Solving Vehicle Routing and Scheduling with Delivery and Installation of Machines using ILS

Valon Kastrati · Arben Ahmeti · Nysret Musliu

the date of receipt and acceptance should be inserted later

Abstract We propose a new method based on iterated local search to solve a Vehicle Routing and Scheduling problem that was recently introduced in the VeRoLog Solver Challenge 2019. Our algorithm includes several neighborhood search operators and destroy/repair heuristics. We propose two new neighborhood operators aiming to address conflicts between requests and trips for better allocation on days, vehicles and routes. Our algorithm was one of the competitors of the VeRoLog Solver Challenge where 13 different teams participated by submitting solutions for the instances from the all-time-best challenge. Results on twenty-five instances presented by the organizer of the challenge show that our approach provided second best result for one of the instances, for six instances third best and for most of the rest fourth best result.

Keywords Vehicle Routing · Scheduling · Metaheuristics

1 Introduction

The class of Vehicle Routing Problems is a family of combinatorial optimization problems that arise in the field of logistics. Capacitated Vehicle Routing Problem (CVRP) is the simplest and most well-known variant, whose objective

Valon Kastrati
Vienna University of Technology, TU Wien
E-mail: e1634370@student.tuwien.ac.at

Arben Ahmeti
Institute of Logic and Computation, DBAI, TU Wien
E-mail: aahmeti@dbai.tuwien.ac.at

Nysret Musliu
Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, Institute of Logic and Computation, DBAI, TU Wien
E-mail: musliu@dbai.tuwien.ac.at

is to find a sequence of visits for each truck and minimize the total distance travelled, where all the customers are served, and the sum of demands for each truck does not exceed the maximum capacity of a truck. CVRP instance that uses only one truck with unlimited capacity is equivalent to the Travelling Salesman Problem (TSP) which serves as proof that CVRP is also NP-Hard problem. In CVRP all the routes start and end at a single depot, and the number of trucks is not important. Different variations and extensions of VRP have been defined and proposed as a result of many practical applications resulting in several types of constraints and objectives. For example, there are variations of VRP that include pickup and delivery requests (VRPPD), soft or hard time window for the requests (VRPTW), multiple depot (MDVRP). Vehicles might have different capacities (Heterogeneous VRP) and they are allowed to make multiple trips (Multitrip VRP). For a more comprehensive overview of the classification of VRP problems see [1].

As the number of models of VRP grows continually, so does the number of methods and approaches proposed by researchers. Classical methods include several construction heuristics, for example, the Clarke and Wright savings algorithm. The Clarke and Wright heuristic, proposed at [2], repeatedly merges two routes (starting from single request routes) by maximizing their savings. Other important construction heuristics are the *nearest neighbor* and the *cheapest insertion*, proposed by [21], that work by inserting requests into routes following a strategy such as nearest neighbor or cheapest insertion.

Well-known classical algorithms include inter-route improvement methods (*k-opt* moves based on TSP problem [11]), intra-route improvements like *λ-interchanges* [15], *Or-opt* [14] and other move operators that basically exchange or relocate a number of customers between different routes or vehicles.

Further improvement of solutions was only possible with the use of metaheuristics based on a single solution and a population of solutions. Simulated annealing was proposed by [15], tabu search was implemented in the algorithm proposed by [3] to solve vehicle routing problems with time windows. Large neighborhood search is used by many authors in general and one of the most successful algorithms was Adaptive LNS, proposed by [16], which was able to solve five different variants of VRP: VRPTW, VRPMD, OVRP, SDVRP, and CVRP. The approach proposed in [7] was to use Variable Neighborhood Search with operators such as 2-Opt, Or-Opt, inter-route relocate, or inter-route exchange embedded in Adaptive Large Neighborhood Search. A heuristic search method that only uses a single neighborhood was proposed by [6]. It focuses on techniques of fast manipulation of memory data during the preprocessing, checks, and different speedups. In [10] the problem is divided into the technicians and the deliveries subproblems, where the first one is treated as a Set Covering Problem and the second part is treated and solved as a Bin Packing Problem.

In this paper, we focus on the problem of Routing and Scheduling of Delivery and subsequent Installations of machines. This variant of VRP was

introduced on the VeRoLog Solver Challenge 2019 [1], organized by EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog [2]) and ORTEC [8].

The challenge was organized in two parts: the first one was called *all-time-best challenge* and the second *restricted resources challenge*. In *all-time-best challenge* there were no restrictions in time or technology that could be used to solve the problem, and there was a set of 25 instances called "Early" that were made available in the challenge. For the second part of the challenge, the participants had to submit an algorithm that was used by organizers to solve instances from the "Hidden" set. In the restricted resources challenge there was a time limit for an instance based on formula $t = f * (10 + n)$, where f is the factor of the computing machine, n is the number of requests in the instance and t is time in seconds.

VeRoLog Solver Challenge 2019 attracted the attention of several researchers and 13 different teams participated by submitting solutions for instances from all-time-best challenge. After this stage of the competition was closed, 8 best teams were selected to participate in the restricted resources challenge. Organizers determined the final ranking for restricted resources challenge after running the solvers submitted by each finalist for the hidden set of instances.

Main contributions presented in this paper are:

- A new approach for Vehicle Routing and Scheduling with Deliveries and Installation of Machines. We present a local search comprised of min-conflicts, nearest neighbor and destroy and repair heuristics embedded in an iterated local search framework.
- Introduction of several neighborhood operators including two new neighborhoods for the VRP problem.
- Evaluation of our approach with three sets of benchmark instances proposed at different stages of the VeRoLog Solver Challenge 2019.

The final results of restricted resources challenge were announced in VeRoLog Meeting in Seville on 4 June 2019 [3], with top three teams giving their presentations at the conference.

2 Problem Description

Vehicle Routing Problem with Delivery and Installation of the Machines (DIM) is one of the newest problems in VRP, that was introduced by the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog) and ORTEC (see [8]). DIM is an extension of CVRP that combines routing and scheduling aspects, which optimizes the distribution and installation of vending machines on specific days. There is a planning horizon spanning over a given number of days, different kinds of machines, and a set of technicians

[1] https://verolog2019.ortec.com/

[2] http://www.verolog.eu/

[3] https://verolog2019.sciencesconf.org

with different skills and different starting locations. Requests from customers contain details such as the first and the last day within which a machine must be delivered, the number of machines and type of machine. If a customer requires different kinds of machines, then a request exists for each kind. Machines have different sizes which must be taken into account when loaded into trucks since trucks have a limited capacity. Delivery of machines is performed by a fleet of identical trucks and each of them starts and ends the day in the depot. Trucks can return many times in depot during a day but they also have a limited maximum daily distance. After deliveries are made, installations must be performed as soon as possible starting from the following day, otherwise, there is a penalty for each day the machine installation is delayed. There is a set of technicians that have different skills, meaning they can only install specified kinds of machines. Each technician may have a different home location, where they start and end a route. They have a limited daily distance, a limited number of locations to visit and they have to take two days off after five consecutive days of work. For each kind of machine, there is a cost associated with each day between delivery and installation. The objective is to minimize the total cost which includes the weighted sum of these components: total distance travelled by trucks, number of total trucks, truck days (a day when a truck is used), total distance travelled by technicians, number of technicians, technicians' days, and waiting times of machines for installations. Distances between locations of the depot, customers, and technicians are calculated as the ceiling of the Euclidean Distance.

In DIM problem, we are given a set of requests $R = \{1, 2, ..., n\}$, and a set of days $D = \{1, 2, ..., d_{max}\}$. Each $r \in R$ has its attributes including location, first and last day between which machines must be delivered, type of machines, and the number of them. The set $T = \{1, 2, ..., t_{max}\}$ contains all the technicians where each of them has a home location and a flag for each kind of machine that shows if the technician is able to perform the installation of that kind of machine. Each location is given as a pair of coordinates (x, y), and the distance between any two locations is calculated with this formula $d = \lceil \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \rceil$

The aim is to design a set of vehicle routes $DR_{dv} = \{dr_1, dr_2, dr_3, ...\}$ for the deliveries, and a set of technician routes for the installations $IR_{dt} = \{ir_1, ir_2, ir_3, ...\}$ for each day $d \in D$, each vehicle, and each technician $t \in T$.

Each route $dr_i \in DR_{dv}$ starts from the depot, delivers the machines to the customers, and ends at the depot. Also, each route $ir_i \in IR_{dt}$ starts from the location of the technician t, performs the installation of the machines, and ends at the starting location. We are also given a weight parameter for each component of the cost, including: (1) vehicle distance, (2) vehicle days, (3) vehicles' number, (4) technicians' distance, (5) technician days and (6) number of technician employed. And at the end, there are the cost parameters for each kind of machine, for the number of days a machine waits for the installation (7). The goal is to minimize the total cost which is a sum of seven components mentioned above and fulfill all hard constraints which are: for each request, delivery must be done between the first and last day, and installation

can be performed starting from the following day of its delivery. A truck has a limited capacity of machines it can deliver during one trip, and maximum distance it can travel during a day. Technicians have a set of skills that specifies which kinds of machines they can install, a maximum distance they can travel during a day, and a maximum number of requests they can execute during a day. Technicians are only allowed to work 5 consecutive days at maximum, and if they do, then they are forced to take two days off.

3 General Description of our Approach

Our approach introduces several innovative ideas such as applying several new neighborhood operators that rely on min-conflicts, destroy-repair, and nearest neighbors heuristics for delivery and installation of machines (DIM) problem. The proposed components operate within the framework of Iterated Local Search (ILS) algorithm (algorithm 1).

ILS framework includes four main components: initial solution, embedded local search, acceptance criteria and perturbation. In this section we describe our approach in details.

3.1 Solution representation

Solution candidates are represented using a data structure that includes several entities: days, vehicles, vehicle trips and routes, delivery requests, technicians, technician routes and installation requests. Since the delivery of the machines is done separately from the installation of the machines, then we refer to them as delivery requests and installation requests.

These entities are organized in a hierarchical way so that they reflect the relationship between each of them. A solution contains a vector of days and each day has two lists: the vehicle routes and technician routes. The vehicle routes contain the list of delivery requests assigned to a vehicle, including the returns of the vehicle to the depot during the day. We use the concept of the trip which represents a part of the route that starts and ends at the depot, to allow the trips to be assigned to different vehicles or even different days. The technician routes contain the installation requests. This approach is known in the literature as 'cluster first-route second' paradigm, where first we create the groups of the request that are assigned to days and vehicles (cluster), and then every cluster is treated as a TSP problem. This hierarchical decomposition into subproblems was proposed by several authors such as [5], [4] and [19]. In the case of our problem, we can perform another decomposition by dividing the deliveries and installation schedules, because they can be treated as two dependent VRP problems.

Algorithm 1 ILS for DIM problem

```
s ← InitialSolution()
s'' ← s
gap ← 0
repeat
   s1 ← MoveTrip(s)  or
   s1 ← SwapTrips(s)
   Through roulette wheel set order of :
   s2 ← RemoveInsertDeliveriesStack(s1)
   s3 ← RemoveInsertInstallationsStack(s2)
   s4 ← RegionGroupingNN(s3)
   s5 ← LoopSwap(s4)
   s6 ← SwapInstallations(s5)
   s ← s6
   if (s < s'') then
      s'' ← s
      gap ← 0
   else
      gap ← gap + 1
      if (gap ≥ k) then
         gap ← 0
         Intensify search :
         s7 ← ReInsertDeliveriesSerial(s)
         s8 ← ReInsertInstallationsSerial(s7)
         s9 ← BestSwapWithNNDays(s8)
         s10 ← BestSwapWithNNDaysInst(s9)
         s ← s10
         if (s < s'') then
            s'' ← s
         end if
         Perturbate s :
         sp1 ← PerturbSwapTrip(s)
         sp2 ← PerturbReInsertDeliveriesStack(sp1)
         s ← sp2
      end if
   end if
until timeExpired
```

3.2 Initial solution

After the empty solution is populated with empty routes for each technician, we start the generation of an initial solution by adding the installations in the first feasible position in available technician routes in the given order of technicians and requests. If no feasible position is found for a request, then we perform backtracking by removing the previous requests and inserting the current one (this approach has been successful for all the instances we experimented with). In the next step, we add the deliveries in the same way, by adding each of them in the first available truck. The insertion of installations is more critical because of the hard constraints such as technician skills, technician days-off, a limited number of visits per day, and limited daily distance. This is not the case for deliveries, because the number of trucks is not limited.

It is worth noting that in this phase we only take care of the feasibility of the solution.

3.3 Local search and neighborhood structures

Local search is based on several neighborhood structures and is conducted in two stages. The first one is executed during every cycle of the local search. The second stage contains other operators and they are executed only after a specific number of cycles of local search don't find a better solution. The operators from the first stage are faster compared to those from the second stage, thus they are given more time as long as they improve the solution. After the first stage fails to find improvement for a certain number of cycles, then the algorithm calls the operators from the second stage.

3.3.1 First stage of local search

The first stage contains these operators:

- MoveTrip,
- SwapTrips,
- ReInsertDeliverStack and ReInsertInstallStack,
- RegionGroupingNN,
- LoopSwap,
- SwapInstallations.

The order of the execution of operators is determined in a probabilistic manner, i.e. in each iteration, a new order of operators is calculated based on probabilities that are determined by an algorithm configurator (we will give more information in the section about experimenting).

MoveTrip - moves a trip from one truck to another within a permitted time window. It uses the idea of min-conflicts heuristic ([13]) to generate the neighborhood in the way that all the trips from a day are treated as variables that are in conflict for a better allocation in one of the available trucks. This neighborhood operator is expected to improve the solution by decreasing the number of trucks or truck days and indirectly changing the configuration of the installations. It also may generate solutions with equal cost but a different structure.

SwapTrips - swaps two trips from different trucks. It also uses min-conflicts heuristic to generate the neighborhood by treating all the possible pairs of trips as variables and finding the best swapping pair. *SwapTrips* and *MoveTrip* are executed alternately, i.e. the former is executed in current loop and the latter in the next. These operators are very efficient when it comes to decreasing the number of vehicles because they treat the problem as a bin packing problem, where the vehicles and their trips are treated as bins and items respectively. In order to evaluate the role of these neighborhood operators, we ran our algorithm with and without operators *MoveTrip* and *SwapTrips*, with a set

of instances comprised of 15 instances from the early set (E01 to E15) and 9 from the late set (L01 to L09). The algorithm was run for each instance, five times in each mode for 500 seconds. We measured the relative improvement of the average costs for each instance. The results are depicted in figure 1. As the chart shows, when the algorithm uses *MoveTrip* and *SwapTrips*, it generates better solutions for most of the instances. For example, we have an improvement ratio of over 8% for the problem instance L09.

The pseudocode of an operator that uses min-conflicts heuristic is given in the algorithm 2. In this procedure, we have an outer loop that is executed as long as there are global improvements. Inside the outer loop, we first set the variable $BestLocalCost$ to infinity. Then we start a loop that performs the moves for as long as there is an improvement compared to $BestLocalCost$. At the end of the inner loop, we check if the resulting solution s is better than $IncumbentSolution$. If we have a global improvement we repeat the outer loop. As it can be noticed, the inner loop will always execute at least two times, because first, we compare the $Cost1$ against ∞. In this way, the first move may result in a worse solution, and in the next cycles only improving moves are accepted. The method *SelectAMove* selects a move depending on the operator, while the method *EvaluateMove(s, Move)* returns the cost of solution s if $Move$ is executed. In the case of *MoveTrip* and *SwapTrips* the method $SelectAMove$ selects a trip to move or a pair of trips to be swapped respectively.

Algorithm 2 Procedure Min-Conflicts

```
s ← CopyOf(IncumbentSolution)
repeat
    LocalImprovement ← false
    BestLocalCost ← ∞
    repeat
        Move ← SelectAMove(s)
        Cost1 ← EvaluateMove(s, Move)
        if Cost1 < BestLocalCost then
            ExecuteMove(s, Move)
            LocalImprovement ← true
            BestLocalCost ← Cost1
        else
            LocalImprovement ← false
        end if
    until LocalImprovement = false
    if s.Cost < IncumbentSolution.Cost then
        IncumbentSolution ← CopyOf(s)
        GlobalImprovement ← true
    else
        GlobalImprovement ← false
    end if
until GlobalImprovement = false
```

ReInsertDeliverStack and ReInsertInstallStack - use destroy and repair heuristic and are similar except for the fact that the former starts destroy-

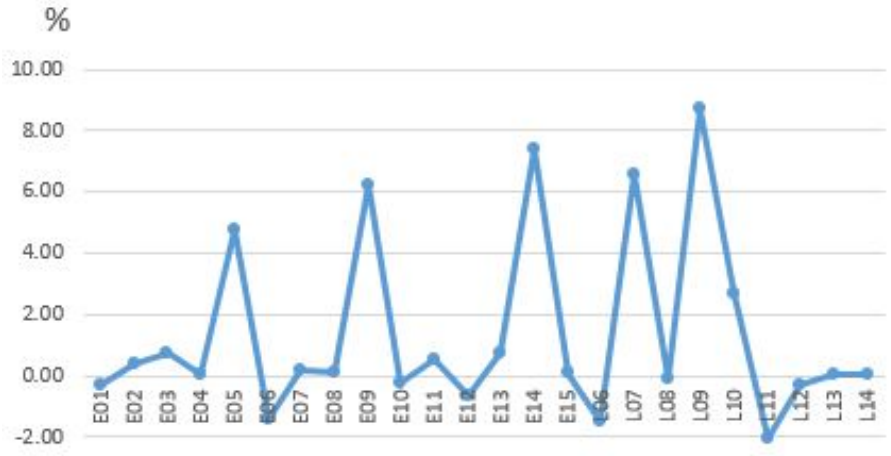

Fig. 1 The ratio between with and without *MoveTrip* and *SwapTrips*

repair on deliveries and the latter starts on installations. We implement several strategies for the destroy part and they are described later in this section. It is worth noting that when we select a request for removal, we remove both the delivery and installation from the schedule. Reinsertion is done in a greedy way for each pair of the delivery and installation, by finding the best positions in the entire feasible time window. This increases the search space because we now consider the best insertion for both the delivery and installation. We have experimented with the order of the insertion, but the procedure was very slow and it did not improve the results significantly. Insertion of deliveries and installations is committed in the order they are removed from the schedule.

RegionGroupingNN - performs removal and insertion of requests by selecting a list of Nearest Neighbors. After a set of requests are selected, they are grouped in a new trip that is assigned to a vehicle. This strategy has the potential to improve the solution because it removes from the schedule the requests that are assigned to different trips and regroups them in an optimized trip.

The three operators mentioned previously are based on the idea of different heuristics for destroy and repair, an approach that has been used by [20], [16] and [18]. This operator implements the min-conflicts heuristic in the same way as algorithm 2.

LoopSwap - performs a sequential swap of requests (nodes) noted as r_i and their positions $p_i, i \in \{1, ..., l\}$, where request r_1 is moved from position p_1 to p_2, r_2 to p_3, ..., r_l to p_1, creating a closed cycle of swapping. This operator affects only deliveries from a certain day. The idea of this operator comes from the classical move called Ejection Chain which was proposed by authors in [17]. The potential of this move relies on the fact that while an improving

single swap of two nodes becomes less likely to happen, increasing the number of swaps in a move increases the possibility to find improvements.

SwapInstallations - swaps two installation requests. The neighborhood is generated so that the first request is selected randomly and then the best swap with all possible installations is executed. In order to extend the selected neighborhood, we also remove the delivery part of the request. The procedure is executed as long as there are improvements and its main target is the subproblem regarding the installations and technicians.

3.3.2 Second stage of local search

The second stage of local search is executed after a parameterized (k) gap of iterations of the first stage without improvement and it uses these operators:

- *ReInsertDeliverSerial* and *ReInsertInstallSerial*
- *BestSwapWithNNDays* and *BestSwapWithNNDaysInst*

ReInsertDeliverSerial and ReInsertInstallSerial - take an input list of requests (delivery and installation) and then relocate each request, including the delivery and installation. This operator, in fact, performs a series of relocations, where the evaluation of the move is done only after all the requests are relocated. The idea behind this is to accept temporarily worse solutions in order to move to more promising search space regions.

BestSwapWithNNDays and BestSwapWithNNDaysInst - These two operators search for the best pair of requests to swap, considering the complete set of requests assigned on a certain day. This procedure is repeated for as long as it finds improvements. Since this generated neighborhood is relatively large, we put these operators in the second stage of local search, which is executed only after the first stage cannot make improvements for a given number of cycles.

Destroy - Remove Heuristics - As mentioned previously, several operators use a method that selects a set of requests for removal so that different heuristics for insertion are used afterward. The simplest way is to select a set of requests from a random day, a random truck, a random trip, or a random technician, depending on the number of requests. The idea behind this is to destroy a particular region of the solution. Another approach is to select randomly a request and then appending the request with the nearest location and with overlapping time windows. In this way, we select a set of requests that have a higher probability to be arranged in a single trip or assigned to a day or truck. Another way to select a set of requests is to find the requests whose partial cost in the total solution cost is highest. This can be done using a function $P(s, r) = E(s) - E'(s, r)$ that calculates the difference of current evaluation $E(s)$ of a solution s, and the cost when the request r is removed from the solution $E'(s, r)$. We refer to this heuristic as $RemoveWorst$.

3.4 Acceptance criteria

In our approach, we accept only better solutions. This logic is implemented inside the min-conflicts algorithm which is used by the mentioned operators.

3.5 Perturbation

Perturbation is implemented through two moves: *PerturbSwapTrip* and *PerturbReInsertDeliverStack*. They are the same operators as *SwapTrip* and *ReInsertDeliverStack* respectively, but modified to accept solution of a worse quality. Perturbation is executed with a parameterized (k) gap of iterations, i.e. for successive k iterations no perturbation takes place. We compensate sparse perturbations with the application of several perturbations in a row and selecting the best current solution as the incumbent solution for the next iteration.

3.6 Parameter tuning

Our algorithm contains a set of operators that are executed in a certain order, and this order appeared to be important. While performing experiments we noticed that for different instances, different orders of operators gave different results. For this reason, we used a method based on roulette wheel to determine the order of operators for each cycle, namely the operators: *ReInsertDeliverStack*, *ReInsertInstallStack*, *RegionGroupingNN*, *LoopSwap* and *SwapInstallations* (respectively $O_1, ..., O_5$). The roulette wheel system takes five parameters $p_i, i \in [1,5], \sum p_i = 1$ as input and produces an order of operators. Each parameter p_i is the probability that operator O_i is selected as j-th operator for $j \in [1,5]$. For example, if all the parameters have the same value, $p_i = 0.2, i \in [1,5]$, then the operators will have the same probability to be chosen at each step in the sequence, which means that all the possible permutations of sequences will have the same probability to be chosen. Or, if parameter $p_1 = 1, p_i = 0, i \in [2,5]$, then the operator O_1 will always be the first one in the sequence, and the four remaining operators will be chosen as the next in the sequence with the same probability. We used the SMAC tool ([9,12]) to determine the best configuration of parameters for our algorithm. The SMAC tool performs intensive experiments using one set of training instances, and one set of testing instances to find the optimal values for each parameter. We ran SMAC tool with these parameters:

- Total time limit was set to 2 days,
- Maximum runtime for instance: 500 seconds
- Training Instances: 8 instances chosen randomly from 'Early' and 'Hidden' sets
- Testing Instances: 27 instances chosen randomly from 'Early' and 'Hidden' sets
- The list of parameters to be tuned: $p_i, i \in [1,4]$, (since $p_5 = 1 - \sum_{i=1}^{4} p_i$)

– The possible values for parameters:

$$\{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.7, 0.8\}$$

The execution of SMAC tool with this setup resulted in parameter settings as shown in table 1. From the table, we see that the operator O_4 (*SwapInstallations*) should be selected as the first parameter in sequence 40% of the time. Operators O_1, O_2, O_5 have equal probability to be selected, and that the O_3 will always be the last operator in sequence.

Table 1 Parameter setting found by SMAC

Parameter	Value
p_1	0.2
p_2	0.2
p_3	0
p_4	0.4
p_5	0.2

4 Computational results

We compared the results of our approach with the results submitted in the challenge website by all the participating teams. In the *all-time-best* challenge took part 13 teams who submitted their solutions for 25 instances. Eight teams, including our team, were selected for the final phase with the *restricted resources*. The organizers of the challenge had made available three sets with 25 instances each:

– Early set: 25 instances for *all-time-best* mode
– Late set: 25 instances used to train the algorithm for *restricted resources* mode
– Hidden set: 25 instances that were used in *restricted resources* mode. This set was made available after the finalists for the second stage were chosen and after the submission of solvers was done.

For both challenges, the participants were allowed to use external state of the art solvers such as: FICO [4], GUROBI [5] or CPLEX [6]

4.1 All-time-best challenge mode

At this stage of the challenge, all the participating teams were able to submit their solutions for each instance from the Early set, and the results were ranked

[4] `https://community.fico.com/s/academic-programs`
[5] `http://www.gurobi.com/academia/academia-center`
[6] `http://ibm.biz/AI_CPLEX128`

for each instance. Since at this stage there was no limitation regarding the run time, the participating teams only submitted their best solution. The challenge website [7] reported the best solutions from all the participating teams for each instance from the Early set. From the table 2 we can see the best result for each instance and the results of our approach. The column *Ratio* expresses the relative difference of the cost of our approach compared to the best result, and the *Rank* column shows the ranking of our algorithm (out of 13 teams) for each instance. We can see that we reached the second place for the first instance, the third place for 6 instances, the fourth place for 13 instances, the fifth place for 4 instances and sixth place for one instance. For 7 instances the ratio is less than 1 percent, which shows that the results of our solver are very close to the best results. A more detailed comparison of all-time-best

Table 2 Best solutions submitted by participants for All-Time-Best challenge

Instance	Best	Our Approach	Ratio	Rank
Early01	3,487,969,810	3,487,996,910	0.0008%	2
Early02	11,149,038,115	11,225,768,410	0.6882%	5
Early03	179,700,885	184,117,395	2.4577%	3
Early04	284,205,965	289,436,220	1.8403%	3
Early05	2,223,814,105	2,479,106,500	11.4799%	4
Early06	24,160,989,040	24,732,913,745	2.3671%	3
Early07	45,815,700	45,853,150	0.0817%	3
Early08	109,798,470	110,016,310	0.1984%	3
Early09	18,075,485	19,762,065	9.3308%	4
Early10	18,500,638,020	18,761,510,110	1.4101%	5
Early11	28,549,460	31,781,875	11.3222%	4
Early12	23,933,097,895	24,447,956,620	2.1512%	5
Early13	582,708,670	584,976,550	0.3892%	4
Early14	94,780,375	99,699,795	5.1903%	4
Early15	1,772,831,110	1,784,630,020	0.6655%	5
Early16	3,287,392,325	3,446,044,045	4.8261%	4
Early17	3,018,108,020	3,035,277,510	0.5689%	6
Early18	5,129,752,375	5,405,790,030	5.3811%	4
Early19	9,290,203	9,457,608	1.8020%	3
Early20	4,764,640	5,296,100	11.1543%	4
Early21	1,292,914,150	1,360,412,960	5.2207%	4
Early22	203,485,635	217,759,862	7.0149%	4
Early23	55,207,660	58,177,435	5.3793%	4
Early24	17,337,730	17,960,220	3.5904%	4
Early25	66,769,325	71,241,680	6.6982%	4

VeRoLog Solver Challenge top solvers results is given in table 3. For each solver/team an each instance the relative gap to to the best is presented. The first column represents the instance, the second shows the number of requests, and the rest of columns show the results of five best teams including our team (last column: AAVK). It can be seen that difference of results for most of the instances among solvers is very narrow.

[7] `https://verolog2019.ortec.com/`

Table 3 Comparison of All-time-best VeRoLog Solver Challenge top solvers results

Inst.	n	MJG	UOS	TCS	COKA	AAVK
E01	150	0.0000	0.0008	0.0009	0.0096	0.0008
E02	300	0.0000	0.1086	0.4372	0.5217	0.6882
E03	450	0.0000	0.3570	4.2765	3.7151	2.4577
E04	600	0.0000	0.7426	2.4175	3.8037	1.8403
E05	750	0.0000	1.0570	7.2300	20.2542	11.4799
E06	900	0.0000	0.6966	3.6447	6.9432	2.3671
E07	150	0.0000	0.0159	0.5373	2.4136	0.0817
E08	300	0.0000	0.0083	0.3387	9.8424	0.1984
E09	450	0.0000	0.7916	5.1190	22.8997	9.3308
E10	600	0.0000	0.0822	1.7774	0.2722	1.4101
E11	750	0.0000	1.7460	3.0933	76.2355	11.3222
E12	900	0.0000	0.8513	2.3508	1.4309	2.1512
E13	150	0.0000	0.0410	0.7328	0.3682	0.3892
E14	300	0.0000	0.4298	4.6534	8.7505	5.1903
E15	450	0.0000	0.0704	0.1740	2.9431	0.6655
E16	600	0.0000	0.8654	2.5741	37.7519	4.8261
E17	750	0.0000	0.1128	0.3475	0.2998	0.5689
E18	900	0.0000	1.0690	3.3212	9.5738	5.3811
E19	150	0.0000	0.2566	1.9767	42.7723	1.8020
E20	300	0.7238	0.0000	11.0304	111.3894	11.1543
E21	450	0.0000	0.7025	3.3241	51.1948	5.2207
E22	600	0.0000	0.6228	3.1012	23.3063	7.0149
E23	750	0.0000	1.9225	2.1815	17.7000	5.3793
E24	900	0.0000	2.2040	2.3669	45.7606	3.5904
E25	150	0.0000	0.4744	4.9185	7.0527	6.6982

4.2 Restricted resources challenge

The late set of 25 instances was used for the second stage of the challenge. Participants had to submit the algorithm and the solution for each instance from this set. Then the organizers ran the solver for these instances in their servers and selected a limited number of best performing teams. Solvers from this short list then were used with the hidden set of instances to further determine the complete ranking of submitted solvers. From 9 runs, the best two and worst two solutions were dropped, and the remaining 5 were used to determine the average and standard deviation. Table 4 shows our results for each instance, including the best cost, average cost, and standard deviation.

For each team, the organizers calculated the rank per instance, and then the final ranking was determined as average of all ranks. The table 5 shows the results of the winning team, and the relative results from fourth, fifth and our solver (teams: orlab, TCS and AAVK). From this table we can see that our solver provided better results than fourth place for seven problem instances.

Table 4 The results of our approach for the hidden set in the restricted resources challenge

Instance	Best	Average	StDev
H01	68,049,230	68,497,477	502,829
H02	1,470,177,245	1,654,579,976	164,820,353
H03	1,388,162,220	1,394,947,955	5,454,085
H04	6,053,945	6,122,060	42,006
H05	4,687,014,190	4,946,153,397	207,877,012
H06	33,537,475	33,642,695	62,003
H07	128,584,875	138,686,310	6,575,414
H08	743,077,790	750,319,371	4,230,922
H09	2,925,297,281	3,087,641,280	120,454,552
H10	47,813,445	49,105,137	848,275
H11	5,050,880,606	5,176,963,402	98,658,731
H12	3,082,704,895	3,095,575,436	8,427,309
H13	5,341,187,251	5,368,906,565	16,223,058
H14	1,394,884,865	1,403,511,603	5,094,711
H15	166,559,140	167,109,473	410,510
H16	61,626,085	63,624,954	1,218,645
H17	27,870,689,964	27,950,302,615	51,793,696
H18	54,778,115	55,313,427	290,138
H19	4,425,251,625	4,450,177,592	20,478,291
H20	208,729,450	231,061,473	14,678,813
H21	38,119,685	38,845,432	477,952
H22	7,473,115	7,575,488	96,316
H23	22,946,775,355	23,021,877,311	48,690,845
H24	32,129,246,375	32,216,184,262	74,165,994
H25	817,352,060	988,807,257	118,314,588

Table 5 Comparison of results in restricted resources challenge

Inst.	UOS	orlab	TCS	AAVK
H01	67,347,510	1.757	1.963	1.708
H02	884,328,838	46.944	9.308	87.100
H03	1,356,206,683	1.245	5.431	2.857
H04	5,470,823	17.165	9.190	11.904
H05	2,438,943,861	46.875	18.939	102.799
H06	33,122,942	0.565	2.041	1.569
H07	102,289,614	29.817	7.889	35.582
H08	729,462,821	1.043	7.975	2.859
H09	1,713,909,634	50.785	18.646	80.152
H10	31,615,173	36.361	14.808	55.321
H11	4,143,464,703	47.843	10.383	24.943
H12	2,988,919,325	1.398	6.547	3.568
H13	5,239,090,235	1.164	3.510	2.478
H14	1,380,531,069	0.843	4.536	1.665
H15	163,861,015	1.025	3.069	1.982
H16	53,561,471	33.553	10.688	18.789
H17	27,322,051,463	0.744	3.341	2.299
H18	53,040,623	7.058	3.951	4.285
H19	4,379,503,211	0.402	2.368	1.614
H20	127,117,759	31.932	11.306	81.770
H21	33,217,241	42.250	8.522	16.944
H22	6,750,354	16.978	3.996	12.224
H23	22,368,503,381	1.233	5.690	2.921
H24	31,373,781,566	0.577	4.962	2.685
H25	549,854,756	23.468	13.028	79.831

5 Conclusions and Future Work

In this work, we presented a new approach for Vehicle Routing and Scheduling with Deliveries and Installation of Machines. Our algorithm includes min-conflicts heuristic, nearest neighbors, and destroy and repair heuristics in the iterated local search framework. Several neighborhood operators are evaluated and additionally, we introduced two new neighborhood operators. The experiments showed that the use of new neighborhoods improved the results for most instances. Overall our approach has been able to provide good solutions for a complex problem that includes scheduling and routing.

We participated in the VeRoLog Solver Challenge 2019 where organizers introduced three sets of problem instances for two different experimental settings: all-time-best and restricted resources challenge. Thirteen teams participated in all-time-best challenge, and eight of them were qualified for the restricted resources challenge, including our team. The approach we proposed provided promising results for instances that were used in the competition. In the future, we plan to extend our metaheuristic approach and investigate its hybridization with exact methods.

References

1. Braekers, K., Ramaekers, K., Van Nieuwenhuyse, I.: The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering **99**, 300–313 (2016)
2. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations research **12**(4), 568–581 (1964)
3. Cordeau, J.F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational research society **52**(8), 928–936 (2001)
4. De Franceschi, R., Fischetti, M., Toth, P.: A new ilp-based refinement heuristic for vehicle routing problems. Mathematical Programming **105**(2-3), 471–499 (2006)
5. Fisher, M.L., Jaikumar, R.: A generalized assignment heuristic for vehicle routing. Networks **11**(2), 109–124 (1981)
6. Geiger, M.J.: A contribution to the verolog solver challenge 2019 (2019). URL `https://verolog2019.sciencesconf.org/244024`. Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog)
7. Graf, B.: An adaptive large variable neighborhood search for a combined vehicle routing and scheduling problem (2019). URL `https://verolog2019.sciencesconf.org/249822`. Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog)
8. Gromicho, J., van't Hof, P., Vigo, D.: The verolog solver challenge 2019. Journal on Vehicle Routing Algorithms **2**(1-4), 109–111 (2019)
9. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International conference on learning and intelligent optimization, pp. 507–523. Springer (2011)
10. Jagtenberg, C., Raith, A., Sundvick, M., Shen, K., Maclaren, O., Mason, A.: Matheuristics for the 2019 verolog solver challenge: Mips and bits (2019). URL `https://verolog2019.sciencesconf.org/250600`. Workshop of the EURO Working Group on Vehicle Routing and Logistics optimization (VeRoLog)
11. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations research **21**(2), 498–516 (1973)

12. Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F.: Smac v3: Algorithm configuration in python. 2017 (2017)
13. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. Artif. Intell. **58**(1-3), 161–205 (1992). DOI 10.1016/0004-3702(92)90007-K. URL `https://doi.org/10.1016/0004-3702(92)90007-K`
14. Or, I.: Traveling salesman type combinatorial problems and their relation to the logistics of regional blood banking. (1977)
15. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of operations research **41**(4), 421–451 (1993)
16. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers & operations research **34**(8), 2403–2435 (2007)
17. Rego, C., Roucairol, C.: A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Meta-Heuristics, pp. 661–675. Springer (1996)
18. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4), 455–472 (2006)
19. Salari, M., Toth, P., Tramontani, A.: An ilp improvement procedure for the open vehicle routing problem. Computers & Operations Research **37**(12), 2106–2120 (2010)
20. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International conference on principles and practice of constraint programming, pp. 417–431. Springer (1998)
21. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)