FIT5216: Modelling Discrete Optimization Problems

Assignment 3: Service Scheduling

1 Overview

For this assignment, your task is to write a MiniZinc model for a given problem specification.

- Submit your work to the MiniZinc auto grading system (using the submit button in the MiniZinc IDE). You must submit via the IDE to be graded and receive marks.
- Submit your model (copy and paste the contents of the .mzn file) and report using the Moodle assignment.

You have to submit by the due date (Friday 30th May 2025, 11:55pm), using MiniZinc and using the Moodle assignment, to receive full marks. You can submit as often as you want before the due date. Late submissions without special consideration receive a penalty of 10% per day. Submissions are not accepted more than 7 days after the original deadline.

This is an **individual assignment**. Your submission has to be **entirely your own work**. We will use similarity detection software to detect any attempt at collusion, and the **penalties are quite harsh**. Similarly you may not use **generative AI** such as ChatGPT, CoPilot or DeepSeek for any part of the project. Note that making your code available to other students undertaking the subject is also an instance of **collusion** that is also punishable. If in doubt, contact your teaching team with any questions!

2 Problem Statement

You are managing a service centre for hot water appliances. Each week you are given a fixed set of services that have been booked. You need to assign the truck to visit and perform each service, while ensuring the trucks have the correct tools for the service and are regularly maintained.

The key data that you need to reason about is the scheduled services. They are represented by the MiniZinc data below:

```
int: endtime;                     % latest end time of services
set of int: TIME = 0..endtime;
enum SERVICE;                     % services to be scheduled
array[SERVICE] of TIME: start;  % start time of service
array[SERVICE] of TIME: end;    % end time of service
enum STYPE;                       % possible service types
array[SERVICE] of STYPE: stype; % type of each service
```

All the scheduling is over the fixed period from time 0 to `endtime`. Each service has a fixed start and end time and a service type.

The key resource for undertaking the service scheduling are trucks. They are represented in MiniZinc data below:

```
enum TRUCK;                       % truck
TRUCK: dtruck;                    % dummy truck
enum TTYPE = { SML, MED, LRG, DUMMY };  % truck type
array[TRUCK] of TTYPE: ttype;     % type of each truck
constraint assert(forall(t in TRUCK)
           (ttype[t] = DUMMY <-> t = dtruck),"dummy type only for dtruck\n");
```

The trucks are given by the enumerated type `TRUCK`, but one of the elements of this type is a dummy truck `dtruck`. Services assigned to the dummy truck are not actually performed. Each truck has a truck type `TTYPE`. There is a dummy truck type which is only used for the dummy truck.

The aim of the model is to assign services to trucks so that as many services as possible are actually done, and minimize the amount of effort on maintenance. Write a MiniZinc model `service.mzn` to tackle this problem. Its recommended that you tackle this problem in stages in the order shown. You can test your model before completing the entire assignment using the checker.

3 Stage A: Assigning Services

The core decisions of the model are, for each service which truck does that service. Note that assigning a service to the dummy truck means that the service is not actually performed.

```
array[SERVICE] of var TRUCK: truck; % which truck takes each service: dtruck if none
```

A first important constraint is that if two services are assigned to the same (non-dummy) truck then they cannot overlap in time. Note that it is perfectly possible to assign two tasks to the same truck where the first has end time equal to the start time of the second task, e.,g the first task starts at 6 and ends at 8 (so duration 2) and the second starts at 8 and ends at 11 (so duration 3).

Given the data file below:

```
endtime = 15;
SERVICE = { S1, S2, S3, S4, S5, S6, S7, S8, S9, S10 };
start =     [0,  0,  0,  7,  3,  5,  6, 10, 11,  12];
end   =     [4,  5,  3, 11,  5,  8,  9, 12, 13,  15];
STYPE = { Gas, Roof, Refit, Solar };
stype =  [ Gas, Refit, Roof, Gas, Solar, Solar, Refit, Roof, Gas, Refit ];
TRUCK = { DD, T1,  T2,  T3,  T4, T5 };
dtruck = DD;
ttype = [ DUMMY, SML, SML, MED, LRG, MED ];
```

Then a possible assignment of trucks is given by:

```
truck = [ T1, T3, T5, T1, T5, T5, T2, T4, T5, T4 ];
```

which we can represent as

```
DD:
T1:S1 S4
```

```
T2:S7
T3:S2
T4:S8 S10
T5:S3 S5 S6 S9
```

which is visualized for each truck as

```
   000000000011111
   012345678901234
DD:...............   0
T1:####...####....   8
T2:......###......   3
T3:#####..........   5
T4:..........#####   5
T5:########...##..  10
```

Where we see the times when the truck is busy and the total work time for each truck.

4 Stage B: Truck Usage, Service Guarantee

Its not economical to use a truck for only very few services in the planning period. We also have to offer a *service guarantee* to our client that some percentage of booked services will occur. The data includes definitions:

```
int: minwork;    % minimal percentage of time each truck that works is working
int: minservice; % minimal percentage of services performed
```

For Stage B you need to ensure that any truck who is assigned any services is assigned work whose total duration is at least `minwork`% of `endtime`. We also need to ensure that the percentage of services that are actually completed is at least `minservice`% of the services booked.

Assuming we set `minwork = 30` and `minservice = 80` a solution is:

```
DD:S5
T1:
T2:
T3:S3 S7
T4:S1 S6 S8 S10
T5:S2 S4 S9
```

with visualization:

```
   000000000011111
   012345678901234
DD:...##..........   2
T1:...............   0
T2:...............   0
T3:###...###......   6
T4:####.###..#####  12
T5:#####..######..  11
```

Notice the trucks that were assigned very little work have disappeared. In this solution not every service is performed (`S5` is not), but more than 80% of services are performed.

5 Stage C: Work Types

There are four types of maintenance services that can be booked: `Gas`, `Roof`, `Refit` and `Solar`, but trucks cannnot be neessarily configured to be able to perform all services. There is a limit on the number of types of service for each truck given by

```
array[TTYPE] of int: maxsty;     % max service types for each truck type
```

For the running example we assume `maxsty = [1,2,3,0];` then a solution that satisfies this constraint is

```
DD:S9 S10
T1:S3 S8
T2:S2 S7
T3:S5 S6
T4:S1 S4
T5:
```

visualized as:

```
   000000000011111
   012345678901234
DD:...........####   5 {Gas, Refit}
T1:###.......##...   5 {Roof}
T2:#####.###......   8 {Refit}
T3:...#####.......   5 {Solar}
T4:####...####....   8 {Gas}
T5:...............   0 {}
```

The kinds of services for each truck are listed afterward. You can see that the SML trucks T1 and T2 only do one kind of service as required, in fact every truck only does one kind of service. The dummy truck of course can be assigned all possible services.

If you only finish to this stage you will be unable to acheive more than 30% marks for most instances. But to do this you probably need to set the maintenance variables `mt` and `ms` below to some good default values.

6 Stage D: Maintenance

Now we get into the more challenging part of the assignment. We also have to ensure that we maintain the trucks, which means that we need to schedule maintenance tasks for them. The data and decisions are defined by:

```
enum MAINTYPE = { NO, SH, LG, MJ }; % NO means no maintenance
array[MAINTYPE] of int: maintdur; % duration of maintenance
```

```
array[MAINTYPE] of int: maxsep;   % max elapsed time since last maint this type or higher
constraint assert(maintdur[NO] = 0 /\ maxsep[NO] = 0, "rules for empty maintenance\n");

int: maxmaint;
set of int: MAINT = 1..maxmaint;
array[TRUCK,MAINT] of var MAINTYPE: mt; % maintenance for each truck (NO=not used).
array[TRUCK,MAINT] of var TIME: ms;     % maintenance start time for each truck
```

So each truck can have up to `maxmaint` maintenances in the schedule. The maintenances are SH (short), LG (long), MJ (major) and NO (none) which is a placeholder for no maintenance used. The duration of each maintenance task is defined in the array `maintdur`.

For each truck we have to decide the type `mt` of each maintenance, and the start time `ms` of each maintenance. All the NO maintenances should be at the end of the list for each truck. The `ms` start time for NO maintenances is irrelevant. The other maintenances must appear in order of increasing start time, the earliest first, then the next, etc.

The maintenance constraints are defined as follows:

- A truck assigned no services needs no maintenance. The remaining constraints apply to any trucks assigned a service.
- There must be a SH/LG or MJ maintenance within `maxsep[SH]` of the start of the schedule and within `maxsep[SH]` of the end of any maintenance task, unless this falls outside `TIME` (i.e. starts at `endtime` or later).
- There must be a LG or MJ maintenance within `maxsep[LG]` of the start of the schedule and within `maxsep[LG]` of the end of any LG or MJ maintenance task, unless this falls outside `TIME`.

The other important constraint is that maintenance tasks and service tasks for the same truck cannot overlap.

Given the data

```
maxmaint = 2;
maintdur = [0,1,3,5];
maxsep = [0,5,10,0];
```

A solution might be

```
truck = [T2, T1, DD, T2, T3, T3, T1, DD, T3, T1];
mt =
[| DD: NO, NO
 | T1: SH, LG
 | T2: LG, LG
 | T3: LG, LG
 | T4: NO, NO
 | T5: NO, NO
 |];
ms =
[| DD: 0,  0
 | T1: 5,  9
 | T2: 4, 12
 | T3: 0,  8
 | T4: 0,  0
 | T5: 0,  0
 |];
```

which is visualized as

```
   000000000011111
   012345678901234
DD:###.......##...   5 {Roof}
T1:#####m###mmm###  11 {Refit}
T2:####mmm####.mmm   8 {Gas}
T3:mmm#####mmm##..   7 {Gas, Solar}
T4:...............   0 {}
T5:...............   0 {}
```

We can see that no maintenance overlaps with service tasks, the start times for each truck are ordered, and the NO maintenances appear at the end. Clearly there is a maintenance within the first 5 time steps for all used trucks, and no period without maintenance for 5 time steps (i.e. for T3 its last maintenance ends at 11 which is within 5 time steps of the endtime 15), and a large maintenance within the first 10 steps for all used trucks.

7 Stage E: Objective

The main criteria for optimisation are two:

- Maximizing the total duration of services performed.
- Minimizing the total maintenance load. The maintenance load at time t is defined by the **square** of the number of simultaneous maintenance tasks occurring at that time (started at or before t and ending after t. The total maintenance load is the sum over all times 0..`endtime-1`.

You should write your model to first minimize the total maintenance load, and then maximize the total duration of services performed.

An optimal solution for the data show `service0.dzn` is given by

```
truck = [DD, T3, T4, T4, T5, T5, DD, T3, T5, T4];
mt =
[| DD: NO, NO
 | T1: NO, NO
 | T2: NO, NO
 | T3: LG, SH
 | T4: LG, SH
 | T5: SH, LG
 |];
ms =
[| DD: 0,  0
 | T1: 0,  0
 | T2: 0,  0
 | T3: 5, 12
 | T4: 3, 11
 | T5: 2,  8
 |];
```

with visualization

```
   000000000011111
   012345678901234
DD:####..###......   7 {Gas, Refit}
T1:...............   0 {}
T2:...............   0 {}
T3:#####mmm..##m..   7 {Roof, Refit}
T4:###mmm.####m###  10 {Gas, Roof, Refit}
T5:..m#####mmm##..   7 {Gas, Solar}
```

which has total maintenance load of 14 (1 active maintenance at times 2,3,4,6,7,8,9,10,11,12 = 10 load, and 2 active maintenances at time 5 = 4 load) and total service duration 24.

If you only finish to this stage you will be unable to acheive more than 75% marks for some instances.

8 Challenge Stage F: Major Maintenance

The remaining difficulty is to schedule major maintenances. Major maintenances are required on a truck every time it has performed `majorwork` time units of service tasks. The data to support this is defined by

```
int: majorwork;                  % max work allowed before major maintenance
array[TRUCK] of int: prework;    % units of work since last major maintenance
```

For each truck t we are given the amount of work they have done before the scheduling period, since the last major maintenance $prework[t]$. We must ensure that no truck performs more than `majorwork` time units of service since its last major maintenance, anywhere in the schedule. For long schedules we may need to include more than one major maintenance for a truck.

Given data

```
majorwork = 30;
prework = [ 0, 5, 10, 15, 20, 25 ];
```

the previous solution is no longer valid since T5 is assigned 7 more time units of service, which added to its 25 would lead to 32 without a major maintenance. Hence the best schedule changes to

```
truck = [T4, T1, T3, DD, T3, T4, T1, T3, DD, T4];
mt =
[| DD: NO, NO
 | T1: SH, LG
 | T2: NO, NO
 | T3: LG, SH
 | T4: SH, LG
 | T5: NO, NO
 |];
ms =
[| DD: 0,  0
 | T1: 5, 10
 | T2: 0,  0
 | T3: 5, 13
 | T4: 4,  8
 | T5: 0,  0
 |];
```

visualized as

```
   000000000011111
   012345678901234
DD:.......######..   6 {Gas}
T1:#####m###.mmm..   8 {Refit}
T2:...............   0 {}
T3:#####mmm..##.m.   7 {Roof, Solar}
T4:####m###mmm.###  10 {Gas, Refit, Solar}
T5:...............   0 {}
```

where we have stopped using T5 and used other trucks which have less prework. The maintenance load is 16 and service time is 25.

Note that this constraint can be tackled using the ideas of sequence dependent scheduling.

Stage G: Report

Obviously the data format for this problem is very complex. There are many implicit assumptions that are likely to hold for realistic data. But we need to be prepared for strange data. Consider the following conditions on the inputs to your model, for each of them explain in a paragraph what your model will do in these circumstances, and more importantly, why:

- `minservice = 0`.
- `minservice = 100`.
- `minwork = 0`.
- `minwork = 100`.
- `maxsty = [1,1,1,0]`.

- `maintdur[SH] > maintdur[LG]`.
- `maxsep[SH] > maxsep[LG]`.
- `maxsep[MJ] > maxsep[LG]`.
- `maxsep[SH] = maxsep[LG] = endtime`.
- `prework = [0 | t in TRUCK]`.
- `majorwork > endtime`.
- `maxsty[ty] = 0` for some `ty` other than `DUMMY`
- `start[s] = 0` and `end[s] = endtime` for some service `s`.
- `start` includes k services with the same start and end time where $k = card(TRUCK)$.
- `card(STYPE) = 1`

You will probably need to experiment by generating new data files. Note that some of these scenarios will have little or no effect, and some will have large and/or obvious effects. You will have to refer to other data values as well to explain some of these conditions. Do your best to explain why the change, or lack of change, results.

After explaining all the cases above, give a summary of what you think are the most important constraints in the model; i.e. those that are most changing the form of the answer. Justify your answer. The report should use no more than 4 A4 pages.

9 Instructions

Edit the provided `mzn` model files to solve the problems described above. You are provided with some sample data files to try your model on. Your implementations can be tested locally by using the *Run+check* icon in the MiniZinc IDE. Note that the checker for this assignment will only test whether your model produces output in the required format, it does not check whether your solutions are correct. The grader on the server will give you feedback on the correctness of your submitted solutions and models.

10 Marking

You will get a maximum of 50 marks for this assigment which is worth 25% of the units overall marks. Part of the marks are automatically calculated. The submission has 10 marks for locally tested data and 20 marks for model testing, for a total of 30 marks by automatic calculation. You will only get full marks if you implement all stages.

The 20 remaining marks are given for code comments and the report marked by the tutors, with the following allocation: Code Comments (6 marks) Report (14 marks).

For the autograded part of the marking you can get most marks having implemented Stages A-E only. Stage F is optional, without implementing it there will be some instances where you can get a maximum of 3/4 of the marks available.

Code commenting should clear explain the role of each variable, constraint and objective defined in the model. You should explain how every constraint is modelled unless this is straightforward: e.g. adding that we use `alldifferent` to ensure all of a set are different is unnecessary detail. You should use good identifier names for variables and procedures you define to help readability. The code and the comments must be formatted to be easy to read and comprehend.

The report requires a discussion of a number of forms of input to the model. Make sure its easy for the reader to determine what everything in this discussion refers to. The written report is worth 14 marks and should answer the questions listed in Stage G. The explanations should be clear and easy to interpret. If you wish you can support your explanation of the reasons for the change or lack thereof by referring to the results of experiments you conducted, but that by itself is not enough to explain behaviour.