

OPTIMA 2026 Hackathon

Airport Turnaround Scheduling — Keeping Melbourne Airport Moving

Optimisation Challenge

Melbourne Connect, 29 July 2026



It is midnight at Melbourne Airport (YMML). Over the next twenty-four hours, hundreds of aircraft will land, park, swap their crews, refuel, swallow a new load of passengers and bags, and push back from their gates. Every one of them needs a place to park and a small army of ground crews to turn it around. Get it right and the apron looks like a well-rehearsed dance. Get it wrong and you have angry passengers, missed connections, a queue of taxiing aircraft burning fuel, and a very expensive day.

You are the operations planner. Your job today is to turn a stack of flight data and an IATA service handbook into a plan that an airport could actually run. We have done the data plumbing for you, and you decide how to model and solve the problem.

What you will build

An optimisation solution for airport-operations problem: given the day's flight schedule, the apron layout, the ground-service crew roster, and the inter-gate travel matrix, produce a plan that assigns every flight to a gate, schedules every ground-handling task and allocates every task to a team unit. Hiring too many crews is wasteful; hiring too few is infeasible. The airport needs a plan that is both feasible and cost-effective.

You could use **use any optimisation technologies**, including MiniZinc, OR-Tools, Gurobi, a hand-rolled SAT encoding, or a pure Python heuristic. The only contract is the JSON instance file you read and the JSON solution file you produce.

Good luck. The apron is yours.

1 Gate Assignment

Every aircraft that lands needs a parking spot, and a spot at Melbourne airport is not just an arbitrary piece of tarmac. Each gate has a fixed stand size, sits at a specific terminal, supports either passenger or freighter operations, and has hours during which it is actually open.

Table 1 gives an example with five real flights. A flight can have different carrier, size and operation type. The carrier can be **domestic** or **international**, the size can be **narrow** or **wide**, and the operation type can be **passenger** or **freighter**. These attributes determine which gates are feasible for that flight.

Flight	Arrival	Departure	Carrier	Size	Type
JST917	00:31	01:16	domestic	narrow	passenger
UAE9314	12:34	14:49	international	wide	passenger
JST034	04:02	04:47	domestic	narrow	passenger
MXD177	04:16	05:01	domestic	narrow	passenger
CPA3128	14:15	16:30	domestic	wide	freighter

Table 1: Example flights and their attributes. All times are local wall-clock.

Table 2 gives an example with six gates to play with. Similarly, a gate can have different stand size, usage and supported type. A gate’s stand size can be **small** or **large**.

Gate	Stand Size	Usage	Supported Type	Open	Close
D02	small	domestic	passenger	00:00	24:00
D04A	small	domestic	passenger	00:00	24:00
D03	large	international	passenger	00:00	24:00
D04	large	international	passenger	00:00	24:00
H03	large	domestic	freighter	06:00	24:00
H03B	large	domestic	freighter	06:00	24:00

Table 2: Example gates and their attributes. All times are local wall-clock.

A feasible gate assignment must respect all of the following.

- **C1.** Every flight is parked at one gate, and the flight’s whole stay (from **arr** until **dep**) must fall inside that gate’s [**open**, **close**] window.
- **C2.** A wide-body flight needs a large stand. Narrow-body flights fit either size.
- **C3.** A passenger flight goes to a passenger gate, a freighter flight to a freighter gate.
- **C4.** An international flight must go to an international gate, and a domestic flight to a domestic gate. Customs takes a dim view of mixing the two.
- **C5.** To handle uncertainty, two flights assigned to the same gate must be separated by at least a **30-minute** inter-turn gap: if flight f_1 departs at **dep**₁ and flight f_2 arrives at **arr**₂ on the same gate, then **arr**₂ − **dep**₁ ≥ 30 minutes.

Once your gate assignment respects C1–C5 you have a **Tier 1** solution. Feasibility is easy on a few flights, and the problem gets interesting when we need to handle the roster of the whole day in our largest instance.

2 Team Scheduling

The aircraft has just landed and rolled into its gate. The clock is running. Before it can leave again it must finish a sequence of ground-handling tasks. Passengers disembark, baggage is unloaded, fuel is pumped, the cabin is cleaned, catering trolleys are loaded, fresh baggage goes on, the next passengers board, the doors close, and a tractor pushes the aircraft back. Some tasks can happen in parallel. Others have to wait. Each of them needs a specific kind of ground crew.

The airport offers you a list of teams that can do the work. You decide which teams to hire and which task each one handles. Hire too few and your schedule is infeasible because no crew is free when you need one. Hire too many and the roster is wasteful. We are looking for a plan with the *minimum total hiring cost* that still gets every aircraft out on time.

There are seven kinds of ground service teams, each capable of doing a specific set of tasks:

- **ramp_team**: ramp operations, including aircraft arrival securing (ARRIVE_SECURE), passenger disembarkation (DISEMBARK), international document (INTL_DOCS) and passenger boarding (BOARDING).
- **baggage_team**: baggage unloading (BAG_UNLOAD) and loading (BAG_LOAD), cargo unloading (CARGO_UNLOAD) and loading (CARGO_LOAD).
- **fuel_team**: aircraft refuelling (FUEL).
- **catering_team**: catering services (CATERING).
- **cabin_clean_team**: cabin cleaning (CABIN_CLEAN).
- **water_waste_team**: water replenishment and waste servicing (WATER_LAV).
- **pushback_team**: aircraft pushback (PUSHBACK) for passenger flight and departure release (DEP_RELEASE) for freighter.

The per-instance roster contains many individual teams of each kind, named `ramp_01`, `ramp_02`, and so on. There is an upper limit on how many teams of each kind you can hire, but whether you actually hire them all is up to you.

Every flight's task list is fixed by its *service profile*, namely the combination of carrier, aircraft size, and operation type. There are three structurally distinct task graphs, one per profile family. They are drawn in Figures 1, 2, and 3. Each task node is colour-coded by the team kind it requires: `ramp_team` `baggage_team` `fuel_team` `catering_team` `cabin_clean_team`

`water_waste_team` `pushback_team`. The nodes are also labelled with the duration range in minutes, in the form *narrow / wide*. Each task has a duration that may vary slightly from flight to flight but remains within the specified range.

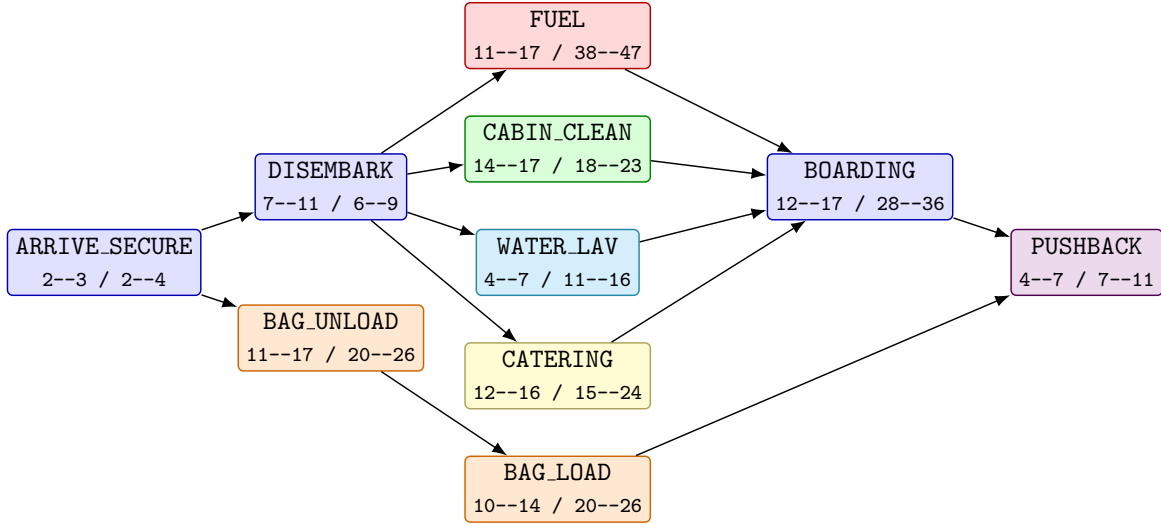


Figure 1: **Domestic passenger profile.** Durations in minutes.

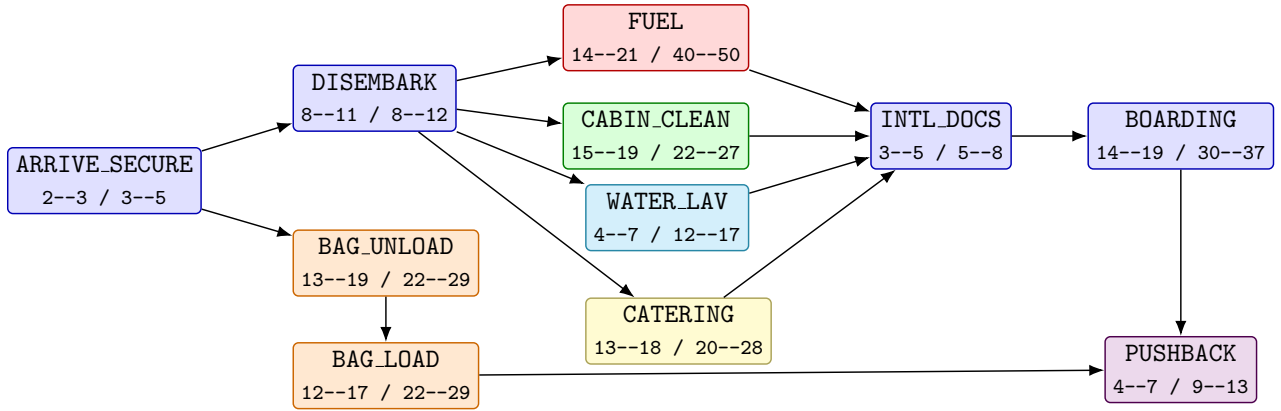


Figure 2: **International passenger profile.** Durations in minutes.

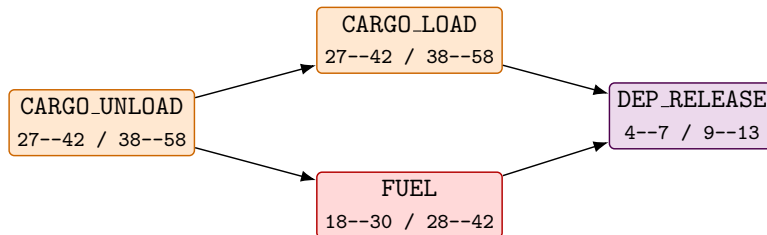


Figure 3: **Freighter profile.** Durations in minutes.

For every flight, we need to additionally decide the start times of each task, and which individual team unit does it. On top of C1–C5 which still must hold, the joint decision of gate assignment and team scheduling must satisfy the following constraints:

- **C6.** The team assigned to a task must be of the kind that task requires. For example, a FUEL task can only be done by a team whose kind is `fuel_team`.
- **C7.** Every task must start at or after the flight’s arrival and end at or before its departure.
- **C8.** Whenever a flight has two tasks whose kinds are in a precedence relation (see Figures 1, 2, and 3), the predecessor must finish before the successor starts.
- **C9.** A single team can only do one task at a time. Two tasks assigned to the same team, whether on the same flight or on different flights, must not overlap in time.

The objective is to minimise the total hiring cost, which is the sum of costs of all team units that are actually used in your plan. Each team has a fixed hiring cost, and two different units of the same kind can (and typically do) differ slightly, so the model must pick individual units rather than treat teams as interchangeable within a kind. Let U be the set of **TeamIDs** that appear at least once in the plan and c_t be the cost of team t . The objective is

$$\min \sum_{t \in U} c_t.$$

Teams that never appear are not hired and cost nothing. Once all constraints C1–C9 are satisfied, you have a **Tier 2** solution.

3 Routing and Rest

The draft plan looks perfect on paper, but you notice two things almost immediately. First, a single fuel-truck unit finishes at gate B22 and may start at H03B one minute later, but the truck takes time to drive across the airfield. Second, a ramp team who has just spent ninety minutes back-to-back on three turnarounds is going to need a break before the next one.

Suppose every team can travel around the apron at the same speed, and that the time to travel between any two gates is known. Two final constraints we need to consider before nailing down the plan are travel time and human fatigue:

- **C10.** A team unit must have enough time to travel between the gates of two consecutive tasks assigned to it. If two tasks are assigned to the same gate, the travel time is zero of course.
- **C11.** Before a team unit starts any task, it must have had at least 15-min idle time for a break within the preceding 90 minutes. Time before a unit's first task counts as idle. Note that travel time between gates does not count as idle time. For example, if a unit does not have any breaks for the last 80 minutes, and it travels for 15 minutes to the next gate, then it needs a 15-minute break before starting the next task.

The instance ships a symmetric integer matrix `travel` indexed by gates, where `travel[g1][g2]` gives the minutes required to move a team unit from gate `g1` to gate `g2`. Same-gate transitions are zero (`travel[g][g] = 0`). For any two of your task assignments (f_1, t_1) and (f_2, t_2) that are assigned to the same team unit, the gap between them must cover the travel time between the two parking gates of f_1 and f_2 :

$$\text{task_start}[f_2, t_2] \geq \text{task_end}[f_1, t_1] + \text{travel}[\text{gate}[f_1], \text{gate}[f_2]].$$

if $\text{task_end}[f_1, t_1] \leq \text{task_start}[f_2, t_2]$. When the two gates are at opposite ends of the airfield it can be the deciding factor that forces you to hire more teams to finish all tasks.

Satisfying all eleven constraints gives a **Tier 3** solution. Constraints C10 and C11 typically push the total hiring cost *up* compared with what C1–C9 alone would allow, because a unit that could previously chain two distant turnarounds now needs travel slack or rest, and you may need to hire an extra unit of that kind to absorb it.

4 Data Schema

The instances and submissions are both in JSON format. The following shows an example instance with five flights and six gates with a six hour horizon.

Listing 1: Instance JSON

```
{
  "horizon": 1440,
  "FlightID": ["JST917", "UAE9314", "JST034", "MXD177", "CPA3128"],
  "GateID": ["D02", "D04A", "D03", "D04", "H03", "H03B"],
  "TeamID": ["ramp_01", "ramp_02", "baggage_01",
    /* ... one entry per team unit ... */],
  "flights": [
    { "arr": 31, "dep": 76,
      "carrier": "domestic", "size": "narrow", "op_type": "passenger",
      "tasks": [
        {"kind": "ARRIVE_SECURE", "duration": 2},
        {"kind": "DISEMBARK", "duration": 7},
        {"kind": "BAG_UNLOAD", "duration": 11},
        {"kind": "FUEL", "duration": 11},
        {"kind": "CABIN_CLEAN", "duration": 14},
        {"kind": "WATER_LAV", "duration": 4},
        {"kind": "CATERING", "duration": 12},
        {"kind": "BAG_LOAD", "duration": 10},
        {"kind": "BOARDING", "duration": 12},
        {"kind": "PUSHBACK", "duration": 4}
      ]
    },
    /* ... one entry per flight ... */
  ],
  "gates": [
    {"stand_size": "small", "usage": "domestic",
      "op_type": "passenger", "open": 0, "close": 1440},
    /* ... one entry per gate ... */
  ],
  "teams": [ {"kind": "ramp_team", "cost": 9},
    {"kind": "ramp_team", "cost": 9},
    {"kind": "baggage_team", "cost": 8},
    {"kind": "fuel_team", "cost": 20},
    {"kind": "cabin_clean_team", "cost": 8},
    {"kind": "water_waste_team", "cost": 11},
    {"kind": "catering_team", "cost": 11},
    {"kind": "pushback_team", "cost": 18},
    /* ... one entry per team unit ... */
  ],
  "travel": [[0, 1, 1, 1, 1, ...], [1, 0, 1, 1, 1, ...], ...]
}
```


The instance JSON has several top-level keys:

- **horizon** is the length of the instance window, in 1-minute slots. Note that times are integer 1-minute slots counted from the start of the instance window: **arr** = 31 means 00:31, **arr** = 754 means 12:34, and so on.
- **FlightID**, **GateID**, and **TeamID** are ordered arrays of identifiers. The order fixes the row/column layout everywhere else: **flights**[*i*] describes the flight named **FlightID**[*i*], and so on for gates, teams and the travel matrix.
- **flights** is an array of records, one per flight, holding everything specific to that flight (time window, service profile, ordered task list). The **tasks** array length varies with the profile: 10 entries for domestic narrow passenger, 11 for international, 4 for freighter.
- **gates** is an array of records, one per gate.
- **teams** is an array of records, one per team unit. Each team has a **kind** and a hiring **cost**.
- **travel** is a $|\text{GateID}| \times |\text{GateID}|$ symmetric matrix with non-negative entries in 1-minute slots.

Note that several fields are enums: flight attributes (**carrier**, **size** and **op_type**), gate attributes (**stand_size**, **usage** and **op_type**) and team **kind** are drawn from a closed set of values, not arbitrary strings.

Solution schema. A submission is also a JSON object with four keys:

- **gate**: array sized $|\text{FlightID}|$, holding the chosen **GateID** for each flight.
- **task_start**: 2D array sized $|\text{FlightID}| \times \text{max_tasks}$ holding the start slot (1-minute) of each task, where **max_tasks** is the length of the longest task list. Padding entries beyond a flight's task count are ignored and should be set to zero.
- **task_team**: 2D array sized $|\text{FlightID}| \times \text{max_tasks}$, holding the **TeamID** performing each task.
- **cost**: the total hiring cost.

Note that **task_end** is not part of the solution, since **task_start**[*f*, *t*] + **flights**[*f*].**tasks**[*t*].**duration** by definition gives the end time of task *t* of flight *f*.

The following shows a submission for the first flight of the example instance, using seven distinct team units from the roster. Every task starts no earlier than 31 and ends no later than 76, the precedence graph is respected, and no team is asked to be in two places at once. The hiring cost is the sum of the seven distinct teams' costs: $9 + 8 + 20 + 8 + 11 + 11 + 18 = 85$.

Listing 2: Example solution submission

```
{
  "gate":      ["D02", /* ... one GateID per flight ... */],
  "task_start": [[31, 33, 33, 40, 40, 40, 40, 44, 54, 66],
                 /* ... one row per flight ... */],
  "task_team": [ ["ramp_01", "ramp_01", "baggage_01",
                  "fuel_01", "cabin_clean_01",
                  "water_waste_01", "catering_01",
                  "baggage_01", "ramp_01", "pushback_01"],
                 /* ... one row per flight ... */],
  "cost": 85
}
```

5 Competition and Ranking

There are multiple instances in this competition. Each instance is scored independently, and your total score is the sum across instances. The leaderboard ranks participating teams by that total.

Points on a single instance are awarded across three tiers. Tier 1 is a flat award as soon as the gate-feasibility constraints hold; Tier 2 and Tier 3 add points that scale with how cheap your plan is compared with the current best on the leaderboard. Table 3 summarises the scoring scheme.

Tier	Constraints	Points formula	Max per instance
1	C1–C5	20	20
2	C1–C9	$50 \times \min(1, \text{best_tier2_cost}/\text{cost})$	50
3	C1–C11	$30 \times \min(1, \text{best_tier3_cost}/\text{cost})$	30

Table 3: Per-instance points scheme.

Suppose `best_tier2_cost` and `best_tier3_cost` are the best costs for Tier 2 and Tier 3 solutions seen so far on the leaderboard for this instance. If your submission satisfies C1–C9 and reports a cost of `cost`, you get 20 points for Tier 1 plus up to 50 points for Tier 2, scaled by how close your cost is to the best so far. If your solution further also satisfies C10 and C11, you can earn up to 30 additional points for Tier 3, again scaled by the best Tier 3 cost so far.

Worked example. On some medium instance, the leaderboard currently shows a Tier-2 best of 220 and a Tier-3 best of 240. You submit a plan that satisfies all eleven constraints and reports a cost of 245. If this cost matches the plan decisions and all constraints C1–C11 are satisfied, your points on that instance are

$$20 + 50 \times \min\left(1, \frac{220}{245}\right) + 30 \times \min\left(1, \frac{240}{245}\right) \approx 20 + 44.9 + 29.4 = 94.3 \text{ points.}$$

The same 245-cost plan submitted with C10 or C11 violated (Tier 2 only) would score $20 + 44.9 = 64.9$ points on that instance. A pure Tier-1 submission with any cost scores a flat 20.

Submitting. You can submit as often as you like through the competition website before 15:30 today. The leaderboard keeps your best solution per instance. The leaderboard will be hidden half an hour before the competition deadline, and the final scoring will be done on the best submission you made.