Getting started

Contact information

If you have any issues during development, you can contact our team using github issues, or alternativelly through bud@loggi.com.

Dataset

The full dataset is available to download here. Make sure you unzip the dataset into the data file. It should contain two directories named `cvrp-instances-1.0` and `delivery-instances-1.0`.

Alternativelly, you can generate the instances yourself from public data using the generation pipeline.

OSRM Server

To correctly evaluate distances, you should use OpenStreetMaps distances provided by the OSRM server. Our recommended way of running OSRM is Docker. To run it, please follow these steps.

1. Download and install docker, follow the instructions according to your operational system.
2. Download our precompiled distance files (5.3Gb compressed, 12.6Gb decompressed).
3. Extract the files into an `osrm` directory.
4. Run an OSRM backend container with the following command:

```
docker run --rm -t -id \
  --name osrm \
  -p 5000:5000 \
  -v "${PWD}/osrm:/data" \
  osrm/osrm-backend osrm-routed --algorithm ch /data/brazil-201110.osrm --max-table-size 100
```

For more information, check our OSRM detailed documentation.

Python API

We provide an API for loading and running Python solvers. It currently supports any Python version >= 3.7.1, which is natively available in most up-to-date operating systems.

Repository setup

This project uses Python Poetry to manage dependencies. You can follow its docs to install it, but a simple

```
pip install poetry
# Or with sudo to install it system-wide
# sudo pip install poetry
```

normally suffices. Check if it worked with `poetry --version`.

Then, at the root of the project install the dependencies with

```
poetry install
```

With everything in place, any Python command can be executed by preceding it with `poetry run` (e.g., `poetry run pytest tests/`). This is usually enough for executing the code in this project, but the user who demands more information can check the Poetry's website.

To implement a new method, we suggest you to create a Python `solve` function that takes an instance and outputs the solution to a file.

Task 1

```
from loggibud.v1.types import CVRPInstance, CVRPSolution


# Implement your method using a solve function that takes an instance and returns a solution
def solve(instance: CVRPInstance) -> CVRPSolution:
    return CVRPSolution(...)


# Loading an instance from file.
instance = CVRPInstance.from_file("path/to/instance.json")

# Call your method specific code.
solution = solve(instance)

# Saving your solution to a file.
solution.to_file("path/to/solution.json")
```

To evaluate your solution inside Python, you can do:

```
from loggibud.v1.eval.task1 import evaluate_solution

distance_km = evaluate_solution(instance, solution)
```

JSON schemas

If you don't use Python, you should implement your own IO functions. The JSON schemas for reading and writing solutions are described below.

CVRPInstance

```
{
  // Name of the specific instance.
  "name": "rj-0-cvrp-0",

  // Hub coordinates, where the vehicles originate.
  "origin": {
    "lng": -42.0,
    "lat": -23.0
  },

  // The capacity (sum of sizes) of every vehicle.
  "vehicle_capacity": 120,

  // The deliveries that should be routed.
  "deliveries": [
    {
      // Unique delivery id.
      "id": "4943245fb66541edaf54f4e3aaed188a",

      // Delivery destination coordinates.
      "point": {
        "lng": -43.12589115884953,
        "lat": -22.89585186478512
      },

      // Size of the delivery.
      "size": 2
    }
    // ...
  ]
}
```

CVRPSolution

```
{
  // Name of the specific instance.
  "name": "rj-0-cvrp-0",

  // Solution vehicles.
  "vehicles": [
    {
      // Vehicle origin (should be the same on CVRP solutions).
      "origin": {
        "lng": -43.374124642209765,
        "lat": -22.790683484127058
      },
      // List of deliveries in the vehicle.
```

```
      "deliveries": [
        {
          "id": "54b10d6d-2ef7-4a69-a9f7-e454f81cdfd2",
          "point": {
            "lng": -43.44893966650845,
            "lat": -22.742762573031424
          },
          "size": 8
        }
        // ...
      ]
    }
    // ...
  ]
}
```

Evaluation scripts

```
poetry run python -m loggibud.v1.eval.task1 \
    --instance tests/results/cvrp-instances/train/rj-0-cvrp-0.json \
    --solution results/rj-0-cvrp-0.json
```

Contributing

First of all, thanks for the interest in the project. If you found a bug or have any question, feel free to open an issue.

If you prefer to contribute with code or documentation, first fork the repository, make the appropriate changes and open a pull request to our `master` branch.

Notice we use Python Poetry to manage dependencies. So if you need to add, remove or update any dependency make sure to use the proper `poetry` commands to write the changes in the `pyproject.toml` and `poetry.lock` files.

Moreover, before opening a pull request, make sure the following were taken care:

- The `black` formatter was run:

```
poetry run black .
```

- The code is conformant with `flake8`:

```
poetry run flake8 .
```

- The tests are still passing:

```
poetry run pytest tests/
```

Note to Windows users

In some cases, Windows uses CRLF as end of line instead of LF, which is the norm in Unix-based systems. This erroneously makes git thinks that a whole file was changed when saved in different operating systems.

To alleviate this issue, we recommend Windows users to do one of the following:

- When installing Git for Windows, choose the option "Checkout Windows-style, commit Unix-style line endings" (see this StackOverflow answer)
- If Git is already installed, write the following in the LoggiBUD repository before making any commit:

```
git config core.whitespace cr-at-eol
```