# SOLVING A STATIC AND DYNAMIC VRP WITH TIME WINDOWS

**OptiML**

Jasper van Doorn, Leon Lan, Luuk Pentinga, Niels Wouda

7 December 2022

# STATIC SOLVER

High-level overview:

- Simplify given baseline
- Tweak local search
- (and more that we do not have time for)

# SIMPLIFY BASELINE

- First, we refactored the entire codebase, and added Python bindings.
- Then, we removed:
  - constructive heuristics
  - circle sectors
  - giant tour representation and split algorithm
  - dynamic growth of granular neighbourhoods and minimum population size

> *Our `Config` object has 26 fields, compared to the baseline's 42.*

# TWEAK LOCAL SEARCH

Noteable differences:

Baseline:

- Always at least two iterations. Later iterations test against empty routes.
- Probabilistically apply intensification.

Ours:

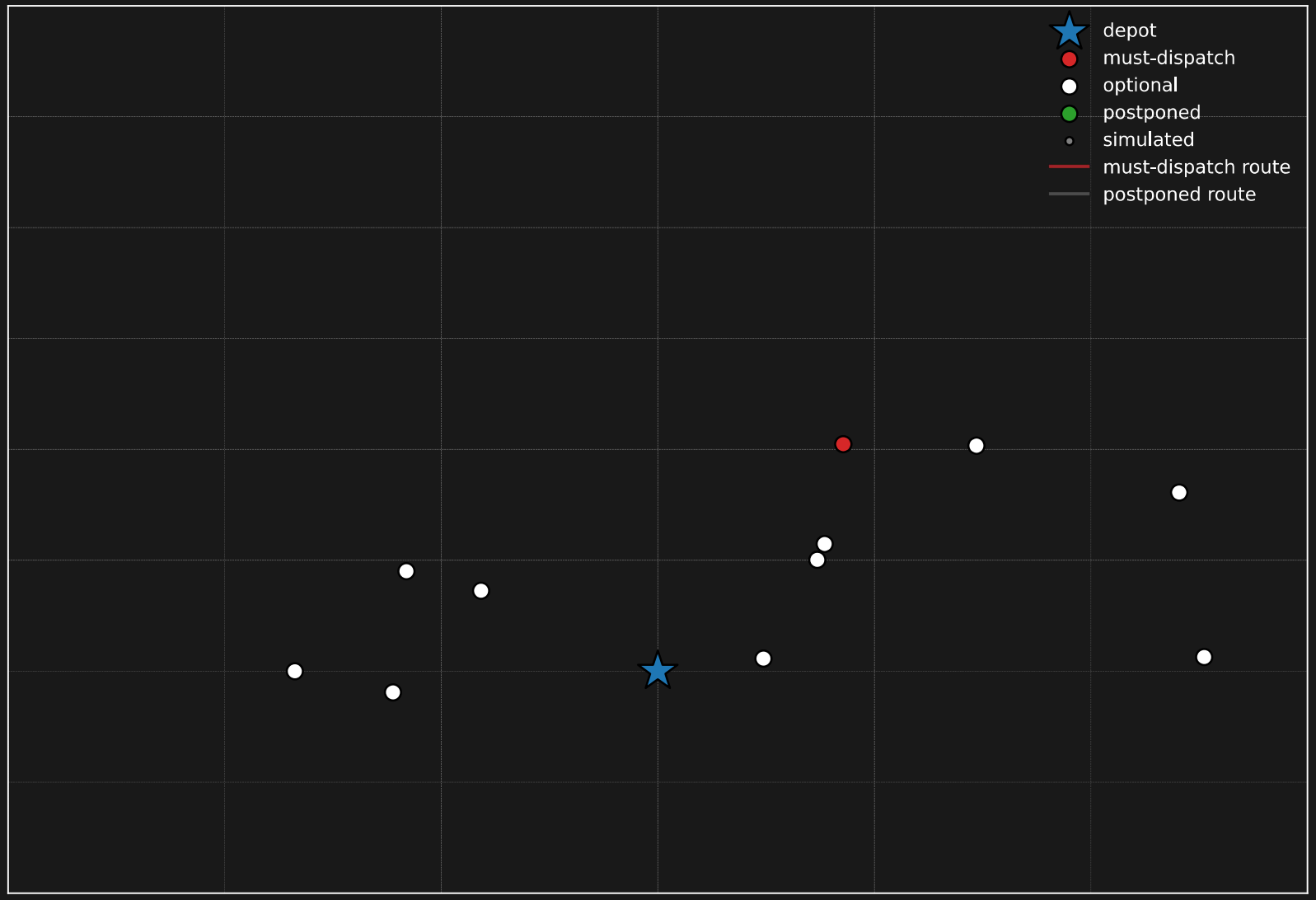
- One iteration if no improvement. Empty routes rarely result in improvement.
- Intensify only new best individuals.
- Templated $(N, M)$ exchange.

# DYNAMIC SOLVER

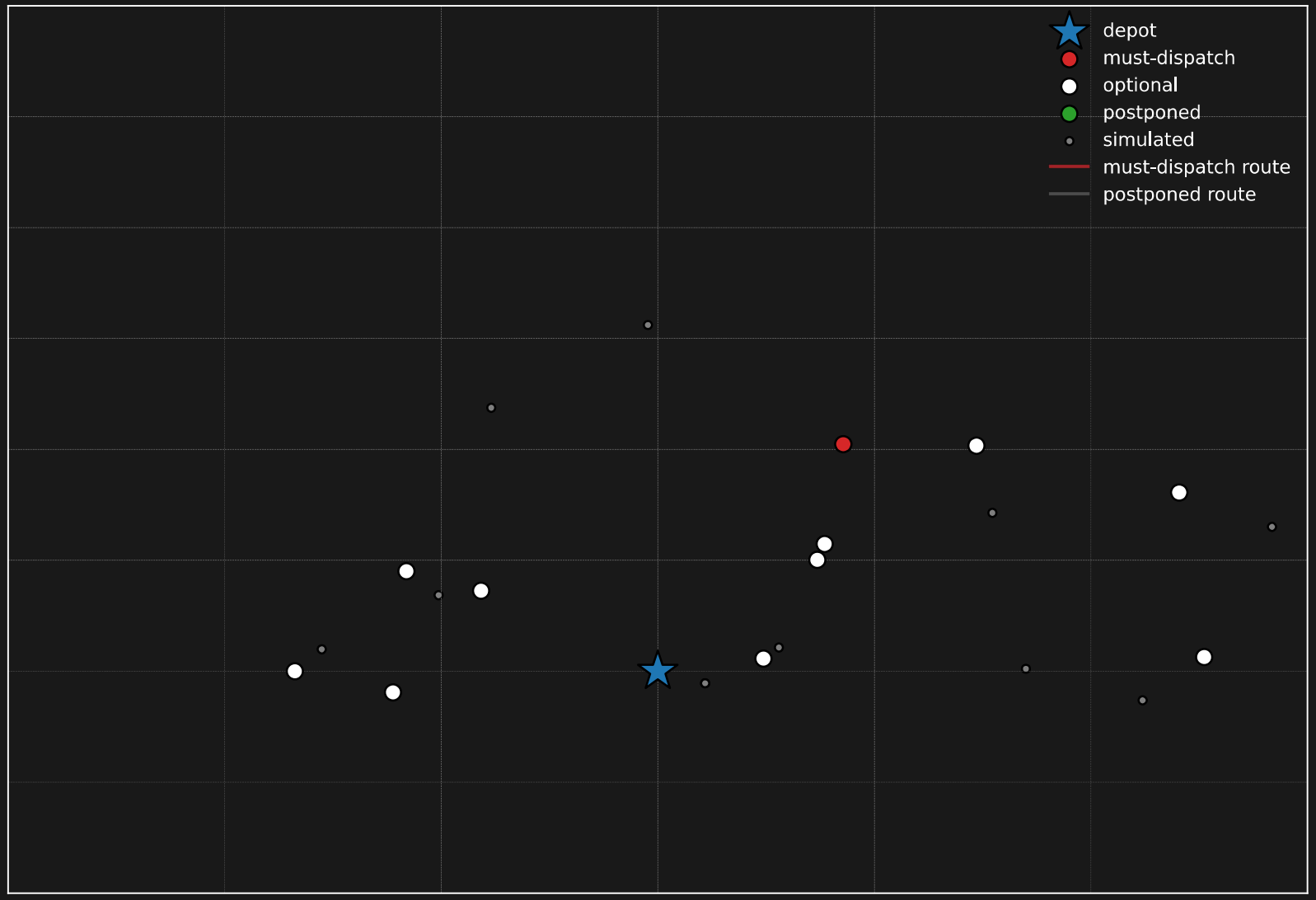
Idea:

*Simulate ahead to determine which requests to postpone*

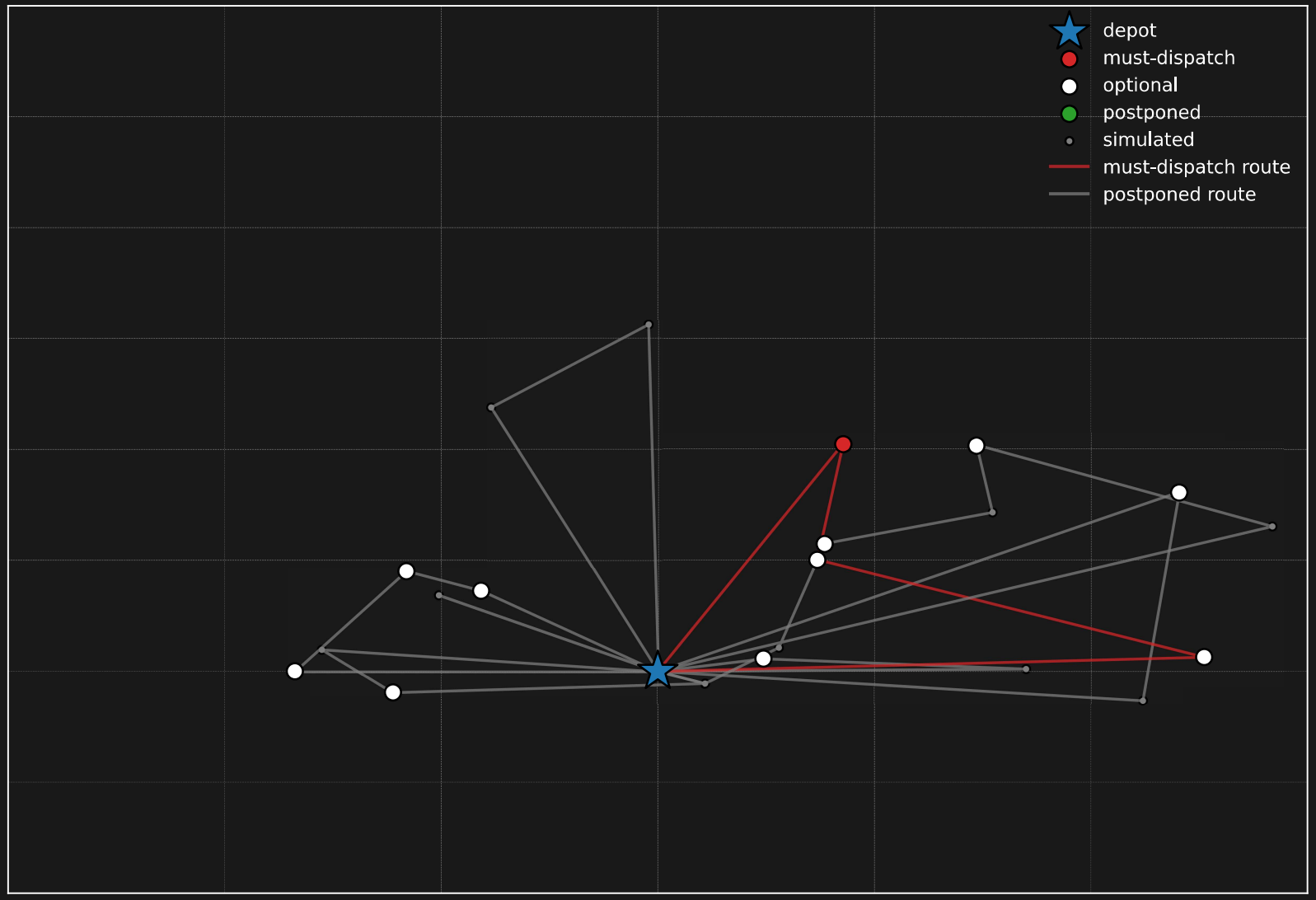
The epoch instance consists of must-dispatch and optional requests.
Optional requests may be postponed to the next epoch.



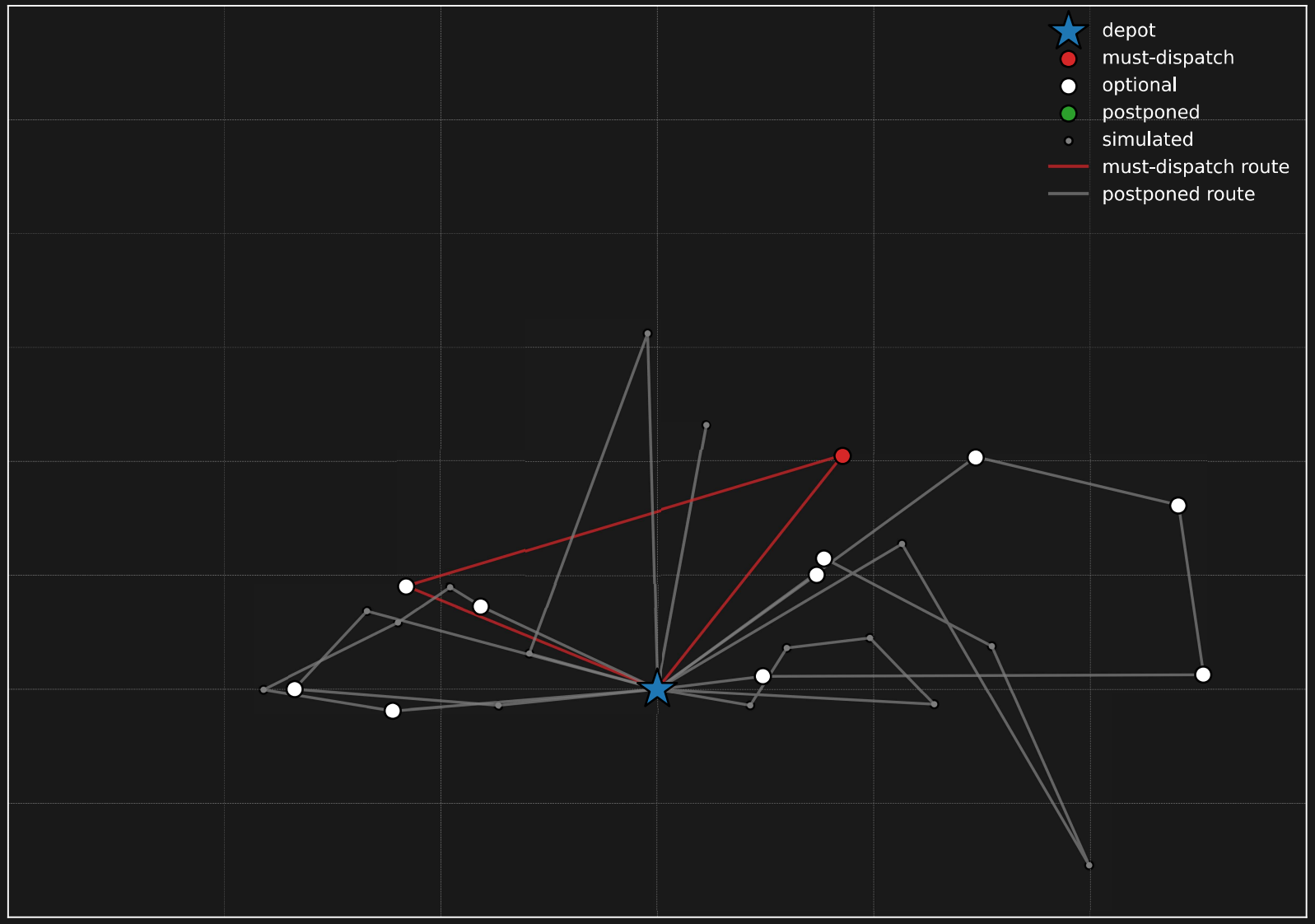
We sample future requests to obtain a simulated instance.
Each simulated request has a nonzero release date.



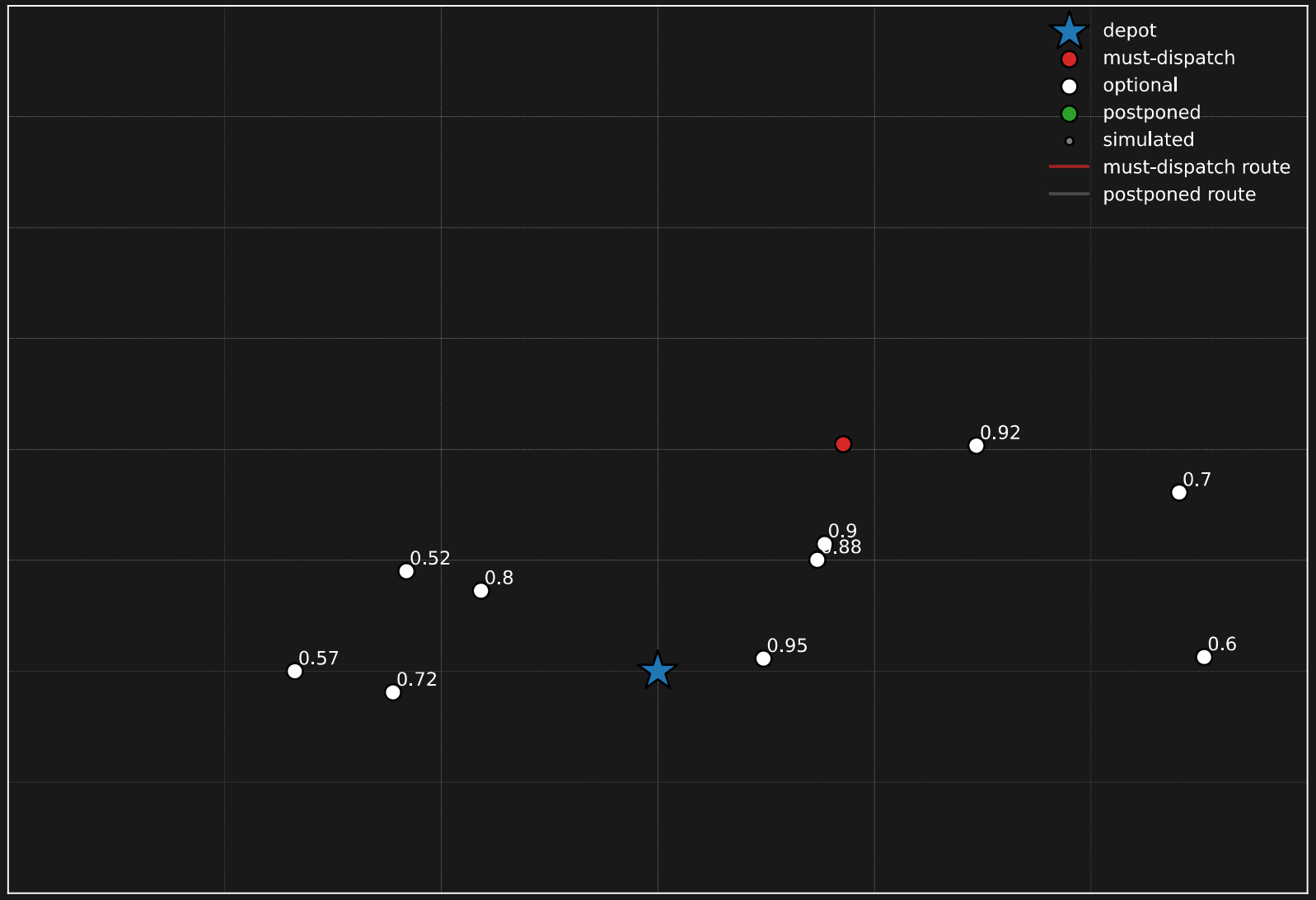
We solve the simulation instance as VRPTW problem with release dates.
The instance is solved very briefly, using less than 0.5 seconds.



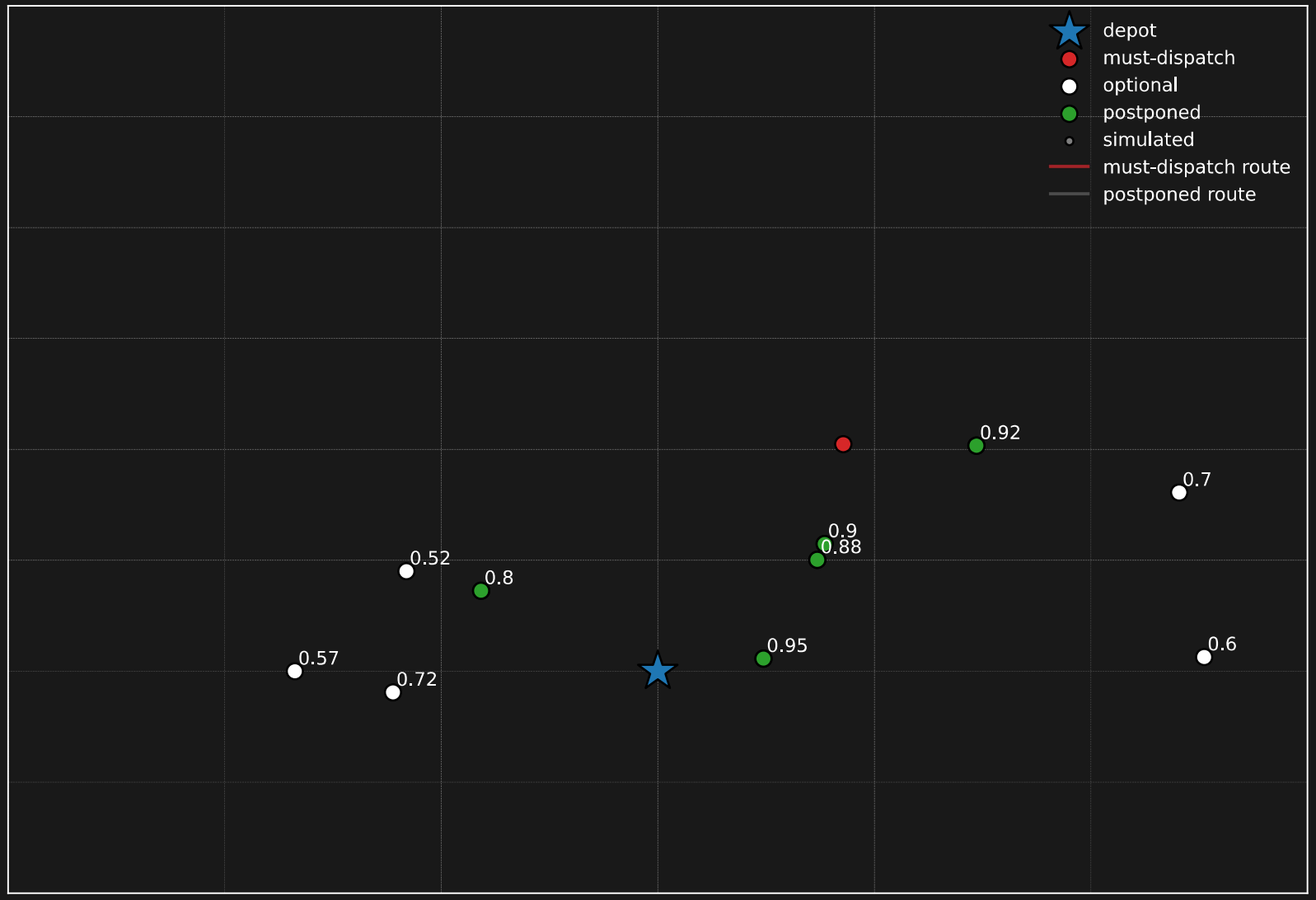
We repeat this for 40 simulation instances.



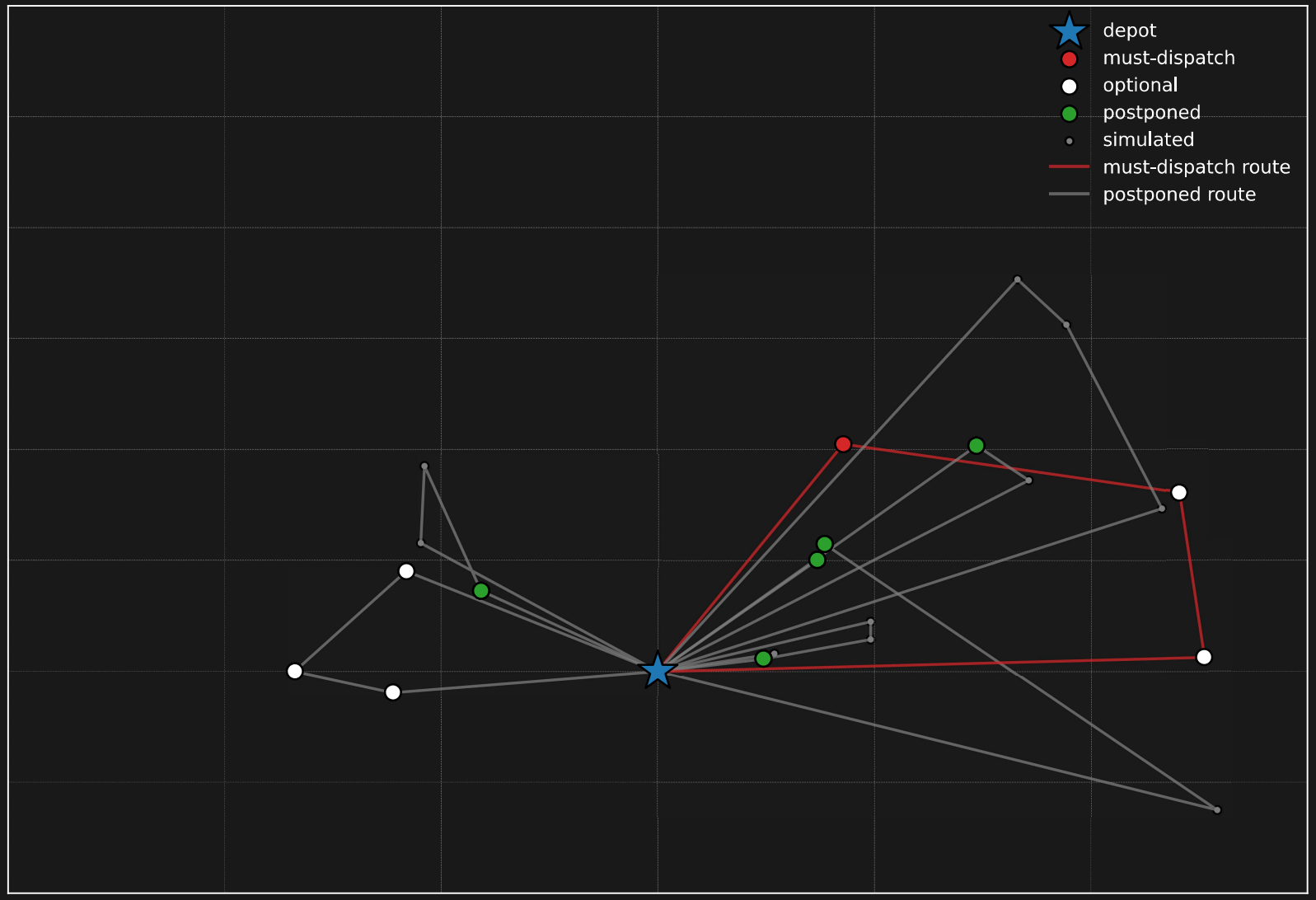
We count for each request how often it was postponed.
We mark all requests with postponement frequency higher than a threshold value.



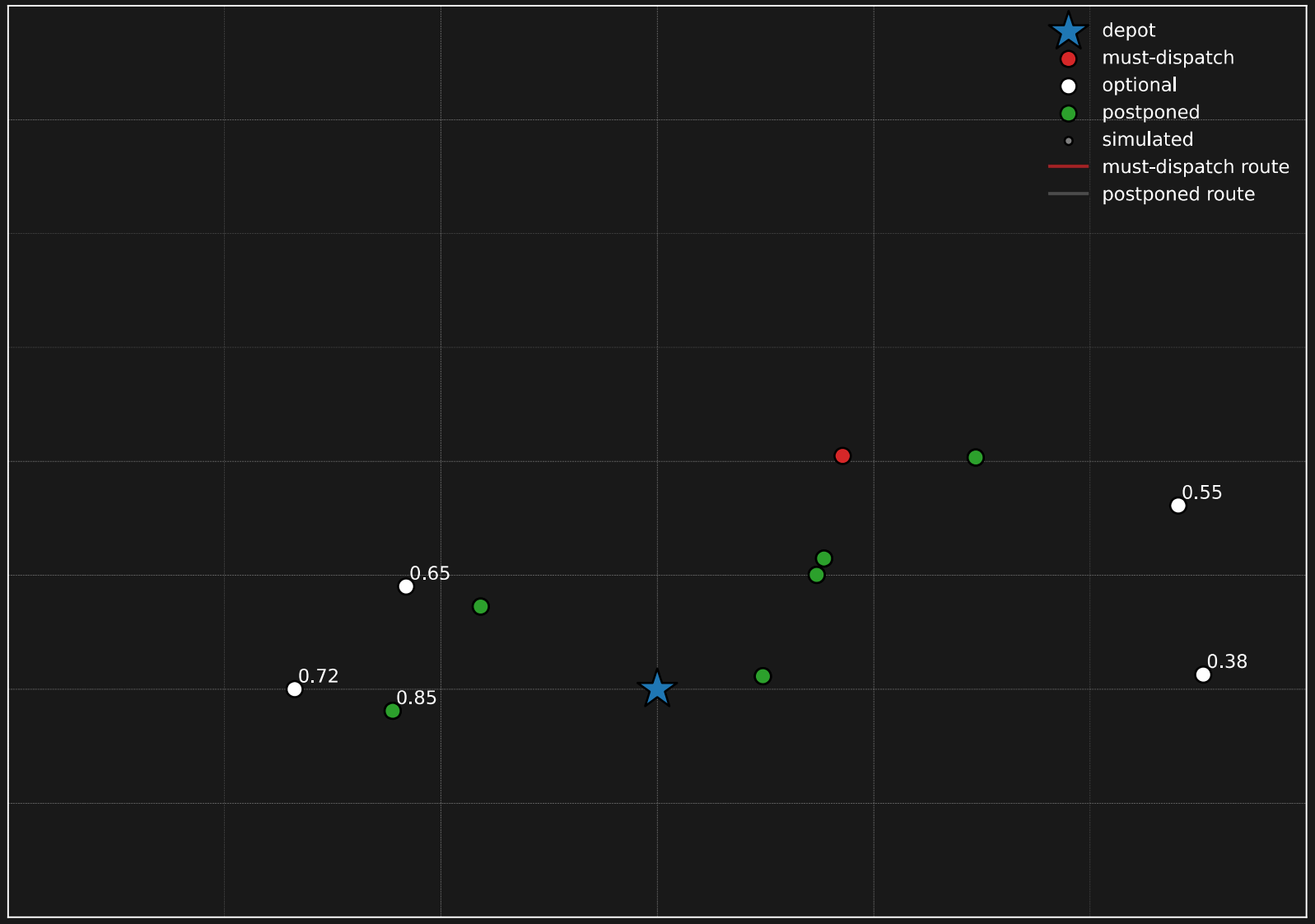
We count for each request how often it was postponed.
We mark all requests with postponement frequency higher than a threshold value.



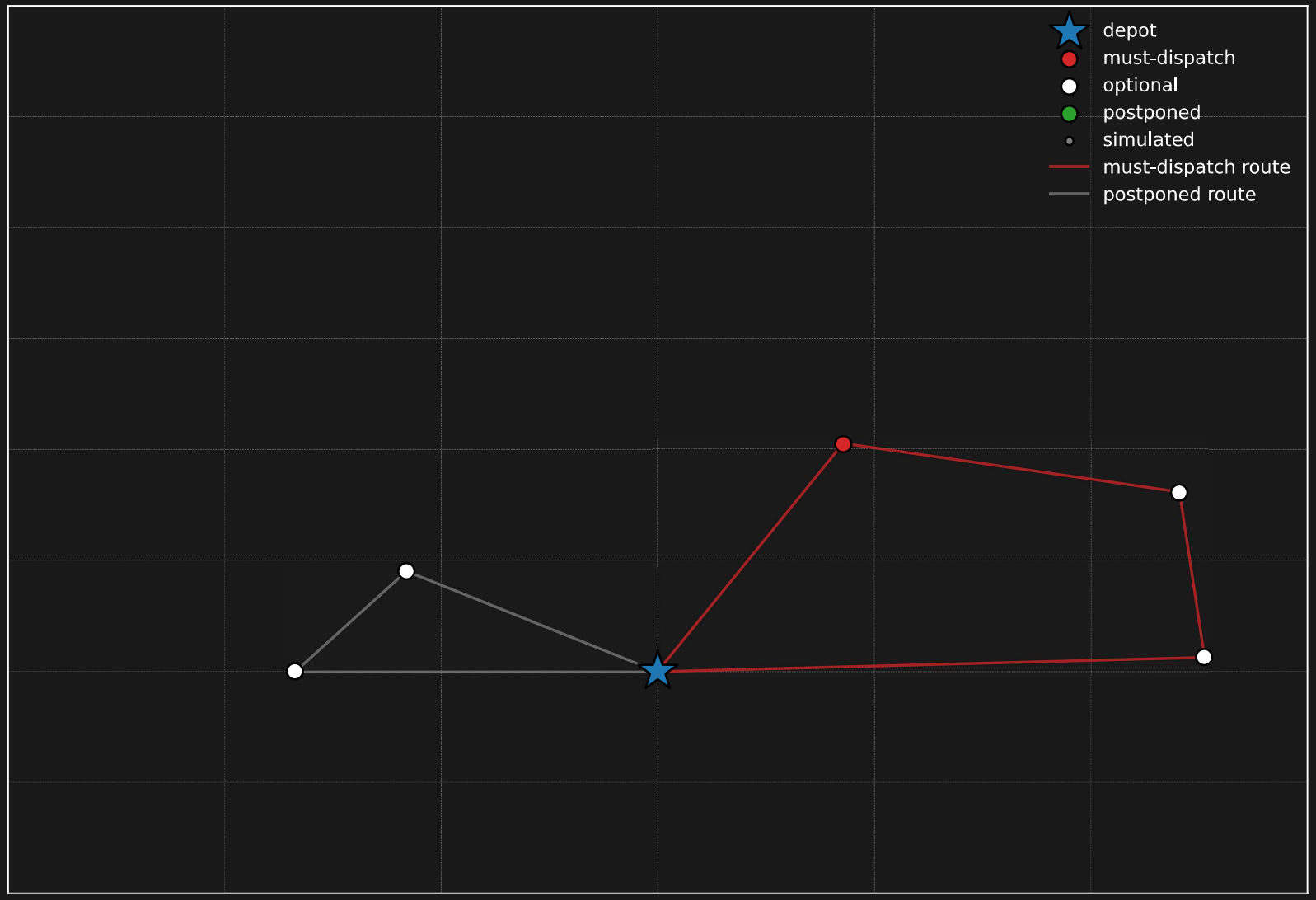
We repeat the full simulation procedure.
The postponed requests now also have a nonzero release date.



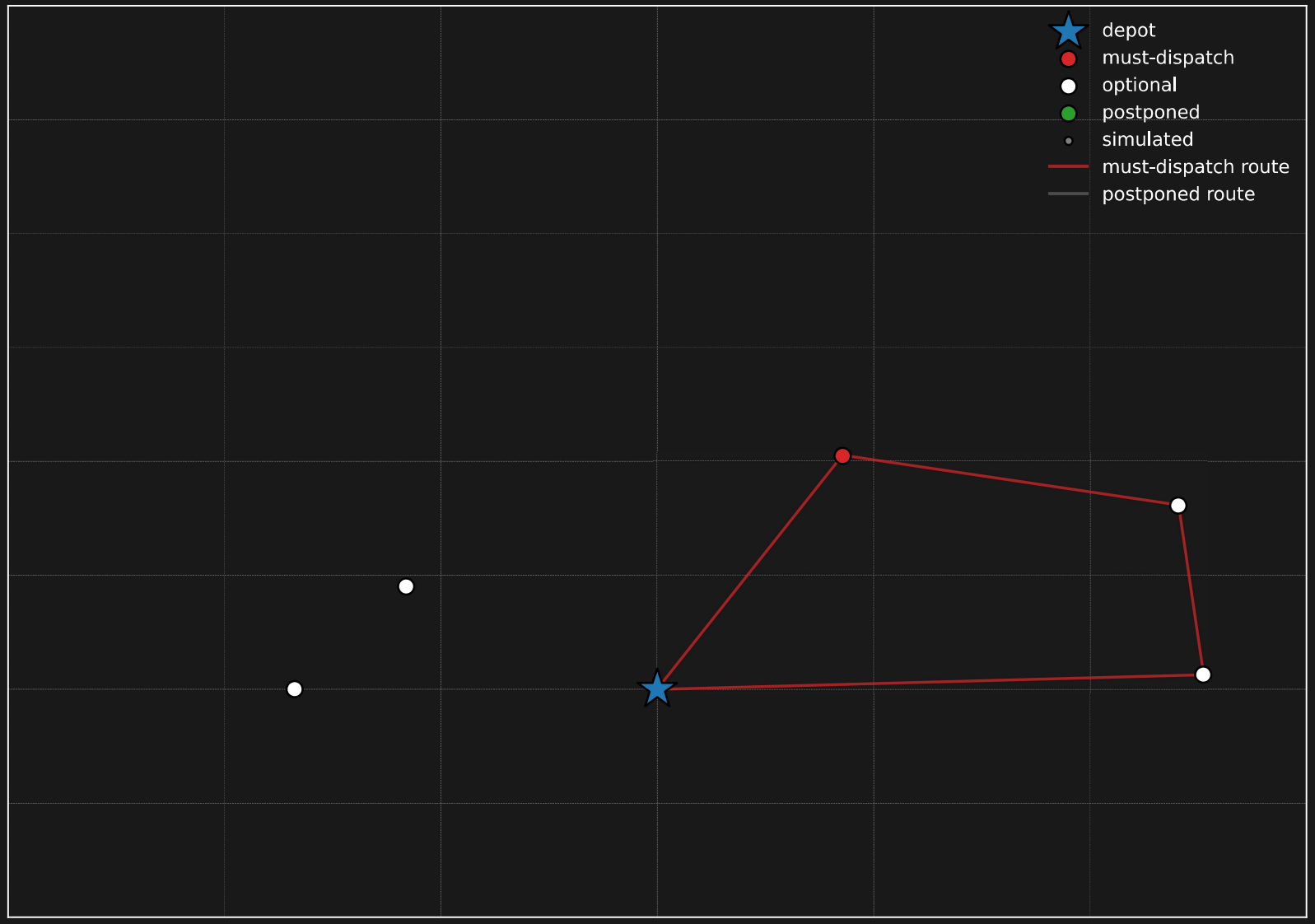
We again mark the requests that were frequently postponed in the simulations.



After finishing the simulation cycles, we remove the postponed requests,
and solve the resulting instance for roughly 20 seconds.



Finally, we remove all postpone-able routes and submit this new solution to the environment.



# THANK YOU

Questions? n.a.wouda@rug.nl

Code: https://github.com/N-Wouda/Euro-NeurIPS-2022