Optimization Challenge 2026

A fast-growing vending company operates a large number of automated machines distributed across a city. Each machine generates revenue only when it is stocked, operational, and properly maintained, and customers expect minimal downtime. Delays in service lead not only to lost sales, but also to reduced customer satisfaction and long-term reliability issues. To remain competitive, the company must carefully plan its daily service operations, deciding when and where to deploy its service units so that needs are addressed on time, preventive actions are taken when beneficial, and travel effort is kept under control.

The company operates a set of machines located at known sites across the city. All service units (vehicles) must begin and end their routes at the depot, with all departures and returns occurring strictly within the day's start and end time. All machines require mandatory replenishment during the day. For each such machine, service has to *start* within a given time window, between earliest allowable time and the latest acceptable deadline. A vehicle that arrives earlier than desired should wait for the earliest allowable service time. Travel times between locations are not time-dependent (i.e., do not change throughout the day). There are a given number of vehicles available. Replenishment at a machine requires a fixed service time. The depot has a fixed closing time, and all vehicles are required to return to the hub by that time (i.e., *day end*).

At the beginning of the day, the company dispatches vehicles according to a given **initial route plan**. Until mid-day, the company observes operations and only records any machine **failures** that occur while vehicles are already en route and may have completed some planned stops. At a fixed decision time (**mid-day**), the company must solve a single **recourse action problem**: given the *current state* of the system (vehicle locations, partial route completion, and the set of failed machines observed so far), update the remainder of the routes for the rest of the day. A trip that began before mid-day for a service must be completed together with that service. If a replenishment service is in progress at mid-day, the vehicle's remaining plan can be updated once that service is completed. Failed machines have to be visited as soon as possible. Service at a failed machine requires a fixed service time. A failure removes the time window restriction on replenishment for the failed machine (if any). Meanwhile, mandatory **replenishment** requirements for non-failed machines must still be satisfied within their specified time windows.

Service Types

A visit to a machine may include one or more of the following actions:

- **Failure repair** (if a failure has occurred at that location),
- **Replenishment** (mandatory for all machines and must occur strictly within the specified time window—except when a machine has failed, in which case the time-window restriction is lifted).

If multiple actions are performed in the same visit, their service times are additive.

Initial Routes

You are given a feasible set of initial routes (one route per vehicle), including the planned visit sequence and planned service start times. Your task is not to construct routes from scratch, but to **update** these routes mid-day after failures are revealed.

Service Severity Information

Each machine has a known demand rate that reflects how critical or costly it is to delay a failure service at that location.

Objective (Weighted Penalty Minimization)

Upon receiving failure information mid-day, you must update the vehicle routes to minimize a weighted penalty that consists of two components below, while ensuring that replenishment windows are met:

1. **Failure response (highest priority):** visit the failure service locations as soon as possible, with penalties weighted by demand rate/criticality.
2. **Conformance to the initial plan:** keep the updated routes as close as possible to the given initial routes (minimize deviation).

A generic scoring form is:

$$\min \quad \lambda_F \cdot \textbf{Failure penalty} + \lambda_D \cdot \textbf{Deviation penalty},$$

where $\lambda_F > 0$ and $\lambda_D > 0$ reflect the intended priority structure (the organizer will provide weight scenarios).

Penalty Definitions

The penalty functions are computed as follows:

- **Failure penalty:** Sum of weighted response times, e.g., $\sum_m c_m \cdot (a_m - f_m)$, where c_m denotes the criticality of failure for vending machine m, f_m is the time failure occurs (before mid-day), and a_m is the time service unit starts working on the failed machine m. a_m is assumed the end of next day (i.e., $\textit{day end} + \textit{day end} - \textit{day start}$) for unaddressed failures of the day .
- **Deviation penalty:** A route-disruption measure, defined as the number of machines whose vehicle assignment changes. Changes to the within-route customer sequence or route length incur no penalty.

Task

Given:

- depot location, a set of machine locations, and travel times,
- an initial feasible route plan for the vehicles,
- a stream of failures with timestamps and priorities,

design a method that updates the routes online to minimize the weighted penalty described above.

Expected Deliverables

You need to submit the following via email (opt-challenge@sabanciuniv.edu) by **April 3, 2026**:

1. A short report presenting:
   - Your solution approach.
   - Relevant implementation details, including the choice of programming language, solver, or any additional software (if applicable), algorithm-specific parameter choices, and where you ran the experiments.
2. Your code file(s).
3. Your solution files in a zip folder:
   - Make sure to submit a separate solution file for each individual instance in the `instances` folder and name it using the convention `sol_instance_ID` where `ID` is the ID number of the instance (ranging from 1 to 15). For example, your solution file for `instance_1.json` should be named `sol_instance_1.json`.
   - See the `example` folder for a sample solution file for `instance_0.json`. You must have a solution file in the given format for every instance you solve. Submissions that do not conform to the sample solution file structure will not be considered.
   - See `readme.pdf` for further information on json structure.
4. Proof of student certificate for each group member (recall that only undergraduate students are eligible).

Evaluation Criteria

Evaluations will be based on the quality of the submitted solutions. The quality of a solution is measured by the score defined above (the smaller, the better). For each problem instance, submissions will be compared against the best obtained solution, and teams will be ranked according to their average relative gap across all problem instances.

Note that you need to register in order to access the problem instances.