Sam Smeaton
Dominic Thompson

Project Report
Routing and Scheduling of Maintenance Fleet for Offshore Wind Farms

MATH3205

Problem

The routing and scheduling of a maintenance fleet for offshore wind farms was first introduced in Dai L, Stålhane M, Utne IB (2015). The problem consists of the creation of an optimal routing schedule for a fleet of maintenance vessels in order to service an offshore wind farm over a multiple time period planning window. Vehicles have associated speeds and fuel costs. Each day vessels must depart from a depot, service turbines, and return to the depot within a time window based on vehicle and period. Each maintenance job requires an amount of non-reusable resources (parts) and reusable resources (technicians) of different types. Technicians can be left at jobs to be completed and picked up later. Non-reusable resources must not exceed vehicle capacity limits, whereas reusable resources must not exceed total personnel vehicle capacity limits (regardless of type) and daily total usage limits (type dependent). There is a time associated with loading / unloading technicians onto each turbine. Each job accumulates penalty costs for the amount of days it takes to service the job from the start of the planning period. The objective is to create a routing and scheduling plan that minimizes the combined cost of vessel travel, employees and delay costs. In the original paper, the problem was solved using a direct IP formulation that proved to be quite slow. The original problem was re-solved using a Branch-and-Price-and-Cut method in Albert H. Schrotenboer, Evrim Ursavas, Iris F. A. Vis (2019).

In this report we will attempt two different solution methods. The first method includes generating a set of feasible sub-routes using a mixed integer linear programming model, from which we join the routes together in a master problem to create an optimal routing and scheduling plan. This proved to be slow and was not guaranteed to eventually solve the routes optimally. The second method improves upon the first, where a direct set generation method is used to generate entire sub-routes directly and use them in the same master problem.

Implementation

Recursive Job Set Generation

Both methods start by generating unique combinations of jobs, from which routes are generated. The generation starts by first looping over time periods and vehicles before calling the recursive generation function with an empty job subset and a starting index of 0. The recursive function then works by looping through the elements in the entire ordered job set, starting at the starting index. It will add the job at that index to the subset and either solve the sub-problem with it, or generate routes directly. If any feasible routes are produced, it will then call recursive generation again with the updated subset, and an incremented starting index. This process ensures that every distinct combination of jobs is only checked once, and its super-sets are never considered. This is illustrated in the following diagram for a job set $J = \{0, 1, 2, 3\}$ with green routes indicating feasible routes generated, and red indicating no feasible routes generated.

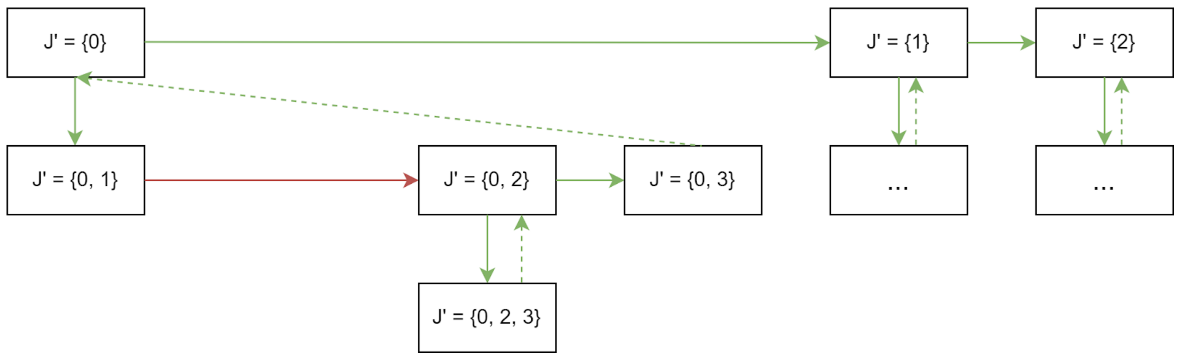


Figure 1: Example Job Subset Generation of $J = \{0, 1, 2, 3\}$

Parts Constraint Abstraction

Once a job subset combination is generated, the validity of the parts constraint is already determined. As it is frequently a binding constraint, it was immediately checked for each subset, with optimal routes only generated if the constraint was not violated. If not the set is marked as infeasible and the generation is terminated on this branch, as if any arbitrary set of jobs violates the maximum parts weight constraint, its super-sets are also guaranteed to violate the constraint.

Previous Result Storing

When generating optimal routes, before either solving the MILP formulation or directly generating routes, solution attempts in previous time windows are checked for the same job set and route to determine if a previous solution can be used to save computational time. If a solution set exists for a previous time period with the same operational time window, that solution set is copied. Otherwise if the job set and vehicle did not provide any feasible solutions for an equal or greater window, there will be no feasible routes generated and generation is cancelled on the branch. If neither case is true, either the MILP or direct generation method is used to find any optimal routes. This results in the following solution flow for a given job set vehicle and period.

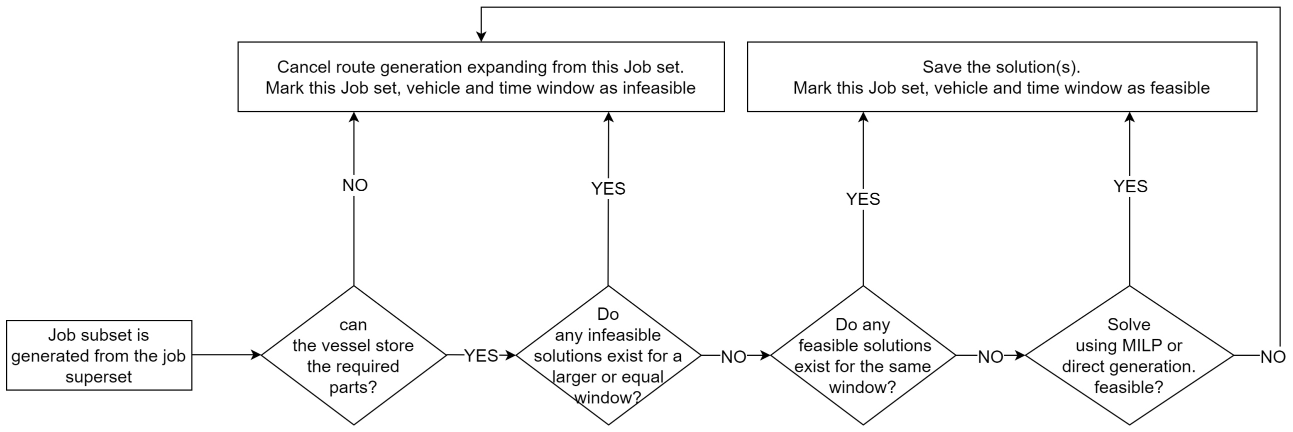

Cancel route generation expanding from this Job set.
Mark this Job set, vehicle and time window as infeasible
Save the solution(s).
Mark this Job set, vehicle and time window as feasible
NO
YES
YES
YES
Job subset is generated from the job superset
can the vessel store the required parts?
YES
Do any infeasible solutions exist for a larger or equal window?
NO
Do any feasible solutions exist for the same window?
NO
Solve using MILP or direct generation. feasible?
NO

Figure 2: Solution flow for a given Job sub-set

Method 1 - Sub-problem Formulation

For the first method sub-routes for the master problem were generated using a MILP formulation for each combination of turbine jobs, period and vehicle. Loading times were including by adding 0.25 (h) to the time of each arc. This sub-problem would return the optimal cost sub-route.

Sets

$N = \{0, 1, 2, \ldots, 2n, 2n+1\}$	set of job drop off and pick up nodes plus depot nodes
$N_d\ (\subset N) = \{0, 1, 2, \ldots n-1\}$	set of job drop off nodes
$N_p\ (\subset N) = \{n, n+1, \ldots 2n-1\}$	set of job pick up nodes
$N^* = N_d \cup N_p$	set of all job drop off and pick up nodes
P	set of technician types

Data

$start = 2n$	drop off node at depot, start point of route, $\in N$
$end = 2n+1$	pick up node at depot, end point of route, $\in N$
$tCost_{i,j}$	travel cost ($) between nodes $i, j \in N$
$tTime_{i,j}$	travel time (hrs) between nodes $i, j \in N$
$jTime_j$	service time (hrs) for job at node $j \in N_d$
$maxTravel$	maximum time vehicle can be used in time period
$capacity$	personnel capacity of vehicle
$pRequired_{p,j}$	personnel of type $p \in P$ required for job at node $j \in N^*$
$pMax_p$	maximum number of personnel of type $p \in P$ available
$pCost_p$	cost ($) of using one person of type $p \in P$ for the period

Variables

$X_{i,j}$	1 if vehicle travels directly from node i to node j where $i, j \in N$. Else 0
Y_j	time vessel visits node $j \in N$
$Z_{p,j}$	number of personnel of type $p \in P$ on the vessel after leaving node $j \in N$
Q_p	number of personnel of type $p \in P$ required to perform entire route

Objective

$$\min \sum_{i \in N} \sum_{j \in N} X_{i,j} * tCost_{i,j} + \sum_{p \in P} Q_p * pCost_p$$

Constraints

$$\sum_{j \in N} X_{j,i} = 1 \qquad \forall i \in N^* \tag{1}$$

$$\sum_{j \in N} X_{i,j} = 1 \qquad \forall i \in N^* \tag{2}$$

$$\sum_{i \in N_d} X_{start,i} = 1 \tag{3}$$

$$\sum_{i \in N_p} X_{i,end} = 1 \tag{4}$$

$$\sum_{i \in N_p} X_{start,i} = 0 \tag{5}$$

$$\sum_{i \in N_d} X_{i,end} = 0 \tag{6}$$

$$Y_{n+i} - Y_i \geq jTime_i \qquad \forall i \in N_d \tag{7}$$

$$Y_{start} = 0 \tag{8}$$

$$Y_{end} \leq maxTravel \tag{9}$$

$$0 \leq \sum_{p \in P} Z_{p,i} \leq capacity \qquad \forall i \in N \tag{10}$$

$$Z_{p,i} \leq pMax_p \qquad \forall p \in P, i \in N \tag{11}$$

$$Z_{p,i} \leq Q_p \qquad \forall p \in P, i \in N \tag{12}$$

$$|Z_{p,i} - pRequired_{p,j} - Z_{p,j}| \leq 2 * pMax_p(1 - X_{i,j}) \qquad \forall i \in N^*,\ j \in N_d,\ p \in P \tag{13}$$

$$|Z_{p,i} + pRequired_{p,j} - Z_{p,j}| \leq 2 * pMax_p(1 - X_{i,j}) \qquad \forall i \in N^*,\ j \in N_p,\ p \in P \tag{14}$$

$$Y_i + tTime_{i,j} - Y_j \leq 2 * maxTravel(1 - X_{i,j}) \qquad \forall i,\ j \in N \tag{15}$$

$$X_{i,j} \in \{0,1\} \qquad \forall i,\ j \in N \tag{16}$$

$$Y_j \geq 0 \qquad \forall j \in N \tag{17}$$

$$Z_{p,j} \in \mathbb{Z}^+ \qquad \forall p \in P,\ j \in N \tag{18}$$

$$Q_p \in \mathbb{Z}^+ \qquad \forall p \in P \tag{19}$$

The objective minimises the combined cost of travel between all turbines and the cost of personnel. (1) ensures that each turbine node is visited once, while (2) ensures each node is departed from once. (3) ensures the vehicle moves from the start node to a drop off node. (4) ensures that the vehicle departs from a pickup node and moves to the depot end. (5) enforces that there is no connection between the start node and any other pickup nodes, where (6) enforces that there is no connection between the end node and any drop off nodes. (7) ensures that all pickup nodes are visited after their corresponding drop off nodes, with at least the amount of time to complete the turbine job between. (8) sets the starting time to 0, while (9) keeps the time reaching the end below the maximum travel time. (10) ensures that at all times the amount of people on the boat between all personnel types is between 0 and the maximum capacity of the ship. (11) sets it so that the amount of personnel of a type on the ship is always less than the maximum amount of that type. (12) sets the total amount of each personnel type to be larger than every instance of the amount of people on the ship, as it is minimized in the objective function this will effectively set it to the largest number of people of that type on the ship at any point. (13) and (14) ensure that if there is a connection between a drop off node and any other node, that personnel constraint flow is satisfied between them. (15) ensures that time constraint flow is satisfied for any two nodes with a connection. (16) keeps all X variables binary, (17) keeps the Y variables positive, and the (18) and (19) constraints variables keeps the Z and Q variables positive integers.

Method 2 - Set Generation

For the second method, the sub-routes were generated directly using a recursive function that returns a set of ordered routes for every combination of turbine jobs, period and vehicle. Loading times were including by adding 0.25 (h) to the time of each arc. The set of sub routes returned do not dominate each other. A sub-route dominates another sub-route if it is cheaper or equal cost, and uses less or equal technicians of each type.

As discussed previously we use a recursive function to generate all the job sets. With each given job set generated, we use a new function that solves this job set producing the best feasible routes for the set. The algorithm for finding and solving feasible routes for the set is as follows:

$$\begin{aligned} v &= \text{vehicle number} \\ t &= \text{time period} \\ J &= \text{job set} \\ R &= \text{route index} \\ S_{v,t,R} &= \text{Ordered route for v in t at index} \\ routeCost_{v,t,R} &= \text{cost of route R with v in t} \\ services_{v,t,R,j} &= \text{v in t on route R services job j} \\ pUsed_{v,t,r,p} &= \text{number techs of type p used on route R with v in t} \end{aligned}$$

```
— Parts Constraint —
parts $\leftarrow$ stored parts amount for route
**if** parts $=$ None **then**
    parts $\leftarrow$ parts required for route
    stored parts for route $\leftarrow$parts
**end if**
**if** parts $>$ vehicle capacity **then**
    return infeasible
**end if**

— Time Window Speed Up —
**if** route has been solved for vehicle in previous period **then**
    **if** greater or equal time window and infeasible **then**
        return infeasible
    **end if**
    **if** same time window and feasible **then**
        $S_{v,t,R} \leftarrow S_{v,t,R}$ for previous route solve
        $routeCost_{v,t,R} \leftarrow routeCost_{v,t,R}$ for previous route solve
        $services_{v,t,R,j} \leftarrow services_{v,t,R,j}$ for previous route solve
        $pUsed_{v,t,r,p} \leftarrow pUsed_{v,t,r,p}$ for previous route solve
    **end if**
**end if**
```

```
— Permutations —
node ← nodes for J
permutations ← get_feasible_permutations(nodes)
solved_routes ← []
for route in permutations do
    feasible ← True
    time, cost, node_index ← $0.25, 0, 0$                          ▷ 0.25 transfer time
    personnel ← dict()                         ▷ keeps track of current personnel on jobs
    dropped ← dict()                                       ▷ keeps track of time dropped
    while node_index < len(route) do
        current_node ← route[node_index]
        time ← time + travel time to current_node
        route_cost ← route_cost + travel cost to current_node
        if drop node then
            dropped[current_node] ← time
            personnel ← personnel + personnel required for job
            update max personnel
        end if
        if pick node then
            time ← time + time until job finished                  ▷ can already be finished
            personnel ← personnel - personnel required for job
            update max personnel
        end if
        if time, capacity, or personnel constraints violated then
            break
        end if
    end while
    time ← time + travel time to depot
    route_cost ← route_cost + travel cost to depot
    if time constrain violation then
        break
    end if
    route_cost ← route_cost + personnel cost
    if feasible then
        solved_routes.append(route)
    end if
end for
if len(solved_routes $>$ 0 then
    for route in solved_routes do
        $S_{v,t,R}$ ← route
        $routeCost_{v,t,R}$ ← route_cost
        $services_{v,t,R,j}$ ← 1 for jobs serviced, 0 for jobs not serviced
        $pUsed_{v,t,r,p}$ ← route personnel
        $R \leftarrow R + 1$
    end for
end if
save time window information for speed up
```

Master Problem Huge IP Formulation

Sets

R	set of routes
V	set of vehicles
T	set of time periods
$R_{v,t}$	set of routes for vehicle $v \in V$, in time period $t \in T$
J	set of turbines
P	set of technician types

Data

$S_{v,t,r}$	list of turbines in route $r \in R$ for vehicle $v \in V$ in time period $t \in T$
$serviceCost_{t,j}$	accumulated cost of taking until period $t \in T$ to service turbine $j \in J$
$routeCost_{v,t,r}$	cost of route $r \in R$ with vehicle $v \in V$ in time $t \in T$
$services_{v,t,r,j}$	1 if route $r \in R$ with vehicle $v \in V$ in time $t \in T$ services turbine $j \in J$
$pMax_p$	maximum number of personnel of type $p \in P$ available each day
$pUsed_{v,t,r,p}$	number of technicians of type $p \in P$ required for route $r \in R$ using vehicle $v \in V$ at time $t \in T$

Variables

$U_{v,t,r}$	1 if route $r \in R$ of vessel $v \in V$ at time $t \in T$ is to be ran, else 0

Objective

$$\min \sum_{v \in V} \sum_{t \in T} \sum_{r \in R_{v,t}} U_{v,t,r} \cdot \left(routeCost_{v,t,r} + \sum_{j \in J} services_{v,t,r,j} \cdot serviceCost_{t,j} \right)$$

Constraints

$$\sum_{r \in R_{v,t}} U_{v,t,r} \leq 1 \qquad \forall v \in V\ , t \in T \tag{20}$$

$$\sum_{v \in V} \sum_{t \in T} \sum_{r \in R_{v,t}} U_{v,t,r} \cdot services_{v,t,r,j} = 1 \qquad \forall j \in J \tag{21}$$

$$\sum_{v \in V} \sum_{r \in R_{v,t}} U_{v,t,r} \cdot pUsed_{v,t,r,p} \leq pMax_p \qquad \forall p \in P,\ \forall t \in T \tag{22}$$

$$U_{v,t,r} \in \{0,1\} \qquad \forall v \in V,\ \forall t \in T,\ \forall r \in R \tag{23}$$

The Objective minimizes the sum of costs for all routes employed over the entire service plan, and the cost of delaying each job until it is serviced. Constraint (20) enforces that each vehicle can do at most one route in each time period. (21) makes it so that each turbine is serviced by exactly one vehicle on one route in one time period. (22) sets the upper bound of personnel available across all vessels for the routes employed in each period for each personnel type. (23) sets the U variables to binary.

Results

MILP Sub-Problem Method

The MILP Sub-Problem Formulation encountered many issues in it's implementation. The largest of which was that it was not guaranteed produce optimal scheduling plans. Because the sub-problem returns the optimal cost route for a time period, vehicle, and set of jobs, the master problem will not consider more expensive routes that may use less personnel. Although these routes are more expensive, they may be used in the optimal scheduling because they may be paired with efficient routes due to their lower technician usage, resulting in a lower total cost. It would be possible to remedy this potential issue by introducing cuts to optimal cost solutions' personnel usage to generate any other possible routes with less personnel. However, it was also orders of magnitude slower than the Set Generation Method, as seen below. Due to these reasons, the direct generation method was used for the rest of the results.

Testcase	Method	Computational Time (s)	Schedule Cost ($)
An10T3-1	1	15.90	29495
	2	2.81	29495
An10T4-1	1	28.28	28334
	2	3.04	28334

Figure 3: Comparison Between MILP Formulation (Method 1) and Direct Route Generation (Method 2)

Set Generation Method

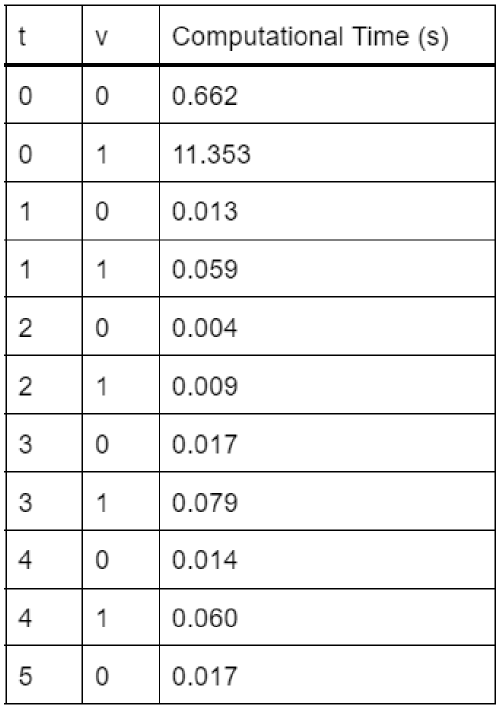
Set Generation produced feasible results in reasonable time for smaller problems. The route was cheaper than the reference solution provided, and was feasible by hand-checking. This was done by calculating the time taken for each vehicle's daily routes, and making these did not exceed their respective time windows. The number of people calculated for each route was also checked by hand, and this did not exceed either the vehicle capacity limits for the total between technician types, or the daily limits on the number of each technician type available. We believe the difference is likely due to a data discrepancy, possibly relating to transfer times. More results are available in the appendix.

Testcase	Computational Time (s)	Schedule Cost ($)	Schedule		
			Day	Vehicle	Route
An10T3-1	2.81	29495	0	0	7d, 5d, 7p, 6d, 5p, 6p
			0	1	0d, 0p, 3d, 9d, 9p, 3p
			1	1	2d, 1d, 1p, 2p
			2	0	4d, 4p, 8d, 8p

Figure 4: An10T3-1 Results with Direct Route Generation

We do not provide results for tests with more than 26 jobs. This is due to the length of time it takes our algorithm to solve the test cases with greater than 26 jobs. The fundamental reason behind the time to compute larger instances is the way the algorithm generates the ordered routes. Firstly the unique job combinations are generated (no ordering), and then for each of these job sets the permutations are generated, which are then solved. By generating the sets first, and then the permutations, we can only cut route generation off at a set level. This is very disadvantageous in terms of time, as even though we can cut off at the set level, the time saving is negligible with the fact that you have to solve all permutations of a set before cutting. This is less of a problem with smaller job counts as there are less permutations for smaller job sets. However, once the job sets increase there are more combinations to generate. So not only do we have to generate permutations more times, the number of permutations generated greatly increases as the jobs sets are larger. This is very noticeable when generating routes for a vehicle that can handle more parts, people, and a greater time window. Given its larger capacity in every aspect the set generation cuts of significantly later, resulting in the number of routes generated increase by orders of magnitude.

An example of this is the An32T10 test. In this test for the first 5 periods the largest time window is only 7 hours, so the set generation cuts off extremely early. This results in the route generation for each vehicle in the periods being completed very quickly, as shown below:



t	v	Computational Time (s)
0	0	0.662
0	1	11.353
1	0	0.013
1	1	0.059
2	0	0.004
2	1	0.009
3	0	0.017
3	1	0.079
4	0	0.014
4	1	0.060
5	0	0.017

Figure 5: An32T10-1 Time Elapsed for Computing Optimal Routes in Each Time Period, Vehicle

As seen above generating sets and permutations for the first 5 periods was reasonably efficient. However problems are encountered in the 6th period route generation. The algorithm generates the routes for vehicle 0 quickly again, as it has a small window (7 hours) and also smaller parts capacity, so it's set generation cuts off quickly. However the algorithm then stalls when generating the routes for vehicle 1 in this time period. This is firstly because the ship can handle more parts, but more importantly because the time window is the largest encountered for this test instance, 9 hours. Because of this, the set generation is cutting off at much higher job counts, resulting in exponentially more possible route permutations being considered.

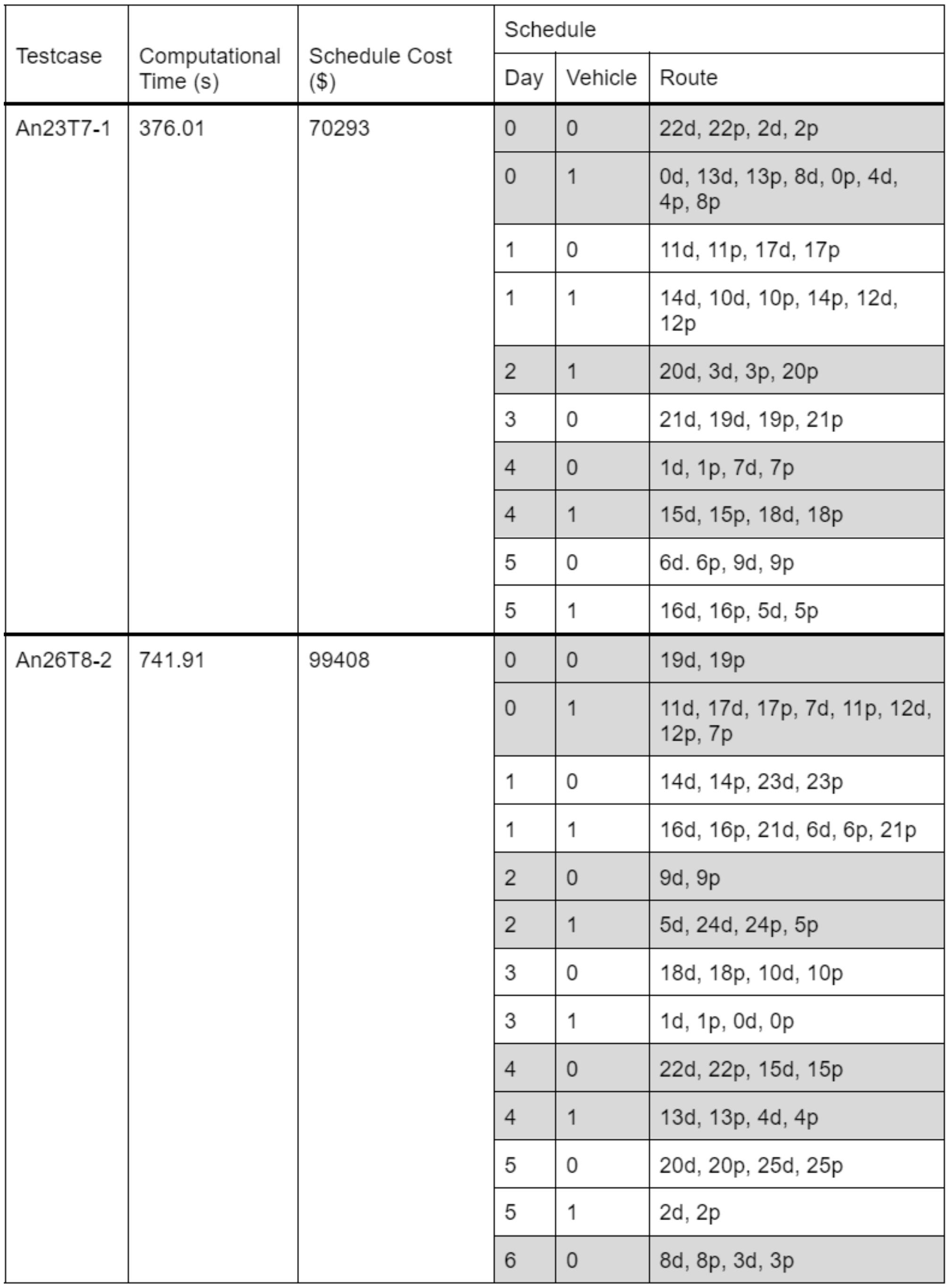
A possibly better way to generate the routes would have been to instead build the permutations recursively, allowing for cut offs at a permutation level. Generating routes recursively and saving finished, feasible routes during generation before building off them, and stopping generating off of routes that are determined to be infeasible. This would stop generation before any unnecessary permutations are generated. Doing it this way would result in significantly faster route generation, and hence not only speed up the solve times of already solved test instances, but also allow for solving larger test instances.

References

Albert H. Schrotenboer, Evrim Ursavas, Iris F. A. Vis (2019) A Branch-and-Price-and-Cut Algorithm for Resource-Constrained Pickup and Delivery Problems. Transportation Science. Published online in Articles in Advance 28 Jun 2019 https://doi.org/10.1287/trsc.2018.0880

Dai L, Stålhane M, Utne IB (2015). Routing and Scheduling of Maintenance Fleet for Offshore Wind Farms. Wind Engineering. 2015;39(1):15-30. Published Feb 1 2015 https://doi:10.1260/0309-524X.39.1.15

Appendix



Testcase	Computational Time (s)	Schedule Cost ($)	Schedule		
			Day	Vehicle	Route
An23T7-1	376.01	70293	0	0	22d, 22p, 2d, 2p
			0	1	0d, 13d, 13p, 8d, 0p, 4d, 4p, 8p
			1	0	11d, 11p, 17d, 17p
			1	1	14d, 10d, 10p, 14p, 12d, 12p
			2	1	20d, 3d, 3p, 20p
			3	0	21d, 19d, 19p, 21p
			4	0	1d, 1p, 7d, 7p
			4	1	15d, 15p, 18d, 18p
			5	0	6d. 6p, 9d, 9p
			5	1	16d, 16p, 5d, 5p
An26T8-2	741.91	99408	0	0	19d, 19p
			0	1	11d, 17d, 17p, 7d, 11p, 12d, 12p, 7p
			1	0	14d, 14p, 23d, 23p
			1	1	16d, 16p, 21d, 6d, 6p, 21p
			2	0	9d, 9p
			2	1	5d, 24d, 24p, 5p
			3	0	18d, 18p, 10d, 10p
			3	1	1d, 1p, 0d, 0p
			4	0	22d, 22p, 15d, 15p
			4	1	13d, 13p, 4d, 4p
			5	0	20d, 20p, 25d, 25p
			5	1	2d, 2p
			6	0	8d, 8p, 3d, 3p

Table A2: More Example Results from Test set A

Testcase	Computational Time (s)	Schedule Cost ($)	Schedule		
			Day	Vehicle	Route
An10T3-1	2.81	29495	0	0	7d, 5d, 7p, 6d, 5p, 6p
			0	1	0d, 0p, 3d, 9d, 9p, 3p
			1	1	2d, 1d, 1p, 2p
			2	0	4d, 4p, 8d, 8p
An10T4-1	3.04	28334	0	0	2d, 2p, 4d, 4p
			0	1	0d, 0p, 6d, 6p
			1	1	8d, 3d, 3p, 8p
			2	0	7d, 7p, 1d, 1p
			2	1	9d, 9p, 5d, 5p
An15T4-1	226.16	46731	0	0	11d, 3d, 3p, 11p, 13d, 13p
			0	1	2d, 2p, 12d, 12p
			1	0	9d, 9p
			1	1	5d, 5p
			2	0	4d, 4p, 14d, 7d, 7p, 14p
			2	1	1d, 1p
			3	0	0d, 0p, 8d, 8p
			3	1	10d, 10p, 6d, 6p
An20T5-1	225.31	71328	0	0	9d, 9p, 11d, 11p
			0	1	13d, 14d, 13p, 2d, 14p, 15d, 15p, 2p
			1	0	8d, 8p, 17d, 17p
			1	1	6d, 6p, 12d, 12p
			2	0	16d, 16p, 4d, 4p
			2	1	1d, 1p, 19d, 19p
			3	0	18d, 3d, 3p, 18p
			3	1	10d, 10p
			4	0	5d, 5p
			4	1	7d, 0d, 0p, 7p

Table A1: Example Results from Test set A

Testcase	Computational Time (s)	Schedule Cost ($)	Schedule		
			Day	Vehicle	Route
Cn22T4-1	21.72	61915	0	1	7d, 5d, 7p, 6d, 5p, 6p
			0	2	13d, 14d, 14p, 17d, 17p, 13p
			1	0	19d, 19p, 5d, 5p
			1	1	16d, 16p, 18d, 18p
			1	2	20d, 20p, 1d, 1p
			2	0	15d, 15p, 11d, 11p
			2	1	12d, 12p, 4d, 4p
			2	2	3d, 2d, 2p, 3p
			3	0	21d, 2p, 0d, 0p
			3	1	6d, 6p
			3	2	8d, 8p, 10d, 10p
Cn26T5-2	30.75	88578	0	0	0d, 0p, 19d, 19p
			0	1	22d, 22p, 5d, 5p
			0	2	7d, 7p, 25d, 25p
			1	0	6d, 1d, 1p, 6p
			1	1	12d, 12p
			1	2	13d, 13p, 8d, 8p
			2	0	21d, 21p, 15d, 15p
			2	1	14d, 14p, 9d, 9p
			2	2	3d, 3p
			3	0	23d, 23p, 10d, 10p
			3	2	24d, 18d, 18p, 24p
			4	0	16d, 11d, 11p, 16p
			4	1	2d, 2p, 17d, 17p
			4	2	20d, 4d, 4p, 20p

Table A3: Example Results from Test set C