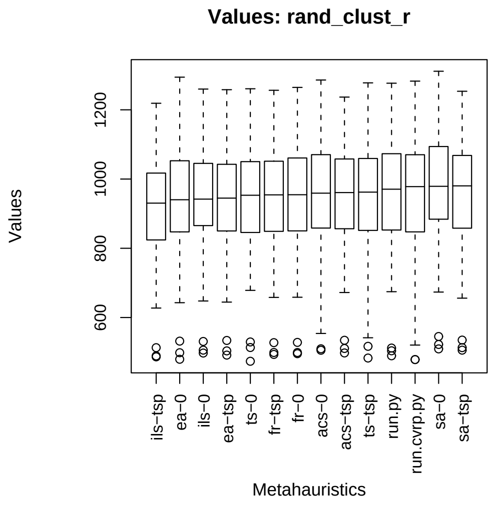
rand_clust_n

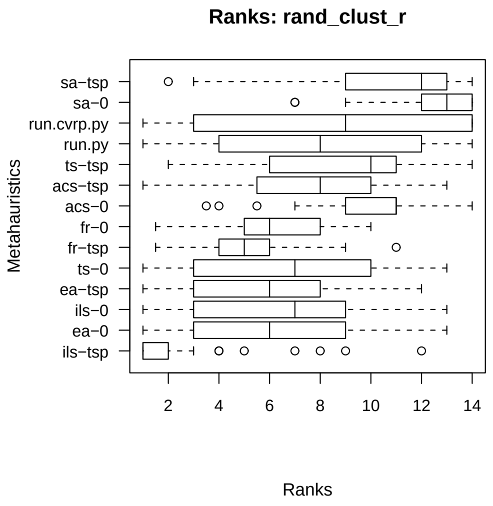
	ils-tsp	ea-tsp	fr-tsp	fr-0	run.cvrp.py	ils-0	ea-0	sa-tsp
ea-tsp	2.5e-05	—	—	—	—	—	—	—
fr-tsp	1.9e-07	1	—	—	—	—	—	—
fr-0	4.3e-08	1	1	—	—	—	—	—
run.cvrp.py	0.00087	0.80528	1	1	—	—	—	—
ils-0	7.2e-08	1	1	1	1	—	—	—
ea-0	1.2e-08	0.21192	1	1	1	1	—	—
sa-tsp	1.4e-11	1.3e-06	0.00010	1.1e-05	1	9.4e-06	0.00016	—
acs-tsp	2.1e-10	0.00033	0.01152	0.00110	1	0.11966	0.13698	0.05415
ts-0	3.8e-07	1	1	1	1	1	1	0.00272
sa-0	1.0e-11	1.0e-11	6.1e-07	6.1e-07	1	5.2e-12	2.1e-10	0.08529
ts-tsp	4.1e-10	9.4e-06	0.00272	0.00136	1	0.51167	0.02439	0.71096
run.py	7.7e-10	0.00110	0.00390	0.02161	1	1	0.43009	1
acs-0	1.0e-11	4.1e-10	5.2e-12	5.2e-12	1	9.6e-05	2.4e-06	1

	acs-tsp	ts-0	sa-0	ts-tsp	run.py
ea-tsp	—	—	—	—	—
fr-tsp	—	—	—	—	—
fr-0	—	—	—	—	—
run.cvrp.py	—	—	—	—	—
ils-0	—	—	—	—	—
ea-0	—	—	—	—	—
sa-tsp	—	—	—	—	—
acs-tsp	—	—	—	—	—
ts-0	0.62472	—	—	—	—
sa-0	4.1e-09	1.2e-08	—	—	—
ts-tsp	1	0.00468	2.6e-05	—	—
run.py	1	0.19174	4.9e-05	1	—
acs-0	0.00427	1.6e-05	0.00540	0.78117	0.15629

P value adjustment method: holm

Figure 15: Results of the metaheuristics implemented on the **rand_clust_n** class of instances.





Pairwise comparisons using Wilcoxon rank sum test

rand_clust_r

	ils-tsp	ea-0	ils-0	ea-tsp	ts-0	fr-tsp	fr-0	acs-0
ea-0	1.4e-07	—	—	—	—	—	—	—
ils-0	4.6e-08	1	—	—	—	—	—	—
ea-tsp	8.1e-07	1	1	—	—	—	—	—
ts-0	1.3e-07	1	1	0.43131	—	—	—	—
fr-tsp	8.6e-08	1	1	1	0.55048	—	—	—
fr-0	6.0e-09	1	1	0.28451	1	6.1e-05	—	—
acs-0	5.7e-12	3.4e-07	0.00016	1.2e-09	0.00026	1.6e-12	2.6e-07	—
acs-tsp	3.6e-10	0.58831	0.48573	0.02061	1	0.00044	0.31458	0.00113
ts-tsp	1.3e-10	0.01496	0.05269	0.00016	0.42894	0.00038	0.04800	0.58831
run.py	2.1e-09	0.58831	0.43031	0.01400	0.78490	0.02061	0.52354	0.43131
run.cvrp.py	1.7e-08	0.48573	0.28251	0.05269	0.49686	0.16554	0.48573	1
sa-0	3.2e-13	1.6e-11	3.2e-13	1.6e-12	1.7e-08	2.1e-12	3.0e-12	1.1e-05
sa-tsp	6.3e-13	0.00018	1.8e-05	2.0e-07	0.00250	9.4e-08	3.6e-06	1

	acs-tsp	ts-tsp	run.py	run.cvrp.py	sa-0
ea-0	—	—	—	—	—
ils-0	—	—	—	—	—
ea-tsp	—	—	—	—	—
ts-0	—	—	—	—	—
fr-tsp	—	—	—	—	—
fr-0	—	—	—	—	—
acs-0	—	—	—	—	—
acs-tsp	—	—	—	—	—
ts-tsp	1	—	—	—	—
run.py	1	1	—	—	—
run.cvrp.py	1	1	1	—	—
sa-0	1.6e-11	1.8e-07	4.4e-07	1	—
sa-tsp	0.00836	0.57755	0.68344	1	0.00398

P value adjustment method: holm

Figure 16: Results of the metaheuristics implemented on the **rand_clust_r** class of instances.

Acknowledgments. We thanks Fabrizio Oliverio (AntOptima) and Tommasio Schiavinotto (Intellektik, Technische Universität Darmstadt) for their contributions to the Second Young Researches' Network Meeting. This work was supported by the *Metaheuristics Network,* a Research Training Network funded by the Improving Human Potential Programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. Laarhoven. Simulated annealing. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, New York, U.S.A., 1997.

[2] J.L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[3] D. J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, 1992.

[4] D. J. Bertsimas, P. Chervi, and M. Peterson. Computational approaches to stochastic vehicle routing problems. *Transportation Science*, 29(4):342–352, 1995.

[5] D. J. Bertsimas, P. Jaillet, and A. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990.

[6] D. J. Bertsimas and D. Simchi-Levi. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research*, 44(2):216–304, 1996.

[7] L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In *Proceedings of PPSN-VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892. Springer Verlag, Berlin, Germany, 2002.

[8] L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO metaheuristic. In *Proceedings of ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 176–187. Springer Verlag, Berlin, Germany, 2002.

[9] J. Bramel and D. Simchi-Levi. *The Logic of Logistics.* Springer, Berlin, Germany, 1997.

[10] W. J. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, NY, 1999.

[11] T. G. Crainic and G. Laporte. Planning models for freight transportation. *European Journal of Operational Research*, 97:409–438, 1997.

[12] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions On Evolutionary Computation*, 1(1):53–66, 1997.

[13] G. Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 1(104):86–92, 1993.

[14] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm superior to simulated annealing. *Journal of Computational Physics*, pages 161–175, 1990.

[15] M. Fisher. In O. M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, chapter Vehicle Routing, pages 1–33. Elsevier, Amsterdam, The Netherlands, 1995.

[16] M. Gendreau, G. Laporte, and R. Séguin. An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation Sciences*, 29(2):143–155, 1995.

[17] M. Gendreau, G. Laporte, and R. Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3), 1996.

[18] B. L. Golden and A. A. Assad, editors. *Vehicle Routing: Methods and Studies.* Elsevier, Amsterdam, The Netherlands, 1988.

[19] M. Haimovitch and A. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10:527–542, 1985.

[20] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[21] P. Jaillet. *Probabilistic Traveling Salesman Problems.* PhD thesis, MIT, Cambridge, MA, 1985.

[22] P. Jaillet. A priori solution of a travelling salesman problem in which a random subset of the customers are visited. *Operations Research*, 36(6):929–936, 1988.

[23] P. Jaillet and A. Odoni. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, chapter The probabilistic vehicle routing problems. Elsevier, Amsterdam, The Netherlands, 1988.

[24] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, November-December 1989.

[25] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, New York, U.S.A., 1997.

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, (4598):671–680, 1983.

[27] G. Laporte and F. Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 33:133–142, 1993.

[28] H.R. Lourenço, O. Martin, and T. Stützle. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, Boston, U.S.A., 2002.

[29] O. Martin, S.W. Otto, and E.W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[30] I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking.* PhD thesis, Department of Industrial Engineering and Management Sciences, Nortwestern University, Evanston, IL, 1976.

[31] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, pages 421–451, 1993.

[32] N. Secomandi. *Exact and heuristic dynamic programming algorithms for the vehicle routing problem with stochastic demands.* PhD thesis, University of Houston, Texas, 1998.

[33] N. Secomandi. A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, 49(5):796–802, 2001.

[34] T. Stützle and H. Hoos. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, chapter Analyzing the Run-time Behaviour of Iterated Local Search for the TSP, pages 589–612. Kluwer Academic Publishers, Boston, U.S.A., 2002.

[35] D. Teodorović and G. Pavković. A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand. *Transportation Planning and Technology*, 16:261–273, 1992.

[36] P. Toth and D. Vigo. *The Vehicle Routing Problem.* SIAM Monographs on Discrete Mathematics, 2002.

[37] V. Černý. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal Of Optimization Theory And Applications*, pages 41–51, 1985.

[38] W. Yang, K. Mathur, and R. H. Ballou. Stochastic vehicle routing problem with restocking. *Transportation Science*, 34(1):99–112, 2000.