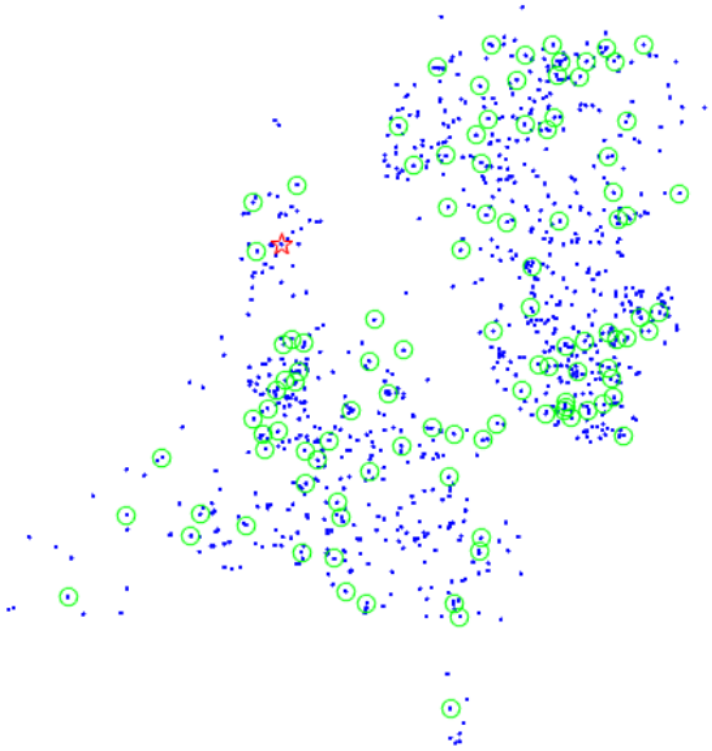
Operations Research Case: VeRoLog 2017



Sara van Bezooijen (2681467), Kim van den Houten (2548011), Finnbar van Lennep (2607436)

Team 10

Vrije Universiteit Amsterdam

January 27, 2021

1 Introduction

In this report we will present our results to an $\mathcal{NP}$-hard optimisation problem, namely the VeRoLog Solver Challenge 2016-2017. The challenge is a simplification of a real-life problem encountered by a cattle improvement company. We assume the reader is already acquainted with the challenge, rather than reiterating the full problem description we refer to [3].
The main objective is to minimise the cost while serving all requests. There are four different types of costs, namely: costs per distance traveled, costs for using a vehicle for a day, costs for using a vehicle at all and costs for each tool type. The complexity of the problem lies in finding an algorithm that is able to minimise the cost for a variety of problem instances, where each problem instance emphasises a different type of cost [2].
This type of vehicle routing problem has not been studied extensively in the literature. Nonetheless, since the end of the challenge a few papers have been published on this topic [1] [2]. We identified two main ways of approaching the problem. Both approaches made a distinction between scheduling and routing, the difference of the two methods lies within the order of solving the problem. Since the approach that first solves the scheduling part and then solves the routing part yielded a solution of higher quality [2], we used this approach as the basis for our algorithm.
For the scheduling and routing part we constructed multiple algorithms, varying in complexity. For the scheduling part we defined: an earliest delivery algorithm, a greedy algorithm and an integer linear program. For the routing part we experimented with the following algorithms: one vehicle per request algorithm, Next Fit algorithm, Best Fit cost algorithm and a Best Fit tools algorithm. With these algorithms we could make different combinations. Regardless of the combination chosen, the final stage in our algorithm was the post-optimisation stage. In this stage the routes found were further improved by randomly selecting improvement heuristics.

2 Methodology

This section outlines our methodology. As mentioned before, scheduling first and routing second, generally, finds a better solution than vice versa. Therefore, we follow this principle. In section 2.1 we clarify the data used and the different types of instances. In section 2.2 three different scheduling algorithms are presented. In section 2.3 four different routing algorithms and their pseudo code are given. We combine these algorithms in our analysis such that we have twelve different scheduling and routing combinations. The final step in our methods involves improving the routes found with various improvement heuristics. This improvement method is called Post-Optimisation and is illustrated and explained in section 2.4.

2.1 Data

For analysis we used the 25 early VeRoLog instances that are provided on the ORTEC website. The challenge distinguishes between different types of instances. The type of instance is specified in the filename, which is of the form *VeRoLog_r100d5_1*. The r in the instance name stands for the number of requests. The d stands for the number of days. The final number indicates the instance cost type. Per cost type the weights of the costs are ordered as follows:

1. tools > vehicles > vehicle days > distance
2. tools > vehicle days > vehicles > distance
3. vehicles > tools > vehicle days > distance
4. vehicle days > tools > vehicles > distance
5. tools > distance > vehicle days > vehicles

For example, when minimising the cost for an instance of type 3 one should focus on minimising the number of vehicles and the tool use rather than minimising the distance. In the problem description and data format file [4] are all input variables explained.

2.2 Scheduling

In this section we exemplify the three different scheduling algorithms constructed, for which the code can be found in our GitHub [6] repository in the file *CreateSchedule.py*.

S1: Earliest delivery

The first method schedules each delivery on the earliest delivery date allowed by the request. Since the duration of the request was given, the pick-up dates were scheduled accordingly.

S2: Greedy algorithm

The greedy algorithm looped through the requests and assigned a delivery day to each request without violating the time window constraint of the request. Per tool type, the algorithm schedules the requests on the day that that type of tool was used the least. For example, if a tool of type 1 is already in use by farm *A* until day 3, the next tool of type 1 will be delivered to farm *B* on day 4 to minimise the tools used per day.

S3: Integer linear programming

The previous scheduling algorithms did not take into account the given constraint on tool use and therefore might not always create feasible schedules. Therefore, we extend our methodology with an integer linear program (ILP) to increase the likelihood of finding a feasible route. For each tool type we constructed an ILP that minimises the number of tools used. In this way we increased the likelihood of finding a feasible routing, since not all tools needed to be reused on the same day to still result in a feasible amount of tools. We solved the ILPs with the Gurobi solver [5].

ILP formulation: Let I_t denote the set of requests where tool type t is requested. And let D denote the maximum number of days for the schedule. Parameters f_i and t_i denote the day from which and to which request i can be delivered respectively. The period length for which the tools from request i will be in use is denoted by p_i. The last parameter n_i denotes the number of tools needed for request i. We formulate the model for tool type t as follows:

min	s_0			objective
s.t.	d_i	$\geq f_i$	$i \in I_t$	(1)
	d_i	$\leq t_i$	$i \in I_t$	(2)
	$\sum_{j=1}^{D} x_{ij}$	$= 1$	$i \in I_t$	(3)
	$\sum_{j=1}^{D} j \cdot x_{ij}$	$= d_i$	$i \in I_t$	(4)
	$\sum_{j=1}^{D} y_{ij}$	$= 1$	$i \in I_t$	(5)
	$\sum_{j=1}^{D} j \cdot y_{ij}$	$= d_i + p_i$	$i \in I_t$	(6)
	$s_{j-1} - \sum_{i \in I} n_i(x_{ij} - y_{ij})$	$= s_j$	$j \in \{1, 2, .., D\}$	(7)
	$x_{ij}, y_{ij} \in \{0, 1\}$			
	$s_j \geq 0$			
	All variables are integers			

Where the following decision variables were used to optimise model:

$$
\begin{aligned}
d_i &= \text{day of delivery for request } i && i \in I_t \\
s_j &= \text{number of tools in depot at the end of the day} j && j \in \{0, 1, 2, .., D\} \\
x_{ij} &= \begin{cases} 1, & \text{if request } i \text{ will be delivered on day } j \\ 0, & \text{otherwise} \end{cases} \\
& (i, j) \in I_t \times \{1, 2, .., D\} \\
y_{ij} &= \begin{cases} 1, & \text{if request } i \text{ will be picked-up on day } j \\ 0, & \text{otherwise} \end{cases} \\
& (i, j) \in I_t \times \{1, 2, .., D\}
\end{aligned}
$$

The objective function minimises the number of tools of type t needed for the schedule. Constraints (1) and (2) make sure that every request is delivered within the given time period. To make sure every request is delivered and picked up on exactly one day we introduce constraints (3) and (5) respectively. To translate the delivery day to a subscript we use constraints (4), this makes sure that $x_{ij} = 1$ when request i is delivered on day j and $x_{ij} = 0$ otherwise. A similar translation is done for pick-up day in constraints (6). The last constraints calculate the number of tools at the end of each day, this is the amount of tools at the depot at the and of the previous day s_{j-1} minus the number of tools delivered that day $\sum_{i \in I} n_i x_{ij}$ plus the number of tools picked up from farms that day $\sum_{i \in I} n_i y_{ij}$. These constraints expects picked-up tools to be delivered to a different farm immediately if another delivery is scheduled on that day. As all stock must be non-negative, s_0 denotes the minimum number of tools needed at the start of the schedule to have enough tools available for every day.

2.3 Routing

For the routing part of the VeRoLog challenge four different routing algorithms were used, ranging from the simplest algorithm we used at the start of the research to the most advanced algorithm we optimised at the end of the research. The implementation of the algorithms can be found in our GitHub [6] repository in the *CreateRoutes.py* file in the functions specified in the pseudo code added in the appendix.

R1: One vehicle per request

The simplest algorithm, implemented at the start of our research, assigned a new vehicle to each request (pick up or delivery). The implementation of this algorithm was easy, especially since there was no need to check the vehicle constraints, i.e. the travelled distance and the weight capacity constraint, when using this algorithm. The pseudo code is added to (A.1) of the appendix.

R2: Next Fit algorithm

Second, we experimented with the Next Fit algorithm. The Next Fit algorithm adds items to a bin or truck until the truck is filled. Thus, when the next to be added item does not fit in the truck anymore, a new truck is opened and the item is loaded onto the new truck. This algorithm does not minimise the distance traveled by the trucks, it only minimises the number of trucks used. Because our program deals with pick ups as well as deliveries, our algorithm was a little more complex than the basic Next Fit algorithm described above. First a truck was filled with as many deliveries as possible according to Next Fit, then the truck was filled with pickups until either the capacity of distance constraint was met. If the distance constrained was met, a new truck was opened, if the weight capacity constraint was met, the truck was returned to the depot and loaded with deliveries again. If there were no deliveries on a given day, the trucks were scheduled with pick ups according to the Next Fit algorithm. The pseudo code is added to appendix A.1.

R3&R4: Best Fit algorithm

The final and most advanced routing algorithms used in the research are algorithms based on Best Fit bin packing. The algorithm decided where on the route of which vehicle to add the new request. It added the request at the location

with the best fit. We made two different rules for best fit. The first (R3) putted the request in the tour of a vehicle where the least tools were taken from the depot, thus minimising the number of tools used for that day. When there were different spots with the same tool use, than the request was scheduled where it added the least kilometers. The second (R4) calculated the cost of the kilometers and cost for tools taken from the depot before adding the new request and after adding the new request. The request was scheduled in a tour where the least cost was added to the existing tour. Both algorithms started scheduling with the heaviest tools and then scheduled pick-ups before deliveries. The pseudo code is added to appendix A.1.

After applying these algorithms we only had single depot tours, i.e. the vehicle only visits the depot at the start and end of its tour. We tried to optimise the number of vehicles by trying to combine the single depot tours into larger tours where the vehicle might visit the depot as an intermediate stop. For each tour was checked whether it could be added to any other tour. We first tried if it is possible to add the tour to the other without a depot visit in between, this might not be possible due to weight restrictions. Then we tried with a depot visit in between.

2.4 Post-Optimisation

In the final stage of our algorithm, the routes found were further improved by randomly selecting improvement heuristics for m iterations. Analysing the results from previous VeRoLog competitors we observed that using various improvement heuristics has the potential of improving the final solution. Moreover, it was observed that both moving and swapping single events in routes or between routes and moving and swapping blocks between routes could potentially lead to better final solution. Where a block was defined as set of consecutive visits, including depot visits [2]. Inspired by these approaches, we did choose for the following improvement heuristic categories. The categories are also visualised in figure 1:

(a) *Single move, one vehicle:* Finds a random vehicle and route within the same day and moves a pick-up or delivery to another position within the same route.

(b) *Single swap, one vehicle:* Finds a random vehicle and route within the same day and swaps a pick-up or delivery with another pick-up or delivery.

(c) *Single move, two vehicles:* Finds a random delivery or pick-up and moves this request to another vehicle within the same day on a random position of the corresponding route.

(d) *Single swap, two vehicle:* Finds two random vehicles and routes within the same day and swaps either a pick-up or delivery with another pick-up or delivery.

(e) *Block move, two vehicles:* Finds a random block of deliveries or pick-ups and move it to a random position on the route of another vehicle. Also implemented for one vehicle.

(f) *Block swap, two vehicles:* Find two random blocks of deliveries or pick-ups, both belonging to another vehicle, and swap the two blocks. Also implemented for one vehicle.

Implementation

The implementation of the post-optimisation can be found in the GitHub [6] map Code under the name *PostOptimisation.py*. The code was build up very generic in the sense that all the move and swap types that are described above were implemented in one single function. The user can indicate with the help of boolean arguments which type he wants to run or can choose to switch between the different types randomly per iteration. For the one vehicle move and swaps, one must set onlySameRoute to true. If the user wishes to only apply moves (no swaps), one must set onlyMove to true. Lastly, if you don't want to move blocks but only single pick-ups or deliveries, one must set maxOne to true. If all are false, everything is randomised. It will be a move if one block length happens to be 0, and it will be within one vehicle if the same vehicle is chosen twice.
The pseudocode is added to appendix A.1.

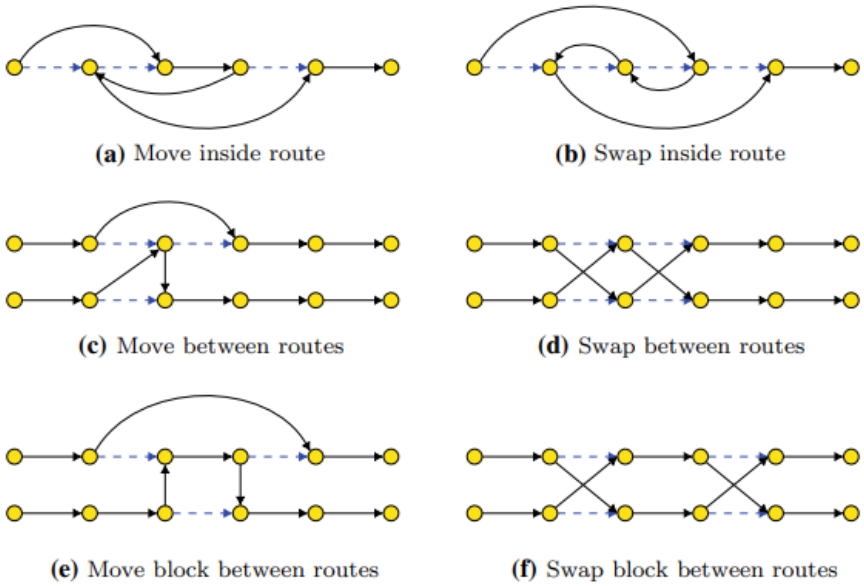


Figure 1: Move and swaps, source: [2]

2.5 Techniques

All the coding and programming in our research was done in Python 3.8. We used some code from the VeRoLog Solution checker [3] to help us read the input data, check constraints and feasibility, create a class that cleverly keeps track of the vehicles and routes, and write our solution to the correct output format.

3 Results

In this section we analyse the results of our different approaches on the 25 early VeRoLog instances. As presented in the methodology, we distinguish 3 scheduling algorithms and 4 routing algorithms. In our analysis we compare all combinations, thereafter we will discuss the effect of post-optimisation. In subsection 3.1 we present one instance and compare the different combinations based on the corresponding output values. We are not looking very specifically at the costs here, but rather at the variables tool use, distance, etc., to get a good picture of how the different algorithms influence these variables. This can help us to get a sense of which algorithms work well with which instance types, because each type has its own cost weights. Then, in subsection 3.2 we summarise for which combinations we obtain feasible solutions. We make a distinction between feasibility that is obtained with post-optimisation and without post-optimisation. We continue in subsection 3.3 with a more general analysis of the performance of the different scheduling and routing combinations and with the help of a ranking system we determine for each instance type the most suitable combination. Using this assignment of algorithms to instance types, we finalise our Results section in 3.4 with a more-depth analysis of the effect of post-optimisation on the final cost. Also, a comparison of our results with the best known solution and our ranking within the challenge if we would have participated is added to appendix A.2.

3.1 Test instance

To see the effect of the different types of routing and scheduling on the variables that determine the costs (excluding the cost itself), we have chosen to compare all combinations of algorithms for one instance. We have left the post-optimisation out of consideration here. *VeRoLog_r100d5_1* is used for the analysis. All combinations that resulted in a infeasible solution were infeasible because they exceeded the maximum tool use. For this instance the available tools per type were [54, 60, 52].
Table 1 shows the results for all possible routing and scheduling combinations. Using either the earliest delivery day scheduling method (s1) or the greedy scheduling method (s2), it was impossible to find a schedule that satisfied the tool use constraint. Still, the total tool use was significantly lower for s2 than for s1. Using the ILP method (s3) we

Table 1: Results for *VeRoLog_r100d5_1*

Scheduling and routing	Feasibility	Distance	Tool use	Total tool use	Vehicle days	Max number of vehicles
s1 - r1	Infeasible	1340048	[64, 67 61]	192	200	78
s1 - r2	Infeasible	1084589	[61, 62 61]	184	112	47
s1 - r3	Infeasible	801139	[57, 61 55]	173	53	21
s1 - r4	Infeasible	792962	[57, 60 54]	171	54	23
s2 - r1	Infeasible	1340048	[59, 63 57]	179	200	76
s2 - r2	Infeasible	1071896	[57, 62 57]	176	110	47
s2 - r3	Infeasible	776600	[54, 56 56]	166	51	22
s2 - r4	Infeasible	780387	[54, 56 54]	164	51	22
s3 - r1	Feasible	1340048	[52, 54 50]	156	200	53
s3 - r2	Feasible	1063285	[49, 54 50]	152	104	35
s3 - r3	Feasible	753528	[38, 45 42]	125	48	17
s3 - r4	Feasible	**741481**	**[38, 46 40]**	**124**	**47**	**16**

succeeded to find a feasible solution. Pinning the routing method to r1 and comparing the difference in total tool use between s1 and s3, we observed the significant difference of $192 - 156 = 36$ tools. This suggested that for instances where tools are expensive, it is always wise to prefer the more complex algorithms. The more advanced schedules also resulted in lower total distance. Again fixing the routing method to r2, r3 or r4 and alternating the scheduling method we observed that the distance decreased if we switched from earliest delivery (s1) to greedy (s2) and also if we switched from greedy to ILP (s3). However, the effect on the distance did not apply if we applied r1, because with this method we simply used a separate vehicle for each pick - up and delivery listed in the schedule. This is also clearly reflected in the fact that for every combination in which r1 is used, the number of vehicle days is 200 (which is equal to 2 times the number of requests). We observed that regardless of the scheduling algorithm we used, we managed to almost halve the number of vehicle days with a simple routing improvement in the form of bin-packing (r2). If we then replaced these routing algorithms with local search (both r3 and r4), the number of days did halve again. A similar effect was shown for the maximum number of vehicles counted in the schedule. Therefore, if one tries to minimise the cost for an instance with relatively high cost for the vehicle days or the vehicle rent one should consider to use algorithm r3 or r4 .The combination that generated the best overall solution is scheduling with and ILP (s3) and routing with a local search algorithm that focuses on distance cost (r4). This algorithm performs best with respect to all variables mentioned in table 1. The cost of the solution that we obtained for this combination was $1,624,082,481$. The best known solution thus far is $1,552,435,049$ (based on the ORTEC Ranking). This is a difference of $71,647,369$ where we left out the possible effect of post-optimisation.

3.2 Feasibility

As the first aim was to find a feasible solution for all instances, we have included Table 2 that summarises the feasible and infeasible outcomes of all instances using the different combinations of the scheduling and routing algorithms. Not all combinations of scheduling and routing directly resulted in feasible solutions for all instances as is shown in Table 2 with red or orange. Remarkably, post optimisation achieved feasibility for some instances that were not feasible initially. The post-optimisation was executed for 150.000 iterations. All instances for which we were able to find a feasible solution only with the help of post-optimisation are colored orange in Table 2. The post-optimisation was more successful on the instances with a somewhat longer planning horizon, which can be deduced from the number of the ratio red/orange cells at the lower part of the table (more orange, longer planning horizon) and the upper part of the table (more red, shorter planning horizon). It is remarkable that *r1000d30 type 1* still has a lot of infeasible final solutions, also after post-optimisation. Analysing this instance in more depth it stands out that the number of available tools is quite tight and therefore a more advanced algorithm is necessary to achieve feasibility for tool use.

Table 2: Feasibility of different algorithms without post optimisation

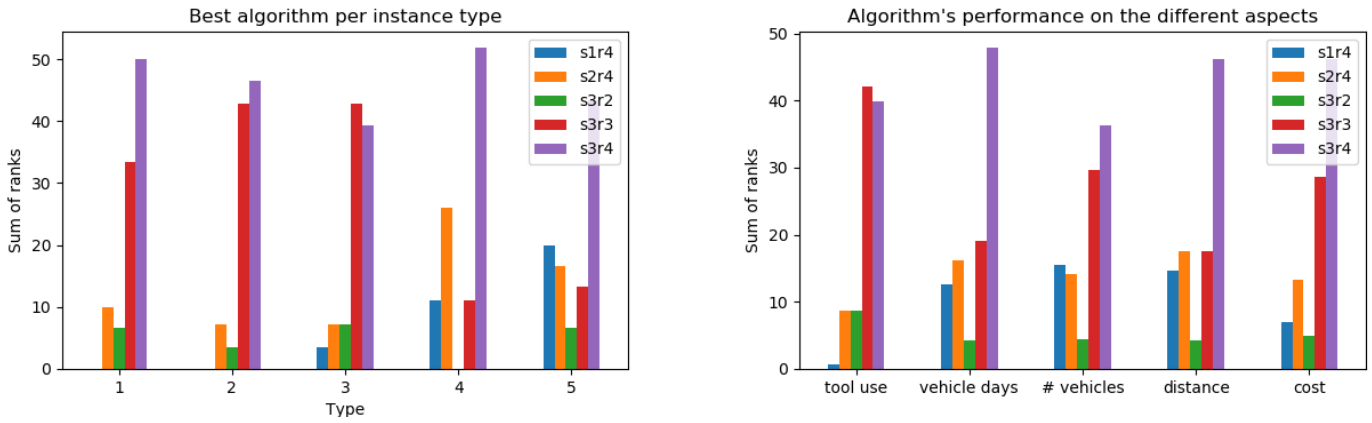
		s1	s1	s1	s1	s2	s2	s2	s2	s3	s3	s3	s3
		r1	r2	r3	r4	r1	r2	r3	r4	r1	r2	r3	r4
r100d5	1	x	x	x	x	x	-	-	-	v	v	v	v
r100d5	2	x	x	x	x	x	x	x	x	-	-	v	v
r100d5	3	x	x	x	x	x	x	x	x	v	v	v	v
r100d5	4	x	x	x	x	x	x	x	x	-	-	v	v
r100d5	5	x	x	x	x	x	x	x	x	-	v	v	v
r100d10	1	-	x	-	v	-	-	v	v	-	v	v	v
r100d10	2	x	x	x	x	-	-	x	-	v	v	v	v
r100d10	3	x	x	x	x	x	x	x	x	-	v	v	v
r100d10	4	-	-	-	-	v	v	v	v	v	v	v	v
r100d10	5	x	x	x	x	x	x	x	x	v	v	v	v
r500d15	1	-	-	v	v	-	-	v	v	v	v	v	v
r500d15	2	-	-	v	v	-	-	v	v	-	v	v	v
r500d15	3	-	-	v	v	-	-	v	v	v	v	v	v
r500d15	4	-	-	-	-	-	-	v	v	v	v	v	v
r500d15	5	-	-	v	v	-	v	v	v	v	v	v	v
r1000d25	1	-	-	v	v	-	-	v	v	v	v	v	v
r1000d25	2	x	x	v	v	-	-	v	v	-	-	v	v
r1000d25	3	x	x	x	x	-	-	-	-	-	-	v	v
r1000d25	4	-	-	v	v	-	-	v	v	v	v	v	v
r1000d25	5	-	-	v	v	-	-	v	v	v	v	v	v
r1000d30	1	x	x	x	x	x	x	x	v	-	v	v	v
r1000d30	2	-	x	v	v	-	-	v	v	v	v	v	v
r1000d30	3	x	x	x	x	-	-	v	-	v	v	v	v
r1000d30	4	-	-	-	v	-	-	v	v	-	v	v	v
r1000d30	5	x	x	v	v	-	x	v	v	v	v	v	v

3.3 Analysis of Scheduling and Routing Algorithms

For analysis of the different algorithm combinations, the algorithm combinations were ranked based on their performance. First, the combinations were only ranked with respect to the cost. The five combinations that achieved the lowest final cost according to this initial ranking were chosen for further analysis, regarding tool use, vehicle days, number of vehicles and the distance. For each instance, the combinations were ranked on these performance metrics and the top three combinations were accredited 3, 2 and 1 points, in order of decreasing rank. Summing over all instances gives the results revealed in figure 2b. Figure 2a shows the sum of the rankings with respect to the cost, grouped by the instance type. In the above mentioned figures (2a & 2b), we see that combination s3r4 is clearly the highest ranking algorithm combination.

The scheduling (S) and routing (R) combinations are numbered from simple to advanced (low to high). Therefor it is expected that s1r1 achieves the lowest results and s3r4 the best. Figure 2b shows as expected that averaged over all instances s3r4 achieves the lowest final cost of all combinations. Furthermore, it is interesting to note that r3 mainly emphasises on minimising the tool use, but s3r3 only has a slight advantage over s3r4 in this area.

In figure 2a we see that the s3r4 combination ranks the highest for every type of instance except for type 3, where the s3r3 combination scores slightly higher than the s3r4 combination. The only area where s3r3 outperforms s3r4 is tool use, in all the other areas s3r4 outperforms the other combinations (see figure 2b). Regarding the number of vehicles, s3r3 ranks nearly as high as s3r4. Therefore it is expected that for type 3 instances minimising the tool use and number of vehicles is important, because s3r3 scores better on type 3 instances than s3r4 does. As explained in section 2.1 for type 3 instances the vehicle cost is the highest followed by the tool costs. However, s3r3 does not



(a) (b)

Figure 2: *The bar charts show the performance of the 5 highest ranking algorithms. 2b shows the the performance over all instances on different aspect of the problem, and 2a shows the the performance with respect to the cost grouped by instance type. In the legend the color assignment of the algorithms is specified .*

perform as well as s3r4 for type 1 instances, even though tool costs are the highest followed by the vehicle cost.
To summarise the results shown in figure 2a, for instances of type 3 algorithm combination s3r3 achieves the best results, and for the other instances of type 1, 2, 4 and 5, s3r4 achieves the best results.

3.4 Analysis of Post-Optimisation

In subsection 3.2 we have already seen that post-optimisation can in some cases ensure that an initially infeasible solution can be turned into a feasible solution by applying random swaps and moves. But also and mainly, the random swaps and moves can reduce costs and optimise the solution. To analyse this hypothesis, we started with initial solutions for each VeRoLog instance. These initial solutions were obtained by applying the algorithm that was assigned as most suitable for that type of instance (the precise assignment can be found in 3.3). The associated cost values are what we call the values without post optimisation. In order to be able to compare between different instances with different starting values for the costs, we decided to make a translation into percentages. The initial cost value that was obtained without post-optimisation was set to 100%. This way, we could easily visualise and compare the relative reductions in costs caused by post optimisation. In figure 3 the results are presented. To average out the random effect, we averaged over 15 different seeds for each instance. Each sub figure contains the relative decrease in cost of one instance type - different types have different cost weights - for the different instance sizes - different instances have a different planning horizon and a different number of requests. The best post-optimisation result was achieved by instances with label *r100d5* and *r100d10*. These are both instances with a small planning horizon (5 and 10 days) and also a relatively small number of requests (both 100), compared to the other instances. So the effect of moves and swaps was relatively more beneficial for the smaller instances. It could be the case that with more iterations the effect also increases with the larger instances. The relative cost decrease was the highest for the instances of type 5. This can be explained by the fact that moving and swapping mainly has a positive effect on the distance the vehicles have to travel. For instances of type 5, the distance costs are relatively high. Therefore it is beneficial to minimise the distance with the help of post-optimisation. For the other instance types, distance is less important and while the distance is minimised the tool usage may increase, which is heavily penalised with high tool costs. To conclude, it must be noted that for all combinations of scheduling and routing algorithms it is beneficial to perform post-optimisation after execution of these algorithms to further reduce costs.

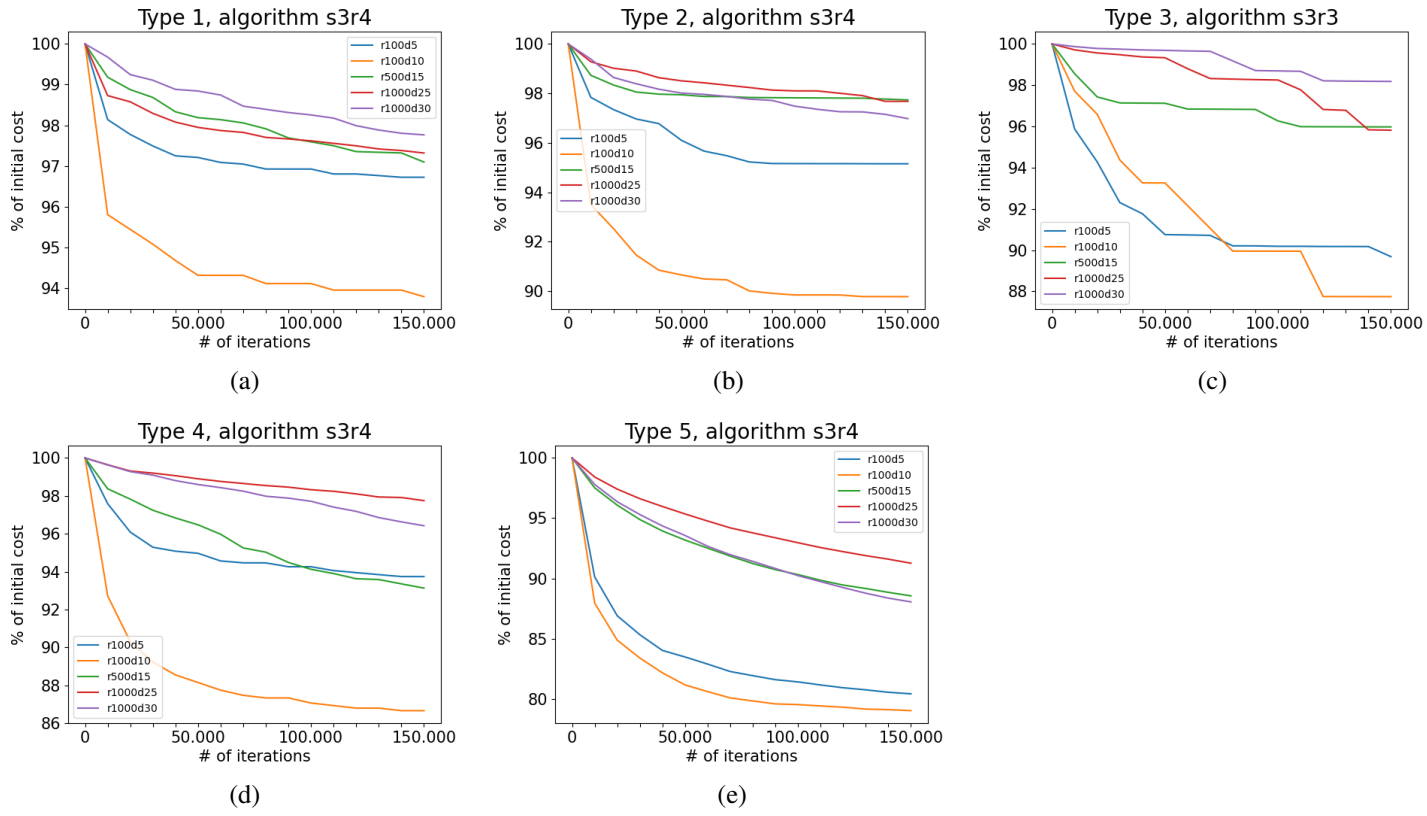


Figure 3: The effect of post-optimisation on cost for different instance types

4 Conclusion and Discussion

Summarising the results, we observed that the ILP method (s3) performed best on all variables that influence the solution cost and that it is therefore always preferable to use this scheduling method. In our research we used 25 VeRoLog test instances for which we could all find a feasible solution. The best performing routing algorithm depended on the instance type and was selected using a ranking method. For type 1, 2, 4 and 5 the best fit algorithm r4 performed best, while for type 3 r3 was assigned as best performing routing algorithm. The reason why certain algorithms worked better for certain instances could be traced back to either the nature of the algorithm or the characteristics of the instance type. Furthermore, we analysed the effect of post-optimisation and were positively surprised with the results. The application of post-optimisation resulted in a significant decrease in the cost of up to 20%, especially for the smaller instances (which have a small planning horizon and a low number of requests). In several cases an initially infeasible solution even became feasible after post-optimisation.

Due to time limitation are research was limited and it was not possible to implement all our concepts. If time would have allowed it we would have applied a local search algorithm to the single routes after the post optimisation. We specifically think this would improve the performance because the post-optimisations swaps requests between routs that have initially been optimised by the best fit algorithm, but after requests are swapped and moved between routes, the resulting routes might not be optimal anymore and could be optimised by a local search algorithm. If this succeeds, we could continue to try to optimise our solution by running the post-optimisation and the local search multiple times after one another.

The post-optimisation algorithm by it self can also be improved by allowing for swaps between days. We did not implement this right away because it makes for more complicated feasibility checks. For instance, deliveries and pick ups of the same request are strictly connected because the pick up has to be completed the day after the client has finished using the tool. Thus, moving the delivery or pick up to another day will always be infeasible unless they are moved correctly together. This, among others, makes between day swapping a little more complicated. Nonetheless, we believe between day swapping will be very effective since our program only optimises tool us between days, therefore between day swapping could cause for significant improvements in other areas like route distance and the number of vehicles used.

5 Collaboration

The team consisting of four people worked predominantly online due to the pandemic. Some members who were located near each other were able to collaborate often in real life and preferably did so. Once a week some members of the team did try to work at the VU on questions that were not easily resolved through online means of communication. Every workday the team did try to meet online via Zoom to discuss e.g. the plan for the day, results and other relevant topics to keep going forward. In general the team tried to divide the tasks when possible, however, not all tasks are easily divisible. For example working on code together can be challenging when communication is limited. GitHub served as a perfect outcome to share the code between team members and combine these parts into an properly functioning complete code. Working in this way allowed the team members e.g. to work in their own branch on new parts of the code they want to modify and to share new code quickly and accurately with all team members and easily find the specific parts of the code that were modified by others and so on. The main idea behind our algorithms was a combination of different ideas tested, adjusted and revised throughout the project. Some parts were taken from the VeRoLog assignment, some parts were directly inspired by the literature, while other parts started out as experiments and out of curiosity. Via thorough discussion these parts were combined into multiple algorithms that were of interest for the whole team. To finalise, the paper was written in overleaf where each person could read, contribute and revise the document.

References

[1] Gromicho, J. A., Haneyah, S., & Kok, L. (2015). Solving a real-life vrp with inter-route and intra-route challenges. *Available at SSRN 2610549*.

[2] Kheiri, A., Dragomir, A. G., Mueller, D., Gromicho, J., Jagtenberg, C., & van Hoorn, J. J. (2019). Tackling a VRP challenge to redistribute scarce equipment within time windows using metaheuristic algorithms. *EURO Journal on Transportation and Logistics, 8*(5), 561-595.

[3] Verolog challenge 2016-2017. https://verolog2017.ortec.com/

[4] The VeRoLog Solver Challenge 2017 *Information for participants* https://verolog2017.ortec.com/pdf/Challenge_problem.pdf

[5] Gurobi https://www.gurobi.com/

[6] Github Repository https://github.com/SaravB/VU-ORcase-Team10

A Appendix

A.1 Pseudo code

r1: One Vehicle per Day

```
def make_day_routes_simplest():
  for each day:
    for each request that day:
      vehicle_route = [0, request, 0]
```

r2: Next Fit

```
def make_day_routes_simple():
  for each day:
      for each request that day:

          # Check distance constraint
          if travelled_dist + dist_to_request + dist_request_depot > Max:
              close old truck
              open new truck

          # Check weight capacity constraint
          elif loaded weight + request_weight > weight_capacity:
              if request is delivery
                  start pickups
              truck returns to depot and starts new tour

          # If both checks passed add item to truck
          else:
              add request to truck
```

r3 & r4: Best Fit

```
def newtourfit_r3():
    tools taken from depot
    added distance

def newtourfit_r4():
    cost for

def BestFit():
    for each day:
        for each tooltype from heavy to light:
            for each request that day of that tooltype:
                for each truck:
                    for each place in tour of truck:
                        calculatenewtourfit()
                        if bestfit > newtourfit:
                            bestfit = newtourfit

                if bestfit is empty:
                    # the request did not fit in any tour
                    new truck
                else:
                    replace old tour with bestfit
```

Post-Optimisation

```
"""
Arguments:
    - nr iterations
    - solution # initial solution
    - onlySameRoute # boolean
    - onlyMove # boolean
    - maxOne # boolean
"""

for i in range(iterations):

    # Find random day
    day = random.choice(all_days)

    # List vehicles on chosen days
    vehicles = all_days[day].Vehicles

    # Find 2 random blocks, ensure that there is no overlap
    while Candidate is None:

        # Choose two random vehicles, choose one for one vehicle moves/swaps
        vehicle1 = random.choice(vehicles)
        vehicle2 = vehicle1 if onlySameRoute is True else random.choice(vehicles)

        # Choose two random indices from each vehicle
        route1 = vehicle1.Route # eg. [0, 1, 2, 3, -4 , -5]
        route2 = vehicle2.Route
        index1 = random.choice(routeLength1)
        index2 = random.choice(routeLength2)

        # Choose random block length
        if maxOne is True:
            blockLength1 = 1
            blockLength2 = random.choice([0,1])

        else:
            blockLength1 = random.choice(routeLength1 - index1)
            blockLength2 = random.choice(routeLength2 - index2)

        if onlyMove is True:
            blockLength 2 = 0

        block1 = route1[index1:index1+blockLength1] # eg. [2, 3]
        block2 = route1[index2:index2+blockLength2]

        # Check overlap between block1 and block2
        if overlap is False:
            candidateVehicle1 = Vehicle(route1)
            candidateVehicle2 = Vehicle(route2)
        else:
            continue

    # Check feasibility of new vehicles
    if candidateVehicle1 is feasible and candidateVehicle2 is feasible:

        # Only apply swap/move if new cost are lower
        if  newCost < oldCost:

            # Update step
            solution = newSolution
```

A.2 Results

VeRoLog Challenge comparison

Table 3 provides an overview of the results obtained for various instances, where we schedule with ILP (s3), route with Best Fit tools (r3) for file type 3 and Best Fit cost (r4) for all other file types. We finalise with 150.000 iterations for post-optimisation.

Table 3: A comparison of our results, the best known solution and our ranking within the challenge if we would have participated

Instance	Best solution	Team 10	Ranking Team 10
VeRoLog_r100d5_1	1,552,435,049	1,603,915,759	14
VeRoLog_r100d5_2	996,709,544	1,031,198,198	13
VeRoLog_r100d5_3	119,957,689	261,516,776	20
VeRoLog_r100d5_4	1,359,088,350	1,868,552,082	14
VeRoLog_r100d5_5	300,125,049,016	333,141,557,021	9
VeRoLog_r100d10_1	1,313,786,538	1,524,827,199	8
VeRoLog_r100d10_2	1,555,438,898	1,857,278,966	11
VeRoLog_r100d10_3	155,316,178	205,845,083	7
VeRoLog_r100d10_4	1,114,888,217	1,790,627,398	13
VeRoLog_r100d10_5	43,871,262,004	58,818,692,007	10
VeRoLog_r500d15_1	3,256,089,143	4,107,403,149	9
VeRoLog_r500d15_2	3,800,352,751	4,231,552,022	8
VeRoLog_r500d15_3	402,839,699	1,156,050,163	15
VeRoLog_r500d15_4	2,807,990,462	5,608,777,302	13
VeRoLog_r500d15_5	251,378,880,010	420,809,045,015	13
VeRoLog_r1000d25_1	7,004,087,706	8,405,601,012	9
VeRoLog_r1000d25_2	6,486,405,100	8,279,609,129	9
VeRoLog_r1000d25_3	207,087,083	422,411,340	12
VeRoLog_r1000d25_4	5,598,405,178	11,652,070,262	11
VeRoLog_r1000d25_5	161,446,120,006	318,369,852,011	13
VeRoLog_r1000d30_1	5,220,068,560	6,693,929,017	9
VeRoLog_r1000d30_2	5,181,409,255	6,896,468,929	9
VeRoLog_r1000d30_3	187,843,389	405,759,344	12
VeRoLog_r1000d30_4	4,562,837,156	9,245,303,902	14
VeRoLog_r1000d30_5	257,497,762,007	513,847,470,020	15