Stochastic local search with learning automaton for the swap-body vehicle routing problem

Túlio A. M. Toffolo[a,b,*], Jan Christiaens[a], Sam Van Malderen[a], Tony Wauters[a], Greet Vanden Berghe[a]

[a] *KU Leuven, Department of Computer Science, CODeS & imec-ITEC - Belgium*
[b] *Federal University of Ouro Preto, Department of Computing - Brazil*

Abstract

This work presents the stochastic local search method for the Swap-Body Vehicle Routing Problem (SB-VRP) that won the First VeRoLog Solver Challenge. The SB-VRP, proposed on the occasion of the challenge, is a generalization of the classical Vehicle Routing Problem (VRP) in which customers are served by vehicles whose sizes may be enlarged via the addition of a swap body (trailer). The inclusion of a swap body doubles vehicle capacity while also increasing its operational cost. However, not all customers may be served by vehicles consisting of two bodies. Therefore swap locations are present where one of the bodies may be temporarily parked, enabling double body vehicles to serve customers requiring a single body. Both total travel time and distance incur costs that should be minimized, while the number of customers visited by a single vehicle is limited both by its capacity and by a maximum travel time. State of the art VRP approaches do not accommodate SB-VRP generalizations well. Thus, dedicated approaches taking advantage of the swap body characteristic are desired. The present paper proposes a stochastic local search algorithm with both general and dedicated heuristic components, a subproblem optimization scheme and a learning automaton. The algorithm improves the best known solution for the majority of the instances proposed during the challenge. Results are also presented for a new set of instances with the aim of stimulating further research concerning the SB-VRP.

Keywords: Swap-Body Vehicle Routing Problem, VeRoLog challenge, metaheuristics, local search, decomposition strategies, learning automaton, neighborhood size reduction

1. Introduction

The Swap-Body Vehicle Routing Problem (SB-VRP) was proposed by the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog) and the PTV Group at the First VeRoLog Solver Challenge (Heid et al., 2014). It is a generalization of the classical Vehicle Routing Problem (VRP) based on real problems faced by industry.

*Corresponding author. *E-mail*: tulio.toffolo@kuleuven.be.

The classical VRP is one of the most studied problems in combinatorial optimization and is defined under capacity and route length constraints (Cordeau et al., 2007). The SB-VRP primarily differs from the VRP insofar as vehicles consist of either one or two bodies (trailers). The lengthened vehicles are called trains and have exactly twice the capacity of the regular vehicles (trucks). Figure 1 shows an example of a truck, a swap body (with a trailer) and a train, respectively.

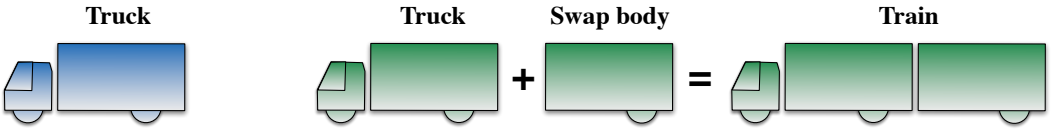


Figure 1: Vehicle type examples

Customers have individual demands and must be served by exactly one vehicle. Three types of customers are considered: those who can only be reached by trucks, those who can be served by both trains and trucks, and those whose demands exceed the capacity of a truck and must be attended to by trains. Customers are geographically dispersed. Travel times and distances between all locations are given.

In addition to the depot and customers' locations, swap locations are present, where one of the bodies of a train may be temporarily left, enabling the vehicle to serve customers with a single body (truck).

The SB-VRP considers both total time and distance to derive costs that should be minimized. These costs vary depending on whether the considered vehicle is a train or truck. Furthermore, additional costs for operations at swap locations are also considered. Vehicles routes are limited by both their capacity and a maximum travel duration.

The present paper proposes a stochastic local search heuristic approach to the problem. Initially, a naive solution is quickly built. Different intensification and diversification strategies are subsequently applied to improve the solution. These strategies include a subproblem optimization scheme and different neighborhood structures, both of which are embedded in a metaheuristic framework. A preprocessing procedure reduces the solution space and thus dramatically increases the heuristic's efficiency. The stochastic local search won the First VeRoLog Solver Challenge and continues to outperform all other proposed approaches for the problem.

The approach has significant practical relevance for a range of business activities including production, distribution and also the transportation sector more generally. The delivery of both perishable and urgently-required goods (fuel, for example), which almost always necessitates transportation by road, becomes greatly optimized. Indeed, very often the transportation costs associated with such products are disproportionate when compared against the cost of the products themselves. Furthermore, the approach ensures efficiency with regard to a number of important economic and ecological factors such as: the number of vehicles, number of drivers, travel distance and time, and the environmental impact.

The present work is organized as follows. Section 2 details the problem. Section 3 presents a literature overview about the SB-VRP and related work. The proposed algorithm

is introduced and described within Section 4. The neighborhood structures considered for the local search are discussed in Section 5. Section 6 presents computational experiments and, finally, Section 7 summarizes the conclusions and indicates future research directions.

2. Problem description

The SB-VRP is a generalization of the classical VRP and, by consequence, is an NP-Hard problem. It can be defined on a graph $G = (V, A)$, where the vertices V are the locations and the arcs A are the connections between these locations. Three vertex categories are considered: depot, customers and swap locations. A single depot vertex is defined.

The customers, represented by the subset $C \subset V$, are divided into three groups: truck-only ($C_1 \subseteq C$), flexible ($C_2 \subseteq C$) and train-only ($C_3 \subseteq C$). These groups are defined according to the types of vehicle that can be employed to visit the customers. Truck-only customers can only be attended to by trucks, flexible customers can have their demands satisfied by both trucks and trains, and train-only customers require trains.

All customers $i \in C$ have an associated demand q_i and service time s_i. These demands must be satisfied with exactly one visit. Since the capacity of a swap body is given by constant Q, truck-only and flexible customers' demands must be bounded by Q, such that $q_i \leq Q \ \forall i \in C_1 \cup C_2$. Contrastingly, train-only customers have demands that trucks cannot satisfy, therefore implying $Q < q_i \leq 2Q \ \forall i \in C_3$.

Swap locations, represented by the subset $S \subset V$, are associated with neither demand nor service time. Nevertheless, depending on the operation executed at a swap location, a certain amount of time is consumed. In total, four operations are possible at a swap location, each consuming varying amounts of time:

park : leaves the back swap body of the train at the swap location;

exchange : leaves the front swap body of the train at the swap location;

pickup : picks up the swap body that was left at a swap location;

swap : leaves the currently attached swap body and picks up the swap body that was left at the swap location.

As Miranda-Bront et al. (2017) have highlighted, the consumed capacity of the bodies may be distributed across routes such that the first action on a swap location is always *park*. *Exchange* generally requires more time than *park*, and thus employing *park* rather than *exchange* results in less time spent in a swap location.

Each arc $(i, j) \in A$ connects location i to location j, has a distance d_{ij} and a travel time t_{ij}. Note that the distances and travel times are asymmetric, meaning d_{ij} and t_{ij} are not guaranteed to be equal to d_{ji} and t_{ji} respectively.

Figure 2 shows a graph representation of a small SB-VRP instance. Triangles represent swap locations, squares indicate truck-only customers, filled circles denote flexible customers and, finally, open circles identify train-only customers.

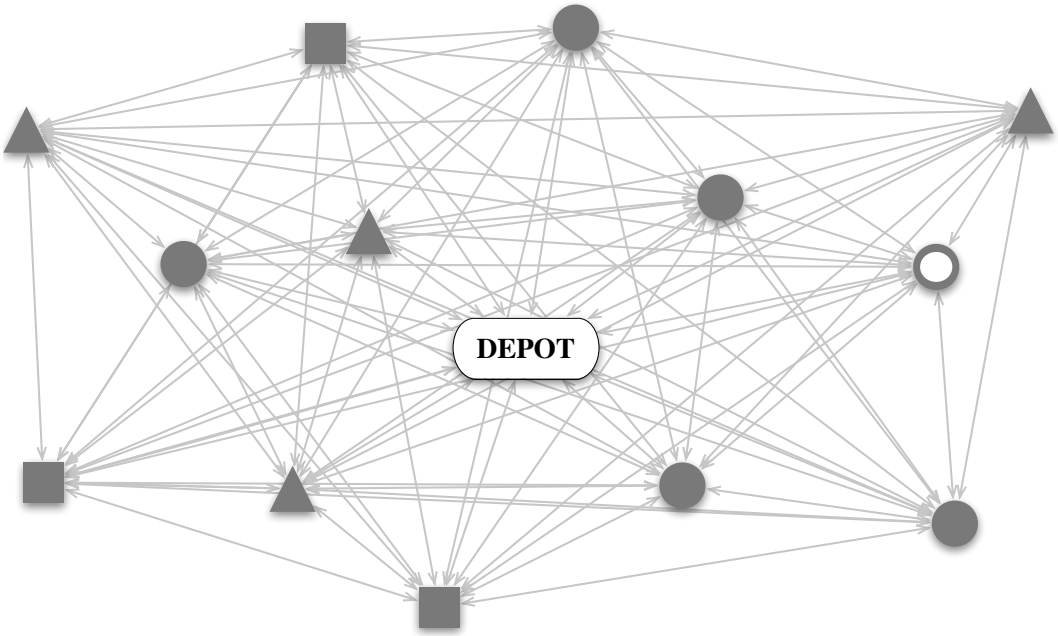


Figure 2: Graph representation of a small SB-VRP instance

Vehicles must leave and return to the depot with the same swap bodies. Crucially, routes must begin and end in the depot and swap bodies may not be exchanged between vehicles. Therefore, if a vehicle leaves a swap body in a swap location, the body must be retrieved later by the same vehicle. Henceforth, the part of the route that comprises of the customers between the two swap location visits will be referred to as a sub-route.

All routes must respect capacity constraints and a maximum route duration T. Each route's duration is given by the sum of its travel times, service times and swap operation times.

In the SB-VRP considered the objective is to minimize the total operation cost, given by the sum of two components:

vehicle/driver costs : consisting of a fixed cost for using a vehicle, a cost per kilometer traveled and a cost per hour (driver's cost);

swap body costs : consisting of a fixed cost per additional swap body and a cost per kilometer traveled with it.

A sample solution for the problem depicted by Figure 2 is shown in Figure 3. This example employs three vehicles: one truck and two trains. Note that one of the routes (route 3) contains a sub-route, therefore indicating it utilizes a swap location. The swap location temporarily stores one of the vehicle's swap bodies, while it visits two truck-only customers. After visiting these two customers (or directly before finishing the sub-route), the vehicle reattaches the parked body and continues towards the next customers.

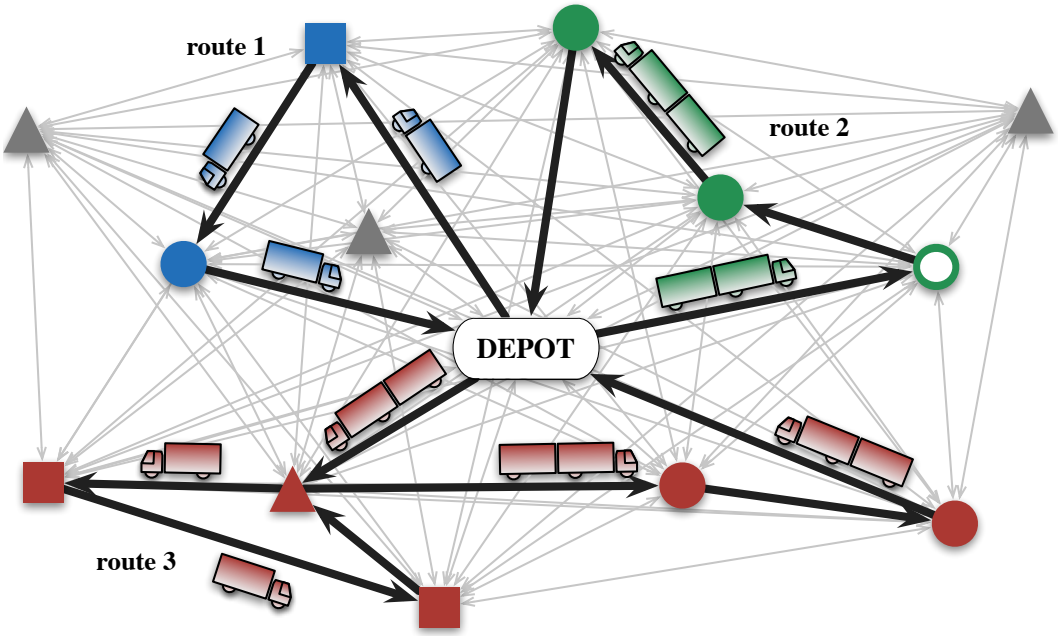


Figure 3: Example of a SB-VRP solution

3. Literature review

The SB-VRP considered by this work was introduced recently in the literature. Huber and Geiger (2014) addressed the SB-VRP with an iterative variable neighborhood search (VNS) procedure. They employed a cluster-first route-second approach to produce initial solutions. Both sequential and parallel versions of the algorithm were evaluated. Lum et al. (2015) applied a VRP-Reduce algorithm to the SB-VRP. They first transform the SB-VRP into a classical VRP and then solve the classical problem employing a simulated annealing (SA) algorithm. Afterwards, a post-processing procedure produces solutions to the SB-VRP, which are subsequently improved with a variable neighborhood descent (VND) method. Miranda-Bront et al. (2017) combined a cluster-first route-second approach with a greedy randomized adaptive search procedure (GRASP). Different constructive heuristics were studied. Absi et al. (2015) proposed a relax-and-repair approach to the SB-VRP in which the problem is initially solved as a heterogeneous-fleet VRP with a memetic algorithm. Next, the results are repaired to produce valid solutions for the SB-VRP. During the optimization process all solutions are stored to be later used as input to a set-partitioning problem aiming at deriving better solutions. Finally, Todosijević et al. (2016) proposed a method that resembles the one proposed by Huber and Geiger (2014). A cluster-first, route-second constructive heuristic for generating an initial solution, and two general variable neighborhood search (GVNS) heuristics. Both sequential and parallel versions were evaluated. Todosijević et al. (2016) also introduced a mixed integer programming formulation for the SB-VRP. Recently, Huber and Geiger (2017) presented a study on the importance of the neighborhoods and their ordering within the VNS algorithm proposed by Huber and Geiger (2014). It was shown that the sequence of neighborhoods matters. In addition, the impact

of the synchronization frequency during the parallel execution on the solution quality was also evaluated. Improved results were reported.

Table 1 summarizes the characteristics of the published SB-VRP methods. Each column represents a different approach: *HG2014/2017* (Huber and Geiger, 2014, 2017), *Lum2015* (Lum et al., 2015), *THUJG2016* (Todosijević et al., 2016), *MB2017* (Miranda-Bront et al., 2017) and *Absi2016* (Absi et al., 2015). Note how all studies addressed the SB-VRP with heuristic-based methods. Moreover, although some classic neighborhoods were adapted to consider SB-VRP characteristics, the limited number of neighborhoods exploiting problem-specific features is noteworthy.

Table 1: Overview of previous SB-VRP strategies in the literature

Feature / characteristics	Challenge participants				Others
	HG2014/2017	*Lum2015*	*THUJG2016*	*MB2017*	*Absi2016*
Construction heuristic	Cluster-first route-second	SA (VRP)	Cluster-first route-second	Cluster-first route-second	Heterogeneous fleet VRP
Metaheuristic strategy	VNS	SA, VND	GVNS	GRASP	Multiple strategies
Population based	-	-	-	-	✓
Parallel computing	✓	-	✓	✓	✓
Intra-route neighborhoods					
relocate	✓	-	✓	-	✓
swap	-	-	-	-	✓
2-opt	✓	✓	✓	✓	✓
3-opt	✓	-	-	-	-
Inter-route neighborhoods					
relocate	✓	-	✓	✓	✓
swap	✓	✓	✓	✓	✓
multiple swap	-	✓	✓	-	-
3-exchange	✓	-	-	-	-
or-opt	-	✓	-	✓	-
Problem-specific neighborhoods					
Change swap location	✓	-	-	-	-
Truck-only customer migration	-	✓	-	-	-
Route downgrade (single truck)	-	-	-	✓	-
Repair procedure(s)	-	-	-	-	✓

Other problems bearing many similarities with the SB-VRP have been studied by various authors. Gerdessen (1996), for instance, presented a study on the Vehicle Routing Problem with Trailers (VRPT). Like the SB-VRP, the VRPT considers two vehicle configurations: trucks and trucks with an attached trailer (trains). Although all customers may be served by trains in the VRPT, it proves inconvenient to visit some customers with a large vehicle, such as those located in the city center. The degree of inconvenience is measured by what Gerdessen calls *manoeuvring time*, which consists of the additional time required by trains, as opposed to trucks, when serving a specific customer. Trailers can be parked at any customer site with two additional simplifications considered: firstly, each trailer is parked exactly once and, secondly, all customers have unit demands. Gerdessen proposed three constructive heuristic algorithms and an improvement heuristic based on local search.

Another similar problem is the Truck and Trailer Routing Problem (TTRP), introduced

by Chao (2002). The TTRP is a real-world extension of the VRP in which a limited fleet of trucks and trailers with fixed capacities serve a set of customers from a central depot. Note that, in contrast to the SB-VRP, obtaining a feasible solution for the TTRP is not trivial, on account of the limited number of vehicles. The objective is to satisfy customers' demands while minimizing the total travel distance. Similarly to the SB-VRP, customers are divided in groups: (i) vehicle customers, reachable by either a complete vehicle (truck and trailer) or by a truck alone, and (ii) truck customers, reachable only by trucks alone. As with the SB-VRP and the VRPT, trailers may be temporarily parked, enabling truck customers to be served exclusively by trucks. Any customer site may serve as a parking place for trailers. Chao (2002) proposed a three-step constructive heuristic and a Tabu Search (TS) method for the TTRP. The algorithm obtained feasible solutions for all instances, despite the high demand-to-capacity ratio (above 90%).

Several papers elaborate upon the TTRP literature. Scheuerer (2006) proposed two constructive algorithms and a TS method, improving the best known results for all instances. Later, Lin et al. (2009) proposed a Simulated Annealing algorithm with a two-level solution representation which employed dummy depots/roots, in conjunction with random neighborhood structures employing three different types of moves. Lin et al. (2009) further improved the best known results for several instances. Caramia and Guerriero (2009) proposed a very interesting hybrid approach based on local search and mathematical programming. They heuristically decompose the problem and use CPLEX to solve two subproblems sequentially. First, each customer is assigned to a route with the objective of minimizing the fleet size. Next, by considering the customers assigned to each vehicle, the individual routes are optimized by minimizing their total tour length. The hybrid algorithm adds constraints to the formulation during each iteration while using a tabu-like customer-route matrix to avoid previously analyzed allocations. Some new best results were produced, in addition to lower bounds for assessing solution quality. Villegas et al. (2013) present a two-phase matheuristic approach to the TTRP which employs locally optimal routes as columns in a set-partitioning formulation. Routes are generated with a metaheuristic consisting of a hybrid GRASP and ILS. Route pool sizes may can be controlled, offering a trade-off between solution quality and running time. Very competitive results are obtained compared to previous methods.

Other papers focus on variants of TTRP such as the relaxed TTRP with unlimited availability of trucks and trailers and the TTRP with time windows (TTRPTW), introduced by Lin et al. (2010) and Lin et al. (2011) respectively. Recently, the TTRPTW was further studied by Parragh and Cordeau (2017), who proposed a tailor-made branch-and-price algorithm capable of optimally solving instances of up to 100 customers.

Much like with the SB-VRP, the majority of the literature on the VRPT and the TTRP address the problem with local search based algorithms. In fact, the majority of the best results for these problems were obtained by metaheuristics relying on local search. This observation, coupled with the NP-hardness of the SB-VRP, further motivated the development of local search algorithms when approaching the problem.

4. Local Search algorithm

A stochastic local search based algorithm is proposed for the SB-VRP. The algorithm begins by building a naive feasible solution. Once an initial solution is obtained, the algorithm proceeds to the local search phase that considers several neighborhood structures. A learning algorithm is responsible for choosing the neighborhood to apply at each iteration. To enable escaping from local optima, a hybridization of the metaheuristics Iterated Local Search (ILS) (Lourenço et al., 2003) and Late Acceptance Hill-Climbing (LAHC) (Burke and Bykov, 2017) is considered.

The algorithm's components are explained throughout the following sections. Section 4.1 details the constructive algorithm while Section 4.2 introduces the hybrid algorithm combining ILS and LAHC. The neighborhood structures employed and the learning mechanism are discussed later in Section 5.

4.1. Constructive algorithm

A simple and straightforward constructive algorithm is considered for quickly producing a feasible solution. The algorithm generates separate routes from the depot to each customer. Both truck-only and flexible customers are initially served by trucks, whereas train-only customers are served by trains.

The produced solution is expected to be very poor in terms of cost. However, it can be generated quickly in $O(|C|)$, where $|C|$ is the number of customers. Additionally, providing good feasible solutions to threshold acceptance algorithms, such as the LAHC, may limit the search space and result in poor final solutions, especially if the initial solution happens to be a local optimum. It is possible to avoid this drawback by feeding the LAHC with a cost larger than the initial solution's. This may, however, result in several worsening modifications at the execution's beginning, eventually wasting the additional effort required to produce a good initial solution.

4.2. Hybrid local search algorithm

The proposed algorithm is a hybridization of the Iterated Local Search (ILS) and the Late Acceptance Hill-Climbing (LAHC) procedures. The ILS metaheuristic was introduced by Lourenço et al. (2003) and relies upon perturbations to escape from local optima. The main principle behind the hybridization is to apply these ILS-like perturbations to the best solution obtained by the LAHC, before re-executing LAHC on the perturbed solution.

The LAHC is a metaheuristic introduced by Burke and Bykov (2008) and further discussed by Burke and Bykov (2017). It is an adaptation of the classic Hill-Climbing heuristic in which the quality of a "late" solution, obtained l iterations before the current, determines whether a new solution is accepted or rejected. This permits the algorithm to accept worsening solutions, enabling it to eventually escape from local optima. Successful applications of the LAHC have been reported by Özcan et al. (2009), Verstichel and Vanden Berghe (2009), Abuhamdah (2010), Goerler et al. (2013) and Yuan et al. (2015), among others.

The LAHC approach to the SB-VRP is presented in Algorithm 1. Three arguments are required: (i) an initial solution S, (ii) the list size l and (iii) the maximum number of

consecutively rejected neighbors m. The algorithm begins by filling the late acceptance list, F, with the initial solution cost (lines 1-2). Following this, the best solution is stored and counters p and r are initialized (lines 3-4), where p is a cyclic pointer to a position in the late acceptance list and r the current number of consecutive rejections. The main loop (line 5) begins by selecting a neighborhood (line 6). The learning algorithm is responsible for this selection. Afterwards, a new neighboring solution is generated (line 7). If this solution is at least as good as the previous or has a lower objective value than the considered entry in the late acceptance list, it is accepted (line 8-9). Note that counter r is reset only if S' is an improving solution over S (lines 10-11). If the best solution S^* is improved, it is updated (lines 12-13). Next, the late acceptance list is updated (line 14) as well as counters p and r (lines 15-16). Finally, the best solution obtained is returned once the main loop finishes (line 17).

Algorithm 1: Late Acceptance Hill-Climbing

Input: Initial solution S, list size l and maximum consecutive rejections m
LAHC(S, l, m)

```
1  foreach p ∈ {0, ..., l − 1} do
2      F_p ← f(S)
3  S* ← S
4  p ← r ← 0
5  while r ≤ m and time limit is not reached do
6      N ← selected neighborhood structure
7      S' ← random neighbor N(S)
8      if f(S') ≤ f(S) or f(S') ≤ F_p then
9          S ← S'
10         if f(S') < f(S) then
11             r ← 0
12         if f(S) < f(S*) then
13             S* ← S
14     F_p ← f(S)
15     p ← (p + 1) mod l
16     r ← r + 1
17 return S*
```

The hybrid algorithm combining ILS and LAHC is presented in Algorithm 2. It begins by producing the initial solution (line 1) and setting the perturbation level ρ to 1 (line 2). The main loop (line 3) first calls the LAHC algorithm (line 4). If the solution produced by LAHC is better than the current best solution, the best solution is updated and the perturbation level is reset to 1 (lines 5-7). Otherwise, the current solution is reset to the best solution (lines 8-9). Afterwards, the perturbation is executed (lines 10-12), which corresponds to applying ρ random moves to the current solution. Next, the perturbation level ρ is increased (line 13) and the loop repeated. Note that the given maximum perturbation level ρ^+ is never exceeded: ρ is reset to 1 after ρ^+ is reached. Once the time limit is reached, the best solution produced is returned (line 14).

Algorithm 2: Hybrid Algorithm (ILS and LAHC)

Input: LAHC list size l, maximum consecutive rejections m and maximum perturbation level ρ^+
SBVRP_Solver(l, m, ρ^+)

```
S* ← S ← initial naive solution
ρ ← 1
while time limit is not reached do
    S ← LAHC(S, l, m)
    if f(S) < f(S*) then
        S* ← S
        ρ ← 1
    else
        S ← S*
    for i = 0 to ρ do
        N ← selected neighborhood structure
        S ← random neighbor N(S)
    ρ ← (ρ mod ρ+) + 1
return S*
```

5. Neighborhood structures

Several neighborhoods were developed to explore the search space of the SB-VRP, being categorized within three groups: (*i*) neighborhood structures based on classical VRP moves, (*ii*) neighborhood structures based on SB-VRP specific moves and, finally, (*iii*) neighborhood structures based on subproblem optimization. On the one hand, the first two groups generate neighbors by applying move(s) to the solution. These neighborhoods are sampled randomly rather than being fully explored, as detailed within Section 4. The neighborhoods based on subproblem optimization, on the other hand, generate neighbors by solving a subproblem. In this case, randomization occurs when generating the subproblem.

All different neighborhoods are considered together as possible approaches in generating new (neighboring) solutions. The probability of selecting each neighborhood in a given iteration is determined by a learning automaton.

Section 5.1 begins by introducing the neighborhood size reduction procedure, responsible for pruning away potentially less attractive neighbors. Next, the different neighborhood structures are explained. Classical VRP neighborhoods are discussed in Section 5.2, those based on SB-VRP specific moves are presented in Section 5.3 and those based on subproblem optimization are described in Section 5.4. Finally, Section 5.5 details the learning automaton.

5.1. Neighborhood size reduction

A neighborhood size reduction procedure is applied to limit the number of solution neighbors. This procedure avoids considering solutions in which certain pairs of customers are visited consecutively in one route. The primary motivation behind this approach is that two geographically distant customers tend not to be visited one after the other, and thus analyzing them as consecutive customers in a route likely results in wasted time.

The procedure requires a preprocessing step. First, a list of possible preceding and succeeding customers in the same route is computed for each customer. Both time and capacity constraints are considered. For instance, two train-only customers can never be in the same route since the sum of their capacities exceeds the capacity of a train. Once the list is built, it is sorted according to the cost of traveling either from or to each other customer, whichever is smaller. This cost is computed as the sum of time and distance costs.

Once the ordered lists are built for each customer, neighborhood reduction may be applied. The procedure limits possible customer allocations to the first η in each customer list (ignoring the other possibilities). Note that $\eta \in \mathbb{Z}^+$ with $\eta \leq |C|$. When $\eta = |C|$, no neighborhood reduction is applied. At the other extreme, when $\eta = 1$ the number of neighboring solutions is drastically limited. Therefore, the value of η determines the possible achievable neighborhood reduction.

During ILS perturbations, η is always set to $|C|$. This disables neighborhood reduction and may lead to diverse solutions, which is the primary objective of the perturbation. In the other contexts, each neighborhood may have a different value for η. Depending on its value and on the instance characteristics, the procedure may cut the optimal solution away from the search space. Hence, setting an appropriate value for η is crucial for the proposed algorithm's efficiency.

A neighborhood is defined by both its structure and the value of η. Therefore, one neighborhood structure is considered in multiple neighborhoods, each with a different value for the neighborhood reduction parameter η. The principle behind this is that the learning algorithm should select the most appropriate neighborhoods, giving very low probabilities for inefficient selections.

5.2. Classical neighborhood structures

Neighborhood structures based on four classical VRP moves are considered. Whenever possible, neighborhood reduction (Section 5.1) is applied. The moves are as follows:

Swap move

The *swap move* consists of simply swapping two customers in the solution. The customers may belong to the same or to different routes.

2-opt move

This move is the classical 2-opt move proposed by Croes (1958). It is important to note, however, that when dealing with the SB-VRP both routes and sub-routes are considered, with sub-routes being treated as independent routes.

Random insert move

This move consists of removing one or more customers and re-inserting them into random routes and positions.

Ejection-chain move

This move is a standard ejection-chain (Glover, 1991, 1996), and consists of re-arranging a chain of consecutive customers in a route. The *size* of the chain is given as a parameter.

5.3. Problem-specific neighborhood structures

Five neighborhood structures were developed to explore specific characteristics of the SB-VRP. Again, neighborhood reduction (Section 5.1) is applied whenever possible. The neighborhood structures are based on the following five moves:

Change swap location move

Given a route that includes a swap location, this move essentially changes the vertex (location) of one of the route's swap locations. Note that moving to certain swap locations may render the solution infeasible due to the limit imposed on route durations. To circumvent this issue, only swap locations close enough (so as to maintain feasibility) are considered. The new vertex is randomly selected from one of these locations.

Convert to route move

This move consists of converting a sub-route into a new (truck) route. The improvement probability of this move is limited, nevertheless it proves useful as a diversification tool.

Add sub-route move

The *add sub-route move* consists of adding a swap location to a train route, resulting in a new sub-route inside the route. Two steps must be executed: (i) defining the vertex (location) of the swap location and (ii) defining the positions in which the swap location will be added. This move requires all customers within the sub-route to be truck-only or flexible customers. Two operations are performed at the swap location: (i) park and then (ii) pickup. Capacity and time constraints must also be checked.

Split to sub-routes move

This move is similar to the *add sub-route* move, but instead of adding one sub-route, it adds two sub-routes at the same swap location, such that each sub-route is conducted with a different swap body. Three operations are defined at the swap location: (i) park, (ii) swap, and finally (iii) pickup.

Merge routes move

This move consists of merging two truck routes into a train route. Initially, two truck routes are selected. Then the swap location closest to both routes is selected and a new route including this swap location created. Afterwards, the two truck routes are added to the route as sub-routes. As with the split to sub-routes move, three operations are performed at the swap location: (i) park, (ii) swap, and finally (iii) pickup.

5.4. Subproblem optimization scheme

Additional neighborhood structures based on a subproblem optimization scheme have been developed. The subproblems require the reallocation of a subset $P \in C$ of the customers while minimizing the solution cost. In other words, subproblems are SB-VRP problems in which a subset $\bar{P}$ of customers, $\bar{P} = C \backslash P$, have their routes predefined.

Two questions arise from this approach. Firstly, how many customers may be added to P without negatively impacting on the subproblem computation time? Secondly, which customers should be added to P? Both questions are addressed in this section.

The initial idea of the subproblem optimization was to solve the subproblems to optimality by an Integer Linear Programming (ILP) approach. However, only very small subproblems (small P) appear solvable in short runtimes by currently available ILP solvers such as CPLEX and Gurobi. Experiments revealed that heuristic (sub-optimal) solutions for larger subproblems yielded much better final results than optimal solutions for very small problems. The difference is more profound when the runtime limit imposed by the VeRoLog challenge of 600 seconds is considered. Hence, the approach proposed by the present paper does not guarantee optimality of subproblem solutions. They are produced by a straightforward best-fit constructive heuristic, henceforth called *Cheapest Inserter* procedure.

The *Cheapest Inserter* procedure operates as follows. First, P is defined as an ordered set and its customers are removed from the solution. Next, they are sequentially inserted following P's order into the routes and positions incurring the lowest cost increase. This procedure executes $O(|P| \times |\bar{P}|)$ operations in the worst case. When applying the neighborhood reduction presented in Section 5.1, only the η closest customers are considered by the *Cheapest Inserter*. This speeds up the procedure by considerably reducing the number of comparisons. The size of P therefore does not prevent the algorithm from quickly producing a solution to the subproblem.

Six strategies have been developed for customer selection in a subproblem. Each strategy requires the maximum number of customers ($size$) as a parameter in addition to the value for the neighborhood size reduction η, applied within the *Cheapest Inserter*. Note that all strategies have a certain degree of randomization.

Ruin and recreate strategy
This strategy randomly selects $size$ customers and adds them to P.

Remove chains strategy
In this strategy, small chains of consecutive customers are added to P. The total number of customers is bounded by the parameter $size$, while the number of chains is bounded by the parameter k. The customers may be added to P in three possible orders: (i) random, (ii) original, or (iii) inverse original route. One of the three sorting criteria is randomly selected.

Remove route strategy
All customers from one or more randomly selected routes are added to P. Similar to the previous strategy, customers may be disposed in three possible orders: (i) random, (ii) route, and (iii) inverse route.

Remove sub-routes strategy
This strategy is similar to the *remove routes strategy* except that, instead of routes, only sub-routes are considered.

Limited ruin and recreate strategy

This strategy is similar to the *ruin and recreate strategy*, but only routes close to one another are considered when selecting customers. Note that the parameter η (Section 5.1) is employed when defining close routes, meaning that only routes containing geographically-nearby customers are selected. This strategy has two parameters: *size*, which indicates the number of customers to be added to P, and k, which indicates the maximum number of routes to be considered.

Ruin and randomized recreate strategy

This strategy is similar to the *ruin and recreate strategy* except for the differing behavior of the *Cheapest Inserter*. Rather than always selecting the cheapest insertion, the algorithm may select another insertion based on a probability given by a Heuristic-Biased Stochastic Sampling (HBSS) (Bresina and Bresina, 1996). Any customer may potentially be chosen, with the best ones having a much higher probability of selection. The chances of selecting the i-th cheaper insertion is given by $f(i) = e^{-i}$.

5.5. Learning automaton

A learning automaton (Narendra and Thathachar, 1989) provides a simple reinforcement learning approach commonly used in single state environments. It maintains and updates a probability distribution γ for all possible actions.

In the proposed algorithm, the learning automaton determines the probabilities of selecting each of the available neighborhoods. Therefore, for each neighborhood k a probability γ_{ke} is assigned to each event (or iteration) e. The value of γ_{ke} is updated according to the feedback provided by the local search algorithm from the previous event, $e-1$. Generally, the probabilities are updated according to Equations (1) and (2). Equation (1) is applied to update the probability of selecting the current neighborhood while Equation (2) is applied to update the probabilities of the remaining neighborhoods.

$$\gamma_{k,e+1} = \gamma_{ke} + \alpha^{+}\beta_e(1-\gamma_{ke}) - \alpha^{-}(1-\beta_e)\gamma_{ke} \tag{1}$$

$$\gamma_{k,e+1} = \gamma_{ke} - \alpha^{+}\beta_e\gamma_{ke} + \alpha^{-}(1-\beta_e)\left(\frac{1}{|M|-1} - \gamma_{ke}\right) \tag{2}$$

In Equations (1) and (2), constants $\alpha^{+} \in [0,1]$ and $\alpha^{-} \in [0,1]$ are the reward and penalty parameters, respectively. $\beta_e \in [0,1]$ indicates the improvement gained by event e and, finally, $|M|$ is the number of available neighborhoods. The objective of this update scheme is to increase the probability of a neighborhood in case of success and to decrease it in case of failure. A linear reward-penalty (L_{R-P}) strategy is employed, meaning that the reward and penalty parameters are equal: $\alpha^{+} = \alpha^{-}$. The improvement gain calculation is also simplified: β_e is defined such that $\beta_e \in \{0,1\}$. It assumes a value of one if successful and zero in case of failure. This approach was shown to be successful for improving heuristic search algorithms by Wauters (2012).

6. Experiments

The proposed algorithm was coded in Java 1.8 and executed on Intel(R) Xeon E5-2680v3 CPU @ 2.5GHz computers with 64GB of RAM memory running Red Hat Enterprise Linux ComputeNode 6.5. All computational experiments were performed according to the VeRoLog challenge rules: the running time was limited to 600 seconds per instance and at most 4 CPU cores were used per execution.

The LAHC list size and the maximum number of consecutively rejected neighbors were manually tuned. After a significant number of experiments considering different values for the parameters, four sets were selected based on their performance: (*i*) l=5000 and m=5000; (*ii*) l=12500 and m=12500; (*iii*) l=25000 and m=25000; (*iv*) l=25000 and m=50000. Each set is considered by the solver in a different thread.

This section begins by presenting the instances provided by the VeRoLog Solver Challenge in Section 6.1. Section 6.2 briefly analyzes the neighborhood size reduction's impact. Section 6.4 shows the behavior induced by the learning automata by presenting the evolution of certain neighborhood selection probabilities. The results obtained for the VeRoLog datasets are presented in Section 6.5 and, finally, Section 6.6 introduces new SB-VRP instances aimed at stimulating future research.

6.1. VeRoLog challenge datasets

In total, six instance sets have been published by the VeRoLoG challenge committee: *small*, *medium*, *large*, *presel*, *final* and *final_random* datasets. The first three datasets were introduced prior to the challenge. The *presel* dataset was used to evaluate algorithms during the pre-selection phase, while the remaining two datasets, *final* and *final_random*, were considered to determine the challenge winner. Each dataset contains three instances:

Table 2: Characteristics of VeRoLog challenge instances

Instance	$\|C\|$	$\|C_1\|$	$\|C_2\|$	$\|C_3\|$	$\|S\|$	T	Q	$\sum q_i$
small_all_with	57	0	56	1	20	8h	500	8445
small_normal	57	15	41	1	20	8h	500	8445
small_all_without	57	57	0	0	20	8h	500	7950
medium_all_with	206	0	206	0	41	11h	1000	24790
medium_normal	206	20	186	0	41	11h	1000	24790
medium_all_without	206	206	0	0	41	11h	1000	24000
large_all_with	548	0	548	0	99	11h	1000	65080
large_normal	548	50	498	0	99	11h	1000	65080
large_all_without	548	548	0	0	99	11h	1000	63090
presel_all_with	550	0	550	0	101	11h	1000	58845
presel_normal	550	50	500	0	101	11h	1000	58845
presel_all_without	550	550	0	0	101	11h	1000	58845
final_normal_all_with	549	0	549	0	102	18h	1000	58785
final_normal	549	50	499	0	102	18h	1000	58785
final_normal_all_without	549	549	0	0	102	18h	1000	58785
final_rand_all_with	549	0	549	0	102	18h	1000	327388
final_rand	549	50	499	0	102	18h	1000	331239
final_rand_all_without	549	549	0	0	102	18h	1000	325815

one in which all customers can be visited by trains (*all_with*), one containing all customer types (*normal*) and one in which no customer can be visited by trains (*all_without*).

Table 2 shows the characteristics of these instances. The nomenclature presented during Section 2 is employed and $\sum q_i$ represents the sum of the demands of all customers $i \in C$. Note that the number of train-only customers ($|C_3|$) is either one or zero. All instance files[1] are available online[2].

6.2. Neighborhood size reduction analysis

A quick analysis on optimal Capacitated Vehicle Routing Problem's (CVRP) solutions was performed insofar as providing some insight regarding the neighborhood size procedure (Section 5.1). The objective is to show some values of the parameter η that would not prevent generating an optimal solution for most instances.

The datasets from Augerat et al. (1995) (sets A, B and P), Christofides and Eilon (1969) (set E), Christofides et al. (1979), Fisher (1994) (set F), Golden et al. (1998), Li et al. (2005) and Uchoa et al. (2017) were employed for the analysis. When no proven optimal solution was available for an instance, the best known solution was considered. In total, 241 instances were evaluated, of which 102 had a proven optimal solution. The evaluation consisted of identifying how many optimal (or best known) solutions would be deliberately pruned for a certain value of η. This occurs whenever a customer $i \in C$ is not connected to one of its η nearest neighboring customers.

Figure 4 depicts a graph visualizing the impact of a relative value for η, or more precisely of setting $\eta = \theta|C|$, where $\theta \in [0, 1]$ is the percentage of customers considered in the neighborhood size reduction. Note that the impact of parameter η depends upon individual instance characteristics, such as how the customers are distributed. Given the small number of SB-VRP instances and the lack of available information concerning optimality, CVRP instances were utilized. The conclusions may therefore not apply directly to the SB-VRP, but continue to provide valuable insights. Figure 4 enables one to conclude that at least one optimal (or best known) CVRP solution satisfies the neighborhood reduction criterion if $\eta \geq 0.45|C|$ for all 241 instances considered. Furthermore, setting $\theta = 0.15$, for instance, would include at least one optimal (or best known) solution for over 93% of the instances. This is a strong indication that neighborhood reduction techniques such as the one presented are worth investigation.

The impact of the neighborhood size reduction on the SB-VRP algorithm's final solution was also evaluated. Figure 5 illustrates the algorithm's performance with and without the neighborhood size reduction. The figure presents the gap to the best solution obtained for each dataset provided by the challenge (*small*, *medium*, *presel*, *final* and *final_random*). For each dataset, two boxplots are presented: the first one represents the algorithm with neighborhood size reduction and the second one represents the algorithm without it. Each dataset contains three instances, *all_with*, *normal* and *all_without*. The algorithm was executed 20 times with different random seeds for each instance, and therefore each boxplot aggregates

[1] The VeRoLog Challenge organizers gently provided the instances and permitted us to publish them.
[2] http://benchmark.gent.cs.kuleuven.be/sbvrp

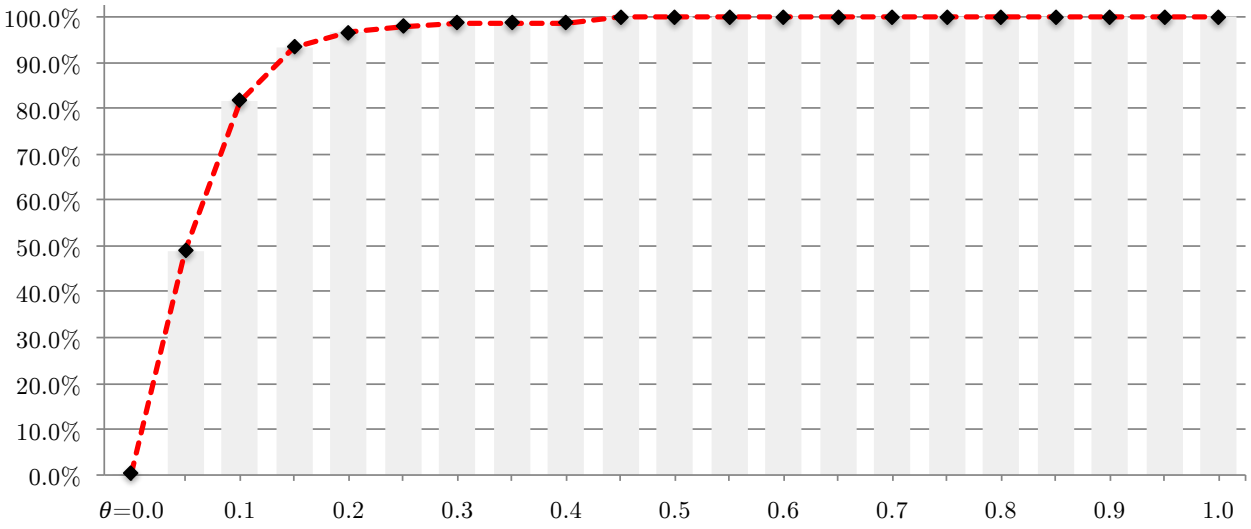


Figure 4: Impact of parameter η, with $\eta = \theta|C|$, on CVRP benchmark instances

60 solutions (20 solutions per instance). The runtime limit of 600 seconds is enforced, and η has been set to 50 or 75, depending on the neighborhood. The learning automaton approach was considered when selecting these values (more details are provided in Section 6.4).

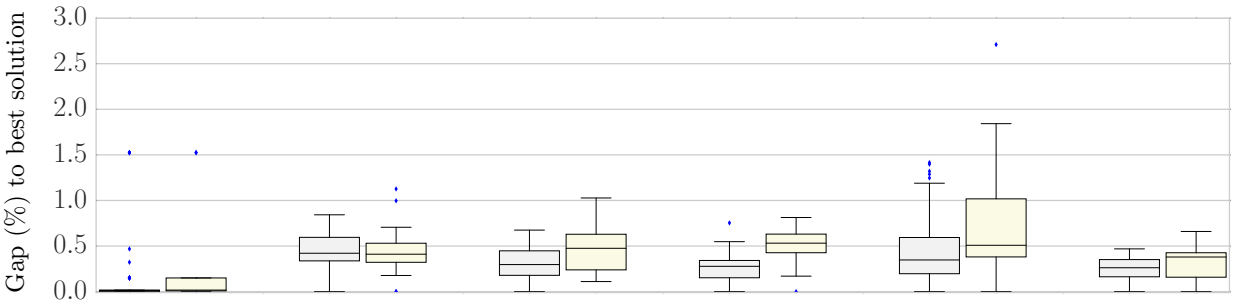


Figure 5: Results (gap to the best solution) produced by the algorithm with (left) and without (right) the neighborhood size reduction procedure

Figure 5 demonstrates how, despite the possibility of pruning away an optimal solution, the neighborhood size reduction procedure positively impacts the algorithm's performance. Permitting a larger number of (improving) iterations within the time limit results in superior solutions. When *small* and *medium* instances are considered, however, the advantage of employing the neighborhood size reduction is less profound. This behavior is expected since small instance solutions have fewer neighbors. Hence, the performance improvement in these cases is smaller, as there is much less pruning by the neighborhood size reduction.

6.3. Neighborhood groups

Experiments isolating each neighborhood group (Section 5) were performed to evaluate their impact on the final solution quality. Let $\mathcal{N}$ be the set containing all developed

neighborhoods. These neighborhoods are divided into three subsets: $\mathcal{N}_1$ represents the classical VRP neighborhoods, $\mathcal{N}_2$ the problem-specific neighborhoods, and $\mathcal{N}_3$ the subproblem optimization neighborhoods. Table 3 compares the results obtained by the algorithm considering different combinations of neighborhood subgroups. The average gap to the solutions obtained with all neighborhoods are presented. The values are aggregated by instance type: all_with, normal and all_without. Each line shows the average gap for 60 executions, 10 for each instance. A runtime limit of 600 seconds was imposed for all executions. The overall average gap is also presented providing a rough ranking with regard to the different strategies.

Table 3: Average gap obtained by employing different neighborhood group combinations

	All neighborhoods	Disabling one neighborhood group			Single neighborhood group		
	$\mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$	$\mathcal{N}_1 \cup \mathcal{N}_2$	$\mathcal{N}_1 \cup \mathcal{N}_3$	$\mathcal{N}_2 \cup \mathcal{N}_3$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$
all_with	0.00%	0.90%	-0.06%	0.15%	0.76%	66.90%	0.12%
normal	0.00%	2.29%	1.21%	0.21%	2.09%	68.81%	1.56%
all_without	0.00%	9.06%	9.45%	0.42%	9.72%	74.41%	10.21%
Average	0.00%	4.08%	3.54%	0.26%	4.19%	70.04%	3.97%

Table 3 enables one to draw some interesting conclusions. Firstly, considering the entire neighborhood set appears to be the best strategy on average. Secondly, it is clear that the subset of problem-specific neighborhoods is incapable of independently producing good quality solutions, as expected. Thirdly, the characteristic of the instances largely impact the relationship between the neighborhoods. Note how excluding the problem-specific neighborhoods was beneficial when dealing with "all_with" instances. These instances enable visiting all customers with trains and therefore the swap locations tend to be less frequently employed, since swap operations become optional. Contrastingly, huge gaps were obtained when the problem-specific neighborhood set was ignored for the "all_without" instances. These instances prohibit visiting customers with trains. Swap locations consequently become essential for storing bodies of trains, as without them only trucks may be used. Finally, the table reveals how including subproblem optimization neighborhoods was on average more beneficial than including classical VRP neighborhoods in almost all situations.

6.4. Learning automaton and neighborhoods

The learning automaton algorithm presented in Section 5.5 has been employed in two different contexts: first as an offline tuning tool to assess the initial probabilities of selecting each neighborhood, and second as an online fine-tuning tool to update these probabilities.

The learning automaton has one parameter: reward-penalty α, also known as the *learning rate*. This parameter defines how much the probabilities may change in each iteration. For offline tuning the reward-penalty was set to $\alpha = 10^{-5}$ and the resulting probabilities were employed to help determining initial neighborhood probabilities. Two main criteria were considered for each neighborhood: its time complexity, given by the number of operations executed to generate a neighbor, and the average probability given by the learning automaton after several rounds of LAHC. The principle behind this is that fast and relatively efficient

neighborhoods should initially have higher chances of selection than slow ones. Due to very low probabilities resulting from the offline tuning phase, some neighborhoods were dropped.

Table 4 shows the selected neighborhoods and their respective "importance", given by v. The probability of each neighborhood is given by its importance divided by the sum of all neighborhoods' importance. The initial importance values considered are $v \in \{1, 5, 10, 20, 50, 100\}$. The table also details the values for parameters *size*, k and η, when required by a neighborhood. If the value for η is hidden, then no neighborhood reduction is applied ($\eta = |C|$).

Table 4: Initial probabilities and considered neighborhoods

Neighborhood	v	Neighborhood	v
Swap(η=50)	50	RuinRecreate(10, η=50)	5
TwoOpt()	100	RuinRecreate(15, η=50)	5
ChangeSwapLocation()	50	RuinRecreate(20, η=50)	5
ConvertToRoute()	10	RuinRecreate(25, η=50)	5
AddSubRoute(η=50)	5	RuinRecreate(35, η=50)	5
SplitToSubRoute()	10	RuinRecreate(50, η=50)	5
MergeRoutes()	10	RemoveRoute(η=150)	1
RandomInsert(*size*=1)	10	RemoveSubRoute()	10
RandomInsert(*size*=2)	10	RemoveSubRoute(η=150)	10
RandomInsert(*size*=3)	10	RemoveChains(*size*=1, k=100, η=50)	10
RandomInsert(*size*=4)	10	RemoveChains(*size*=2, k=100, η=50)	10
EjectionChain(*size*=3, η=50)	20	RemoveChains(*size*=3, k=100, η=50)	10
EjectionChain(*size*=4, η=50)	20	RemoveChains(*size*=4, k=100, η=50)	10
EjectionChain(*size*=5, η=50)	20	RemoveChains(*size*=5, k=100, η=50)	10
EjectionChain(*size*=10, η=50)	20	RemoveChains(*size*=6, k=100, η=50)	10
EjectionChain(*size*=15, η=50)	10	RemoveChains(*size*=7, k=100, η=50)	10
EjectionChain(*size*=20, η=75)	10	RemoveChains(*size*=8, k=100, η=50)	10
EjectionChain(*size*=25, η=75)	10	LimRuinRecreate(*size*=4, k=4, η=50)	10
EjectionChain(*size*=30, η=75)	10	LimRuinRecreate(*size*=2, k=5, η=50)	10
RuinRecreate(*size*=1, η=50)	5	LimRuinRecreate(*size*=5, k=2, η=50)	10
RuinRecreate(*size*=2, η=50)	5	RuinRandRecreate(*size*=1)	5
RuinRecreate(*size*=3, η=50)	5	RuinRandRecreate(*size*=2)	5
RuinRecreate(*size*=4, η=50)	5	RuinRandRecreate(*size*=3)	5
RuinRecreate(*size*=5, η=50)	5	RuinRandRecreate(*size*=4)	5

The online fine-tuning phase, present in the final algorithm, considers a lower reward-penalty $\alpha = 10^{-6}$. The goal is to avoid drastic changes in the probabilities while still fine-tuning them. After each execution of the LAHC algorithm, the probabilities are reset to their initial values.

6.5. Results

Table 5 presents the average and best results from 20 algorithm executions for all instances published by the First VeRoLog Challenge. These results are compared against the best results reported in the literature:

HG2014 : Huber and Geiger (2014) executed their approach 30 times on an Intel Xeon X5650 2.66 GHz;

Lum2015 : Lum et al. (2015) reported the best results obtained on a 2012 MacBook Air (precise computer unspecified);

Absi2016 : Absi et al. (2015) employed an Intel Xeon 2.8 GHz and ran their relax-and-repair heuristic 10 times;

THUJG2016 : Todosijević et al. (2016) experimented both their Parallel and Sequential GVNS on an Intel i7-4900MQ 2.80 GHz;

MB2017 : Miranda-Bront et al. (2017) executed their algorithm 10 times on an Intel Core i5-3320M;

HG2017 : Huber and Geiger (2017) utilized an Intel Xeon X5650 2.66 GHz. They report the best solutions out of a very large set of experiments considering different parameters and algorithm versions. Each of these experiments consists of 30 algorithm executions.

With the exception of Absi et al. (2015), who included results from 20-minute runs, runtimes were limited to 10 minutes by all other authors. Table 5 highlights how the algorithm proposed in this paper clearly outperforms all approaches described in the literature. For the majority of the instances, the average over all runs provides a better value than the best solution generated by other algorithms. The produced solutions are available online[3].

Table 5: Results for VeRoLog challenge instances

Instance	Proposed Algorithm		*HG2014*	*Lum2015*	*Absi2016*	*THUJG2016*	*MB2017*	*HG2017*
	Average	Best						
small_all_with	4716.50	**4715.78**	4730.92	4873.05	4716.58	4731.02	4728.93	4716.58
small_normal	4797.69	**4797.55**	4804.97	4959.00	4802.38	4847.63	4806.97	**4797.55**
small_all_without	4858.53	**4839.64**	**4839.64**	5356.36	4981.70	5249.18	4855.62	**4839.64**
medium_all_with	7745.80	**7708.63**	7755.43	8335.57	7763.13	7754.39	7847.30	7734.61
medium_normal	7818.29	7803.54	7817.83	8297.25	7810.93	7834.78	7942.22	**7795.98**
medium_all_without	8007.11	**7980.01**	8045.47	8628.37	8058.89	8382.80	8169.69	7982.76
large_all_with	19851.04	**19806.75**	20215.26	21317.00	20495.70	20066.40	20516.70	20058.99
large_normal	20039.92	**19985.88**	20524.54	22051.40	20760.30	20496.40	20738.50	20298.82
large_all_without	20728.43	**20640.72**	21255.51	22419.40	21580.60	22310.60	21522.50	21003.51
presel_all_with	24475.56	**24402.67**	25072.36	26658.10	25021.70	24965.10	25573.20	24767.63
presel_normal	24859.76	**24800.16**	25425.85	26712.40	25529.50	25443.20	25894.40	25069.86
presel_all_without	25510.38	**25448.55**	25835.85	26712.40	25975.50	26515.90	26524.50	25719.19
final_all_with	33118.08	**33014.03**	-	-	-	-	34997.90	33753.48
final_normal	34254.51	**33927.04**	-	-	-	-	36305.80	34649.78
final_all_without	36574.23	**36347.28**	-	-	-	-	38826.20	36814.84
final_random_all_with	129356.15	**129049.18**	-	-	-	-	131445.00	129257.44
final_random	132758.74	132341.83	-	-	-	-	135509.00	**132295.68**
final_random_all_without	144588.22	**144331.84**	-	-	-	-	152587.00	144725.57
Gap from best result:	0.33%	0.01%	1.36%	6.83%	1.97%	2.93%	3.28%	0.81%

Figure 6 presents boxplot graphs illustrating the proposed approach's performance for instance groups *large*, *presel*, *final* and *final_random*. The dashed lines indicate previous

[3] http://benchmark.gent.cs.kuleuven.be/sbvrp

best solutions. For 9 out of 12 instances, every solution produced by the proposed algorithm lies below the dashed line. This further supports the claim that the proposed approach outperforms all methods in the literature. Figure 6 also enables one to conclude that the algorithm is very robust. In fact, the maximum gap between two solutions obtained by the algorithm after 600 seconds is 1.5% (instance *small_all_without*: costs of 4839.64 and 4913.39). The average gap to the best solution when considering all 360 solutions produced for the 18 instances is only 0.33%.

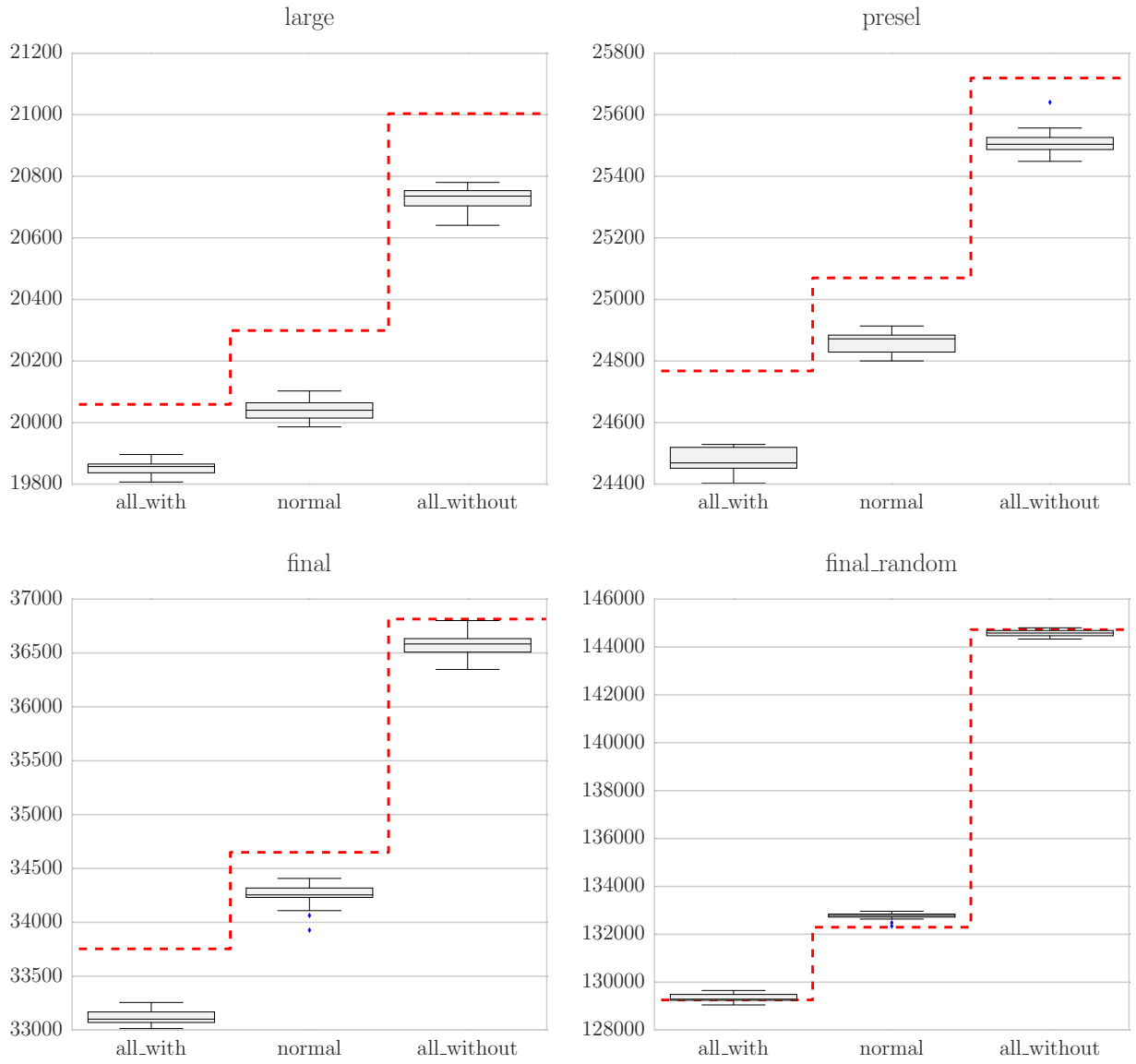


Figure 6: Boxplots comparing the solutions obtained with the proposed approach and the best results reported in the literature for large instances (more than 500 customers)

6.6. Additional instances

New instances are proposed to stimulate additional research in the field, varying many characteristics in contrast to those proposed by the VeRoLog Solver Challenge. These new instances are generated from Uchoa et al. (2017)'s benchmark CVRP instances which are produced by distributing customers within a 1000×1000 grid in accordance with various strategies. Uchoa et al. (2017)'s instances are modified via the addition of either 4, 20 or 100 swap locations and the definition of costs for trains and trucks. Swap locations are generated by first selecting a random customer before randomly selecting a point on a circle of radius 100 from the selected customer's location. If this point lies within the CVRP's 1000×1000 grid then the point is accepted as a swap location. This process continues until the required number of swap locations have been added to the instance. The cost of a truck is fixed at one per distance unit. The cost of a train is given by c per distance unit, with $c \in \{1.2, 1.4, 1.6\}$. Customers are distributed within truck-only and flexible customers, such that $|C_1| = \lfloor k \times |C| \rfloor$ and $|C_2| = |C| - |C_1|$, where $|C_1|$ and $|C_2|$ are the number of truck-only and flexible customers, respectively. No train-only customers are generated. Moreover, swap body capacity simply corresponds to a vehicle's capacity in the CVRP instances.

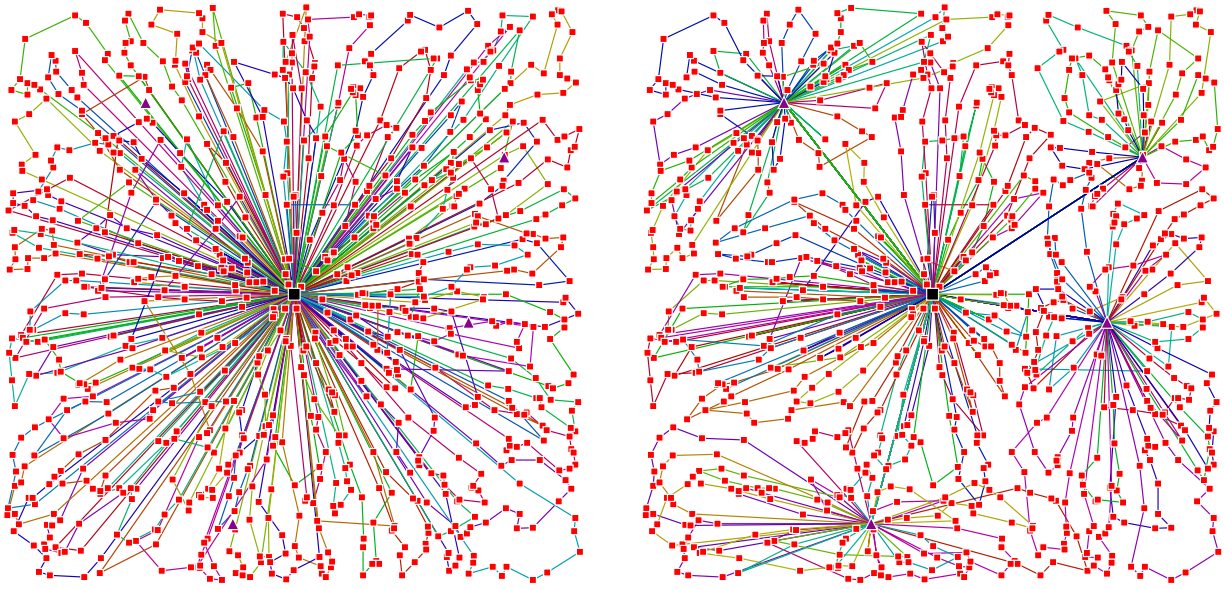
Note that these new instances have significant differences from those proposed during the VeRoLog Solver Challenge. The instances proposed here have symmetric distances, impose no limits on route durations and do not include service times at customers or operation times at swap locations.

An example[4] comparing CVRP and SB-VRP is shown in Figure 7. Instance sbvrp-n936-s4 adds four swap locations to instance X-n936-k151. Customers may be visited only by trucks, and the train's cost is set to 120% of the truck's cost. Note how the four swap locations are utilized extensively within this example, resulting in a 2.5% reduction in the objective function value (129633.4 against 132926.0).

Table 6 presents the characteristics of the generated instances (including characteristics of the original CVRP instances). The columns indicate:

- $|C|$: number of customers;
- *Dep* : placement of the depot, which can be at the center (C), at a random position (R) or at the corner of the grid (E);
- *Cust* : placement of the customers, which may be random (R), in n clusters (C(n)), or "randomly-clustered" (RC(n));
- *Dem* : demand distribution, which may be completely unitary (U), uniformly distributed in a range between n_1 and n_2 (n_1-n_2), depending on quadrant (Q) or with "many small and few large values" (SL);
- Q : vehicle capacity (or capacity of each swap body);
- $|S|$: number of swap locations;
- c : cost of a train (trucks have cost fixed to 1.0 per distance unit);

[4]For more examples, the reader is directed to http://benchmark.gent.cs.kuleuven.be/sbvrp, where solution files and the software for visualizing them are provided.



(a) CVRP solution for X-n936-k151 (b) SB-VRP solution for sbvrp-n936-s4

Figure 7: Example of CVRP and SB-VRP solutions for instances derived from Uchoa et al. (2017)

k : coefficient of customers categorization, $|C_1| = \lfloor k \times |C| \rfloor$ and $|C_2| = |C| - |C_1|$.

Detailed information concerning the characteristics of the original CVRP instances may be found within Uchoa et al. (2017). Table 6 also presents the average and best results out of 20 algorithm executions with a runtime limit of 10 minutes and 1 hour, respectively. Note that the proposed algorithm was executed using the same parameters as those employed in the experiments of Section 6.5.

Table 6: Characteristics of the proposed instances and computational results

#	Instance	Characteristics (Uchoa et al., 2017)					SB-VRP			Results (10min)		Results (1h)	
		$\|C\|$	*Dep*	*Cust*	*Dem*	Q	$\|S\|$	c	k	Avg.	Best	Avg.	Best
1	sbvrp-n101-s100	100	R	RC(7)	1-100	206	100	1.2	0.9	22540.70	22494.80	22506.68	22492.80
2	sbvrp-n106-s20	105	E	C(3)	50-100	600	20	1.6	1.0	23661.37	23615.40	23643.09	23629.60
3	sbvrp-n110-s4	109	C	R	5-10	66	4	1.4	0.5	15920.79	15891.60	15897.86	15891.60
4	sbvrp-n115-s100	114	C	R	SL	169	100	1.4	0.1	12302.95	12191.00	12223.85	12186.20
5	sbvrp-n120-s4	119	E	RC(8)	U	21	4	1.6	0.0	13503.40	13484.20	13486.12	13390.60
6	sbvrp-n125-s20	124	R	C(5)	Q	188	20	1.2	0.1	36515.15	36461.60	36476.56	36439.20
7	sbvrp-n129-s4	128	E	RC(8)	1-10	39	4	1.6	1.0	30469.39	30386.60	30408.99	30355.40
8	sbvrp-n134-s100	133	R	C(4)	Q	643	100	1.4	0.9	9960.20	9896.60	9918.03	9890.40
9	sbvrp-n139-s20	138	C	R	5-10	106	20	1.2	0.0	12679.34	12647.60	12660.45	12647.60
10	sbvrp-n143-s100	142	E	R	1-100	1190	100	1.4	0.5	14824.00	14736.40	14754.40	14736.40
11	sbvrp-n148-s4	147	R	RC(7)	1-10	18	4	1.6	0.9	45448.99	44096.80	44030.44	43252.00
12	sbvrp-n153-s20	152	C	C(3)	SL	144	20	1.2	0.0	14815.98	14770.80	14797.89	14773.20
13	sbvrp-n157-s20	156	R	C(3)	U	12	20	1.6	0.1	15477.34	15426.80	15417.12	15376.00
14	sbvrp-n162-s100	161	C	RC(8)	50-100	1174	100	1.4	0.5	13807.98	13719.40	13745.48	13719.40
15	sbvrp-n167-s4	166	E	R	5-10	133	4	1.2	1.0	19815.63	19767.20	19777.78	19695.20
16	sbvrp-n172-s100	171	C	RC(5)	Q	161	100	1.2	0.0	30032.52	30024.00	30026.21	30024.00
17	sbvrp-n176-s4	175	E	R	SL	142	4	1.6	0.9	49130.29	46707.60	46866.35	45484.00

(continued on next page)

Table 6: Characteristics of the proposed instances and computational results (continued)

#	Instance	Characteristics (Uchoa et al., 2017)					SB-VRP			Results (10min)		Results (1h)	
		$\|C\|$	*Dep*	*Cust*	*Dem*	Q	$\|S\|$	c	k	Avg.	Best	Avg.	Best
18	sbvrp-n181-s20	180	R	C(6)	U	8	20	1.4	0.5	22143.19	22091.00	22104.73	22056.80
19	sbvrp-n186-s4	185	R	R	50-100	974	4	1.2	1.0	25651.45	25553.40	25491.21	25131.80
20	sbvrp-n190-s100	189	E	C(3)	1-10	138	100	1.6	0.1	15463.47	15359.40	15372.54	15301.00
21	sbvrp-n195-s20	194	C	RC(5)	1-100	181	20	1.4	0.1	37900.65	37695.80	37658.32	37471.20
22	sbvrp-n200-s100	199	R	C(8)	Q	402	100	1.6	1.0	50723.48	50592.40	50579.62	50427.00
23	sbvrp-n204-s4	203	C	RC(6)	50-100	836	4	1.2	0.5	20133.14	20068.60	20079.93	20043.40
24	sbvrp-n209-s20	208	E	R	5-10	101	20	1.4	0.0	26738.25	26658.80	26668.98	26622.40
25	sbvrp-n214-s100	213	C	C(4)	1-100	944	100	1.6	0.9	10822.29	10758.00	10763.21	10704.80
26	sbvrp-n219-s4	218	E	R	U	3	4	1.2	0.9	101295.38	100729.00	100978.25	100649.00
27	sbvrp-n223-s20	222	R	RC(5)	1-10	37	20	1.4	0.5	36704.39	36459.00	36516.64	36417.00
28	sbvrp-n228-s20	227	R	C(8)	SL	154	20	1.2	1.0	22649.59	22431.20	22515.39	22427.60
29	sbvrp-n233-s4	232	C	RC(7)	Q	631	4	1.4	0.1	19258.70	18954.60	19107.78	18941.80
30	sbvrp-n237-s100	236	E	R	U	18	100	1.6	0.0	25454.56	25234.20	25279.33	25205.40
31	sbvrp-n242-s20	241	E	R	1-10	28	20	1.2	1.0	62482.20	62171.20	62266.32	62108.60
32	sbvrp-n247-s100	246	C	C(4)	SL	134	100	1.4	0.1	28221.79	28058.60	28058.08	27911.40
33	sbvrp-n251-s4	250	R	RC(3)	5-10	69	4	1.6	0.9	41014.91	37218.60	37626.60	37010.80
34	sbvrp-n256-s4	255	C	C(8)	50-100	1225	4	1.6	0.5	22023.10	21687.80	21639.21	19426.80
35	sbvrp-n261-s100	260	E	R	1-100	1081	100	1.2	0.0	21136.92	20986.60	21041.61	20986.80
36	sbvrp-n266-s20	265	R	RC(6)	5-10	35	20	1.4	1.0	63312.60	63079.20	63118.29	62917.80
37	sbvrp-n270-s20	269	C	RC(5)	50-100	585	20	1.2	0.5	31361.16	31195.00	31243.23	31118.80
38	sbvrp-n275-s4	274	R	C(3)	U	10	4	1.4	0.0	18940.07	18888.80	18903.71	18871.80
39	sbvrp-n280-s100	279	E	R	SL	192	100	1.6	0.9	32197.82	31886.20	31959.69	31722.20
40	sbvrp-n284-s100	283	R	C(8)	1-10	109	100	1.6	0.1	19381.85	19202.60	19235.35	19110.60
41	sbvrp-n289-s20	288	E	RC(7)	Q	267	20	1.4	0.0	71748.75	71639.00	71542.48	71434.60
42	sbvrp-n294-s4	293	C	R	1-100	285	4	1.2	0.9	50338.40	47632.80	47703.31	45380.20
43	sbvrp-n298-s100	297	R	R	1-10	55	100	1.4	1.0	31715.41	31502.60	31485.55	31246.20
44	sbvrp-n303-s4	302	C	C(8)	1-100	794	4	1.6	0.1	22832.06	22711.40	22717.58	22645.20
45	sbvrp-n308-s20	307	E	RC(6)	SL	246	20	1.2	0.5	23563.78	23324.00	23185.37	22928.60
46	sbvrp-n313-s4	312	R	RC(3)	Q	248	4	1.2	0.5	75694.01	73087.00	72755.57	72562.20
47	sbvrp-n317-s100	316	E	C(4)	U	6	100	1.4	0.0	58438.25	58312.80	58380.68	58279.40
48	sbvrp-n322-s20	321	C	R	50-100	868	20	1.6	0.1	31153.38	30819.20	30791.26	30634.00
49	sbvrp-n327-s4	326	R	RC(7)	5-10	128	4	1.2	1.0	26987.69	26801.80	26763.68	26489.60
50	sbvrp-n331-s100	330	E	R	U	23	100	1.4	0.9	28513.69	28260.20	28105.76	27788.20
51	sbvrp-n336-s20	335	E	R	Q	203	20	1.6	0.5	123044.32	122533.40	122361.76	121921.00
52	sbvrp-n344-s4	343	C	RC(7)	5-10	61	4	1.2	0.9	45302.97	41666.20	43216.76	40842.00
53	sbvrp-n351-s20	350	C	C(3)	1-100	436	20	1.6	1.0	26112.07	26021.00	26010.77	25919.00
54	sbvrp-n359-s100	358	E	RC(7)	1-10	68	100	1.4	0.0	42537.92	42286.60	42303.67	41983.60
55	sbvrp-n367-s20	366	R	C(4)	SL	218	20	1.6	0.1	22322.04	22101.60	21996.53	21656.80
56	sbvrp-n376-s100	375	E	R	U	4	100	1.4	0.1	111893.74	111476.20	111517.40	111264.00
57	sbvrp-n384-s4	383	R	R	50-100	564	4	1.2	0.0	45656.69	45470.40	45441.98	45304.80
58	sbvrp-n393-s100	392	C	RC(5)	5-10	78	100	1.6	1.0	36832.35	36519.40	36552.43	36286.60
59	sbvrp-n401-s20	400	E	C(6)	Q	745	20	1.4	0.9	53737.96	53452.00	53495.55	53338.80
60	sbvrp-n411-s4	410	R	C(5)	SL	216	4	1.2	0.5	19278.39	18803.60	18770.09	18603.60
61	sbvrp-n420-s4	419	C	RC(3)	1-10	18	4	1.2	0.5	95750.59	93630.60	91132.09	90620.80
62	sbvrp-n429-s100	428	R	R	50-100	536	100	1.4	0.9	56865.53	56636.00	56494.73	56307.80
63	sbvrp-n439-s20	438	C	RC(8)	U	12	20	1.6	0.0	35549.16	35016.40	35294.73	35009.60
64	sbvrp-n449-s100	448	E	R	1-100	777	100	1.2	0.1	41953.47	41570.80	41412.53	40982.20
65	sbvrp-n459-s20	458	C	C(4)	Q	1106	20	1.4	1.0	25046.39	24685.40	24520.72	24187.20
66	sbvrp-n469-s4	468	E	R	50-100	256	4	1.6	0.1	190409.37	189868.80	189656.37	189426.60
67	sbvrp-n480-s4	479	R	C(8)	5-10	52	4	1.6	0.5	97438.81	89458.20	87876.17	86695.20
68	sbvrp-n491-s20	490	R	RC(6)	1-100	428	20	1.2	0.0	46023.17	45883.40	45806.53	45679.60
69	sbvrp-n502-s100	501	E	C(3)	U	13	100	1.4	1.0	54480.57	54261.80	54327.01	54200.00
70	sbvrp-n513-s100	512	C	RC(4)	1-10	142	100	1.6	0.9	24686.73	24472.40	24460.20	24326.80
71	sbvrp-n524-s4	523	R	R	SL	125	4	1.4	0.5	133072.77	131259.60	129852.68	128261.40
72	sbvrp-n536-s20	535	C	C(7)	Q	371	20	1.2	0.9	71373.16	71191.00	71130.96	70945.80
73	sbvrp-n548-s100	547	E	R	U	11	100	1.2	0.1	63100.78	62789.20	62610.65	62343.00
74	sbvrp-n561-s4	560	C	RC(7)	1-10	74	4	1.6	1.0	52142.10	51781.60	51764.28	51511.60
75	sbvrp-n573-s20	572	E	C(3)	SL	210	20	1.4	0.0	39728.09	39521.00	39515.33	39401.00
76	sbvrp-n586-s20	585	R	RC(4)	5-10	28	20	1.2	0.9	145081.50	144678.00	144430.92	144269.80

(continued on next page)

Table 6: Characteristics of the proposed instances and computational results (continued)

#	Instance	Characteristics (Uchoa et al., 2017)					SB-VRP			Results (10min)		Results (1h)	
		$\|C\|$	*Dep*	*Cust*	*Dem*	Q	$\|S\|$	c	k	Avg.	Best	Avg.	Best
77	sbvrp-n599-s4	598	R	R	50-100	487	4	1.6	0.0	96508.84	96209.40	95975.48	95793.60
78	sbvrp-n613-s100	612	C	R	1-100	523	100	1.4	1.0	54242.47	53891.00	53516.76	53113.40
79	sbvrp-n627-s4	626	E	C(5)	5-10	110	4	1.6	0.1	60551.98	59217.40	58944.11	58026.20
80	sbvrp-n641-s20	640	E	RC(8)	50-100	1381	20	1.4	0.5	56985.91	56630.00	56207.74	55939.40
81	sbvrp-n655-s100	654	C	C(4)	U	5	100	1.2	0.5	73269.46	73186.60	72968.71	72824.00
82	sbvrp-n670-s4	669	R	R	SL	129	4	1.2	0.1	98832.02	98178.20	98238.34	97818.20
83	sbvrp-n685-s20	684	C	RC(6)	Q	408	20	1.6	0.0	64405.40	63970.60	63740.19	63284.20
84	sbvrp-n701-s100	700	E	RC(7)	1-10	87	100	1.4	0.9	69105.02	68307.80	68172.87	67454.60
85	sbvrp-n716-s20	715	R	C(3)	1-100	1007	20	1.4	1.0	39078.70	38738.60	38450.13	38168.20
86	sbvrp-n733-s4	732	C	R	1-10	25	4	1.6	0.9	155822.25	141618.40	146352.27	135869.80
87	sbvrp-n749-s100	748	R	C(8)	1-100	396	100	1.2	0.1	56116.02	55881.00	55723.82	55502.80
88	sbvrp-n766-s100	765	E	RC(7)	SL	166	100	1.6	1.0	106061.99	105462.80	104980.10	104365.80
89	sbvrp-n783-s4	782	R	R	Q	832	4	1.4	0.0	61655.59	61231.80	60879.23	60608.80
90	sbvrp-n801-s20	800	E	R	U	20	20	1.2	0.5	61576.71	60632.80	60179.16	59891.00
91	sbvrp-n819-s100	818	C	C(6)	50-100	358	100	1.4	0.5	125217.12	124521.60	124092.90	123375.80
92	sbvrp-n837-s4	836	R	RC(7)	5-10	44	4	1.2	0.1	134346.01	133712.00	132976.66	132397.20
93	sbvrp-n856-s20	855	C	RC(3)	U	9	20	1.6	1.0	88029.31	87743.80	86955.92	85840.20
94	sbvrp-n876-s20	875	E	C(5)	1-100	764	20	1.6	0.9	90739.96	90187.60	89977.46	89734.80
95	sbvrp-n895-s4	894	R	R	50-100	1816	4	1.2	0.0	42107.46	41907.60	41747.53	41656.80
96	sbvrp-n916-s100	915	E	RC(6)	5-10	33	100	1.4	0.0	241110.22	240702.40	240140.84	239844.80
97	sbvrp-n936-s4	935	C	R	SL	138	4	1.2	1.0	158350.03	154993.20	133332.20	129633.40
98	sbvrp-n957-s100	956	R	RC(4)	U	11	100	1.4	0.9	73653.74	73133.60	72926.08	72555.80
99	sbvrp-n979-s20	978	E	C(6)	Q	998	20	1.6	0.5	109252.51	107932.20	107572.13	106815.60
100	sbvrp-n1001-s4	1000	R	R	1-10	131	4	1.6	0.1	76996.90	76172.80	75520.17	74188.80

7. Conclusions

The Swap-Body Vehicle Routing Problem (SB-VRP) concerns a relevant VRP generalization which poses interesting challenges with respect to developing optimization approaches. This work presented the stochastic local search algorithm for the SB-VRP that won the First VeRoLog Solver Challenge (2014).

The presented algorithm explores the problem structure and clearly outperforms all published approaches to the SB-VRP. It has improved the best reported solution in the literature for the majority of the instances introduced during the First VeRoLog Solver Challenge. For *large*, *presel* and *final* instance sets, the worst solution obtained after 20 algorithm runs is better than the previous best known result, indicating the algorithm's superior performance. To encourage further research on the problem addressed by this paper, the CVRP instances proposed by Uchoa et al. (2017) were adapted to the SB-VRP and an automated benchmark website[5] was produced including instances, solutions and a visualization tool.

All algorithmic components were discussed and analyzed, including the hybrid metaheuristic, the considered neighborhoods, the neighborhood size reduction and the learning automaton. An analysis of the neighborhood size reduction in the context of (academic) benchmarks was also presented, based on experiments including SB-VRP and CVRP instances. These combined analyses resulted in various insights concerning the development of competitive local search algorithms to the SB-VRP.

[5] http://benchmark.gent.cs.kuleuven.be/sbvrp

Future research directions include simplifying the presented algorithm. This paper described the winning approach of the First VeRoLog Challenge as it was implemented. However, the large number of neighborhoods and components hinder the understanding of the algorithm's behavior and prove a challenge in terms of reproduction. Additionally, core components such as the learning automaton may be further explored to prune neighborhoods, enabling simplification without significant loss in terms of solution quality. Finally, some of the ideas here presented should be applied to related problems, such as the VRPT and the TTRP.

Acknowledgements

Work supported by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be), IWT 130855 grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) in cooperation with Conundra (www.conundra.eu), and Leuven Mobility Research Center. The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Hercules Foundation and the Flemish Government – department EWI.

In addition, we would like to thank Luke Connolly for providing editorial consultation.

References

Absi, N., Cattaruzza, D., Feillet, D., Housseman, S., 2015. A relax-and-repair heuristic for the swap-body vehicle routing problem. Annals of Operations Research, 1–22.
URL http://dx.doi.org/10.1007/s10479-015-2098-8

Abuhamdah, A., 2010. Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. In: IJCSNS- International Journal of Computer Science and Network Security, Vol. 10 No. 1, January.

Augerat, P., Belenguer, J., Benavent, E., Corberán, A., Naddef, D., Rinaldi, G., 1995. Computational results with a branch and cut code for the capacitated vehicle routing problem. Tech. rep., University Joseph Fourier, Grenoble, France.

Bresina, J., Bresina, L., 1996. Heuristic-Biased Stochastic Sampling. In: AAAI-96 Proceedings. pp. 271–278.

Burke, E. K., Bykov, Y., 2008. A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems. In: Proccedings of PATAT 2008 conference.

Burke, E. K., Bykov, Y., 2017. The late acceptance hill-climbing heuristic. European Journal of Operational Research 258 (1), 70 – 78.
URL http://www.sciencedirect.com/science/article/pii/S0377221716305495

Caramia, M., Guerriero, F., 2009. A heuristic approach for the truck and trailer routing problem. Journal of the Operational Research Society 61 (7), 1168–1180.
URL http://dx.doi.org/10.1057/jors.2009.59

Chao, I. M., 2002. A tabu search method for the truck and trailer routing problem. Computers and Operations Research 29 (1), 33–51.

Christofides, N., Eilon, S., 1969. An algorithm for the vehicle-dispatching problem. Operational Research Quarterly (20), 309–318.

Christofides, N., Mingozzi, A., Toth, P., 1979. The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (Eds.), Combinatorial Optimization. Vol. 1. Wiley Interscience, pp. 315–338.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., Vigo, D., 2007. Chapter 6 Vehicle Routing. In: Barnhart, C., Laporte, G. (Eds.), Handbooks in Operations Research and Management Science. Vol. 14. Elsevier, pp. 367 – 428.
URL http://www.sciencedirect.com/science/article/pii/S0927050706140062

Croes, G. A., 1958. A method for solving traveling-salesman problems. Operations Research 6 (6), 791–812.
URL http://dx.doi.org/10.1287/opre.6.6.791

Fisher, M. L., 1994. Optimal solution of vehicle routing problems using minimum k-trees. Operations Research 42 (4), 626–642.
URL http://dx.doi.org/10.1287/opre.42.4.626

Gerdessen, J. C., 1996. Vehicle routing problem with trailers. European Journal of Operational Research 93 (1), 135–147.

Glover, F., 1991. Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem. Tech. rep., Leeds School of Business, University of Colorado, Boulder.

Glover, F., 1996. Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65, 223–253.

Goerler, A., Schulte, F., Voß, S., 2013. An application of late acceptance hill-climbing to the traveling purchaser problem. In: Pacino, D., Voß, S., Jensen, R. (Eds.), Computational Logistics. Vol. 8197 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 173–183.
URL http://dx.doi.org/10.1007/978-3-642-41019-2_13

Golden, B., Wasil, E., Kelly, J., Chao, I.-M., 1998. The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results. In: Crainic, T., Laporte, G. (Eds.), Fleet Management and Logistics. Centre for Research on Transportation. Springer US, pp. 33–56.
URL http://dx.doi.org/10.1007/978-1-4615-5755-5_2

Heid, W., Hasle, G., Vigo, D., 2014. Verolog solver challenge 2014 – VSC2014 problem description.
URL http://verolog.deis.unibo.it/news-events/general-news/verolog-solver-challenge-2014

Huber, S., Geiger, M., 2014. Swap body vehicle routing problem: A heuristic solution approach. In: González-Ramírez, R., Schulte, F., Voß, S., Ceroni Díaz, J. (Eds.), Computational Logistics. Vol. 8760 of Lecture Notes in Computer Science. Springer International Publishing, pp. 16–30.
URL http://dx.doi.org/10.1007/978-3-319-11421-7_2

Huber, S., Geiger, M. J., 2017. Order matters - A variable neighborhood search for the swap-body vehicle routing problem. European Journal of Operational Research 263 (2), 419 – 445.
URL http://www.sciencedirect.com/science/article/pii/S0377221717303934

Li, F., Golden, B., Wasil, E., 2005. Very large-scale vehicle routing: new test problems, algorithms, and results. Computers & Operations Research 32 (5), 1165 – 1179.
URL http://www.sciencedirect.com/science/article/pii/S0305054803003150

Lin, S.-W., Yu, V. F., Chou, S.-Y., 2009. Solving the truck and trailer routing problem based on a simulated annealing heuristic. Computers and Operations Research 36 (5), 1683–1692.

Lin, S.-W., Yu, V. F., Chou, S.-Y., 2010. A note on the truck and trailer routing problem. Expert Systems With Applications 37 (1), 899–903.
URL http://dx.doi.org/10.1016/j.eswa.2009.06.077

Lin, S.-W., Yu, V. F., Lu, C.-C., 2011. A simulated annealing heuristic for the truck and trailer routing problem with time windows. Expert Systems With Applications 38 (12), 15244–15252.
URL http://dx.doi.org/10.1016/j.eswa.2011.05.075

Lourenço, H. R., Martin, O. C., Stützle, T., 2003. Iterated local search. In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics. Vol. 57 of International Series in Operations Research & Management Science. Springer US, pp. 320–353.

Lum, O., Chen, P., Wang, X., Golden, B., Wasil, E., 2015. A Heuristic Approach for the Swap-Body Vehicle Routing Problem. In: 14th INFORMS Computing Society Conference. pp. 172–187.

Miranda-Bront, J. J., Curcio, B., Méndez-Díaz, I., Montero, A., Pousa, F., Zabala, P., 2017. A cluster-first route-second approach for the swap body vehicle routing problem. Annals of Operations Research 253 (2), 935–956.

URL http://dx.doi.org/10.1007/s10479-016-2233-1

Narendra, K. S., Thathachar, M. A. L., 1989. Learning Automata: An Introduction. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Özcan, E., Bykov, Y., Birben, M., Burke, E. K., 2009. Examination timetabling using late acceptance hyper-heuristics. In: Proceedings of the Eleventh conference on Congress on Evolutionary Computation. CEC'09. IEEE Press, Piscataway, NJ, USA, pp. 997–1004.
URL http://dl.acm.org/citation.cfm?id=1689599.1689731

Parragh, S. N., Cordeau, J.-F., 2017. Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows. Computers & Operations Research 83, 28–44.
URL http://www.sciencedirect.com/science/article/pii/S0305054817300266

Scheuerer, S., 2006. A tabu search heuristic for the truck and trailer routing problem. Computers and Operations Research 33 (4), 894–909.

Todosijević, R., Hanafi, S., Urošević, D., Jarboui, B., Gendron, B., 2016. A general variable neighborhood search for the swap-body vehicle routing problem. Computers & Operations Research, –.
URL http://www.sciencedirect.com/science/article/pii/S0305054816300120

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A., 2017. New benchmark instances for the Capacitated Vehicle Routing Problem. European Journal of Operational Research 257 (3), 845–858.
URL http://dx.doi.org/10.1016/j.ejor.2016.08.012

Verstichel, J., Vanden Berghe, G., 2009. A late acceptance algorithm for the lock scheduling problem. In: Voss, S., Pahl, J., Schwarze, S. (Eds.), Logistik Management, Hamburg, 2-4 September 2009. Springer.
URL https://lirias.kuleuven.be/handle/123456789/249443

Villegas, J. G., Prins, C., Prodhon, C., Medaglia, A. L., Velasco, N., 2013. A matheuristic for the truck and trailer routing problem. European Journal of Operational Research 230 (2), 231 – 244.
URL http://www.sciencedirect.com/science/article/pii/S037722171300324X

Wauters, T., 2012. Reinforcement learning enhanced heuristic search for combinatorial optimization. Ph.D. thesis, KU Leuven.

Yuan, B., Zhang, C., Shao, X., 2015. A late acceptance hill-climbing algorithm for balancing two-sided assembly lines with multiple constraints. Journal of Intelligent Manufacturing 26 (1), 159–168.
URL http://dx.doi.org/10.1007/s10845-013-0770-x