Exercise 5: Ant Colony Optimization

Due date: *November 20th, 2025 (push to GitHub repo before the lecture)*
General Requests: *Same as before.*
Specific Requests: *Please at least* **try** *to consider GitFlow and commit/issue conventions for your repo, I'm begging you.*

5.1 Ant Colony Optimization

We've now applied metaheuristics to simple benchmarks (Hill Climbing), parameter tuning (Genetic Algorithms), regression (GA variants), and path-finding (Simulated Annealing). This week, we'll explore Ant Colony Optimization[1, Chapters 10, 18, and 19], a metaheuristic inspired by real ant foraging - originally for path-finding problems like the TSP, but also powerful for solving multi-constraint optimization problems.

This time, you can choose to implement one of the two options below:

Option 1: solve the TSP again using ACO (standard Ant System or Max-Min Ant with pheromone bounds and elite deposition). The heuristic function could, e.g., look like $\eta_{ij} = 1/(1+\text{distance}_{ij})$. Compute it once in the beginning, then only access matrix entries in the loops.

- **data**: use either last week's Austria cities dataset or this localized Carinthian towns dataset. both again contain a city_coordinates.json (geoinformation of each town), a routes_summary.json (compact information on all available pairwise routes), and files containing individual pairwise routes. if you want to generate your own data, you can use this data retrieval script
- **visualization**: you can use the provided visualization function or do your own thing, whatever works for you
- some good parameter choices might be $\alpha = 1$, $\beta = 3 - 5$, $\rho = 0.5$, n_ants$= 30 - 50$, n_iter$= 100$

Option 2: solve the Nurse Scheduling Problem (NSP), another highly complex and interesting combinatorial problem. this involves constructing an optimal schedule for **ten nurses**, **seven days**, and **3 shifts** per day that allows hospital operations to run smoothly under the following constraints:

hard constraints: constraints that **need to be met** no matter what.

1. at least two nurses per shift
2. following a night shift, a nurse needs to rest (cannot work the subsequent morning shift)
3. every nurse may work at most two shifts per day

soft constraints: constraints that **should** be met but may be ignored if push comes to shove.

1. shifts should be distributed fairly among all nurses
2. each nurse should have at least one day off every week[1]

(Important!) A few suggestions and comments:

- take a look at this skeleton code offering a basic setup of schedule updating and heuristic function
- once again, **start small**, then scale up once you know it works
- represent the pheromone information as a $3D$**-tensor** (matrix), i.e., $\tau_{n,d,s}$, with n, d, s the specific nurse, day, and shift. the heuristic $\eta_{n,d,s}$ here sadly cannot be pre-computed like in the TSP because it will depend on prior schedule decisions each of your ants made leading up to that specific schedule element
- represent the full schedule as a **binary matrix** $\text{schedule}_{n,s,d}$ (1: nurse n works shift s on day d, 0: she doesn't). this is important: the matrices τ and η would correspond to τ_{ij} and η_{ij} for selecting single edges of the full path in the TSP; but here, you select single schedule elements (i.e., nurse x working shift y on day z) of the full schedule. just like you had to store the full path for the TSP in the pseudocode I showed you, you'll have to store the full schedule here to trace which nurses you already scheduled where for evaluating the fitness L_k of the schedule generated by ant k

[1] We could introduce additional constraints such as single nurses not wanting to work a night shift, etc.

- there are generally two ways of **handling contraints** using metaheuristics: (a) implement candidate generation in a way that ensures compliance with the constraints or (b) penalize violations via the objective function. in (b), the algorithm learns that violations lead to bad results, which is usually smarter than taping off entire sections of your search space from the get-go. in the skeleton code, you'll see a possible penalty setup for our NSP constraints[2]
- some good parameter choices might be $\alpha = 1$, $\beta = 3 - 5$, $\rho = 0.5$, n_ants$= 50 - 100$, n_iter$= 100$

5.2 Analysis and Reporting

Add a streamlit page on ACO to the documentation you created for the last exercises.
Your final documentation webpage should contain the following sections:

1. Introduction - Overview of the algorithm: basics, strengths/weaknesses, complexity, ...
2. Methods - Describe your implementation: pros/cons, relevant parameters/functions, critical design choices, ...
3. Results - Display your results (obtained optimum paths for Option 1, obtained optimum schedule for Option 2) and allow different ACO parameter settings
4. Discussion - Analyze your findings: expected versus unexpected results, solution quality, efficiency, limitations, complexity of the implementation, possible improvements, ...

The documentation doesn't need to be absurdly long, but it should give a comprehensive overview of the topic that you can use later on to study for the exam.

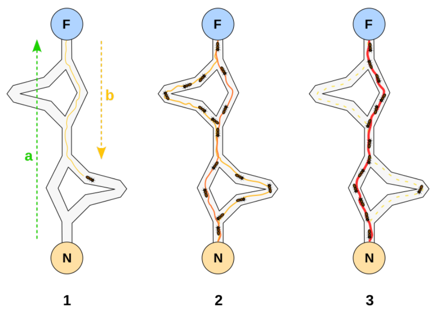

Figure 1: Another nice figure of how ants explore the environment and identify shortest paths via pheromone deposition, taken from [2].

References

[1] D. Simon, Evolutionary Optimization Algorithms. Wiley, 2013. [Online]. Available: https://research-1ebsco-1com-1195qzf320241.perm.fh-joanneum.at/c/kofjhs/search/details/4sh2uyq6wn?db=nlebk&db=nlabk
[2] ``Ameisenalgorithmus,'' Wikipedia. [Online]. Available: https://de.wikipedia.org/wiki/Ameisenalgorithmus

[2]Note: hard constraints should be penalized **heavily** (i.e., a penalty of 1000) to make violations very unattractive, while soft constraints should be penalized **lightly** (i.e., a penalty of 100; there should be an order of magnitude between hard and soft constraints). Penalties within soft and hard conditions may vary as well; each single constraint can be weighted differently according to our perceived importance of it.