CO 2024 Case report of team B5

Khanh Ngo, Stijn Smoes, Stelian Munteanu, Dhruv Singh, Jelle Overwater

2738543, 2712533, 2733912, 2668483, 2742364

May 20, 2024
Vrije Universiteit Amsterdam

Combinatorial Optimization
Faculty of Science

1 Abstract

This paper aims to solve the problem for the VeRoLog Solver Challenge 2019. The problem addresses a logistical issue encountered by a major international company that manages the delivery and installation of vending machines. These machines must be delivered within a specified time frame tailored to each customer, and a technician must install them promptly following their delivery. A greedy algorithm was used as an initial approach by creating and updating the "brute route schedule" that only considers the cheapest delivery costs for the unfulfilled requests at the moment. Despite not utilizing any optimizer, it took around 1-2 seconds for our greedy algorithm to solve every instance, which is fast and well within polynomial time. In hopes of improving the results, we attempted to make a different program with the implementation of Gurobi [1], which is an optimization solver. The program was also able to solve all instances, but for large instances, the runtime was longer. Regarding performance, the Gurobi program did better than the initial greedy algorithm for all instances. Based on our best values shown in Table 1, we believe our performance is quite decent overall, with some instances achieving nearly 0% above the best-of-class benchmarks.

2 Literature review

The vehicle routing problem (VRP) involves a set of problems where the goal is to determine the most cost-effective routes for vehicles to visit a set of customers. The costs include factors such as travelled distance and vehicle operating expenses. This set of problems is classified as NP-complete (see Section 3 for proof), meaning it cannot be solved in polynomial time. Vehicle routing problems generally consist of a set of customers, a fleet of identical vehicles and a central depot from which the vehicles start and to which they return at the end of the day. Algorithms for solving the VRP have been extensively discussed in the literature. For instance, Mole and Jameson (1976) [2] explored a route-building algorithm using a generalized version of the Clarke and Wright savings algorithm [3]. Their method selects the next potential customer based on the distance to the current customer and the size of said customer's request. Once all customers are visited, the constructed routes are rechecked and potentially altered by interchanging customers to reduce costs without violating constraints, using a 2-approximation algorithm. The algorithm applies to various VRP variants, including those with distance and capacity constraints VRPs have evolved and special cases with certain constraints received their names and abbreviations. Examples of popular and frequently recurring problems are the vehicle routing problem with time windows (VRPTW), the multiple-period vehicle routing problem (MPVRP) [4] and CVRPTW, variants of VRPTW where there are capacity constraints that need to be adhered to. These variants of the vehicle routing problem will be discussed in sections 2.1 and 2.2 with possible solutions found in the literature. A simple algorithm that can determine the route every vehicle has to complete, Prim's algorithm for the minimum spanning tree, can be used as an approximation. The fundamentals of this algorithm are discussed in section 1.3. 1.4 will talk about other solutions that have been used to solve the Verolog solver challenge of 2019.

2.1 The vehicle routing problem with time windows (VRPTW)

In the VRPTW (Vehicle Routing Problem with Time Windows), each customer must be visited within a specified time frame. Hard time windows do not allow early or late arrivals; vehicles must wait until the time window starts and cannot be late [5]. Solomon (1985) [6] investigated various algorithms for solving the VRPTW, including the Clarke and Wright savings algorithm [3], a time-oriented nearest-neighbour algorithm and insertion heuristics. Solomon discovered that the insertion heuristics algorithm, which involves inserting a customer into an existing route at the optimal position to minimize distance and time, yielded the best results. This algorithm was

particularly effective in scenarios where vehicles had many customers to serve. Kolen et al. (1986) [7] employed a branch-and-bound algorithm to address the VRPTW. Their approach utilized a search tree where each node represents a partial solution, adding customers to existing routes based on calculated lower bounds. The optimal partial solution was determined by selecting the one with the lowest lower bound. They used Dijkstra's algorithm on a directed graph to find the shortest routes.

An adaptation of the VRP is the MPVRP, where customers can have multiple time windows for deliveries. Christofides and Beasley (1984) [8] proposed heuristic algorithms to solve the MPVRP by treating it as both a median problem and a travelling salesman problem. Initially, the algorithms determine which customers are visited on which day, followed by an interchange procedure that refines this initial plan to minimize distribution costs. Their method was tested on problem sets with up to 126 customers, achieving a 13% improvement over the algorithm used by Russell and Igo (1979) [9].

2.2 The capacitated vehicle routing problem with time windows (VRPTW)

The capacitated vehicle routing problem (CVRP) is an extension of the VRP in which vehicles have a limited carrying capacity. The items have a quantity, such as weight or volume, and the vehicles have a maximum capacity that they can carry. Popular solution methods for the CVRP are for example Branch-and-Bound. This method systematically explores all possible routes, pruning those that exceed capacity constraints early on. Nearest Neighbour builds a route by repeatedly adding the closest customer that does not violate capacity constraints. A very simple but quite effective one is a greedy Knapsack: This involves selecting customers based on a value-to-weight ratio (e.g., demand-to-distance ratio) and iteratively adding them to a route until the vehicle's capacity is reached. This is simple and quick, providing a good starting solution that can be further refined using other techniques [10]

2.3 Minimum spanning tree: Prim's algorithm

In the literature on VRP variants (Sections 1.1-1.3), a frequently used approach for finding the shortest routes involves selecting the nearest customer to the current location. This method parallels Prim's algorithm, which constructs a minimum spanning tree among a set of nodes. Prim's algorithm operates by starting with a single node and iteratively adding the nearest unvisited node to the tree, ensuring no cycles are formed. The process continues until all nodes are included in the tree [11], [12]. This greedy algorithm is considered optimal for constructing a minimum spanning tree [13].

2.4 Other solutions to Verolog solver challenge of 2019

The 2019 VeRoLog Solver Challenge focused on a unique vehicle routing problem that involved a combination of the distribution and installation of equipment, such as vending machines. The best solution provided to the challenge was created by Benjamin Graf, as part of his PhD thesis at the University of Osnabrück. The paper introduces a method that combines large neighbourhood search and local search heuristics with a decomposition approach, guided by an adaptive layer that takes into account the specific problem instance, a given time limit, and the performance of the computing environment. [14] Another solution provided by COKA-coders tackled the routing challenge by breaking down the problem into manageable components treated as set partitioning problems as a matheuristics approach. Using a "column-wise neighbourhood search" method, precise exploration of a vast solution space is done. Through iterative mixed-integer programming, high-quality solutions for each subproblem can be achieved, followed by a straightforward local search "fusion" heuristic that enhances the overall solution's quality.[15]

3 Proof of NP-completeness

To prove that the problem is NP-complete, we need to show that the problem is **in NP** and the problem is **at least as hard as another known NP-complete problem:**

3.1 The problem is in NP

Given a solution to the problem, we can easily verify its correctness in polynomial time by:

- Verifying that each truck's route starts and ends at the depot (the depot as the endpoint is not explicitly included in the solution file), respects capacity constraints, and does not exceed the maximum daily travel distance.
- Checking that each request is fulfilled within its specified delivery window, and that the delivery date precedes the installation date by at least one day.
- Ensuring that each machine is installed by a technician within the appropriate time frame and skill set.
- Ensuring that technicians' schedules adhere to the maximum consecutive working days and other relevant constraints.
- Checking that the total cost is correctly computed based on the objective cost function.

3.2 The problem is at least as hard as the Traveling Salesman Problem (TSP)

We chose the well-known Traveling Salesman Problem (TSP) and a TSP instance can be defined as: given a set of cities and distances between them, we aim to find the shortest route that visits each city exactly once and returns to the starting city. We reduced the TSP to our problem as follows:

- Each city in the TSP corresponds to a customer location in our problem.
- The distance between cities in the TSP corresponds to the distance between customer locations in our problem.
- A solution for the TSP can be transformed into a solution for our problem by assigning trucks (technicians) to follow the route and deliver (install) machines at each customer location.
- For the TSP, the goal is to find the shortest route, as described earlier. The same can also be applied to the goal of our problem, which is to minimize the total cost.

4 Algorithm

At the beginning of this project, we made a fairly simple program to have at least a feasible solution. After that, we started making a program with the use of Gurobi to try to get solutions with a better score for the instances. Gurobi is an optimization solver designed for mathematical programming, it provides tools to model and solve a variety of optimization problems. After making the program with Gurobi, the program gave better scores for all some instances. However, we also decided to show how we made the simple algorithm in addition to the gurobi algorithm.

4.1 Simple Algorithm

The first algorithm that we have created is mainly based on approaching the deliveries first, then the scheduling part and finally a connection between the both. Let us describe each part separately.

The routes for the delivery of the vending machines were created in the following way. First, all the requests have been assigned to be delivered on their last possible day. The next step is to create all possible combinations of requests within each day (i.e. possible routes), to check which of

them are feasible, i.e. within the maximum capacity and the maximum distance of a truck. Finally, starting from the longest routes, we add them to the final "brute delivery schedule", checking at the same time that no two routes contain the same request.

Further, the newly created "brute delivery schedule" is used to create an installation schedule. Now, we create a schedule of possible instalments based on the deliveries, i.e. all the requests are moved one day later, since the installation of the machines for a request can not take place on the same day as their delivery. Next, we have again created combinations of possible requests within each day. We then iterate through all the combinations (again in decreasing order) and within an inner loop, we iterate through the technicians (based on the decreasing order of their maximum distance) and check whether for a given technician a given route is feasible, given the constraints for technicians.

Given the fact that we iterate through the combinations of routes and technicians using a given order (i.e. in the decreasing order of their length and their maximum allowed distance) and not all technician-route combinations are checked, some requests may remain uninstalled. That is why we decided to perform a sanity check (RouteScheduleConnection part). First, this step requires checking which requests remained uninstalled. For each unfulfilled request, we check what is the cheapest day it may be delivered (based on the previously created "brute route schedule") and reassign it to be delivered on that day by changing the "brute route schedule" and performing all the further steps described above. This is the connection step between the routing and the delivery part. This is the greedy part of the algorithm, only by considering the cheapest delivery costs for the unfulfilled requests at the moment, the "brute route schedule" is recreated.

The output that can be expected is not the optimal one. This is because of how the delivery and installation routes are created. The combinations of possible routes are created only between the requests that have the same latest possible delivery day. However, the case of considering other combinations between the requests with different possible delivery days is not taken into account. Moreover, from the instalment route perspective, the objective will not be optimal since not all the technician-route combinations are considered, as mentioned above.

In addition, there might be feasible instances that are not solved by the algorithm. This could be caused by the "connection part" where we check for unfulfilled requests. In that case, as we mentioned above, the requests for which the algorithm can not create an installation schedule, are decided to be delivered on the cheapest possible day. However, there could be cases when the algorithm would not find an installation schedule based on the newly-created delivery schedule too, so it would be blocked in an infinite while loop.

Concerning the complexity of the algorithm, it is $O(n^4)$ since we used 3 inner loops in the def createsschedule(schedulecombinations, techschedule), in the GreedySchedule.py file, and in the last inner loop, definitions that contain one for loop each are called. The runtime, however, is very optimal, being around 1–2 seconds for all the instances. This can be justified by the low complexity of the algorithm and also because we have not used any optimization software or extra-tools.

4.2 Gurobi Algorithm

For the Gurobi algorithm [1], we will show you how we obtained the solutions by showing the program as an MIP. The MIP goes as follows:

$\mathcal{W}$ is the set of trucks used in the program. The amount of trucks used in the program is constant and needs to be given before running the Gurobi program. $\mathcal{R}$ is the set of all possible truck routes, where the distance of the truck route is less than or equal to the maximum distance a truck can travel. In addition, a truck route can only drive to a maximum of 4 different locations besides its starting and ending location to avoid the long computational time when there are a lot of locations in an instance. $\mathcal{O}$ is all the requests of the instance. Each request has a location_id

which is where the request needs to be delivered, start_day which is when the request can start being delivered, end_day which is when the request needs to have been delivered, machine_id which is what type of machine the request consists from and quantity which is how many machines need to be delivered in the request.$\mathcal{T}$ is all the technicians of the instance. Where each technician has a location_id which is the location of the technician, max_distance_per_day which is the max distance the technician can travel, max_installations_per_day which is the maximum amount of installations the technician can install in a day and machine capabilities which shows what type of machines the technician can install.$\mathcal{H}$ is the set of all possible technician routes. It contains, for each technician, all possible routes with a maximum of 3 different locations (similar to the truck routes) where the starting and ending location is the location of that specific technician. In addition, these possible routes are less than or equal to the max distance that the technician can travel.$\mathcal{S}$ is the set of all possible schedules, where each set is a sequence of increasing days and does not contain any set of 5 consecutive days.$\mathcal{D}$ is the set of days.
The constants and variables [16] in the MIP are:

1. Constant n_{trucks} is the number of trucks that is assigned for the program.
2. Decision variable $x_{d,w,r,o} = 1$ if request o $\in \mathcal{O}$ is delivered on day d $\in \mathcal{D}$ by truck w $\in \mathcal{W}$ using truck route r $\in \mathcal{R}$, and 0 otherwise. It is enforced 0 if the day d is outside the time window in which request o needs to be delivered. In addition, $x_{d,w,r,o}$ is enforced 0 if the location of request o is not in truck route r.
3. Decision variable $y_{d,t,h,o} = 1$ if request o $\in \mathcal{O}$ is installed on day d $\in \mathcal{D}$ by technician t $\in \mathcal{T}$ using technician route h $\in \mathcal{H}$, and 0 otherwise. It is enforced 0 if the location of request o is not in technician route h or if the starting location of request o is not the location id of technician t or if technician t can not install request o.
4. Decision variable $u_{d,w,r} = 1$, if truck route r $\in \mathcal{R}$ is used on day d $\in \mathcal{D}$ by truck w $\in \mathcal{W}$, and 0 otherwise.
5. Decision variable $i_{d,t,h} = 1$, if technician route h $\in \mathcal{H}$ is used on day d $\in \mathcal{D}$ by technician t $\in \mathcal{T}$, and 0 otherwise.
6. Decision variable l_o is the number of idle days for request o $\in \mathcal{O}$, being the number of full integer days that elapse between its delivery and installation.
7. Decision variable $k_{t,s} = 1$, if technician t $\in \mathcal{T}$ uses schedule s $\in \mathcal{S}$, and 0 otherwise.
8. Constant c_o is the cost paid for each idle day for request o $\in \mathcal{O}$.
9. Constant c_h is the cost of using technician route h $\in \mathcal{H}$, which is the cost of using a technician for a day and the cost of the distance the technician has travelled by technician route h.
10. Constant c_r is the cost of using truck route r in $\in \mathcal{R}$, which is the cost of using a truck for a day and the cost of the distance the truck has travelled by truck route r.
11. Constant s_o is the size of request 0 $\in \mathcal{O}$.
12. Constant $b_{s,d} = 1$ if schedule s $\in \mathcal{S}$ contains day d $\in \lceil$, and 0 otherwise.
13. Constant d_r is the distance of truck route r $\in \mathcal{R}$.
14. Constant d_h is the distance of technician route h $\in \mathcal{H}$.
15. Constant q_o is the quantity of request o $\in \mathcal{O}$.
16. Constant c_{truck} is the cost of one truck.
17. Constant $c_{technician}$ is the cost of one technician.

The objective function is:

$$
\begin{aligned}
\min \quad & \sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\sum_{r\in\mathcal{R}} c_r u_{d,w,r} && \text{truck daily and distance costs} \\
& + n_{\text{trucks}} \cdot c_{\text{truck}} && \text{truck fleet size cost} \\
& + \sum_{d\in\mathcal{D}}\sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}} i_{d,t,h} c_h && \text{technician daily and distance costs} \\
& + \sum_{o\in\mathcal{O}} c_o l_o && \text{idle costs} \\
& + \sum_{t\in\mathcal{T}}\sum_{s\in\mathcal{S}} c_{\text{technician}} k_{t,s} && \text{technician work size cost}
\end{aligned}
$$

The constraints [17] for the MIP are:

$$\sum_{r\in\mathcal{R}} d_h u_{d,w,r} \leq maxtruckdistance \qquad \forall w \in \mathcal{W} d \in \mathcal{D} \qquad \text{Maximum truck distance enforced}$$

$$\sum_{o\in\mathcal{O}} q_o s_o x_{d,w,r,o} \leq truckcapacity \qquad \forall w \in \mathcal{W} d \in \mathcal{D} r \in \mathcal{R} \qquad \text{Maximum truck capacity enforced}$$

$$\sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\sum_{r\in\mathcal{R}} x_{d,w,r,o} = 1 \qquad \forall o \in \mathcal{O} \qquad \text{Every request is being delivered:}$$

$$\sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\sum_{r\in\mathcal{R}} x_{d,w,r,o} u_{d,w,r} = 1 \qquad \forall o \in \mathcal{O} \qquad \text{Every request has a truck route:}$$

$$\sum_{d\in\mathcal{D}}\sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}} y_{d,t,h,o} = 1 \qquad \forall o \in \mathcal{O} \qquad \text{Every request is being installed:}$$

$$\sum_{d\in\mathcal{D}}\sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}} y_{d,t,h,o} i_{d,t,h} = 1 \qquad \forall o \in \mathcal{O} \qquad \text{Every request has a technician route:}$$

$$\sum_{h\in\mathcal{H}}\sum_{o\in\mathcal{O}} y_{d,t,h,o} \leq maxinstallations \qquad \forall d \in \mathcal{D} t \in \mathcal{T} \qquad \text{Max amount of installations enforced}$$

$$\sum_{h\in\mathcal{H}} i_{d,t,h} \leq \sum_{s\in\mathcal{S}} b_{s,d} k_{t,s} \qquad \forall d \in \mathcal{D} t \in \mathcal{T} \qquad \text{No 5 consecutive working days}$$

$$\sum_{s\in\mathcal{S}} k_{t,s} \leq 1 \qquad \forall t \in \mathcal{T} \qquad \text{Technician has one schedule}$$

$$\sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\sum_{r\in\mathcal{R}} dx_{d,w,r,o} \leq \sum_{d\in\mathcal{D}}\sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}} dy_{d,t,h,o} \qquad \forall o \in \mathcal{O} \qquad \text{Request is being installed after delivery}$$

$$\sum_{h\in\mathcal{H}} d_h i_{d,t,h} \leq maxdistancetechnician \qquad \forall d \in \mathcal{D} t \in \mathcal{T} \qquad \text{Maximum technician distance enforced}$$

$$l_o = \sum_{o\in\mathcal{O}} \left(\sum_{d\in\mathcal{D}}\sum_{t\in\mathcal{T}}\sum_{h\in\mathcal{H}} dy_{d,t,h,o} - \sum_{d\in\mathcal{D}}\sum_{w\in\mathcal{W}}\sum_{r\in\mathcal{R}} dx_{d,w,r,o}\right) \qquad \text{Calculation of idle days}$$

By running this program on the instances, we get a schedule of all requests where we know on which day the request is delivered, which truck delivers the request when the request is installed and which technician installs the request. With this information, we can calculate the truck distance, number of truck days, number of trucks used, technician distance, number of technician days, number of technicians used, idle machine cost and at last the total cost.

Concerning the complexity of this algorithm, it is $O(n^5)$ since 5 loops are used at line 107 in the code. A loop for technicians, days, technician routes, requests, and machine capabilities of a technician. The runtime of this algorithm depends on the find_routes function in the code, which makes all possible routes of all locations with a maximum of 3 locations in the route. For the instances with 1 to 10 locations, the algorithm is done in 1-5 seconds, but for the instances with 11 or more locations, the algorithm takes around 15-360 seconds. Not only does the number of locations cause that, but the number of technicians causes this long computational time too. As there are more technicians in the instance, more starting locations are thrown in the find_routes function to get all technician routes. As there are more technician routes, more decision variables are created for Gurobi and therefore Gurobi takes longer to find a solution. So the runtime of the algorithm depends mostly on the amount of locations and technicians in the instance.

5 Performance

Table 1: Best found solutions for all 20 instances

Instance	1	2	3	**4**	5
Objective value	256,440	272,325	536,189	**28,720**	13,075
% above best-of-class	0.54%	0.44%	22.72%	**0.00%**	3.98%
Instance	6	7	**8**	9	10
Objective value	313,592	9,799,680	**8,362,470**	152,930	790,015
% above best-of-class	0.11%	12.16%	**0.00%**	3.15%	26.62%
Instance	11	12	13	14	15
Objective value	678,469,700	95,695,694	326,720	1,295,678	93,523,512
% above best-of-class	0.48%	0.54%	7.25%	2.38%	3.50%
Instance	16	17	18	19	20
Objective value	1,178,900	120,237,615	110,228,528	522,014	395,090
% above best-of-class	7.20%	0.12%	4.53%	4.69%	7.98%

6 Result analysis

Considering the model's results when executed for each instance, it is clear that its performance has some variability upon looking at the percentage deviation from the best of the class. To start, the objective values shown in Table 1 have clear instances with outliers. Despite that, the overall results are a significant improvement over the solutions provided by the simple algorithm. A measure of that significance can be found in instance 1, where the objective value was less than half that of the simple algorithm. A pattern that was spread throughout the other instances.

Looking at these instances, they can be grouped into two performance levels, with the near-optimal instances being in **best-of-class** $\leq 5\%$, with the less optimal instances being everything above that. This separation leads to 14 near-optimal solutions and 6 less so. The spread of the instances and their optimum relative to the best-in-class, alongside the two optimality groups, can be seen in Figure 1.

Upon further investigation of Figure 1, it becomes clear that *All instances* has 2 outliers that fall under the *Less optimal* grouping. Those are the solutions for instances 3 and 10. It needs to be kept in consideration that the 600-second time limit acted as a statement of intent, and is not the case for all instances but serves as a reference point. Which could be pointed out as the reason those instances performed less optimally. Some instances are extended to 900 seconds, and others

go on indefinitely. One of those indefinite cases is that of instance 20, due to the size of the requests, technicians and locations, it was deemed not viable to assign a time limit for a good result relative to the rest of the class. What is also noticeable is that the algorithm stays consistent throughout runs, given that the time limit remains unchanged. Increasing that time limit would result in an overall improvement that has to be considered on a case-by-case basis, as the differences between the instances affect the improvement with the number of variables increasing each time.

Looking at the algorithm itself, the most time-consuming aspect is that of the schedule and route creation, forming a bottleneck that may not make it possible for some to execute it. This highlights the importance of shortening the possible route length and limiting the routes, which may reduce the optimality but keep the model functioning at all times. Furthermore, the performance of the model can be considered generally near-optimal for this time limit, with half the instances being close to the optimum. The other half is mostly limited by time, as an extended calculation period would lead to a more optimised result that could then fall under the near-optimal definition. What is noticeable is that the model does not have a specific strength or weakness, going over each instance with the same methodology and providing accurate solutions for some and less optimal solutions for the other. A notable occurrence of this is the difference between the percentage optimality of instances 17 and 18, leading to the notion that the model is relatively optimal but has room for improvement.

7 Conclusion

To conclude, the Gurobi algorithm parses the data provided in the instance file into a format that can then be turned into constraints and variables within the solver of the Gurobi program. It does so by generating the possible routes and schedules both trucks and technicians can have, choosing those that cost the least, and then parsing them as such. The model does not have a clear strength or weakness, handling both the smaller and bigger instances adequately, with outliers in each. Discerning this lack of clarity can be done in Table 1, where even in the same range of objective values, the accuracy can differ significantly. This leads to the primary improvement, developing consistent optimality in the same orders of magnitude alongside the implementation of a route finder that considers edges instead of entire routes as is done now.

8 Roles and responsibilities

All members of the team worked hard and contributed to this project, collaboration, and communication were good and members assisted each other with the tasks at hand if necessary.

9 Appendix

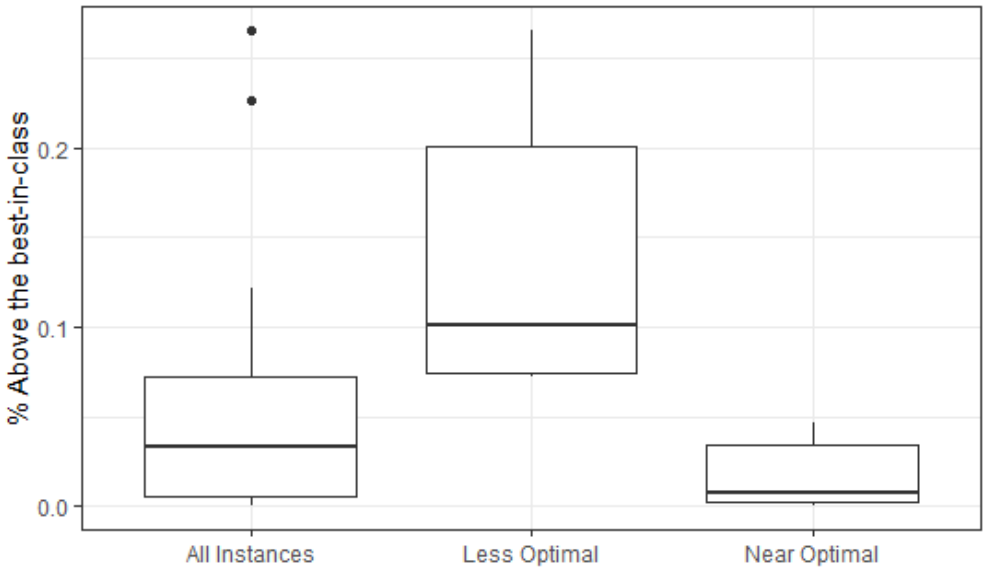

Figure 1: Boxplot van de prestatieniveaus over alle instanties

References

[1] Aug. 2022. URL: https://www.gurobi.com/documentation/10.0/refman/index.html.

[2] S. H. Jameson R. H. Mole. "A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion". In: (1976).

[3] Fadlah Tunnisaki and None Sutarman. "Clarke and Wright Savings Algorithm as Solutions Vehicle Routing Problem with Simultaneous Pickup Delivery (VRPSPD)". In: *Journal of physics. Conference series* 2421.1 (Jan. 2023), p. 012045. DOI: 10.1088/1742-6596/2421/1/012045. URL: https://doi.org/10.1088/1742-6596/2421/1/012045.

[4] J. Desrosies M. M. Solomon. "Time Window Constrained Routing and Scheduling Problems". In: (1998). URL: https://pubsonline.informs.org/doi/epdf/10.1287/trsc.22.1.1.

[5] O. B.G. Madsen M. M. Solomon B. Kallehauge J. Larsen. *Vehicle Routing Problem with Time Windows*. 2005.

[6] Solomon M. M. "Algorithms for the Vehicle Routing and Scheduling Problems With Time Window Constraints". In: (1985).

[7] A. H. G. Rinnooy Kan A. W. J. Kolen and H. W. J. M. Trienekens. "Vehicle Routing With Time Windows". In: (1986).

[8] J. E. Beasley N. Christofides. "The period routing problem". In: (1984).

[9] R. Rusell W. Igo. "An assignment routing problem". In: (1979).

[10] V. Praveen et al. "Vehicle routing optimization problem: A study on capacitated vehicle routing problem". In: *Materials today: proceedings* 64 (Jan. 2022), pp. 670–674. DOI: 10.1016/j.matpr.2022.05.185. URL: https://www.sciencedirect.com/science/article/abs/pii/S2214785322034654.

[11] C. H. Papadimitriou S. Dasgupta and U. V. Vazirani. *Algorithms*. 2006.

[12] J. Medak. "Review and Analysis of Minimum Spanning Tree Using Prim's Algorithm". In: (2018).

[13] H. J. Greenberg. "Greedy Algorithms for Minimum Spanning Tree". In: (1998).

[14] Benjamin Graf. "Adaptive large variable neighborhood search for a multiperiod vehicle and technician routing problem". In: *Networks (New York, N.Y. Online)/Networks* 76.2 (June 2020), pp. 256–272. DOI: 10.1002/net.21959. URL: https://doi.org/10.1002/net.21959.

[15] Caroline J. Jagtenberg et al. "Columnwise neighborhood search: A novel set partitioning matheuristic and its application to the VeRoLog Solver Challenge 2019". In: *Networks (New York, N.Y. Online)/Networks* 76.2 (June 2020), pp. 273–293. DOI: 10.1002/net.21961. URL: https://doi.org/10.1002/net.21961.

[16] Aug. 2022. URL: https://www.gurobi.com/documentation/10.0/refman/variables.html.

[17] Aug. 2022. URL: https://www.gurobi.com/documentation/10.0/refman/constraints.html.