Combinatorial Optimization
Lab 07: Network Flows

Industrial Informatics Research Center
http://industrialinformatics.fel.cvut.cz/

April 1, 2020

Overview of the tutorial

- Revision of the network flows (30 mins)
- Interesting applications (15 mins)
- Pilots exercise (20 mins)

Part 1: Revision

After the lecture, each student should be able to answer the following questions:

- How is the problem of the network flows **defined**? What are the inputs, and what is the output?

  The input is 5-tuple (G, l, u, s, t), where G is some oriented graph, l and u are functions mapping edges to non-negative real (integer) numbers giving lower and upper bounds on the flow, respectively. Vertex s is the source and t is the target (also called sink).

  The output is feasible flow f.

- What is the **flow**?

  Flow $f : E(G) \to \mathbb{R}_0^+$ is a mapping from edges to the real numbers, such that Kirchhoff law, given by Equation (1), is valid for every vertex, except for the source and the sink.

$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e), \ \forall v \in V(G) \setminus \{s, t\} \tag{1}$$

  Set $\delta^+(v)$ is defined as $\{v \in V(G) \mid \exists e = (v, x) \in V(G)\}$, i.e. it is a set of outgoing edges from vertex v. Similarly, $\delta^-(v)$ is set of edges entering vertex v.

- What is the **feasible flow**?

  We say that the flow is feasible, iff

$$l(e) \leq f(e) \leq u(e), \ \forall e \in E(G) \tag{2}$$

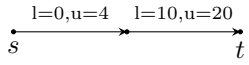
- What is optimized?

  We want to maximize the total flow outgoing from the source vertex, i.e.

$$\max_f \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e). \tag{3}$$

  We want to have maximal possible flow going from the source to the sink.

- Does some feasible flow always exist? No, see the following example. It shows that if the bounds are somehow 'incompatible', a feasible flow does not need to exist.



- How can we solve the problem of **maximal flow** using only the techniques that we have already learned?

  We can use algorithms, such as Ford-Fulkerson, Push-Relabel etc., or we can formulate it as a **linear program** and solve it by the simplex (or other) algorithm.

- Write the LP formulation of the flow problem.

$$\max \sum_{e \in \delta^+(s)} f_e - \sum_{e \in \delta^-(s)} f_e \tag{4}$$

$$\text{s.t.} \sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e, \forall v \in V(G) \setminus \{s,t\} \tag{5}$$

$$l(e) \leq f_e \leq u(e), \forall e \in E(G) \tag{6}$$

$$f_e \in \mathbb{R}_0^+, \forall e \in E(G) \tag{7}$$

  We can see that the formulation as practically same as the definition of the problem. So yeah, we are done! Or not?

- Why should we study the network flow problem, when it is possible to solve it by standard LP algorithms?

  Well, there is always exchange between expressivity and complexity. Remember ILP – the framework is rich, and we can use it to formulate many problems, but the solving algorithms will not be able to use, e.g., some specific properties of our problem, and therefore will be slow, compared to some specialized algorithms. Yes, maximum flow problem can be solved by LP, but the general simplex algorithm will probably be slower compared to specialized flows algorithms. So what are some other techniques that can be used to solve the problem? Let's see some examples.

- **Exercise**: there is a graph and fortunately somebody gave us a feasible flow as well. Can we improve it somehow?

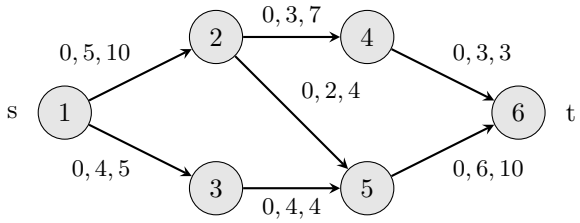


Figure 1: Example graph and a feasible flow. Edges are labelled by lower bound/flow/upper bound.

  You should find out that the flow can be improved by 2 – incrementing the flow along the edges of path 1-2-5-6. Try to formulate it generally – what is the **capacity** of the augmenting path?

  It is minimum of possible improvements along the edges.

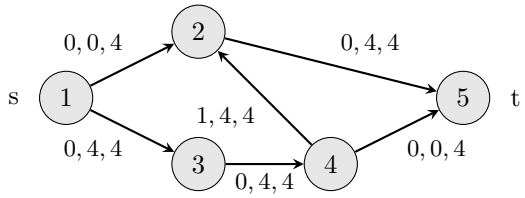
  How many potential improving paths between s and t are there in the graph?

  You will probably find at least 3, but there are, in fact, four.

  All paths (and their capacities):
  1-2-4-6, capacity 0 (because of 4-6)

1-2-5-6, capacity 2 (because of 2-5)
1-3-5-6, capacity 0 (because of 3-5)
1-3-5-2-4-6, capacity 0 (because of 3-5)

Now, think about the following example, and try to find the augmenting paths.



All paths (and their capacities):
1-2-5, capacity 0 (2-5)
1-2-4-5, capacity 3 (4-2)
1-3-4-5, capacity 0 (1-3)
1-3-4-2-5, capacity 0 (1-3)

Figure 2: Example graph with backward edge. Edges are labelled by lower bound/flow/upper bound.

It can be seen that back-edges cannot be disregarded. Therefore – augmenting paths does not respect orientation of edges. Capacity of edge can be computed as a difference between upper bound and flow (forward edges) or difference between flow and lower bound (backward edges).

By change, we came up with similar ideas as Ford and Foulkerson, so let's now discuss their method.

- What is the main idea of the **Ford-Fulkerson** algorithm? What is the complexity? Are there any disadvantages?

  The algorithm tries to find augmenting paths (see the lectures, labelling algorithm) from the source to the sink, i.e., it iteratively tries to improve the flow going from s to t. The flow is maximal iff there is no augmenting path. (Do not forget the back-edges).

  The **complexity** of the algorithm for integer bounds is $\mathcal{O}(|E| \cdot F)$, where F is volume of the maximum flow; finding of some augmenting path takes time $\mathcal{O}(|E|)$ (e.g. by DFS). Expressed like that, the algorithm is only pseudo-polynomial (bounds can be encoded by log bits, see the discussion here https://stackoverflow.com/questions/19647658/what-is-pseudopolynomial-time-how-does-it-differ-from-polynomial-time). But, if we always choose the shortest augmenting path, the complexity will be $\mathcal{O}(|E|^2 \cdot |V|)$ – Edmonds-Karp algorithm (BFS for the augmenting paths).

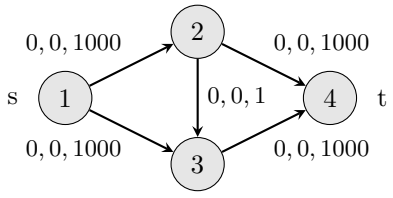


Figure 3: Graph for which Ford-Foulkerson might improve the flow only by 1 per iteration.

The **problem** is that in order to start, it needs some initial flow.

- When is it easy to find some initial flow?

  When all of the lower bounds are zero – just set the initial flow to zero.

Part 2: Interesting applications

- Edge-disjoints paths

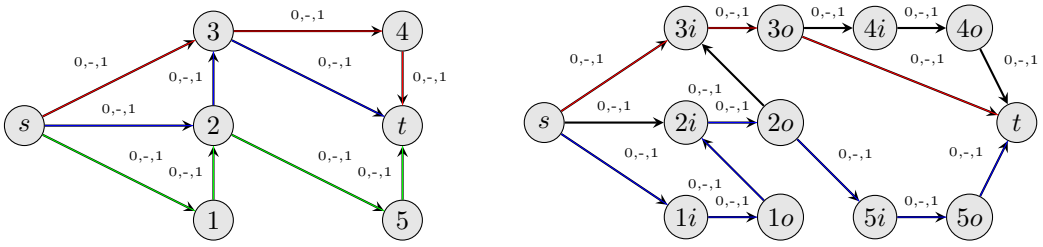
  The question is: How many edge-disjoint paths are there in the graph G between vertices s and t?

  The formulation by maximum flow is easy. Just take the original graph and define $l(e) = 0$, $u(e) = 1$ for each edge $e \in E(G)$. Then find the maximal flow. The paths can be reconstructed from the flow by modified DFS.

- Vertex disjoint paths

  Modification of the previous problem. We disallow the paths going through the same vertex. How to do it by flows?

  Just split each vertex v to v_1 and v_2 connected by edge $e = (v_1, v_2)$ with $l(e) = 0$, $u(e) = 1$. Edges going to v now go to v_1 and edges going from v go from v_2.



Edge-disjoint paths | Vertex-disjoint paths (transformed graph)

Figure 4: Edge and vertex disjoint paths

- Escape problem

  Given a finite 2d-grid and three prisoners standing somewhere in the grid, the question is whether they can reach the borders such that their paths do not overlap.

  This problem can be solved by vertex-disjoint paths. Source is connected to all starting positions and the sink is connected to all border vertices.

For other interesting applications of the flows, you may read the following text http://jeffe.cs.illinois.edu/teaching/algorithms/book/11-maxflowapps.pdf (optional).

Part 3: Pilots exercise

Airline company must organize regular training and certification of qualification for its employees (pilots). It is required to train n pilots $\{P_1, \ldots, P_n\}$, whose license would expire otherwise. For these purposes, m training sessions will be organized at times (days) $\{T_1, \ldots, T_m\}$. Availability of the pilots is given by parameters $A_{i,j}$, where

$$A_{i,j} = \begin{cases} 1, & \text{if pilot } P_i \text{ can attend training date } T_j \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Assume that each pilot should attend only a single training session.

Now formulate this problem as a flow problem (define the graph, edges, bounds, etc.).

Afterwards add some more restrictions: There must be commissioners present during the training (one commissioner looks after one pilot). There are k commissioners available and their availability is given by $D_{l,j}$, where

$$D_{l,j} = \begin{cases} 1, & \text{if commissioner } D_l \text{ can attend training date } T_j \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Moreover, each commissioner can attend at most q training dates.

Formulate the problem by maximum flows.

Optional variant: What if each day T_i had fixed maximum capacity C_i? Nodes corresponding to t can be split to two.

Solution: Solution can be obtained by constructing the graph step by step. How to model time availability of pilots? → layer between pilots and dates. Each pilot must be supervised by one instructor; time availability of instructors; each instructor at most q dates. → see how the lower bounds and the upper bounds are constructed. Try to understand the structure of the graph.

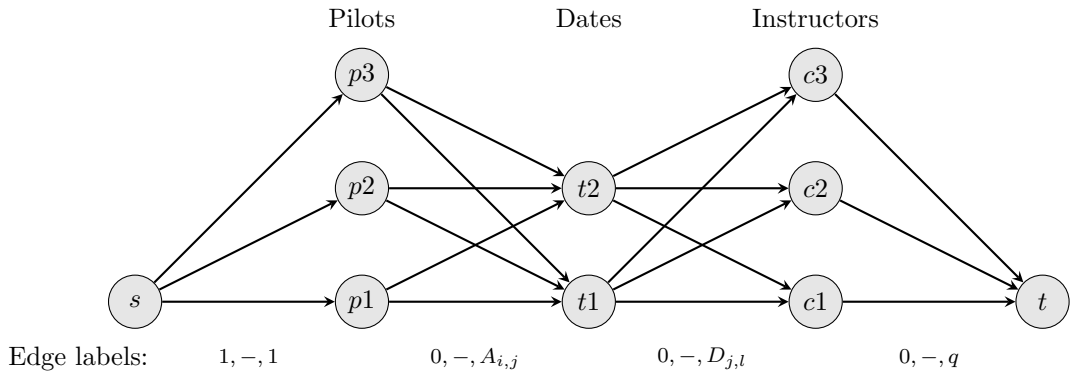


Figure 5: Structure of the graph for the pilots exercise.

Summary

After going through this lab, you should be able to describe the networks flow problem formally. You should know some basic facts about the Ford-Foulkerson algorithm and you should be able to use the network flows framework to solve some simple problems.