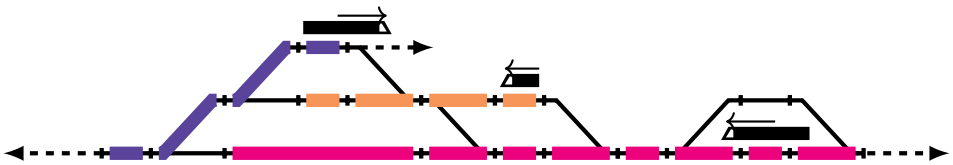


DISPLIB: Train Dispatching benchmark library

Effective management of dense railway traffic using algorithms has proven to be very hard. DISPLIB is a set of challenging problem instances derived from real-world railawy traffic management systems. This document describes the DISPLIB problem definition with the associated file format.

Any updates to this document will be published on the DISPLIB web page. On the web page you will also find the problem instance files and a Python solution verification program.

Changelog

- *2025-09-17*: Moved the problem and file format specification over from the competition document to this document.

Contents

1 DISPLIB file format

Given a set of trains traveling on a railway, the **Train Dispatching Problem** is the operational problem that occurs when some trains have become delayed with respect to their prescribed timetable, and we want to make routing and scheduling adjustments to minimize the total delay on the railway.

The DISPLIB problem format is a conceptual model and a JSON format for describing instances of the train dispatching problem. The format is designed to be as simple as possible while still being capable of capturing a wide range of real-world dispatching problems.

1.1 Conceptual model

Conceptually, the format describes a **set of trains** and an **objective**. Each train consists of a directed acyclic graph of operations, called the **operations graph**. This graph has exactly one node that has no incoming edges, called the **entry operation**, and exactly one node that has no outgoing edges, called the **exit operation**. Each operation has the following properties associated with it:

- A minimum duration, i.e., the shortest allowable time from the start of the operation to the end of the operation. The minimum duration may be zero time units.
- Lower and upper bounds on when the operation may start.
- A set of **resources**, all of which need to be exclusively allocated to the train to perform the operation. Each resource used by the operation also has a release time, which is an additional duration that the operation occupies the resource after the operation has ended. The release time may be zero time units.

Resources may represent anything that can only be used by one train at a time, but typically represent physical sections of the railway track that cannot be shared with other trains according to safety regulations. Note that some operations require more than one resource. For example, in Figure 1, operation 1 requires *both* of the resources r_1 and r_2, representing the fact that a long train is streching over two consecutive track sections. Note that the set of all resources defined for the problem is given only implicitly, as the union of the sets resources referred to by all the operations.

A **solution** to the problem consists of an ordered sequence of operation **start events** across all trains. Each start event specifies the start time of the corresponding operation. From this global sequence of events, we can extract the subsequence of events for each specific train. In the train's subsequence, any start event (except the first event) is also the **end event** of the train's

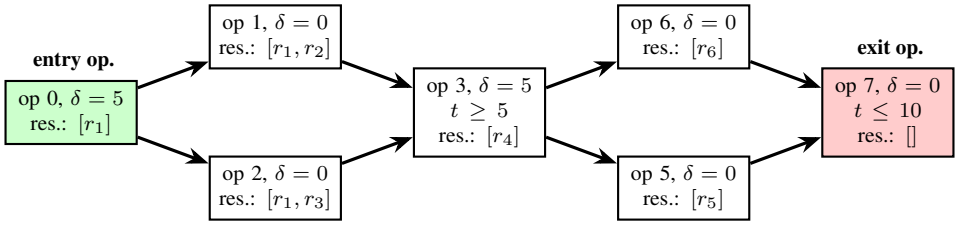


Figure 1: A train operation graph. The boxes represent operations, labeled with a minimum duration δ, and a list of resources $r_1, r_2, \ldots$ to which it needs exclusive access. The green node labeled "op 0" is the entry operation (it has no incoming edges), and the red node labeled "op 7" is the exit operation (it has no outgoing edges). The operations "op 3" and "op 7" have lower and upper bounds, respectively, defined on their start time t.

previous operation. All time and duration values must be given as non-negative integers. This solution sequence is feasible if:

- The sequence of events is given in chronological order, i.e., the time of each event is greater than or equal to the time of all preceding events.
- The subsequence of operations applying to each train forms a path through that train's operations graph, starting at the entry operation and ending at the exit operation.
- The time value of each start event satisfies the lower and upper bounds on the start time of the operation.
- The duration from each start event to the corresponding end event must be greater than or equal to the minimum duration of the corresponding operation.
- For any pair of operations o_1 and o_2, where:
  - the start event of o_1 precedes the start event of o_2 in the global sequence, and
  - o_1 and o_2 belong to different trains, and
  - o_1 and o_2 use a common resource r,

  the following must hold:
  - the end event for o_1 precedes the start event for o_2 in the global sequence, and
  - the duration from the end event of o_1 to the start event of o_2 is greater than or equal to the release time of resource r in o_1.

  Note that this implies that the global ordering of events is important beyond just the time values. A resource may be released by one train and allocated by another train at the same time value, but this is only feasible if the end event of the previous usage occurs before the next usage in the global ordering (see also the example in Sec. 2.3).

Note that the exit operation has no end event, and so any resources occupied by the exit operation of the train will never be released.

1.1.1 Objective

The objective defines the **cost** of a solution, and the goal of solving the DISPLIB problem instances is to find a feasible solution with the minimum cost.

The objective is described as a sum of objective components. Each objective component describes an **operation delay** as a threshold time $\bar{t}$ for when an operation becomes delayed, and

the cost associated with the delay. The cost value v_i of an operation delay i is:

$$v_i = c \cdot \max\left\{0, t - \bar{t}\right\} + d \cdot H(t - \bar{t})$$

where c and d are non-negative constants, t is the start time of the referenced operation, and H is the Heaviside step function that takes the value 0 for negative arguments and 1 otherwise. If that operation is not part of the solution, then $v_i = 0$. Note that, typically, either c or d is zero, but the formula contains both to be able to model both step-wise and linear functions of delay.

1.2 JSON format

The JSON object for a DISPLIB problem instance contains two keys: `trains` and `objective`, giving the following overall structure of the JSON file, where `...` is a placeholder:

```
{ "trains": [[{ ... operation ... }, ... ], ... ],
  "objective": [{ ... component ... }, ... ] }
```

1.2.1 Trains

The top-level `trains` key contains a list of trains, where each train is a list of operations. References to specific trains are given as a zero-based index into the `trains` list, and references to operations are given as a zero-based index into a specific train's list of operations. Each operation is a JSON object with the following keys:

- **`start_lb`** (optional, number): the earliest start time of the operation. Must be a non-negative integer. If the key is not present, defaults to 0.
- **`start_ub`** (optional, number): the latest start time of the operation. Must be a non-negative integer. If the key is not present, defaults to infinity (i.e., no upper bound).
- **`min_duration`** (number): the minimum duration, i.e., the minimum time between the start time and the end time of the operation. Must be a non-negative integer.
- **`resources`** (optional, list): a list of resources used by the train while performing the operation. If the key is not present, defaults to the empty list. Each resource usage is given as a JSON object with the following keys:
  - **`resource`** (string): the name of a resource.
  - **`release_time`** (optional, number): the release time for the resource, i.e., the minimum duration between the end time of this operation and the start time of any subsequent operation (of a different train) using the same resource. If the key is not present, defaults to 0.
- **`successors`** (list): a list of alternative successor operations, given as zero-based indices into the list of this train's operation. The list must be non-empty unless this operation is the *exit operation*.

For example, an operation that starts between time 7 and 8 and has a duration of 5 time units, uses the resource `r1`, and releases it immediately after the end event, and must be succeeded by the operation with index 2, would be formatted as follows:

```
{ "start_lb": 7, "start_ub": 8, "min_duration": 5,
  "resources": [{ "resource": "r1" }], "successors": [2] }
```

For each train, the list of operations is ordered topologically, i.e., all successors for an operation appear after it in the list. Note that this also means that the entry operation will always be at index 0, and the exit operation will always be the last operation in the list.

1.2.2 Objective

The top-level `objective` key contains a list of objective components. Each objective component is a JSON object containing the key `type` for determining an objective component type, and other keys depending on the type. Only one type, called **operation delay** (see Sec. 2.1), is defined in DISPLIB 2025 (the `type` key is included for forward compatibility).

The operation delay objective component is decribed as a JSON object with the following keys:

- **`type`** (string): must contain the string `"op_delay"`.
- **`train`** (number): reference to a train as an index into the top-level `trains` list.
- **`operation`** (number): reference to an operation as an index into the list defining the train's operation graph.
- **`threshold`** (number): a time after which this delay component is activated, as defined in the formula above. If the key is not present, defaults to 0.
- **`increment`** (number): the constant d in the formula above. Must be a non-negative integer. If the key is not present, defaults to 0.
- **`coeff`** (number): the constant c in the formula above. Must be a non-negative integer. If the key is not present, defaults to 0.

For example, an objective component that measures the time that train 0's operation 5 is delayed beyond time 10, would be formatted as follows:

```
{ "type": "op_delay", "train": 0, "operation": 5,
  "threshold": 10, "coeff": 1 }
```

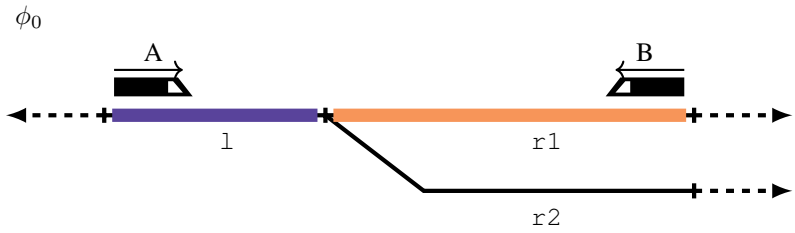
1.2.3 Solution format

Solutions to a DISPLIB problem are given as a JSON object (typically in a separate file from the problem instance), containing the following keys:

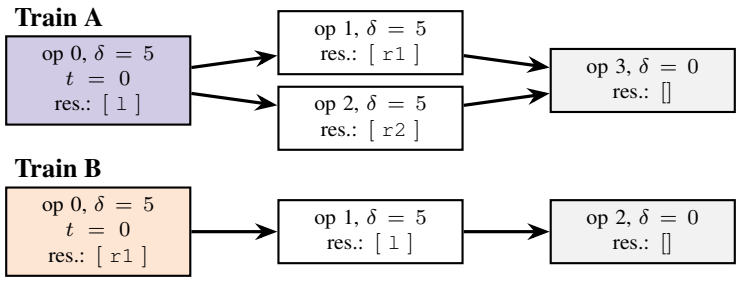
- **`objective_value`** (number): is the objective value of the solution (according to the `objective` from the problem instance).
- **`events`** (list): is an ordered list of start events, i.e. a list of references to operations to be started in the given order. Each start event is described by a JSON object with the following keys:
  - **`time`** (number): the time at which to start the operation. Must be a non-negative integer.
  - **`train`** (number): reference to a train as an index into the top-level `trains` list.
  - **`operation`** (number): reference to an operation as an index into the list defining the train's operation graph.

1.3 An example problem

Let's consider two trains meeting at a junction, coming from opposite directions. Train A is currently occupying the left part of the track (resource `l`), and may proceed either to the upper right track (resource `r1`), which train B is currently occupying, or to the lower right track (resource `r2`), which is currently free.



The operation graphs for the two trains would look as follows:



The objective is to minimize the time when train B finishes traveling through `l`. The JSON problem instance for this problem is:

```
{
  "trains": [
    [{ "start_ub": 0,
       "min_duration": 5,
       "resources": [{ "resource": "l" }],
       "successors": [1, 2] },
     { "min_duration": 5,
       "successors": [3],
       "resources": [{ "resource": "r1" }]},
     { "min_duration": 5,
       "successors": [3],
       "resources": [{ "resource": "r2" }]},
     { "min_duration": 0,
       "successors": []}],
    [{ "start_ub": 0,
       "min_duration": 5,
       "resources": [{ "resource": "r1" }],
       "successors": [1]},
     { "min_duration": 5,
       "resources": [{ "resource": "l" }],
       "successors": [2]},
     { "min_duration": 0,
       "successors": []}]],
  "objective": [{ "type": "op_delay",
                  "train": 1,
                  "operation": 2,
```

```
                    "coeff": 1}]
}
```

The following JSON object is a feasible (and optimal) solution to the problem instance:

```
{ "objective_value": 10, "events": [
  {"time": 0, "train": 0, "operation": 0},
  {"time": 0, "train": 1, "operation": 0},
  {"time": 5, "train": 0, "operation": 2},
  {"time": 5, "train": 1, "operation": 1},
  {"time": 10, "train": 1, "operation": 2},
  {"time": 10, "train": 0, "operation": 3}] }
```

Note that this is an example where the global ordering of events is important beyond the ordering implied by the time values: if we switch the ordering of the two events at time 5, the result is not a feasible solution since operation 1 of train B allocates `1` while it is still in use by train A.