# Homework 5

MSSC 6000 — Scientific Computing — Spring 2023

Due Monday, April 24, on D2L, by 11:59pm

For each of the two problems below, your task is to

- Implement a simulated annealing solution in python.
- Explain all design choices (for example, your tweak function, your choice of parameters, your choice of stopping conditions, etc).
- Then, use any method we've learned so far (simulated annealing, or anything else) to try to find the best solution you can to the particular test case given. I will be awarding bonus points to the best solutions in the class.

**Please do not hesitate to let me know if you need help understanding these problems, or understanding the code I have provided to get you started.**

1. A university has asked you to help them figure out where to place those blue light emergency call buttons on their campus to minimize the distance between any of the buildings and one of the blue lights. The campus has $n$ buildings $b_1, b_2, \dots, b_n$ that are represented by $x, y$-points in the Cartesian plane, all inside of the square with corners $(0,0)$ and $(1,1)$. They want to place $k$ lights $\ell_1, \ell_2, \dots, \ell_k$ inside of the unit square.

   Recall that the Euclidean distance between two points $(x_1, y_1)$ and $(x_2, y_2)$ is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

   For a given building $b_i = (x_{b_i}, y_{b_i})$, the distance to the closest light is

   $$\min_{1 \leqslant j \leqslant k} \left( \sqrt{(x_{\ell_j} - x_{b_i})^2 + (y_{\ell_j} - y_{b_i})^2} \right),$$

   and so the quantity that the university is trying to minimize is

   $$\sum_{i=1}^{n} \left( \min_{1 \leqslant j \leqslant k} \left( \sqrt{(x_{\ell_j} - x_{b_i})^2 + (y_{\ell_j} - y_{b_i})^2} \right) \right).$$

   Write a simulated annealing procedure that takes the building locations and the quantity $k$ as input and returns the best location for the lights that you can find.

   I have written some code snippets to help you. In the file `blue_lights_helper.py`, there is a class called `Campus` with a scoring function that you can use, `random_campus` to generate random campuses for you, and files to write a campus to a file or read it from a file.

   Your submitted code should have a function called `SA` that takes a campus as an input (as well as other optional parameters if you want, e.g., $\alpha$, $T_0$), and that then runs your procedure:

   ```
   SA(C, ...)
   ```

   You should also submit in a separate text file your best solution to the campus with 100 buildings in the file `campus.txt` using 10 lights. In the first line of your text file, print the locations of the 10 lights that represents your best solution. In the second line, write what you think you score is. The best solutions will receive bonus points.

**Guidelines for the competition:**

- You may use any of the techniques we've discussed in this course.
- You must use your own code to find solutions – for example, you cannot use external tools for solving optimization problems like OR-Tools, Gurobi, Matlab, etc.
- It is fine (and encouraged) to do extra research on our course methods to figure out ways to improve them, but you should not do research on this specific problem.

2. Let $G$ be a graph with $n$ vertices, where $n$ is even. We will consider only simple, undirected graphs, which means the edges do not have a defined direction, between any two vertices there can be at most a single edge, and there cannot be an edge from one vertex to itself. (These are the kinds of graphs we discussed earlier in this class.)

   Let $V$ be the set of vertices. Suppose we have split $V$ into two equally sized subsets $V_1$ and $V_2$ (so they both have size $n/2$). This is called a *partition* of the vertices. The score of a particular partition is equal to *the number of edges that have one vertex in $V_1$ and another vertex in $V_2$*. Your goal is to minimize this score. In other words, you want to find the partition of the vertices into two equally sized parts that minimizes the number of edges in the graph that cross between the parts.

   (a) Invent a "real-world" situation that this problem could model. It doesn't have to exactly fit every detail, but basically I'm looking for some kind of problem you could solve with this model, possibly with some adjustments/modifications.

   (b) Write a simulated annealing procedure that takes a graph and returns the best partition of the vertices that you can find. The actual thing that should be returned is just one part of the partition (the other one can be derived from it by just taking the set complement).

   I have written some code snippets to help you. In the file `graph_partitioning_helper.py`, there is a class called `Graph`. The input to make a `Graph` object is `n`, the number of vertices (labeled from 1 to $n$), and `edges`, which should be a list/set of pairs of vertices that are connected by edges. For example

   ```
   G = Graph(4, [(1,2), (1,4), (3,4)])
   ```

   creates a graph with 4 vertices and 3 edges. This class has a scoring function for you called `unpenalized_score` (this is a bit of a hint: you might get better results if you try a penalized version where the partition parts don't have to be the same size [or you might not!]). For example, to check the score of the partition $V_1 = \{1,2\}$, $V_2 = \{3,4\}$, you can call:

   ```
   G.unpenalized_score([1,2])
   ```

   which should return 1. This file also includes a function to make a random graph and a function that finds the brute force solution (this will only work for fairly small $n$), both to help you with your development and testing. Lastly, there are functions to write graphs to a file and read them back from a file.

   Your submitted code should have a function called `SA` that takes a graph as an input (as well as other optional parameters if you want, e.g., $\alpha$, $T_0$), and that then runs your procedure:

   ```
   SA(G, ...)
   ```

   (c) You should also submit in a separate text file your best solution to the graph with 1000 vertices and edge probability 1% in the file `graph-1000-0p01.txt`. In the first line of your text file, print the list of vertices in $V_1$ of your partition. In the second line, write what you think you score is. The best solutions will receive bonus points. The guidelines are the same as for question 1.