Prof. Dr. Stephan Held
Daniel Blankenburg, M. Sc.

Programming Exercise 2

Exercise P.1. Let $G = (V, E)$ be an undirected graph and $k = |V|$. Let $C \subseteq V$ and $f : V \setminus C \to \{1, \dots, k\}$ be a placement function (in particular f is injective).

(a) Compute positions $g : V \to [1, k]$, extending f, approx. minimizing

$$\sum_{e=\{v,w\}\in E} |g(v) - g(w)|^2$$

This can be achieved by choosing any initial positions and then iteratively picking the average position of all neighbors for each circuit (this is equivalent to perform Gauß-Seidel iterations on the linear equation system defined in Proposition 4.18 in the lecture). Stop if the maximum circuit movement in one iteration is below 0.1. Note that g is not required to be injective.

(b) Compute positions $g : V \to \{1, \dots, k\}$, extending f, minimizing

$$\sum_{e=\{v,w\}\in E} |g(v) - g(w)|$$

Again g is not required to be injective. One way to solve this problem approximately is to use *subgradient descent*. The subgradient of a convex function $h : \mathbb{R}^n \to \mathbb{R}$ is defined as the set

$$\partial h(x) := \{v \in \mathbb{R}^n : h(y) \geq h(x) + \langle v, y - x\rangle \ \forall y \in \mathbb{R}^n\}.$$

If $h(x)$ is differentiable in x then $\partial h(x) = \{\nabla h(x)\}$. Note that for $v_i \in \partial h_i(x)$ it holds that $\sum_{i=1}^m v_i \in \partial \sum_{i=1}^m h_i(x)$. Subgradient descent works in the following way:

- Let $x^{(0)} \in \mathbb{R}^n$.
- for $i = 1, ..., K$:
  - Let $v^{(i)} \in \partial h(x^{(i-1)})$.
  - $x^{(i)} := x^{(i-i)} - \alpha_i v^{(i)}$
- return $x^* \in \{x^{(1)}, ..., x^{(K)}\}$ with minimum h-value.

Here $\alpha_i \in \mathbb{R}_{>0}$ should be a step-size rule that satisfies $\sum_{i=1}^{\infty} \alpha_i = \infty$, $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$, e.g. $\alpha_i = 1/i$.

To apply subgradient descent to this exercise you can define $h : \mathbb{R}^V \to \mathbb{R}$, $h(x) := \sum_{\{v,w\} \in E(G)} |x_v - x_w|$. In order to satisfy the constraints on the x variables you should leave $x_v := f(v)$ for $v \in V \setminus C$ fixed and project x_v into $[1, k]$ after every iteration for $v \in C$.

Alternatively (to get bonus points), you can use an open-source LP solver of your choice to solve the problem exactly. This can be for example the academically free program *QSopt* through the API in lp.h that is available on the website or the free LP solver from the Google OR-tools. For QSopt there is also an example program explaining the usage of that API, on which you can build upon. Read the README file for further information! The program compiles under Linux and under Windows/Cygwin (`http://www.or.uni-bonn.de/lectures/ss22/cd_ex/lp_solver.tar.gz`). For compiling type 'make' in the extracted directory 'mss'. If you encounter problems building mss, do not hesitate to contact me.

(c) Finally extend the placement f to a placement $f : V \to \{1, \dots, k\}$. First use the program from the first, then from the second task obtaining some g which is not necessarily injective and integral. Sort the circuits by their position w.r.t. g, pick the median unassigned circuit and assign it to the median available position in f. Assign all the other circuits to either the left or the right side depending on their positions w.r.t. g and recursively solve these two smaller instances, finally yielding a placement f that is integral and distinct.

Run your programs on the instances on the website. For each of the tasks output the linear length and the sum of the quadratic lengths. Moreover print the positions g of the circuits C for the first two tasks and the positions f for all vertices for the third task, by printing a single line for each circuit containing its index in the input and its computed position.

The instances are given in DIMACS format:

- The first line starts with a p (problem) and has the following format:

  p edge k m

  where k = number of vertices $(1, \dots, k)$ and m = number of edges $(1, \dots, m)$.

- Lines starting with an e define edges and have the following format:

e i j

where the edge is joining the vertices indexed by i and j.

- Lines starting with an n define vertex-positions and have the following format:

  n i p

  where i is the vertex index and the integer $p \in \{1, \ldots, k\}$ is its position. Vertices without fixed position, for which you should compute a position, will have $p = -1$.

There is a C-routine for reading in a graph in the given format provided on the website.

The program must be written in `C` or `C++` and must compile and run on Linux. You are allowed to use any any ISO `C` or `C++` standard including `C++20`. You can use any tool available in the standard library. Your program must compile with either Clang (any version $\geq$ 3.4.2) or Gcc (version $\geq$ 4.8.3) with `-Wall -Wextra -Wpedantic -Werror` and cannot link to any external library except the standard library. To achieve the maximum score, your program must not leak any memory and must be well documented.

(12+12+12 points)

Deadline: June 14, via email to blankenburg@or.uni-bonn.de. The websites for the lecture with all exercises and test instances can be found at:

`http://www.or.uni-bonn.de/lectures/ss22/chipss22_ex.html`