Optimization for Computer Science Assignment 1

October 18, 2016

Submission: Upload your implementation files in a single zipped file (`MatrNr.zip`) to the TU-Graz TeachCenter using the MyFiles extension.
Deadline: November 1st, 2016 at 23:58h.

1 Shortest Path as a Linear Program

Computing the shortest path in a graph is a classical problem in computer science that arises in a multitude of applications. Consider a directed graph $G = \{\mathcal{V}, \mathcal{E}\}$, with vertices $V_i \in \mathcal{V},\ i = 1, \ldots, N$ and edges $e_{ij} \in \mathcal{E}$ that connect vertex V_i to V_j. The direction of the edges is encoded by the ordering of the indices: edge e_{ij} is directed from vertex V_i to vertex V_j. Additionally we assign to each edge a weight $w_{ij} \geq 0$.

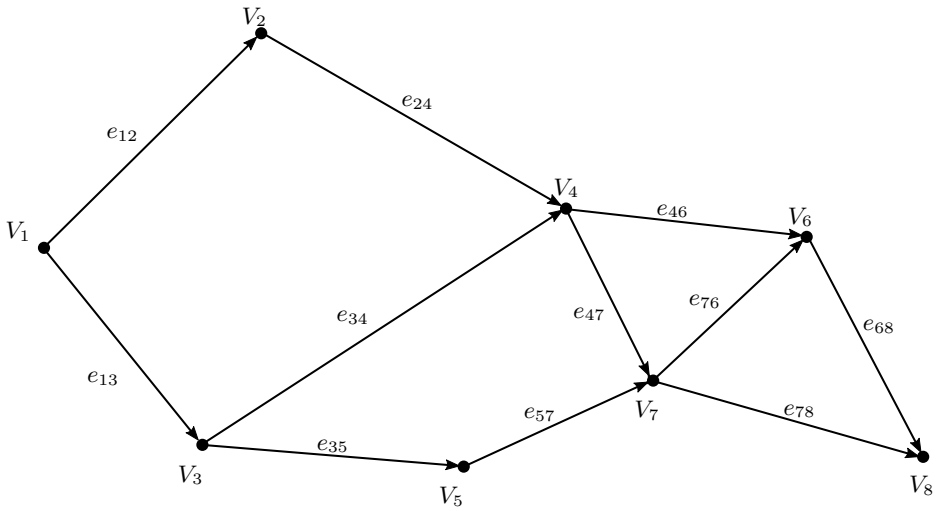

Figure 1: A directed graph.

We specify the graph as follows:

- Vertices are given by $V \in \mathbb{R}^{2\times N}$, where each column of the matrix contains x and y position.

- Edges are specified by a connectivity matrix $E \in \mathbb{R}^{N\times N}$ containing 0 and 1. A 1 at index (i, j) means that vertex V_i is connected to vertex V_j (in this direction). From E, we can deduce the number of edges M in the graph as $M = \sum_{i,j} e_{ij}$, i.e. the number of nonzero entries in E.

Finding the shortest path in such a graph can be modeled as a linear program. A path $\mathcal{P}$ is a sequence of vertices $\{V_1, \dots, V_n\}$ such that $[V_{i-1}, V_i]$ is an edge for $1 < i \leq n$, and the shortest path is the one for which the quantity

$$C = \sum_{w_{ij}\in\mathcal{P}} w_{ij} \tag{1}$$

is minimal.

1.1 Modeling the LP

We start modeling the LP by re-indexing the edges e_{ij}. Instead of using indices (i, j) to indicate which vertices are connected by the edge, we assemble all edges into a set $\mathcal{E} = \{e_m\},\ m = 1, \dots, M$, where M is the number of edges in the graph[1]. Next, we introduce the decision variable $x \in \mathbb{R}^M$. Each entry x_m of the vector x is an indicator wether the corresponding edge is part of the path ($x_m = 1$) or not ($x_m = 0$). Hence we can directly write the *cost function* of the LP as

$$\min_x \sum_{m=1}^{M} w_m x_m = \min_x w^T x, \tag{2}$$

where w_m is the weight of edge e_m.

Minimizing eq. (2) alone is not enough, we also need to make sure that the solution is actually a valid path. For this, the following constraints have to be fulfilled:

- Let V_s be the source vertex and V_t the target vertex, the path must start at V_s and end at V_t.
- The edges of the path must form a connected sequence. This can be expressed by the constraint that for inner vertices (all vertices of the path except V_s and V_t) the number of incoming edges must be equal to the number of outgoing edges. Obviously, we have for V_s that the number of incoming edges is 0 and the number of outgoing edges is 1 and vice-versa for V_t.

We formalize this as

$$\forall i \quad \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1 & \text{if } V_i = V_s \\ -1 & \text{if } V_i = V_t \\ 0 & \text{else} \end{cases}, \tag{3}$$

where x_{ij} is the indicator of the corresponding edge e_{ij}. Eq. (3) can be written in matrix form as $Ax = b$. Adding the constraints to the cost function (2), we arrive at a linear program with equality constraints in standard form

$$\min_x w^T x \quad \text{s.t.} \quad x \geq 0,\ Ax = b \tag{4}$$

[1] We have to agree on some ordering convention to convert index pairs (i, j) to a unique index m.

- Design a LP to find the shortest path between the first and the last vertex of the graph, i.e. $V_s = V_1$ and $V_t = V_N$
- Assemble the matrix A and the vectors b, w
- Use the linear programming toolbox to solve the LP (`scipy.optimize.linprog`)
- Verify your solution by running the program multiple times with randomly initialized w

2 Framework

We provide a python script with the basic framework (reading input data, drawing a graph. . .). Your task is to implement and solve the LP as detailed in section 1.1.

Starting the script The script can be started directly from the command line. It accepts either no parameters, in which case graph data is loaded from the files `V.txt`, `E.txt` and `w.txt`, or exactly 3 parameters with filenames for V, E and w respectively.

Example:

```
$ python shortest_path.py
$ python shortest_path.py my_v.txt my_e.txt my_w.txt
```

You are encouraged to change the input data, invent your own datasets etc. Basic input sanitizing functionality is provided in the script. You are free to change the script to your liking, add your own classes etc., as long as you pay attention to the submission rules in the next paragraph.

Submission Please make sure that your script follows these rules:

- The script name must be `shortest_path.py` and it must run in python version 2.7
- The script must accept three parameters to load graph data (already implemented).
- The script must produce an image (using the function `draw_graph()`) of the shortest path called `shortest_path.png` (already implemented).
- The script must not import additional python modules besides the ones already present.
- The script must finish without waiting for user input, i.e. remove all `plt.show()` or other calls that block execution.
- Please follow exactly the submission format: Create a single zip file with your implementation. *Do not use other formats such as tar, tar.gz, bzip. . .* Name the zip file according to your matriculation number, e.g. `0123456.zip`.

Violation of these rules will result in deduction of points!

Note that we will run a plagiarism check on the source code.