Traveling salesman problem

Given a set of locations and distances between each pair of locations, the goal of the *traveling salesman problem* (TSP) is to find the shortest cyclic route visiting each location exactly once. This assignment is about addressing TSP by implementing a metaheuristic algorithm from scratch. The main purpose of this exercise is to experience and explore common pitfalls and difficulties you might encounter and to get familiar with an optimization process.

You will work in pairs (as previously chosen in IS). The organization of the work is up to you. However, the workload division in the pair should be fair and reasonably balanced.

Tasks

You will get TSP instances and solver templates for your convenience. Your tasks will be to:

1. Implement a TSP solver based on the *large neighborhood search* (LNS) metaheuristic
2. Integrate classical search operators (k-opt, k-swap) into the LNS solver
3. Summarize your experience in a report (required: 1 PDF file)
    - Implementation of LNS solver + classical operators (required: 1–2 A4 pages, 10–12 pt font size)
    - Statement on generative AI utilization
    - Clear description of authors' contributions

Before you start coding, make sure to clarify for yourself all points in the **'Before coding' checklist**. The checklist covers the key design decisions before you start your implementation. Feel free to experiment with different approaches to the tasks or seek inspiration from the related literature. We will test your solver on both the provided and hidden instances. Summarize your overall experience in the report. Briefly address all the points from the 'Before coding' checklist. Mention all used sources and literature references. Discuss what turned out to be complicated. Cover the utilization of AI and clearly state individual authors' contributions.

Provided materials

Make yourself completely familiar with the provided materials. The instructions and technical requirements included are an important part of this assignment. The package with the additional materials includes:

- TSP instances, data format description (see `./data/README.md`)
- Implementation requirements and instructions + code template (see `./solver_template/README.md`)
- Tool for visualization of solutions (see `./tsp_viz/README.md`)

Implementation requirements

Detailed implementation requirements are covered in the provided package. We emphasize the following:

- The solver must be implemented in Python 3
- The solver must conform to the required interface (see `./solver_template/README.md`)
- Only standard Python libraries are allowed (e.g., `json`, `math`, `random`, `sys`, `time`)
- The solver must run on a single CPU only
- The solver executions respect the given instance timeouts
- The submitted code must work properly on `aisa.fi.muni.cz`

Usage of generative AI

Using generative AI tools for programming is allowed and welcome under the following conditions:

1. AI tools are only used to generate well-testable functions (e.g., 2-opt operator)
2. Each AI-generated function is covered with a human-written unit test
3. Each such usage is clearly described and documented in the report
    - Persistent links to the relevant conversations with AI are provided
    - You may only use AI tools that provide this functionality (e.g., ChatGPT)

Using such tools is purely optional and up to your decision.

Submission details

- Each pair of students uploads their code (ZIP archive) and report (PDF file)
- A dedicated submission vault in IS is open for each pair of students
- Follow the requirements in the **'Before submission' checklist**
- **Deadline 29. 10. 2024 21:00**

Before coding

The checklist summarizes important questions that need to be addressed before implementing your solver. The list is not necessarily exhaustive. However, it presents a general guideline from which you can start.

- ☐ **Solution representation**: which one is convenient?
- ☐ **Infeasible solutions**: how to handle them?
- ☐ **Initial solution(s)**: how to obtain them?
  - ☐ Trivially
  - ☐ How to generate better initial solution(s)?
- ☐ **Destroy/repair methods**: which to consider?
- ☐ **Cost function**: formulation, evaluation
  - ☐ Trivial evaluation
  - ☐ Incremental evaluation
- ☐ **Solution acceptance**: which solutions to accept?
- ☐ **Termination criteria**: when to stop?
- ☐ **Parameterization**
  - ☐ What are the parameters of the algorithm?
  - ☐ How to properly set their values?

Before submission

The checklist serves as a sanity check before homework submission. Please make sure that all the listed points are met before you submit your work.

Solver implementation

- ☐ The solver conforms to the required interface (see `./solver_template/README.md`)
- ☐ The solver respects the given instance-specific runtime limits
- ☐ The solver runs properly on `aisa.fi.muni.cz`
- ☐ The solver does not produce an excessive amount of logs
- ☐ All your code is enclosed within one ZIP archive

Report

- ☐ The report is a single PDF file
- ☐ Each page contains authors' full names, UČOs, and a page number
- ☐ Solver implementation (LNS + classical operators)
  - ☐ 1–2 A4 pages, 10–12 pt font size
  - ☐ All points from the 'Before coding' checklist are addressed
  - ☐ Used sources and additional references are mentioned
  - ☐ Experience and complicated points are discussed
- ☐ Usage of generative AI is clearly described and follows the required rules
- ☐ Contributions of individual authors are stated