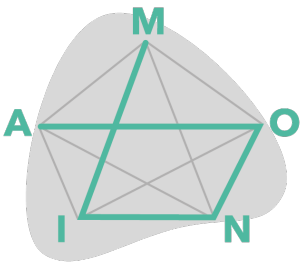

MINOA Research Challenge: Input/Output Description Format

M.A.I.O.R. srl, University of Pisa

Version 1.2, February 18, 2022

Contents

MINOA has received funding from the Europeans Union's EU Framework Programme for Research and Innovation Horizon 2020 under the Marie Skłodowska-Curie Actions Grant Agreement No 764759.

1 Introduction

This document provides the description of the format of the input JSON files in which test instances of the Integrated Timetabling and Vehicle Scheduling Problem (ITTVS) of the challenge (for more details, see [1, 2, 3]) will be provided to participants, and the format of the output JSON files in which the participants must produce the corresponding solutions.

For easier understanding, the names and the types of object are written in typewriter style following the syntax `name:Object`.

The format of input and output files is different according to the category: for 'Junior' or 'Senior' some data is not needed, i.e., some fields will not be present. We will describe the more general format, but we will make it clear which fields will not be present for Junior or Senior with the annotations [P] and [S]: the former indicates that the field is only required in the category 'Professional', while the latter indicates that the field is only required in the Professional and Senior categories.

2 Input

The input file is a JSON file containing all necessary information about the instance, organized in the following sections.

2.1 Time Horizon

The time horizon H represents the period of time (typically, a part of a day) in which activities need be planned. It consists of k *time windows* defined by $k+1$ time instants $t_0 < t_1 < \cdots < t_k$, where t_0 and t_k are the initial and final time instants of H. The time horizon is represented by `timeHorizon:List<int>` $= [t_0, \ldots, t_k]$. Any time-related quantity is expressed as an integer, measuring seconds. Times less than or equal to t_0 and times greater than t_k are considered external to the time horizon.

In the following, for all objects that have some properties related to time windows, these will be represented in a list with a size equal to k. The i-th element of the list (whatever its type), for $i = 1, \ldots, k$, is assumed to be relative to the i-th time window $(t_{i-1}, t_i]$. To refer to this property we will say that the list is 'indexed on H'.

2.2 Network nodes

The nodes of the *public transportation network* (PTN) are described by a list of network nodes `nodes:List<NetworkNode>`. For each network node, represented by a `node:NetworkNode` object, we will indicate:

- the name of the node (unique) with `nodeName:String`;
- the node's parking capacity `breakCapacity:int` [S];
- the electrics vehicle slow charge spots with `slowChargeCapacity:int` [S];
- the electrics vehicle fast charge spots with `fastChargeCapacity:int` [S];
- a list of minimum and maximum breaking time (indexed on H), with `breakingTimes:List<StoppingTime>`, where `stoppingTime:StoppingTime` is a simple structure containing:
  - the minimum stopping time, indicated with `minStoppingTime:int`;
  - the maximum stopping time, indicated with `maxStoppingTime:int`;

The node names are unique. Other objects will refer to a network node using his name. The object `nodes` will always contain a node with name '`dep`' which represent the only depot in the PTN.

2.3 Deadhead arcs

The list of deadhead arcs `deadheadArcs:List<DeadheadArc>` contains information about the links between the depot and the other nodes and vice versa. For each deadhead arc `deadhedArc:DeadheadArc`, we will indicate:

- the unique identifying code of the arc with `deadheadArcCode:int`;
- the name of the terminal node `terminalNode:String`;
- the type of the deadhead arc, `deadheadType:String{"pullOut","pullIn"}`, where `"pullOut"` means that the deadhead arc starts at the depot and ends at `terminalNode`, while `"pullIn"` means that the deadhead arc starts at `terminalNode` and ends at the depot;
- the length of the arc (in km) with `arcLength:float`;
- the list of travel times for each time window (indexed on H) with `travelTimes:List<Integer>`.

2.4 Directions

The directions in the lines in PTN are indicated as `directions:List<Direction>`. Each direction is described by a `direction:Direction` object and is characterized by the following fields:

- the (unique) line name with `lineName:String` (this has no use in the model, but it is useful to keep track of the practical meaning of the direction);
- The type of direction with `directionType:String{"inBound", "outBound"}` (this has no use in the model, but it is useful to keep track of the practical meaning of the direction);
- the start terminal node, `startNode:String`;
- the end terminal node, `endNode:String`;
- the list of headways for each time window (indexed on H) `headways:List<Headway>`; for each `headway:Headway` we indicate:
  - the minimum headway for that time window with `minHeadway:int` [P];
  - the ideal headway for that time window with `idealHeadway:int` [P];
  - the maximum headway for that time window with `maxHeadway:int`.
- a list of potential trips, with `trips:List<Trip>`. For each `trip:Trip` we will indicate:
  - the unique id of the trip with `tripId:int`;
  - the start time of the trip with `startTime:int`;
  - the end time of the trip with `endTime:int`;
  - the arrival time at the main stop of the route with `mainStopArrivalTime:int`;
  - the length (in km) of the trip with `lengthTrip:float`;
  - if the trip can be the initial or final trip for this direction with `isInitialFinalTT:String{"null", "initial", "final"}`.

2.5 Fleet

For the fleet `fleet:Fleet` we will indicate:

- the list of vehicleType with `vehicleList:List<VehicleType>`. For each vehicle type `vehicleType:VehicleType` we define:

  - the vehicle type name (say, `"lowRangeElectric"`, `"highRangeElectric"`, `"ICE"`) with `vehicleTypeName:String` (this has no use in the model, but it is useful to keep track of the practical meaning of the vehicle type);
  - its usage cost (a fixed cost, regardless the usage time) with `usageCost:float`;
  - the coefficient to be multiplied by the pull-in and pull-out time with `pullInOutCost:float`;
  - Each type of vehicle can be either a traditional Internal Combustion Engine (ICE) vehicle or an electric vehicle. If the type of vehicle is an ICE vehicle there is the field `iceInfo:IceInfo` containing its CO_2 cost coefficient `emissionCoefficent:float`. Otherwise the type of vehicle is electric and there is the field `electricInfo:ElectricInfo` containing the fields:
    1. The number available of vehicles of that type `numberVehicle:NumberVehicle` [S];
    2. Its autonomy (in km) `vehicleAutonomy:int` [S];
    3. Its maximum charging time (0-100% charging time, in seconds) `maxChargingTime:int` [S];
    4. Its minimum charging time (in seconds) `minChargingTime:int` [S].

    We remark that electric vehicles do not have any CO_2 cost coefficient, while there is no bound on the available number of ICE vehicles which have unlimited autonomy and do not need recharges.

- The fast charge coefficient, that is, a positive coefficient smaller than one that provides the maximum charging time on a fast charge spot as a fraction of that in a slow charge spot, `phi:float` [S].

2.6 Global Cost

This section includes the list of coefficients needed to define the objective function. In `globalCost:GlobalCost` we indicate:

- the break cost coefficient, that is the multiplicative coefficient for all unnecessary breaks (the break time spent at the nodes by a vehicle block which does not include the minimum stopping-times and the recharge times) `breakCostCoefficient:float`;
- the coefficients of the quadratic penalty function of the relative deviation of each (feasible) actual headway:
  - the quadratic coefficient `alpha2:float` [P];
  - the linear coefficient `alpha1:float` [P];
  - the constant term `alpha0:float` [P].

  Let be i, j two consecutive trips in the time table of one direction and v_{ij} the absolute value of the relative deviation of actual headway respect to the ideal headway, the penalty function is the following

$$p(v_{ij}) = \begin{cases} \texttt{alpha2}\, v_{ij}^2 + \texttt{alpha1}\, v_{ij} + \texttt{alpha0} & \text{if } v_{ij} > 0 \\ 0 & \text{if } v_{ij} = 0 \end{cases}$$

  The TT-cost is the sum of this penalty on every pairs of consecutive trips in the time table for each direction (for detail see [1]).

3 Output

The output format of a test instance solution contains all objects already described in the input format description and two new objects: the vehicle block list and the report of the solution.

To make the format easier to read, unlike the input file, the output file will not contain the entire list of potential trips per line, but only the trips belonging to the solution.

3.1 Vehicle block list

The list of vehicle blocks `vehicleBlockList:List<VehicleBlock>` contain the vehicle blocks that make up the output solution. Each vehicle block `vehicleBlock:VehicleBlock` contains:

- the vehicle type name, `vehicleTypeName:String`;
- the list of the activities `activityList:List<Activity>`. This list must be sorted by activity execution time. Each activity could be:
  - an `activityTrip:ActivityTrip` containing
    - the field `idTrip:int` relative to the trip performed by the vehicle block;
  - a `deadhead:Deadhead` that contains
    - the starting time `startingTime:int`;
    - the ending time `endingTime:int`;
    - the code of the deadhead arc used by the vehicle block `deadheadArcCode:int`;
  - an `break:Break`. Each break will include:
    - the name of the node where the break takes place `nameNode:String`;
    - the list of break time windows `breakTimeWindows:List<BreakTimeWindow>`. For each `breakTimeWindow:BreakTimeWindow` we indicate
      - the start time of the time window `startTime:int`;
      - the end time of the time window `endTime:int`;
      - the type of slot that the vehicle is occupying `typeSpot:String{"parking", "slowCharge", "fastCharge"}` [S];
      - if the vehicle is charging or not `isCharging:bool` [S].

      We remark that the field `isCharging` is useful to clarify whether a vehicle that occupies a charge spot is charging or used it to park. Note that if the vehicle remains on the charging slot after reaching 100% charge, the break time window has to terminate and it has to be followed by another break time window, in the same node, where the field `isCharging` is equal to false. If a vehicle is in a parking spot, the time spent in the parking spot is not considered as a recharge even if the field `isCharging` is true. Note that even if `endTime` and `startTime` are expressed in seconds the difference beetween the two must be an integer number of minutes, i.e., `endTime - startTime` must be a multiple of 60.

3.2 Report solution

The report of the solution `reportSol:ReportSol` contains the other information necessary for the evaluation of the solution within the context of the challenge:

- the list of CPU type `listCpuType:List<CpuType>`. For each `cpuType:CpuType` we indicate:
  - the description of CPU `description:String` that should contain a human readable description of the CPU type, maker and all relevant characteristics (number of cores/hardware threads, frequency), something like `"Intel(R) Xeon(R) Gold 5120 CPU @ 2194.845 MHz, 8 core, 16 threads"`;
  - the number of CPU cores of that type used to compute the solution (if the CPU cores have multiple hardware threads each thread count as a core unless the feature is disabled) `numberCpu:int`;
  - the CPU integer index of performance `cpuIntegerIndex:float`;
  - the CPU floating-point index of performance `cpuFloatIndex:float`;
- the objective function value `upperBound:float` [optional];
- the lower bound value `lowerBound:float`;

- the execution time (in seconds) that the algorithm required to compute the solution and the lower bound (excluding file reading and writing but including all the rest) `executionTime:float`; note that this time will be multiplied by the total number of CPU used, each scaled by its performance index as reported in the `cpuType` field.

The `upperBound` field is not mandatory. If the validator finds this field it compares it with the computed objective function value of the solution and returns a warning if the values are different. This can be useful in the initial phase of the challenge to facilitate alignment between the validator's objective function and that of the competing software. In the final phase of the challenge the validator will ignore this value.

4 Glossary

Headway: In 'transit speak', headway is the amount of time between transit vehicle arrival at a stop. A route that has a vehicle once an hour have a 60 minute headway.

Line: A line is a grouping of routes that is generally known to the public by a similar name or number

Route: A route is a link sequence, defined by an ordered sequence of (two or more) points on route. A *route* may pass through the same route point more than once, as in the case of a loop.

References

[1] *MINOA Research Challenge: Description problem - Professional*. https://minoa-itn.fau.de/?page_id=968.

[2] *MINOA Research Challenge: Description problem - Senior*. https://minoa-itn.fau.de/?page_id=968.

[3] *MINOA Research Challenge: Description problem - Junior*. https://minoa-itn.fau.de/?page_id=968.