TU Wien
Institute for Logic and Computation
Algorithms and Complexity Group

Heuristic Optimization Techniques, WS 2019

Programming Exercise: Assignment 2

v1, 2019-12-10

Note: This programming exercise is **optional**. If you develop one more advanced hybrid metaheuristic, you can gain up to six bonus points. If you develop additionally a second more advanced hybrid metaheuristic, one third of the final exam is about the latter.

Work on the problem within your programming exercise group. You can and are encouraged to split tasks evenly but **everybody has to understand and be able to explain** all the concepts involved, your overall implementation, and the submitted report. Hand in your complete report **and** submit the solutions for the instances **and** upload your source code via TUWEL by **Sunday, 12th January 2020, 23:55**. For further questions please send an e-mail to `heuopt@ac.tuwien.ac.at`

The second programming assignment is to **develop a more advanced hybrid metaheuristic** for the Traveling Salesperson Problem with Target Distances for Multiple Drivers (TSPTDMD) in your preferred programming language. You can build upon everything you implemented for the first programming exercise. The problem description is given in the *general information* document.

We have selected another subset of instances with fewer vertices for this programming assignment to be found in TUWEL. The general aim is to come up with an approach that is particularly well suited for this more restricted class of instances.

The tasks for this exercise are:

1. Implement one or two of the following or some other more advanced metaheuristic strategy you did not yet consider:
   - Adaptive Large Neighborhood Search (ALNS)
   - Genetic Algorithm (GA)
   - Ant Colony Optimization (ACO)
2. Use this metaheuristic in a meaningful combination with (some of) the local search or greedy algorithms you have developed in the first exercise or consider now as meaningful extension. Remember that in a hybrid optimization approach the individual components should meaningfully augment each other in order to combine the strengths and that a good balance of exploration versus exploitation needs to be achieved.
3. Run experiments, compare at least two substantially different variants of your metaheuristic (e.g., with and without one of your major components), and write a report as discussed in the problem description.
4. Submit your best solution for each instance in TUWEL.

For the development and the report **consider the following points**:

- The distance matrix for the Euclidean instances grows in size with $\mathcal{O}(n^2)$. Although the target instances are now smaller, you may still do the following in order to avoid storing the distance matrix explicitly:
  - calculate its elements not beforehand but on demand in a function, or

  - in Python, calculate its elements beforehand using a $n \times n$ `numpy` array and the `scipy` function `cdist`[1], **not** with two nested Python loops. Perform the corresponding rounding of the distances by adding 0.5 to the whole array at once and converting it to integers using `astype`[2], **not** with two nested Python loops.

- You may switch to a *two-stage approach* instead of the big-M method, especially for the incomplete graph instances. In this case, the construction is bound to find a feasible solution and the subsequent improvement addresses the optimization in the feasible region only. Think about how you possibly can use the classical TSP solver `Concorde`[3] [4] to solve the Hamiltonian Cycle Problem (HCP).

- When implementing a GA, define at least one suitable operator for

  - *selection*,
  - *recombination*, and
  - *mutation*

  of solutions.

- When implementing an ACO, define at least one suitable

  - structure for the *construction graph*,
  - *pheromone model*, and
  - a concept for *heuristic information*

  to guide the decisions during the construction of solutions.

- When implementing an ALNS:

  - Define at least two suitable *destroy operators* addressing both the tour and assignment aspect of the problem.
  - Define at least two corresponding *repair operator*.
  - Report the *relative success* of the operators over runs on difficult instances.
  - Compare the ALNS with and without using the *Metropolis acceptance criterion* and report your findings.

- Remember that when conducting experiments with randomized algorithms, multiple runs over the same instances have to be performed and the mean objectives/runtimes and standard deviations have to be reported.

- Observe the evolution of the objective (for ALNS) or its min/mean/stddev (for GA/ACO) over iterations for selected instances in a plot.

- Think about what values you use for your parameters and document them in the report. You optionally can use `irace`[5] to tune your parameter values.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html`
[2] `https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.astype.html`
[3] `http://www.math.uwaterloo.ca/tsp/concorde.html`
[4] `https://github.com/jvkersch/pyconcorde`
[5] The tool can be found at `http://iridia.ulb.ac.be/irace`