# Assignment 3 Report 35224436

## Introduction

This report presents a sensitivity analysis of a service scheduling optimization model. By varying key parameters (e.g., `minservice`, `maxsty`, `maintdur`, etc.), I examine their effects on solution feasibility, service coverage, maintenance load, and runtime, highlighting the most impactful constraints.

## Methodology

The model has multiple key parameters and constraints. To assess their individual impacts, I adopted the following structured methodology:

The solver used is **Chuffed 0.13.2**, with a fixed solving time limit of **60 seconds** for each run. The evaluation involved these steps:

1. **Data file choose:** I chose **service2.dzn** as the sample dataset because its service times are highly overlapping but spread across the timeline. This structure — with overlapping high-load periods and narrow gaps for inserting maintenance — makes it a good case to test how feasible or infeasible the problem becomes under different parameter settings (e.g., `minservice`, `minwork`, `maxsep`, `maintdur`, `majorwork`), and how the maintenance strategy needs to adjust accordingly.
2. **Modify parameters one-by-one:** Following the instructions from Stage G, I changed one key parameter at a time and compared the results with the baseline. I observed whether the result was optimal, and compared key indicators like `totalLoad`, `totalService`, and `solving time`.

## Parameter-by-Parameter Analysis

### minservice = 0

When `minservice` is set to 0, the model no longer considers service completion and focuses only on minimizing **totalLoad**.
In this case, running service2.dzn results in
**totalLoad = 0** and `totalService = 0`.
This is because, without the minimum service requirement, the model prefers assigning all tasks to the dummy truck, which doesn't need any maintenance (since it doesn't perform any services).

### minservice = 100

When `minservice` is set to 100, the model fails to find a feasible solution for `service2`.
The reason is a conflict between the two goals: completing all services and minimizing maintenance. The model can't find a way to fit all the required services and legal maintenance into the limited gaps defined by **maxsep[SH]** or **maxsep[LG]**.
As soon as any truck is assigned a service, it becomes impossible to satisfy both "full service coverage" and "valid maintenance", so the model returns
**UNSATISFIABLE**.

### minwork = 0

There is no change in `totalLoad` or `totalService`. In the original dataset, `minwork` is 20, meaning a truck only needs to work for 20% of `endtime = 15`, which is 3 time units.
Since all services in service2 last no more than 3 units, it's easy to meet this requirement. So with `minwork = 0`, the constraint is still naturally satisfied.

### minwork = 100

When `minwork` is set to 100, not only `service2` but all datasets become unsolvable.
This is because
`minwork = 100` forces any truck in use to spend 100% of its time performing services. However, no truck in any dataset can spend the entire time window from 0 to `endtime` on services due to the mandatory maintenance constraints.

As a result, the solver immediately returns
**UNSATISFIABLE**.

## maxsty = [1,1,1,0]

When `maxsty` is set to `[1, 1, 1, 0]`, it means each truck can only perform **one** type of service.
In this case, the model returns
**totalLoad = 62** and **totalService = 24**. Because trucks are limited to handling just one type, the model has to assign the same type of service to different trucks. This increases the total number of trucks in use and leads to more maintenance.
As a result, the maintenance burden (totalLoad) increases sharply to 62.
Even though the load is higher, the model still manages to complete more services, so totalService remains close to baseline.

## maxsty[ty] = 0 for some ty other than DUMMY (eg. maxsty[MED] = 0)

This setting means trucks of a certain type are **not allowed** to perform any services.
After testing, we found that when
**maxsty[SML]** or **maxsty[MED]** is set to 0, service2 becomes unsolvable. However, setting `maxsty[LRG] = 0` (e.g. `maxsty = [1, 2, 0, 0]`) still allows the model to find an optimal solution with `totalLoad = 38` and `totalService = 23`, which is the same as the baseline.
This is because, in the service2 dataset, apart from two LRG-type services, the remaining trucks (two SML and two MED trucks) can cover 6 different service types in total — enough to meet the demands of 10 services across 4 service types.

## maintdur[SH] > maintdur[LG]

To test this setting, I changed the original `maintdur = [0, 1, 3, 5]` to `maintdur = [0, 4, 3, 5].` After testing, the result was: `totalLoad = 52` and `totalService = 24`.
Originally, SH maintenance took less time than LG. Now with SH taking even longer than LG, the model must re-arrange the overall maintenance plan to minimize total maintenance load. Because SH became more “expensive,” the optimizer prefers to schedule fewer SH and more LG tasks instead. However, LG must also follow stricter interval rules (
**maxsep[LG]**), so the model still needs to insert a few SHs in between to fill timing gaps.
As a result, both the number of maintenance tasks increased. The total maintenance load rose from 38 to 52. Still, to maximize service coverage, the model successfully completed all 24 tasks — so
`totalService` slightly increased compared to the baseline.

## maxsep[SH] > maxsep[LG]

To test this, I changed the original `maxsep = [0, 4, 8, 0]` to `maxsep = [0, 10, 8, 0]`.
The result was
**totalLoad = 12** and **totalService = 26**, and solving time arise to nearly 5 seconds. After this change, the model allows trucks to skip `SH` for longer periods. Although `LG` maintenance may increase, it is more spread out across different trucks and happens less frequently overall, so the actual maintenance load decreases — from 30 to 12.
Also, when
**maxsep[SH] = 10**, trucks don’t need any SH within the first 10 time units, reducing the number of SH arrangement. This frees up time to schedule shorter services at the beginning, so
**totalService** increases from 23 to 26.

## maxsep[MJ] > maxsep[LG]

In this case, I changed `maxsep = [0, 4, 8, 0]` to `maxsep = [0, 4, 8, 10]`. The result was `totalLoad=38` and `totalService=23`, which matches the baseline result.
In the original dataset, MJ maintenance was not subject to minimum interval constraints (since **maxsep[MJ]=0**), meaning it was not mandatory to insert at least one MJ within the period. After changing it to `maxsep[MJ]=10`, although the model now requires "at least one MJ every 10 time units", the original solution still works. This is because even though `maxsep[MJ]=0` previously allowed no MJ maintenance within the period, the optimal solution already included MJ maintenance due to the prework, majorwork, and stage F MJ constraints. Now with the forced MJ insertion, since the total time endtime is only 15 (not much larger than `maxsep[MJ]=10`), at worst we only need to schedule one MJ for each "active" truck, so the original solution still satisfies all constraints.

## maxsep[SH] = maxsep[LG] = endtime

After setting maxsep to [0, 15, 15, 0], the model found an optimal solution on the service2 dataset with totalLoad = 0 and totalService = 31.
With maxsep[SH] = maxsep[LG] = 15, trucks don't need to perform any SH or LG maintenance within the entire 0..endtime scheduling window. Therefore, the total maintenance load naturally becomes 0, and all trucks are fully available to complete services, allowing totalService to reach its maximum value of 31.

## prework = [0 | t in TRUCK]

The optimal solution remains at `totalLoad = 38` and `totalService = 23.`
For the original data, with
**prework = [0, 5, 10, 15, 20, 25]** and **majorwork = 30**, only trucks T5 and T4 required mandatory MJ within the prework[t] + endtime timeframe. When setting all prework values to 0, it effectively resets the initial workload for all vehicles to 0, meaning each truck is treated as if it just completed a major maintenance at time 0, eliminating the need to insert MJ before t=0.
However, the arrangement of the three maintenance types within the period and their concurrency remain unchanged from the original plan, so totalLoad stays at 38 and totalService stays at 23. In other words, while the scheduling window changes, the total maintenance requirements remain the same.

## majorwork > endtime

I tested different values for majorwork. After multiple tests, when 25 <= majorwork <= 55, the model consistently found the same optimal solution as the original dataset, with totalLoad=38 and totalService=23. For other values, no feasible solution was found.
When majorwork is less than 25 (e.g., 24), the model becomes unsatisfiable because "prework[T5] = 25" directly exceeds majorwork=24, violating the hard constraint that "the total service time between any two MJs must not exceed majorwork hours". Therefore, the model immediately returns UNSATISFIABLE without attempting to schedule any services.
When majorwork > 55, it triggers a redundant constraint in my model (added for pruning purposes)

```
constraint majorwork >= total_service_dur + max(prework) -> forall(t in TRUCK, m in MAINT)(mt[t,m] != MJ);
```

This leads to prohibiting MJ maintenance arrangements, which conflicts with the mandatory SH/LG/MJ maintenance constraints from stage D, resulting in UNSATISFIABLE.

## start[s] = 0 and end[s] = endtime for some service s

Testing shows that setting most services to start[s] = 0 and end[s] = endtime leads to an increase in totalLoad. This is because such services require a truck to be occupied for the entire period (length endtime=15). Only one truck can be "sacrificed" to handle this super long service, while the remaining trucks must compete for available time slots to perform other services and maintenance.
With fewer available trucks, additional maintenance must still be inserted on other vehicles to satisfy maintenance constraints, resulting in a significant increase in totalLoad.

## start includes k services with the same start and end time where k = card(TRUCK)

For `service2.dzn`, `card(TRUCK) = 6`

I modified the original data:
start = [0, 0, 0, 7, 3, 5, 6, 10, 11, 12];
end = [4, 5, 3, 11, 5, 8, 9, 12, 13, 15];

to:
start = [0, 0, 0, 7, 0, 5, 0, 0, 11, 0];
end = [4, 5, 3, 11, 5, 8, 5, 5, 13, 5];

Testing showed that the model could not find a feasible solution.
In this case, k = 6 services must all start at the same time (e.g., t=0) and end at the same time (e.g., t=5), requiring 6 different non-Dummy trucks to perform these 6 services in parallel. However, service2 only has 5 actual working trucks available, so the service assignment constraints cannot be satisfied. Additionally, even with more trucks, the subsequent maintenance requirements would make maintenance scheduling impossible due to the high concurrency. Therefore, the model returns UNSATISFIABLE.

## card(STYPE) = 1

When all service types are changed to the same type, for example: **STYPE = { All }; stype = [ All | i in 1..10 ]**;
The optimal solution remains the same as the baseline.
Reducing the service types to one (from the original four) effectively relaxes the maxsty constraints, making service assignment simpler. Therefore, the optimal solution cannot be worse. However, since maintenance constraints are not affected by service types, the total load remains unchanged. Additionally, since the duration of each service remains the same, there is no significant change in totalservice.

# Summary

After analysis, the most important constraints are as follows:

1. **Non-Overlapping Services and Maintenance (Stage A)**

   This `cumulative` constraint ensures that "at any point in time, a (non-Dummy) truck can only perform one task (service or maintenance)". In simple terms, any service or maintenance assigned to truck `t` cannot overlap on the timeline. If this constraint is removed, trucks could "simultaneously" perform multiple services or conduct services and maintenance in parallel, which would greatly expand the scheduling space and completely alter maintenance concurrency and service completion numbers. Under various extreme parameters (such as `minservice = 100` or `maxsep = endtime`), as long as this constraint remains, all service and maintenance conflict checks will either succeed or fail. Therefore, this is the most important constraint.

2. **Minservice constraints（Stage B)**

   This constraint controls the minimum proportion of services that must be completed, effectively reducing the number of feasible solutions and directly influencing the decision of service assignment to dummy trucks. When minservice=0, all services would be assigned to the dummy truck, resulting in meaningless solutions; when minservice=100, the solution space is significantly reduced, making it almost impossible to find solutions for all datasets, and the related constraints from stage D would make maintenance time too tight to complete all services. Therefore, this constraint significantly impacts truck selection and solving efficiency.

3. **Maxsty constraints（Stage B)**

   The `maxsty` constraint limits how many service types each truck type can handle during scheduling, directly affecting service flexibility. When tightened from `[1,2,3,0]` to `[1,1,1,0]`, previously allowed flexibility for MED and LRG trucks is lost. This forces same-type services to be split across multiple trucks, increasing concurrent maintenance and sharply raising `totalLoad`. If `maxsty` for any non-DUMMY type is set to 0, it would immediately make certain services impossible to execute by any available truck, resulting in UNSATISFIABLE.
   Therefore, maxsty fundamentally alters the feasible solution space and optimal solution structure, making it one of the most critical constraints that can significantly change the scheduling pattern in the model.

4. **Stage D's Mandatory Maintenance and Maintenance Interval Constraints**

   These constraints force maintenance to be inserted within the scheduling period, not only reducing available service time but also increasing total maintenance load. Additionally, the relative size between maxsep[SH] and maxsep[LG] determines whether the maintenance strategy will be SH-dominated or LG-dominated, adding solution branches.
   In Stage G's various extreme tests, whenever maxsep changes, not only does the maintenance insertion strategy completely change, but it also limits how long "continuous service" can be. Therefore, this constraint has a non-negligible impact on the model's performance.

5. **Major Maintenance Workload Constraint (majorwork-related Stage F)**

   This constraint limits how much service a truck can do between two MJs, preventing long continuous work. MJ typically lasts long (e.g., `maintdur[MJ] = 5`) and blocks future service slots. Frequent MJ insertions reduce service time; infrequent MJ risks violating `majorwork`, causing infeasibility or requiring more SH/LG to delay overload.
   So, majorwork helps decide how often trucks need major maintenance and how long they can keep working without it. In other words, it directly limits the available time for service tasks.