INTRODUCTION TO LINEAR OPTIMIZATION

Mandatory homework 3 and 4: LPs in practice

The following is the final mandatory homework. There are three tasks, with some subtasks. Altogether, the tasks are worth 50 points. Please submit your solution by email by Wednesday, Jan 31.

EXERCISE ONE [15 points] The new fast food chain STEAMBOAT WILLIE'S has asked you to do preliminary research in Poland in terms of how many restaurants will they need.

To be specific, their initial plan is:

1. To only open STEAMBOAT WILLIE'S inside towns (any town from the dataset).
2. To have enough STEAMBOAT WILLIE'S stores so that every town in Poland has one within 50km.

You are to use the provided dataset of cities in Poland to do the research.

Your tasks are the following:

1. First, use integer programming to compute how many stores does STEAMBOAT WILLIE'S need to open for the restriction of 50km, as above.
2. Suppose, for example, that the right answer is 42. Then, create a table with rows indexed by $1, 2, \ldots, 42$, and for each row – corresponding to opening exactly $k \leq 42$ stores – compute the minimum distance D so that there exists a placement of k restaurants such that every town in Poland can have a STEAMBOAT WILLIE'S within D km. By this logic (and using our arbitrary example), for 42 stores, this number D should be below 50.
3. Finally, compute the linear programming relaxation of the integer programming problem. In other words, if we are allowed to open *fractions* of stores in cities, how many stores do we need to open so that every town in Poland has at least one within 50km?

Didactic notes:

- Please use the dataset provided.
- Your homework should consist of combination of code, tables, plots and discussion about your solution and the observed data. The easiest solution is using a Jupyter notebook and mix data and code. Alternatively, the code can be separate, but the remaining three should be together in a PDF document.
- Discussion on the observed data is necessary. Comment on for example how long does it take your computer to compute the given instance and if this is a significant problem of your code, or if it is to be expected.
- If your program cannot handle all cities in Poland from the dataset, you are free to pre-select some large subset and work with that. By my sample solutions, it should be possible to solve subsets with more than 500 cities.
- You can use other languages than Python and other solvers than Gurobi, if you wish. However, you should not use any toy solvers that solve only very small instances.
- Please use the dataset provided as a benchmark for your LPs. You can of course select subsets of towns in it, but, as an example, it will not be enough to claim your code works on metric spaces with at most 10 points. Make sure you find the largest subset that works for you.
- You are free to use other Python libraries and search for advice on the internet. However, make sure that you are not pasting large parts of other people's code into your own. Some bits can be adapted – all with proper reference to the source, of course – but do not paste in the first result of "Christofides algorithm Python" or such.
- Similarly, I have used code plagiarism detection tools in the past, and in fact have caught people copying another person's code and renaming the variables. Make sure the final code is

indeed yours and yours only. I strongly discourage reading anybody else's full solution – this usually leads to plagiarism being detected.
- I have not checked that the dataset is a full metric, although its construction comes from a metric space (the Earth). I have checked some smaller subsets to verify the triangle inequality, and even if it fails in some extreme cases (when three cities lie on an arc), it should not affect the approximation ratio of the Christofides algorithm.

Exercise two [20 points]

If you do not know the Metric Traveling Salesman Problem, here is a recap: Given a metric space M – a set of points X and a symmetric distance function $d : X \times X \to \mathbb{R}_0^+$ that satisfies the triangle inequality, find a sequence of visiting all the points such that each point is visited exactly once and the overall travel length is minimized.

The decision version of this problem is NP-hard, but it can be also efficiently approximated in polynomial time.

Your task will be to explore the metric TSP problem in a practical setting.

1. First, implement the integer LP formulation for metric TSP which contains exponentially many constraints. There are several possible ones, one can be the *Dantzig-Fulkerson-Johnson* formulation from `https://en.wikipedia.org/wiki/Travelling_salesman_problem`.
   Use the provided dataset as a benchmark and report (in the homework) on how large an instance are you able to solve using this formulation with exponentially many constraints.
2. Next, implement a "lazy row generation" version where you add constraints progressively until a solution is a tour that visits all vertices once. Adding new constraints into the model sounds daunting, but this is actually really well covered in the Gurobi example set, for example here: `https://www.gurobi.com/jupyter_models/traveling-salesman/`. Report, in the homework, how large a dataset you can solve with this variant.

Exercise three [15 points]

We continue with the metric travelling salesman problem.

Every programmer should know the *Christofides algorithm* for approximating the metric TSP problem. A description can be found for example here:

`https://en.wikipedia.org/wiki/Christofides_algorithm`. Your task is to indeed implement the Christofides algorithm. Run the Christofides algorithm on the chosen subset of the dataset and compare (with plots) its efficiency and its running time to the optimal ILP solution from Point 1 or Point 2 in Task 2.

The most difficult part is to find a library for *perfect matching of minimum cost*, which you need to find as part of the Christofides algorithm. You are of course free to use a library for this from the Python package universe or elsewhere.

Please follow the two rules below:

- If you are using any library, do not forget to explicitly cite it in your homework. Any copying of code from the internet without proper citation/credit will be considered plagiarism.
- To make sure your implementation is sound in terms of complexity, please include in your documentation what is the worst-case running time complexity of the library, and if it matches the usual worst-case running time of the Christofides algorithm.