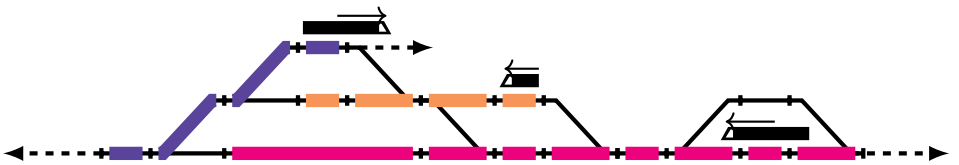


DISPLIB 2025

Train dispatching competition

Effective management of dense railway traffic using algorithms has proven to be very hard. The DISPLIB competition challenges you to advance the state-of-the-art in real-time train dispatching and contribute to more environmentally friendly transportation!

This document describes the DISPLIB competition (Sec. 1) and the formal problem definition and file format (Sec. 2).

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

- *2024-08-22*: Draft version.

Contents

1 The DISPLIB 2025 Competition

1.1 Spirit

The DISPLIB competition on train dispatching aims to encourage research into algorithms for optimized train dispatching that are suitable for real-time usage and for integration into in a real-world railway system. To encourage submission of algorithmic ideas that have not been carefully implemented or fine-tuned for real-time usage, the time limit per instance has been set to 10 minutes, which is higher than the 1-2 minute limit typically used in this application domain. Also, the competition will rely on self-reporting of time usage, and the organizers will only verify solution feasibility and objective values.

All the instances on which the algorithms will be evaluated are extracted from real-world railway information systems from different countries around the world. Instances with a wide range of characteristics will be used, including passenger-dominated versus freight-dominated railways, and coarse-grained infrastructure (stations and lines) versus fine-grained infrastructure (signals in station areas).

1.2 General rules

- Any programming language and computer may be used to solve the problem instances. The use of existing general-purpose optimization software is allowed (including commercial solvers such as CPLEX, Gurobi, Xpress, etc.).
- All reported results must be computed using no more than 8 CPU threads and 16 GB of RAM with a **time limit of 10 minutes per instance**.
- Problem instances and solutions will be given in the JSON formats described below. A solution will be considered feasible if the provided Python program for solution verification reports it to be feasible, and the objective value used to evaluate the submission's quality will be the one reported by the program. We have done our best for the verification program to adhere to the problem instance specification below. However, if any bugs are discovered in the verification program, the program may be modified and the new program announced on the DISPLIB web page during the competition. Deliberate exploitation of bugs in this program is not allowed, and such bugs should be reported to the competition organizers.
- Offline tuning or learning phases of the algorithm are allowed, and do not count against the time limit for computing the solution. However, adaptation email to specific problem instances are not allowed.

1.3 Evaluation criteria

The competition will be split into two phases, each with its own submission deadline:

- **Phase 1**: the first phase will contain some small test instances with corresponding optimal solutions, and a set of instances based on real-world use cases. The early phase instances will be publised on the DISPLIB web page on 1st October 2024.
  - Phase 1 submissions must be made at the latest on **31st January 2025** (AoE).

- **Phase 2**: second phase of the competition will contain larger and more challenging problem instances based on real-world use cases. The late phase instances will be

published on the DISPLIB web page on 15th February 2025.
  - Phase 2 submissions must be made at the latest on **30th April 2025** (AoE).

Points are awarded to each solution based on the position among its competitors and the phase (phase 2 instances are given more points). The top competitors will score points according to the scale in Table 1 for each instance in each phase. When there are two or more solutions tied for the same positions, the points granted by these positions are split evenly between competitors (rounded up in case of fractional points). When a team does not provide any solution for an instance, it is awarded zero points for that instance.

The ordering of all competitors will be based on the sum of points for all instances from both phases. The winner of the competition will be the solver with the highest total number of points. In the event of any ties for a position, the competitor with superior results (based on descending order of lexicographic ordering of points awarded) will gain precedence.

Table 1: Points awarded for an instance.

Position	Phase 1	Phase 2
1st	10	15
2nd	7	11
3rd	5	8
4th	3	6
5th	2	4
6th	1	3
7th		2
8th		1

1.4 Submission instructions

Everyone is welcome to participate! Collaboration in teams of any size is encouraged. To gain the most points, a team should send one submission per phase.

The submission should consist of the following:

- A short report describing the algorithm. The report should be maximum 10 pages in Springer LNCS format.
- A zip file containing at most one solution per problem instance in the phase.

The submission should be sent to the competition organizers within the deadline of the phase using the following email addresses:

- Giorgio Sartor <giorgio.sartor@sintef.no>
- Bjørnar Luteberget <bjornar.luteberget@sintef.no>

Preliminary results will be announced on the DISPLIB webpage after the first phase.

The top three teams, after evaluation following the last submission deadline, will be invited to present their algorithms at the International Conference on Optimization and Decision Science (ODS) 2025. At ODS 2025, the organizers will present a summary of results of the competition and the ranking of the teams, including which team is the winner, will

be announced. Final results and rankings for all teams will be published on the DISPLIB webpage after ODS 2025.

Please also watch the web page for amendments of the rules, instances, and the verification program. Such amendments will only be made when absolutely necessary to achieve a fair competition based on realistic problem instances in the spirit described in Sec. 1.1.

The competition is organized by Bjørnar Luteberget, Giorgio Sartor, Oddvar Kloster, and Carlo Mannino.

2 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.

2.1 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** that 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 the set of all relevant resources for the whole problem is given only implicitly, as the union of the sets resources referred to by 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 previous operation. All time and duration values must be given as non-negative integers. This solution sequence is feasible if:

- 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.

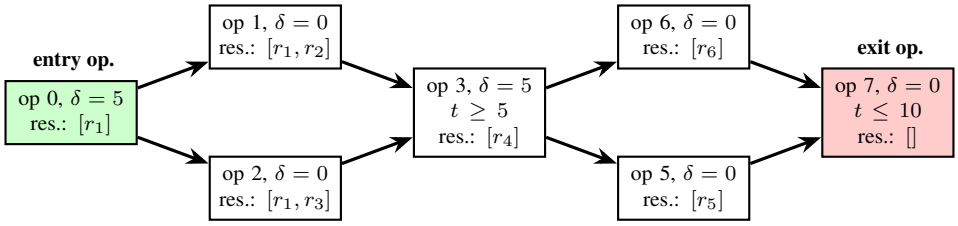


Figure 1: A train operation graph. The boxes represent operations, labeled with a minimum duration δ, and a list of resources $r_1, r_2, \ldots$ to which it needs exclusive access. The green node labeled "op 0" is the entry operation (it has no incoming edges), and the red node labeled "op 7" is the 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.

- The time value of each start event satisfies the lower and upper bounds on the start time of the operation.
- For each operation, the end event occurs no earlier than after the minimum duration has passed since the start event.
- For any start event for an operation o_1 that uses a resource r and any subsequent start events for another operation o_2 that also uses resource r:
  - the end event for o_1 must precede the start event for o_2, and
  - the start event for o_2 must occur no earlier than after the release time of resource r in o_1 has passed since the end event of o_1.

  Note that this implies that the global ordering of event 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.

2.1.1 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 for when an operation becomes delayed, and the cost associated with the delay. The cost value v_i of an operation delay i is:

$$v_i = c \cdot \max\left\{0, t - \overline{t}\right\} + d \cdot [t \geq \overline{t}]$$

where c and d are non-negative constants, t is the start time of the referenced operation, and $[t \geq \overline{t}]$ takes the value 1 if $t \geq \overline{t}$ and 0 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.

2.2 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 ... }, ... ] }
```

2.2.1 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 of the next operation to use this 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] }
```

2.2.2 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 trains' 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 }
```

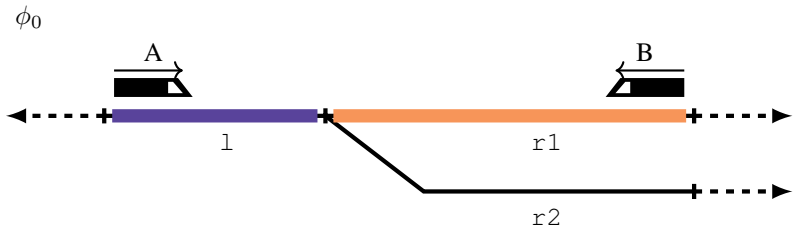
2.2.3 Solution format

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

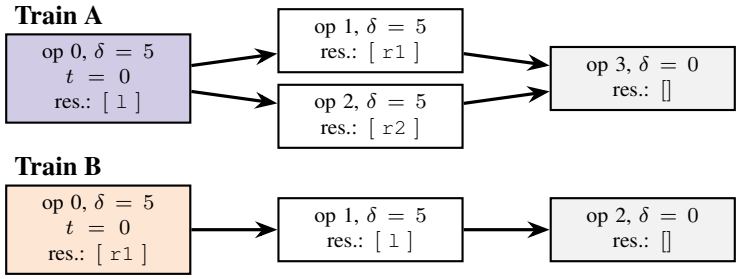
- **`objective_value`** (number): is the objective value of the solution (according to the `objective` from the problem instance).
- **`events`** (list): is an ordered list of start events, i.e. a list of references to operations to be started in the given order. Each start event is described by a JSON 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 trains' operation graph.

2.3 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": 5,
       "successors": []}],
    [{ "start_ub": 0,
       "min_duration": 5,
       "resources": [{ "resource": "r1" }],
       "successors": [1]},
     { "min_duration": 5,
       "resources": [{ "resource": "l" }],
       "successors": [2]},
     { "min_duration": 5,
       "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": 1, "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 of where the global ordering of events is important beyond the ordering implied by the time values: if we switch the ordering of the two middle events (both at time 5), the result is not a feasible solution since train 1's operation 1 allocates `1` before train 0's operation 0 ends.