FIT5216: Modelling Discrete Optimization Problems
Assignment 3: Train 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).
- 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 (24th May 2024, 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% of the available marks 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**. You may not use **large language models** for any part of this assignment. If in doubt, contact your teaching team with any questions!

2 Problem Statement

Your job is to build a detailed train timetable from a list of routes and services.

The rail network is made up of STOPs. We include a dummy stop `dstop` which does actually exist to pad arrays. Each stop has a minimal wait time to let passengers get on and off. Each stop has a `skip_cost` which is the cost if we skip the station in order to run the service faster (i.e. wait less than the minimal wait time). Each stop has a number of platforms available. Each stop is either an ordinary station, a hub station where many lines meet, or a terminus station where services can begin and end. There is a travel time matrix which records the travel time between two directly connected stops, or has absent `<>` if they are not connected. Each directly connected pairs of stops have a line type for the line connecting them: SING, a single track for both directions (so we need to have one train leave the connection before we can start a train in the opposite direction, and trains going the same direction cannot pass); DOUB, a single track for each of the two directions between the stops (so trains going the same directions cannot pass, but the two directions are independent); QUAD, two lines in each direction allowing arbitrary train passing in both directions; or NONE, there is no direct connection between the stops. There is a minimum separation `min_sep` (in terms of time units) between two services taking the same link from STOP to STOP in the same direction, if the connection is SING or DOUB.

```
enum STOP; % set of stops
STOP: dstop; % dummy stop
array[STOP] of int: minimal_wait;         % minimum wait time at station
```

```
array[STOP] of int: skip_cost;            % cost to skip the station (per service)
array[STOP] of int: platform;             % number of platforms at station
enum STYPE = { ORDINARY, HUB, TERMINUS };
array[STOP] of STYPE: stype;              % type of STOP
array[STOP,STOP] of opt 0..infinity: travel_time; % travel time from each stop to another
enum LINE = { SING, DOUB, QUAD, NONE };
array[STOP,STOP] of LINE: line;           % line type between stops
int: makespan;                            % end of time considered;
set of int: TIME = 0..makespan;
TIME: min_sep;                            % min separation of services same line same dirn
```

The services follow one of a fixed number of ROUTEs. There is a maximum route length `max_route_length`, and for each route its length `rlength` and the route itself is defined by an array of stops, with dummy stops filling in any unused stops for shorter routes.

```
enum ROUTE;                               % a sequence of STOPs
int: max_route_length;                    % longest route in number of stops
set of int: STOPNO = 1..max_route_length;
array[ROUTE] of STOPNO: rlength;          % length of routes
array[ROUTE,STOPNO] of STOP: route;       % route definitions;
```

The SERVICEs are the train services to be scheduled. Each has a route `sroute` as well as an earliest start time `service_start` and preferred end time `service_end`.

```
enum SERVICE;                           % the set of services to be schedules
array[SERVICE] of ROUTE: sroute;        % the route for each service
array[SERVICE] of TIME: service_start; % from when the service can start
array[SERVICE] of TIME: service_end;   % when the service ideally ends
```

Finally we need to assign engines to routes. There is a set of ENGINEs that run the services. Each engine has a start location. An engine will be used for multiple services, we connect them together using the `prev` decisions which maps each service to the previous service using the same engine, or an engine if this service is the first to use the engine.

```
enum ENGINE;                            % set of engines available
array[ENGINE] of STOP: start;           % start location for engines
enum SERVICEX = s(SERVICE) ++ e(ENGINE);  % extended type for prev array
```

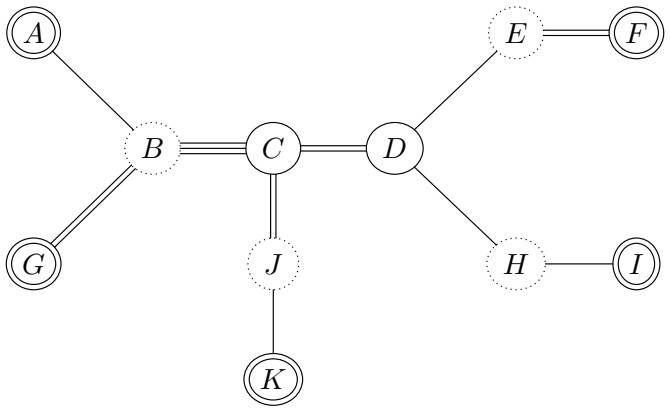
A sample data file `trains00.dzn` is given below for a network of 11 stops with a dummy, 5 routes, 11 services, and 7 engines.

```
STOP = {A,B,C,D,E,F,G,H,I,J,K,dummy};
dstop = dummy;
minimal_wait = [ 10, 4, 8, 8, 4, 10, 10, 8, 10, 6, 10, 0];
skip_cost = [0, 6, 0, 0, 20, 0, 0, 5, 0, 5, 0, 0 ];
platform = [2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 0];
```

```
stype = [TERMINUS, ORDINARY, HUB, HUB, ORDINARY, TERMINUS,
         TERMINUS, ORDINARY, TERMINUS, ORDINARY, TERMINUS, ORDINARY];
                % A   B   C   D   E   F   G   H   I   J   K  dummy
travel_time = [|  0,  7, <>, <>, <>, <>, <>, <>, <>, <>, <>, 0
               |  7,  0,  8, <>, <>, <>, 12, <>, <>, <>, <>, 0
               | <>,  9,  0, 10, <>, <>, <>, <>, <>,  5, <>, 0
               | <>, <>, 10,  0,  8, <>, <>,  4, <>, <>, <>, 0
               | <>, <>, <>,  8,  0, 15, <>, <>, <>, <>, <>, 0
               | <>, <>, <>, <>, 15,  0, <>, <>, <>, <>, <>, 0
               | <>, 12, <>, <>, <>, <>,  0, <>, <>, <>, <>, 0
               | <>, <>, <>,  4, <>, <>, <>,  0,  6, <>, <>, 0
               | <>, <>, <>, <>, <>, <>, <>,  6,  0, <>, <>, 0
               | <>, <>,  6, <>, <>, <>, <>, <>, <>,  0, 10, 0
               | <>, <>, <>, <>, <>, <>, <>, <>, <>, 10,  0, 0
               |  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0
               |];
line = [| NONE, SING, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE
        | SING, NONE, QUAD, NONE, NONE, NONE, DOUB, NONE, NONE, NONE, NONE, NONE
        | NONE, QUAD, NONE, DOUB, NONE, NONE, NONE, NONE, NONE, DOUB, NONE, NONE
        | NONE, NONE, DOUB, NONE, DOUB, NONE, NONE, SING, NONE, NONE, NONE, NONE
        | NONE, NONE, NONE, DOUB, NONE, DOUB, NONE, NONE, NONE, NONE, NONE, NONE
        | NONE, NONE, NONE, NONE, DOUB, NONE, NONE, NONE, NONE, NONE, NONE, NONE
        | NONE, DOUB, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE
        | NONE, NONE, NONE, SING, NONE, NONE, NONE, NONE, SING, NONE, NONE, NONE
        | NONE, NONE, NONE, NONE, NONE, NONE, NONE, SING, NONE, NONE, NONE, NONE
        | NONE, NONE, DOUB, NONE, NONE, NONE, NONE, NONE, NONE, NONE, SING, NONE
        | NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, SING, NONE, NONE
        | NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE, NONE
        |];
makespan = 240;
min_sep = 4;

ROUTE = anon_enum(5);
max_route_length = 6;
rlength = [ 6, 5, 6, 6, 5 ];
route = [| A, B, C, D, E, F
         | G, B, C, J, K, dstop
         | F, E, D, C, B, A
         | K, J, C, D, H, I
         | I, H, D, E, F, dstop
         |];

SERVICE = { R1a, R1b, R1c, R2a, R2b, R3a, R4a, R4b, R5a, R5b, R5c };
sroute = [ ROUTE[1], ROUTE[1], ROUTE[1], ROUTE[2], ROUTE[2], ROUTE[3],
           ROUTE[4], ROUTE[4], ROUTE[5], ROUTE[5], ROUTE[5]];
```

```
service_start = [ 0, 120, 120,  0,  90,  90,  0, 120,   0,  60, 120];
service_end   = [60, 150, 200, 50, 130, 150, 90, 220, 120,  90, 150];

ENGINE = { E1, E2, E3, E4, E5, E6, E7 };
start = [A, A, A, G, G, I, I];
```

The network encoded in the data file above has "shape" below: where TERMINUS are shown with two circles, HUB with one circle and ORDINARY as dotted circles. Connections are show as single line for SING, double line for DOUB and triple line for QUAD.



There are five routes $A \to B \to C \to D \to E \to F$, $G \to B \to C \to J \to K$, $F \to E \to D \to C \to B \to A$, $K \to J \to C \to D \to H \to I$, and $I \to H \to D \to E \to F$.

The decisions for the model are as follows:

```
array[SERVICE,STOPNO] of var TIME: arrive;    % arrive time at each stop
array[SERVICE,STOPNO] of var TIME: depart;    % depart time at each stop
array[SERVICE,STOPNO] of var TIME: wait;      % wait time at each stop
array[SERVICE,STOPNO] of var bool: stopped;   % which stops are actually made
var 0..infinity: delay_obj;                   % delay objective
var 0..infinity: skip_obj;                    % skip objective
array[SERVICE] of var ENGINE: engine;         % which engine is used for each service
array[SERVICE] of var SERVICEX: prev;         % previous service or engine
```

For each service and stop we have an arrival time at the stop, a depart time from the stop, and a wait time at the stop. For each service and stop we have a Boolean determining if we actually stopped there (for the minimal wait time) or we went straight through the stop without picking up passengers. We have two parts of the objective: the delay costs of the plan, and skip costs of the plan. Finally we need to decide which engine is assigned to each service, and which is the previous service using the same engine.

The assignment is in stages, please attempt them in order. Note that if you do not complete all stages you cannot get full marks.

Stage A - Schedule Constraints

Create a model `trains.mzn` that takes data in the format specified above and decides the arrival times, depart times and wait times for each service and stop. For stages A–B you can just set

`stopped` to be always true. For stages A–C you should leave the `engine` and `prev` variables unconstrained. For stages A–B leave `delay_obj` and `skip_obj` as unconstrained.

The schedule constraints are:

- No service starts (arrives) at its first station before it service start time.
- The wait time at each stop is the depart time minus the arrive time
- The wait time at each stop is greater than or equal to the minimal wait time if the service stopped at the stop (which for Stage A is always true)
- The arrival time at the next stop is at least the travel time from the previous stop to this one plus the departure time of the previous stop
- For dummy stops the arrival time is the departure time from the previous stop, the wait time is 0 and the departure time is the arrival time. This means all these times duplicate the real end time of the service.

A possible solution for the data above to stage A is:

```
R1a: A: 0-10-10 B: 17-4-21 C: 29-8-37 D: 47-8-55 E: 63-4-67 F: 82-10-92
R1b: A: 120-10-130 B: 137-4-141 C: 149-8-157 D: 167-8-175 E: 183-4-187 F: 202-10-212
R1c: A: 120-10-130 B: 137-4-141 C: 149-8-157 D: 167-8-175 E: 183-4-187 F: 202-10-212
R2a: G: 0-10-10 B: 22-4-26 C: 34-8-42 J: 47-6-53 K: 63-10-73
R2b: G: 90-10-100 B: 112-4-116 C: 124-8-132 J: 137-6-143 K: 153-10-163
R3a: F: 90-10-100 E: 115-4-119 D: 127-8-135 C: 145-8-153 B: 162-4-166 A: 173-10-183
R4a: K: 0-10-10 J: 20-6-26 C: 32-8-40 D: 50-8-58 H: 62-8-70 I: 76-10-86
R4b: K: 120-10-130 J: 140-6-146 C: 152-8-160 D: 170-8-178 H: 182-8-190 I: 196-10-206
R5a: I: 0-10-10 H: 16-8-24 D: 28-8-36 E: 44-4-48 F: 63-10-73
R5b: I: 60-10-70 H: 76-8-84 D: 88-8-96 E: 104-4-108 F: 123-10-133
R5c: I: 120-10-130 H: 136-8-144 D: 148-8-156 E: 164-4-168 F: 183-10-193
```

which shows for each service, the stop and arrival-wait-depart time. Note how the arrival + wait = departure, the minimal wait time constraint holds, and the travel time from one stop to another is satisfied.

Stage B - Platform Limits

Each stop only has a limited number of platforms. For this stage you need to add a constraint to ensure that no more than `platform[st]` trains are ever waiting at stop `st`. A train is at the stop from its arrival time to its departure time.

A possible solution for the data above to stage B is:

```
R1a: A: 0-10-10 B: 17-4-21 C: 29-8-37 D: 47-8-55 E: 63-4-67 F: 100-10-110
R1b: A: 120-10-130 B: 141-4-145 C: 153-8-161 D: 175-8-183 E: 191-4-195 F: 210-10-220
R1c: A: 120-10-130 B: 137-4-141 C: 149-8-157 D: 167-8-175 E: 183-4-187 F: 220-10-230
R2a: G: 0-10-10 B: 22-4-26 C: 37-8-45 J: 50-6-56 K: 66-10-76
R2b: G: 90-10-100 B: 112-4-116 C: 124-8-132 J: 146-6-152 K: 162-10-172
R3a: F: 90-10-100 E: 115-4-119 D: 127-8-135 C: 145-8-153 B: 162-4-166 A: 173-10-183
R4a: K: 0-10-10 J: 20-6-26 C: 32-8-40 D: 55-8-63 H: 67-8-75 I: 81-10-91
R4b: K: 120-10-130 J: 140-6-146 C: 157-8-165 D: 183-8-191 H: 195-8-203 I: 209-10-219
R5a: I: 0-10-10 H: 16-8-24 D: 28-8-36 E: 44-4-48 F: 63-10-73
```

```
R5b: I: 60-10-70 H: 76-8-84 D: 88-8-96 E: 104-4-108 F: 123-10-133
R5c: I: 120-10-130 H: 136-8-144 D: 148-8-156 E: 164-4-168 F: 183-10-193
A:R1a    0-10  R1b  120-130 R1c  120-130 R3a  173-183
B:R1a   17-21  R1b  141-145 R1c  137-141 R2a   22-26  R2b  112-116 R3a  162-166
C:R1a   29-37  R1b  153-161 R1c  149-157 R2a   37-45  R2b  124-132 R3a  145-153 R4a   32-40  R4b  157-165
D:R1a   47-55  R1b  175-183 R1c  167-175 R3a  127-135 R4a   55-63  R4b  183-191 R5a   28-36  R5b   88-96  R5c  148-156
E:R1a   63-67  R1b  191-195 R1c  183-187 R3a  115-119 R5a   44-48  R5b  104-108 R5c  164-168
F:R1a  100-110 R1b  210-220 R1c  220-230 R3a   90-100 R5a   63-73  R5b  123-133 R5c  183-193
G:R2a    0-10  R2b   90-100
H:R4a   67-75  R4b  195-203 R5a   16-24  R5b   76-84  R5c  136-144
I:R4a   81-91  R4b  209-219 R5a    0-10  R5b   60-70  R5c  120-130
J:R2a   50-56  R2b  146-152 R4a   20-26  R4b  140-146
K:R2a   66-76  R2b  162-172 R4a    0-10  R4b  120-130
```

We now show the time intervals where each stop is being used by different services. Notice how service `R1a` arrives later at stop F because service `R3a` is using the single platform at stop F until time 100.

Stage C - Objectives

From now on, we allow `stopped` variables to be set by the solver (i.e. dont set them all to true). We are only allowed to skip ORDINARY stations. We still must enforce that all the HUB and TERMINUS stations are stopped at for each service.

The delay objective tries to finish each service at the service end time. A service ends at its depart time from the last stop in the route. Calculate the `delay_obj` as the sum over all the services of the difference from the end of the service to its preferred service end time.

The skip objective pays a penalty for each stop that is skipped in a service. The n^{th} stop of service s if skipped if `stopped[s,n]` is set `false`. Note that if a service skips a stop the wait time for that service can be less than the minimal wait time of the stop. We pay a skip cost equal to the skip cost for that stop for each service that skips it. Calculate `skip_obj` as the sum of all skip costs.

Change your model to minimize the sum of `delay_obj` and `skip_obj`. Note how the two objectives work against each other.

A possible solution for the data above to stage C is:

```
R1a: A: 0-10-10 B: 17-4-21 C: 40-8-48 D: 58-8-66 E: 74-4-78 F: 100-10-110
R1b: A: 120-10-130 B: 141-4-145 C: 153-8-161 D: 175-8-183 E: 191-4-195 F: 212-10-222
R1c: A: 120-10-130 B: 137-4-141 C: 149-8-157 D: 167-8-175 E: 183-4-187 F: 202-10-212
R2a: G: 0-10-10 B: 22-4-26 C: 34-8-42 J: 47-6-53 K: 63-10-73
R2b: G: 90-10-100 B: 112-4-116 C: 124-8-132 J: 137-6-143 K: 153-10-163
R3a: F: 90-10-100 E: 115-4-119 D: 127-8-135 C: 145-8-153 B: 162-4-166 A: 173-10-183
R4a: K: 0-10-10 J: 20-6-26 C: 32-8-40 D: 50-8-58 H: 62-8-70 I: 76-12-88
R4b: K: 120-10-130 J: 143-6-149 C: 157-8-165 D: 183-8-191 H: 195-8-203 I: 209-10-219
R5a: I: 0-10-10 H: 16-8-24 D: 28-8-36 E: 44-4-48 F: 110-10-120
R5b: I: 60-10-70 H: 76-8-84 D: 88-8-96 E: 104-4-108 F: 123-10-133
R5c: I: 120-10-130 H: 136-8-144 D: 148-8-156 E: 164-4-168 F: 183-10-193
A:R1a    0-10  R1b  120-130 R1c  120-130 R3a  173-183
B:R1a   17-21  R1b  141-145 R1c  137-141 R2a   22-26  R2b  112-116 R3a  162-166
C:R1a   40-48  R1b  153-161 R1c  149-157 R2a   34-42  R2b  124-132 R3a  145-153 R4a   32-40  R4b  157-165
D:R1a   58-66  R1b  175-183 R1c  167-175 R3a  127-135 R4a   50-58  R4b  183-191 R5a   28-36  R5b   88-96  R5c  148-156
E:R1a   74-78  R1b  191-195 R1c  183-187 R3a  115-119 R5a   44-48  R5b  104-108 R5c  164-168
F:R1a  100-110 R1b  212-222 R1c  202-212 R3a   90-100 R5a  110-120 R5b  123-133 R5c  183-193
G:R2a    0-10  R2b   90-100
H:R4a   62-70  R4b  195-203 R5a   16-24  R5b   76-84  R5c  136-144
I:R4a   76-88  R4b  209-219 R5a    0-10  R5b   60-70  R5c  120-130
J:R2a   47-53  R2b  137-143 R4a   20-26  R4b  143-149
K:R2a   63-73  R2b  153-163 R4a    0-10  R4b  120-130
```

```
delay_obj = 312; skip_obj = 0;
```

Here we have not skipped any station, but paid a delay cost of 312: calculated as 50 + 72 + 12 + 23 + 33 + 33 + 2 + 1 + 0 + 43 + 43.

```
R1a: A: 0-10-10 B: 17-4-21 C: 29-8-37 D: 47-8-55 E: 63-4-67 F: 82-10-92
R1b: A: 120-10-130 B: 137-4-141 C: 151-8-159 D: 171-8-179 E: 187-4-191 F: 208-10-218
R1c: A: 120-10-130 B: 137-0-137 C: 145-8-153 D: 163-8-171 E: 179-4-183 F: 198-10-208
R2a: G: 0-10-10 B: 22-4-26 C: 34-8-42 J: 47-6-53 K: 63-10-73
R2b: G: 90-10-100 B: 112-4-116 C: 124-8-132 J: 137-0-137 K: 147-10-157
R3a: F: 92-10-102 E: 117-0-117 D: 125-8-133 C: 143-8-151 B: 160-0-160 A: 167-10-177
R4a: K: 0-10-10 J: 20-0-20 C: 26-8-34 D: 55-8-63 H: 67-0-67 I: 73-17-90
R4b: K: 120-10-130 J: 140-0-140 C: 153-8-161 D: 179-8-187 H: 191-0-191 I: 210-10-220
R5a: I: 0-10-10 H: 16-0-16 D: 20-8-28 E: 36-0-36 F: 102-10-112
R5b: I: 60-10-70 H: 76-0-76 D: 80-8-88 E: 96-0-96 F: 112-10-122
R5c: I: 120-10-130 H: 136-0-136 D: 140-8-148 E: 156-0-156 F: 171-10-181
A:R1a    0-10  R1b  120-130 R1c  120-130 R3a  167-177
B:R1a   17-21  R1b  137-141 R1c  137-137 R2a   22-26  R2b  112-116 R3a  160-160
C:R1a   29-37  R1b  151-159 R1c  145-153 R2a   34-42  R2b  124-132 R3a  143-151 R4a   26-34  R4b  153-161
D:R1a   47-55  R1b  171-179 R1c  163-171 R3a  125-133 R4a   55-63  R4b  179-187 R5a   20-28  R5b   80-88  R5c  140-148
E:R1a   63-67  R1b  187-191 R1c  179-183 R3a  117-117 R5a   36-36  R5b   96-96  R5c  156-156
F:R1a   82-92  R1b  208-218 R1c  198-208 R3a   92-102 R5a  102-112 R5b  112-122 R5c  171-181
G:R2a    0-10  R2b   90-100
H:R4a   67-67  R4b  191-191 R5a   16-16  R5b   76-76  R5c  136-136
I:R4a   73-90  R4b  210-220 R5a    0-10  R5b   60-70  R5c  120-130
J:R2a   47-53  R2b  137-137 R4a   20-20  R4b  140-140
K:R2a   63-73  R2b  147-157 R4a    0-10  R4b  120-130
delay_obj = 256; skip_obj = 132;
```

Here we have reduced the delay objective by skipping stops: e.g. `R2b` skips stop `J` (wait time of 0). The delay objective 256 = 32 + 68 + 8 + 23 + 27 + 27 + 0 + 0 + 8 + 32 + 31, while the skip objective is skipping B twice for 12, E 4 times for 90, H five times for 25 and J 3 times for 15 totalling 132. But overall this is worse than the solution above: $256 + 132 > 312 + 0$.

Notice that if we spend zero time waiting at a stop (i.e. we skip it) then we dont need a platform for the service (we assume there is a bypass track). Most methods of encoding the platform limit constraint will handle this correctly already, but you should think about it.

Stage D - Engine Assignment

Now we need to assign engines to each service, and assign a previous "service" to each service. The `prev` variables are the key decisions here:

- If `prev[`s`] = e(`e`)` then this means service s is the first to use the engine e. The start station for this service and the engine should be the same.
- If `prev[`s`] = s(`s'`)` then this means service s follows service s' using the same engine. The first stop for service s' should be the last stop for service s, and the start time for service s' should be no earlier than the end time of service s, and the engines used by s and s' should be the same.

Clearly two services cant have the same previous "service".

A sample solution for the data given is:

```
R1a: A: 0-10-10 B: 17-4-21 C: 29-8-37 D: 47-8-55 E: 63-4-67 F: 82-10-92
R1b: A: 120-10-130 B: 137-4-141 C: 149-8-157 D: 167-8-175 E: 183-4-187 F: 202-10-212
R1c: A: 121-10-131 B: 145-8-153 C: 161-8-169 D: 179-8-187 E: 195-4-199 F: 214-10-224
```

```
R2a: G: 0-10-10 B: 22-0-22 C: 30-8-38 J: 43-0-43 K: 53-10-63
R2b: G: 90-10-100 B: 112-0-112 C: 120-8-128 J: 133-6-139 K: 149-10-159
R3a: F: 92-10-102 E: 117-4-121 D: 129-8-137 C: 147-8-155 B: 164-4-168 A: 175-10-185
R4a: K: 63-10-73 J: 83-0-83 C: 89-8-97 D: 107-8-115 H: 119-0-119 I: 125-10-135
R4b: K: 159-10-169 J: 179-0-179 C: 185-8-193 D: 203-8-211 H: 215-8-223 I: 229-10-239
R5a: I: 5-21-26 H: 32-25-57 D: 61-12-73 E: 81-7-88 F: 107-13-120
R5b: I: 60-10-70 H: 76-8-84 D: 88-8-96 E: 104-4-108 F: 123-10-133
R5c: I: 135-10-145 H: 151-2-153 D: 157-8-165 E: 173-4-177 F: 192-10-202
A:R1a    0-10  R1b  120-130 R1c  121-131 R3a  175-185
B:R1a   17-21  R1b  137-141 R1c  145-153 R2a   22-22  R2b  112-112 R3a  164-168
C:R1a   29-37  R1b  149-157 R1c  161-169 R2a   30-38  R2b  120-128 R3a  147-155 R4a   89-97  R4b  185-193
D:R1a   47-55  R1b  167-175 R1c  179-187 R3a  129-137 R4a  107-115 R4b  203-211 R5a   61-73  R5b   88-96  R5c  157-165
E:R1a   63-67  R1b  183-187 R1c  195-199 R3a  117-121 R5a   81-88  R5b  104-108 R5c  173-177
F:R1a   82-92  R1b  202-212 R1c  214-224 R3a   92-102 R5a  107-120 R5b  123-133 R5c  192-202
G:R2a    0-10  R2b   90-100
H:R4a  119-119 R4b  215-223 R5a   32-57  R5b   76-84  R5c  151-153
I:R4a  125-135 R4b  229-239 R5a    5-26  R5b   60-70  R5c  135-145
J:R2a   43-43  R2b  133-139 R4a   83-83  R4b  179-179
K:R2a   53-63  R2b  149-159 R4a   63-73  R4b  159-169
E1:A R1b:A:120 - F:212
E2:A R1a:A:0 - F:92 R3a:F:92 - A:185
E3:A R1c:A:121 - F:224
E4:G R2b:G:90 - K:159 R4b:K:159 - I:239
E5:G R2a:G:0 - K:63 R4a:K:63 - I:135 R5c:I:135 - F:202
E6:I R5b:I:60 - F:133
E7:I R5a:I:5 - F:120
prev = [e(E2), e(E1), e(E3), e(E5), e(E4), s(R1a), s(R2a), s(R2b), e(E7), e(E6), s(R4a)];
delay_obj = 354; skip_obj = 37;
```

Note how the engine `E2` is used by `R1a` (`prev[R1a] = e(E2)`) going from stop `A` to `F` and then `R3a` (`prev[R3a] = s(R1a)`) going from stop `F` to `A`. Similarly engine `E5` starts with `R21` from `G` to `K`, then `R4a` from `K` to `I`, then `R5c` from `I` to `F`.

Recall that the test `prev[`s`] in e(ENGINE)` succeeds if the previous service of s is an engine (i.e. the service is the first to use an engine), and the expression `e^-1(prev[`s`])` returns the engine. Similarly the test `prev[`s`] in s(SERVICE)` succeeds if the previous service of s and is another service, and the expression `s^-1(prev[`s`])` returns the service.

Stage E - Double Track Constraints - Challenge Stage

A double track between stop st_1 to st_2 is two single tracks, one for each direction. This means while trains going in different directions are independent, trains going the same direction are constrained, by minimum separations constraints and cant pass. Add a constraint for each pair of services s_1 and s_2 on a double track which move from stop st_1 to st_2 that

- Either service s_1 departs station st_1 at least `min_dep` minutes after s_2, or vice versa.
- If service s_1 departs st_1 before s_2 then it arrives at st_2 at least `min_sep` minutes before s_2, and vice versa (if s_2 departs first, then it arrives first).

Note this can be quite challenging since gathering the data to express the constraints can be tricky.

A sample solution for the data given is:

```
R1a: A: 0-10-10 B: 17-4-21 C: 29-8-37 D: 47-8-55 E: 63-4-67 F: 82-10-92
R1b: A: 133-14-147 B: 154-4-158 C: 166-9-175 D: 185-8-193 E: 201-4-205 F: 220-10-230
R1c: A: 120-10-130 B: 137-4-141 C: 149-8-157 D: 167-10-177 E: 191-4-195 F: 210-10-220
R2a: G: 0-10-10 B: 22-4-26 C: 34-8-42 J: 47-6-53 K: 63-10-73
R2b: G: 90-10-100 B: 112-0-112 C: 120-8-128 J: 133-6-139 K: 149-10-159
```

```
R3a: F: 140-10-150 E: 165-4-169 D: 177-8-185 C: 202-8-210 B: 219-4-223 A: 230-10-240
R4a: K: 79-13-92 J: 102-6-108 C: 114-10-124 D: 134-8-142 H: 146-8-154 I: 160-10-170
R4b: K: 159-10-169 J: 179-1-180 C: 186-8-194 D: 204-8-212 H: 216-8-224 I: 230-10-240
R5a: I: 9-89-98 H: 104-8-112 D: 116-8-124 E: 132-4-136 F: 151-10-161
R5b: I: 98-10-108 H: 114-8-122 D: 126-8-134 E: 143-4-147 F: 162-10-172
R5c: I: 172-11-183 H: 189-2-191 D: 195-8-203 E: 211-4-215 F: 230-10-240
A:R1a    0-10  R1b  133-147 R1c  120-130 R3a  230-240
B:R1a   17-21  R1b  154-158 R1c  137-141 R2a   22-26  R2b  112-112 R3a  219-223
C:R1a   29-37  R1b  166-175 R1c  149-157 R2a   34-42  R2b  120-128 R3a  202-210 R4a  114-124 R4b  186-194
D:R1a   47-55  R1b  185-193 R1c  167-177 R3a  177-185 R4a  134-142 R4b  204-212 R5a  116-124 R5b  126-134 R5c  195-203
E:R1a   63-67  R1b  201-205 R1c  191-195 R3a  165-169 R5a  132-136 R5b  143-147 R5c  211-215
F:R1a   82-92  R1b  220-230 R1c  210-220 R3a  140-150 R5a  151-161 R5b  162-172 R5c  230-240
G:R2a    0-10  R2b   90-100
H:R4a  146-154 R4b  216-224 R5a  104-112 R5b  114-122 R5c  189-191
I:R4a  160-170 R4b  230-240 R5a    9-98  R5b   98-108 R5c  172-183
J:R2a   47-53  R2b  133-139 R4a  102-108 R4b  179-180
K:R2a   63-73  R2b  149-159 R4a   79-92  R4b  159-169
E1:A R1b:A:133 - F:230
E2:A R1a:A:0 - F:92 R3a:F:140 - A:240
E3:A R1c:A:120 - F:220
E4:G R2a:G:0 - K:73 R4a:K:79 - I:170 R5c:I:172 - F:240
E5:G R2b:G:90 - K:159 R4b:K:159 - I:240
E6:I R5b:I:98 - F:172
E7:I R5a:I:9 - F:161
prev = [e(E2), e(E1), e(E3), e(E4), e(E5), s(R1a), s(R2a), s(R2b), e(E7), e(E6), s(R4a)];
DOUB
B-G
C-D L(R1a):37-47 L(R1b):175-185 L(R1c):157-167 L(R4a):124-134 L(R4b):194-204
C-J L(R2a):42-47 L(R2b):128-133
D-E L(R1a):55-63 L(R1b):193-201 L(R1c):177-191 L(R5a):124-132 L(R5b):134-143 L(R5c):203-211
E-F L(R1a):67-82 L(R1b):205-220 L(R1c):195-210 L(R5a):136-151 L(R5b):147-162 L(R5c):215-230
B-G R(R2a):10-22 R(R2b):100-112
C-D R(R3a):185-202
C-J R(R4a):108-114 R(R4b):180-186
D-E R(R3a):169-177
E-F R(R3a):150-165
delay_obj = 587; skip_obj = 16;
```

The output shows all the services going left on the DOUB track segments, and their depart-arrive times on the segment; then the same information for services going right on the segment. Note how going from stop `E` to `F` services `R1b` and `R1c` overlap in the usage of the segment, but `R1c` departs and arrives at least `min_sep` minutes ahead of `R1b`.

Stage F - Single Track Constraints - Challenge Stage

A single track between stop st_1 to st_2 means we cant have two trains going in different directions on the track segment. Add a constraint for each pair of services s_1 and s_2 on a single track segment which move from stop st_1 to st_2 or stop st_2 to st_1 that

- If they both move from st_1 to st_2 then there is a `min_dep` difference in departure times, and a `min_dep` difference in arrival times, and the maintain the same order (the constraints for Stage E, applied to SING track segments).
- If they move in opposite directions then s_1 finishes using the segment before s_2 starts using the segment or vice versa

Note this is quite tricky to do well.

A sample solution for the data given is:

```
R1a: A: 0-10-10 B: 17-4-21 C: 29-8-37 D: 47-8-55 E: 63-4-67 F: 82-10-92
R1b: A: 132-11-143 B: 150-4-154 C: 162-8-170 D: 180-8-188 E: 196-4-200 F: 215-10-225
R1c: A: 122-10-132 B: 139-4-143 C: 151-8-159 D: 169-8-177 E: 185-4-189 F: 204-10-214
R2a: G: 0-10-10 B: 22-4-26 C: 34-8-42 J: 47-0-47 K: 57-10-67
R2b: G: 90-10-100 B: 112-4-116 C: 124-8-132 J: 137-6-143 K: 153-10-163
R3a: F: 92-10-102 E: 117-4-121 D: 129-8-137 C: 147-8-155 B: 164-4-168 A: 175-10-185
R4a: K: 67-10-77 J: 87-0-87 C: 93-8-101 D: 111-8-119 H: 123-0-123 I: 129-10-139
R4b: K: 163-10-173 J: 183-4-187 C: 193-8-201 D: 211-8-219 H: 223-0-223 I: 229-10-239
R5a: I: 2-55-57 H: 63-8-71 D: 75-8-83 E: 91-4-95 F: 110-10-120
R5b: I: 60-10-70 H: 76-8-84 D: 88-8-96 E: 104-4-108 F: 123-10-133
R5c: I: 139-10-149 H: 155-0-155 D: 159-8-167 E: 175-4-179 F: 194-10-204
A:R1a    0-10  R1b  132-143 R1c  122-132 R3a  175-185
B:R1a   17-21  R1b  150-154 R1c  139-143 R2a   22-26  R2b  112-116 R3a  164-168
C:R1a   29-37  R1b  162-170 R1c  151-159 R2a   34-42  R2b  124-132 R3a  147-155 R4a   93-101 R4b  193-201
D:R1a   47-55  R1b  180-188 R1c  169-177 R3a  129-137 R4a  111-119 R4b  211-219 R5a   75-83  R5b   88-96  R5c  159-167
E:R1a   63-67  R1b  196-200 R1c  185-189 R3a  117-121 R5a   91-95  R5b  104-108 R5c  175-179
F:R1a   82-92  R1b  215-225 R1c  204-214 R3a   92-102 R5a  110-120 R5b  123-133 R5c  194-204
G:R2a    0-10  R2b   90-100
H:R4a  123-123 R4b  223-223 R5a   63-71  R5b   76-84  R5c  155-155
I:R4a  129-139 R4b  229-239 R5a    2-57  R5b   60-70  R5c  139-149
J:R2a   47-47  R2b  137-143 R4a   87-87  R4b  183-187
K:R2a   57-67  R2b  153-163 R4a   67-77  R4b  163-173
E1:A R1b:A:132 - F:225
E2:A R1c:A:122 - F:214
E3:A R1a:A:0 - F:92 R3a:F:92 - A:185
E4:G R2a:G:0 - K:67 R4a:K:67 - I:139 R5c:I:139 - F:204
E5:G R2b:G:90 - K:163 R4b:K:163 - I:239
E6:I R5a:I:2 - F:120
E7:I R5b:I:60 - F:133
prev = [e(E3), e(E1), e(E2), e(E4), e(E5), s(R1a), s(R2a), s(R2b), e(E6), e(E7), s(R4a)];
DOUB
B-G
C-D L(R1a):37-47 L(R1b):170-180 L(R1c):159-169 L(R4a):101-111 L(R4b):201-211
C-J L(R2a):42-47 L(R2b):132-137
D-E L(R1a):55-63 L(R1b):188-196 L(R1c):177-185 L(R5a):83-91 L(R5b):96-104 L(R5c):167-175
E-F L(R1a):67-82 L(R1b):200-215 L(R1c):189-204 L(R5a):95-110 L(R5b):108-123 L(R5c):179-194
B-G R(R2a):10-22 R(R2b):100-112
C-D R(R3a):137-147
C-J R(R4a):87-93 R(R4b):187-193
D-E R(R3a):121-129
E-F R(R3a):102-117
SING
A-B L(R1a):10-17 L(R1b):143-150 L(R1c):132-139 R(R3a):168-175
D-H L(R4a):119-123 L(R4b):219-223 R(R5a):71-75 R(R5b):84-88 R(R5c):155-159
H-I L(R4a):123-129 L(R4b):223-229 R(R5a):57-63 R(R5b):70-76 R(R5c):149-155
J-K L(R2a):47-57 L(R2b):143-153 R(R4a):77-87 R(R4b):173-183
delay_obj = 371; skip_obj = 30;
```

The output shows all the services going left or right on the SING track segments, and their depart-arrive times on the segment.

Stage G - Report

In order for the input data to be correct it has to satisfy many requirements. For this problem the input data is complex and there are many ways to write it so that no solution is possible, or erroneous results are returned. In your model write MiniZinc assertions to check the input data satisfies the following conditions:

- Minimal wait times are all non-negative.
- Skip cost is non-negative, skip cost for non-ordinary stations is zero.

- Platform numbers are positive, except for the dummy stop.
- Wait times, skip cost and platform numbers for the dummy stop are all 0.
- Travel time to the dummy stop and from a stop to itself is 0.
- Minimum separation is non-negative.
- Service end is no earlier than service start for each service.
- The line type is symmetric, i.e. the line type from stop st_1 to st_2 is the same as the line type from stop st_2 to st_2.
- The line type agrees with the travel time, i.e. line type is NONE iff travel time is absent or travel time is 0.
- The route lengths agree with the definitions in route, i.e. if a route is length smaller than the max route length, then the extra stops are filled with dummy stops, and dummy stops only appear at the end of a route.
- Each two consecutive (non-dummy) stops occuring in a route are connected (not absent in travel time).

Make sure that your assertions produce clear error messages when the assertions are violated. Make sure the error message pinpoints exactly where the data goes wrong.

Explain in your report how you test each of the last three conditions, using a paragraph for each. Add small sample input files that violate each of the last three conditions and submit them with your report. In the report show what output your model gives for these sample files. These example should be generated by you *independently*, we will check for similarity. Note that none of these conditions is violated by the given input data or the hidden test data that does violate assertions. IMPORTANT: your last MiniZinc submission should have the assertions in it, this version will be run on hidden test data.

Most **importantly** explain what your model is likely to do if each assertion above was violated by the data but not tested and the model ran on the erroneous data. For example suppose the first assertion was violated, e.g. a stop had a negative wait time, then the model may treat this as if the minimal wait time was zero, or it may allow trains to depart earlier from the stop then they arrive. Which depends on the how the constraints for stage A are written. The explanation of how your model would react to erroneous data if the assertions were missing should be given for all 11 assertions above. We have made the first answer easy for you, what would your model do?

3 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. The default solver (and likely best) is `chuffed`, and this is the solver that will be used for model testing.

When you submit it is **important** that you do not print intermediate solutions. The default setup of the project will do this, but it means if you dont set a time limit, you may get no answer unless optimality is proven. So if you check the box: "Print intermediate solutions" in the solver configuration, while you are testing your model, remember to uncheck it before submission.

4 Marking

You will get a maximum of 50 marks for this assignment 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 15 for model testing, for a total of 25 marks by automatic calculation. You will only get full marks if you implement all stages.

The 25 remaining marks are given for code comments and the report marked by the tutors, with the following allocation: Code Comments (7 marks) Report (13 marks) Assertion testing (5 marks).

For the autograded part of the marking you can get most marks having implemented Stages A-D only. You will not get any marks unless you implement at least up to Stage C. Without implementing Stage D you can get at most 1/2 marks of that available. Stages E and 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 unecessary 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 should explain how the last three input data conditions are checked, and explain the result on the sample data file, as well as giving the output of your model on them. This is worth 6 of the 13 marks for the report.

The report should then explain the likely behaviour of data files that break the assertions if they were run without being checked. It should have 11 separate explanations, one for each assertion. It should reason about likely effects as shown in the example for the first assertion. Usually 1-2 sentence per assertion should be sufficient for the explanation. These explanations are worth 7 of the 13 marks for the report.

We will run your last MiniZinc submission against erroneous data to see that the assertions you added work correctly, and give meaningful errors messages. This is worth 5 marks of the assignment.