Combinatorial Algorithms
Lab 10: Minimum Cost Flow

Industrial Informatics Department
http://industrialinformatics.fel.cvut.cz/

April 24, 2024

Overview of the tutorial

- Revision of the min-cost flow (10 mins)
- Warehousing example (20 mins)
- Reconstruction of image by projections (20 mins)
- Cycle cancelling (15 mins)

Part 1: Revision

During the previous labs, we studied problems of the shortest paths in graphs and maximum flows in networks. Both of these problems address different components of a more general framework. Shortest paths consider edge costs, but no flow capacities, whereas maximum flow problem considers capacities, but the cost structure is very simple. The minimum cost flow problem, studied in this lab, combines both components. We consider edge costs and balances of the nodes.

Now, let's define the problem of min-cost flows formally.

- What is the **input**?

  We have 5-tuple (G, l, u, c, b), where G is an oriented graph, $l, u : E(G) \to \mathbb{R}_0^+$ represent lower bound and upper bound on the feasible flow, $c : E(G) \to \mathbb{R}$ is arc cost (for a unit flow) and $b : V(G) \to \mathbb{R}$ is the balance, representing consumption / supply of a node.

  We assume that balances satisfy $\sum_{v \in V(G)} b(v) = 0$, i.e., the total balance of the network is zero.

- What is the **output**?

  We want to find flow $f : E(G) \to \mathbb{R}_0^+$ with minimal cost $\sum_{e \in E(G)} f(e) \cdot c(e)$, which is feasible, i.e., it satisfies given bounds and balances:

$$l(e) \leq f(e) \leq u(e), \ \forall e \in E(G), \tag{1}$$

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v), \ \forall v \in V(G). \tag{2}$$

  It holds that nodes with positive balance ($b(v) > 0$) generate some flow, nodes with negative balance ($b(v) < 0$) consume some flow and nodes with zero balance ($b(v) = 0$) satisfy the Kirchhoff law.

- How to **solve** this problem?

  Again, the problem can be encoded as a linear program and solved by standard solvers.

$$\min_f \sum_{e \in E(G)} c(e) \cdot f(e) \quad \text{s.t.} \tag{3}$$

$$l(e) \leq f(e) \leq u(e), \ \forall e \in E(G), \tag{4}$$

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v), \ \forall v \in V(G), \tag{5}$$

$$f(e) \in \mathbb{R}_0^+, \ \forall e \in E(G). \tag{6}$$

  Besides this approach, there are many algorithms, such as the cycle cancelling algorithm, primal-dual algorithm, relaxation algorithm, network simplex algorithm etc.

  Transformation of the max-flow to the min-cost-flow: Note that the maximum flow problem can be transformed to min-cost flow problem by following transformation:

  1. Add edge $\tilde{e}$ from t to s with $u(\tilde{e}) = \infty$ and $c(\tilde{e}) = -1$.
  2. Set the cost of any other edge $e \in E(G)$ to 0.
  3. Set $b(v) = 0$ for all vertices (including s and t).

  Now the minimum cost circulation (negative) maximizes the flow on the edge $\tilde{e}$.

Now, it is time to solve some examples!

Part 2: Ice cream example (warehousing)

Let us solve a simple example. Imagine, that you have a company producing delicious ice cream. During the year, demand d_i of the customers varies depending on the season of the year $i \in \{1, 2, 3, 4\}$. Accordingly, the price c_i for making the ice-cream varies too (higher price when the demand is high).

Now, you have an idea – could it be possible to produce the ice cream during the seasons i when the production price c_i is low, and sell it later, when the demand (price) will be higher? Certainly, but we would need a warehouse! Let R be the capacity of the warehouse. Let h_j be the price for storing the ice-cream between season j and $(j+1)$. Let u_j be the capacity of the production. You, as the ice-cream maker, want to plan the production for the following year with minimal cost.

Exercise: Try to solve this example assuming that the warehouse needs to be empty at the beginning and end of the year.

Solution: The resulting graph will have one node for each season plus one source and one terminal. Balance of the source will be equal to the sum of all demands, while balance of the terminal will be minus the sum of all demands. Nodes corresponding to the seasons will divide the ice creams between the warehouse and the customers.

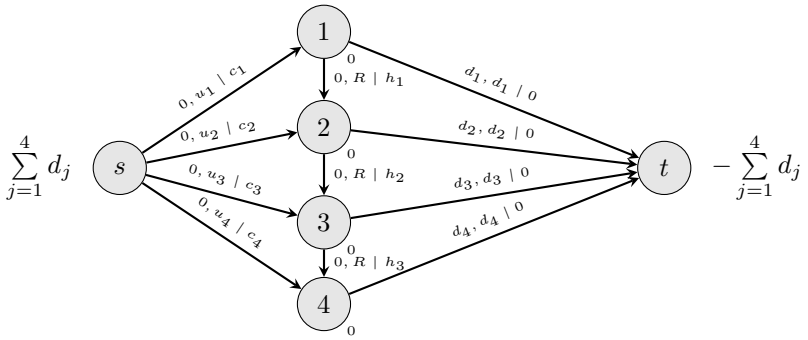


Figure 1: Min-cost flow formulation of the ice-cream problem (edges labelled by $l(e), u(e) \mid c(e)$), balances are written next to each node.

Exercise: What will change if we assume that the warehouse is not empty, i.e., that there is y pieces of ice-cream inside the warehouse at the beginning?

We could just add a vertex, representing the state of the warehouse. We would also need to adjust the balance of the source vertex accordingly.

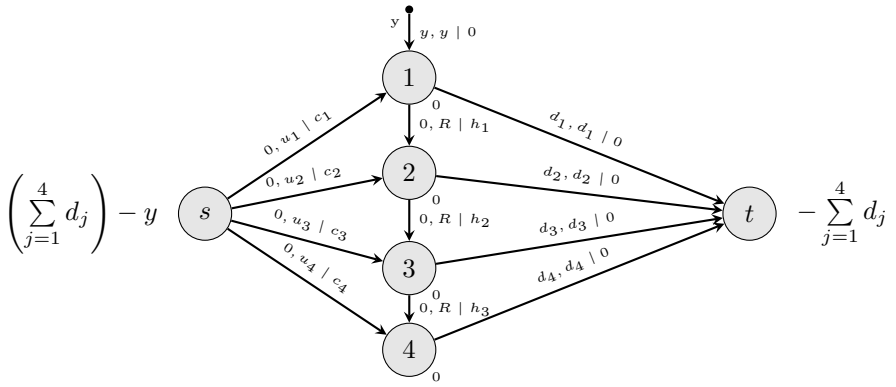


Figure 2: Solution for non-zero initial amount of ice-creams.

Question: Are you able to reduce the size of the shown graph by removing some of its vertices?

Part 3: Reconstruction of image using projections

Motivation: In medicine, in order to reconstruct a 3D image of an object, projections scanned by an X-ray machine can be used. In this example, we will assume a slightly simpler scenario of reconstructing a 2D image using 1D projections.

Example: Let us have horizontal and vertical projections of a simple 2D binary image. Given these projections, we want to reconstruct the original image. However, there might be multiple solutions to the problem. We can eliminate some of them by using additional projections (such as diagonal projections). One can see that even four projections would not be enough to reconstruct the simple image unambiguously. Still, with more projections, we can obtain more precise reconstruction.

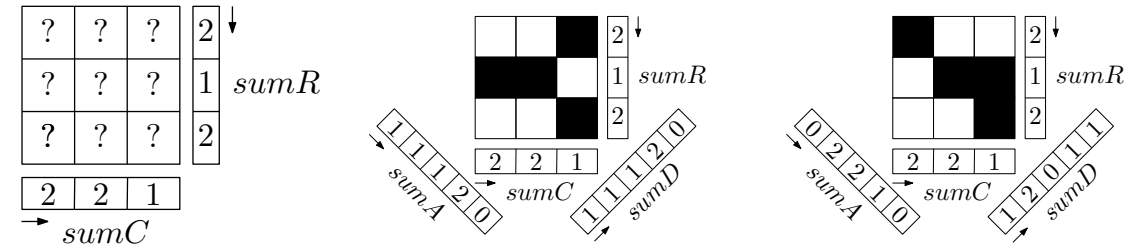


Figure 3: Projections and possible reconstructions.

Now, think how you would solve this problem using the min-cost flow formulation.

Network flows formulation: Given two projections of $(n \times n)$ binary image, we can reconstruct the image the following way. Nodes of the network will correspond to the projections, while each edge can be associated with the pixel at the specific position. Its maximal flow is limited to one, as the image is binary. Note that the number of edges is n^2, since the number of related pixels is invariant.

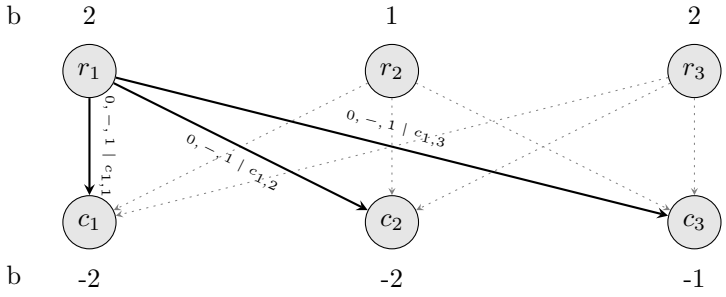


Figure 4: Network for projections $\mathcal{R}$ and $\mathcal{C}$ of the example image shown in Figure 3.

Example of the network for two projections is shown in Figure 4. The algorithm would try to reconstruct the image for all possible pairs of projections. For each pair, it builds the network (bipartite graph) and computes the min-cost flow. Balances correspond to the sums in the respective projection (positive for one projection and negative for the other one).

The cost can be computed as

$$c_{i,j} = 1 - I_{k,l},$$

where $I_{k,l}$ is the pixel value from the last iteration and $c_{i,j}$ is the cost of the edge corresponding to this pixel. By this definition of the cost, we prefer the new image to be similar to the old one.

The image is reconstructed from the flow simply by coloring the pixels corresponding to the edges with non-zero flow. The algorithm iterates until image is stable or given number of iterations was reached.

Part 4: Cycle cancelling

One of the ways to solve the min-cost flow problem is to use the **cycle cancelling algorithm** (CC). Similarly to Ford-Fulkerson (FF), it needs some initial feasible flow. Contrary to FF, which tries to find augmenting paths leading to the increase of the flow, CC iteratively finds the cycles with potential of decreasing the flow (i.e., the objective).

The initial feasible flow can be established by solving the maximum flow algorithm (add new source, connected to all nodes v with $b(v) > 0$, and new sink connected from all nodes w with $b(w) < 0$).

The CC algorithm uses the **residual graph** to find the cycles, and terminates when there are no more negative cost cycles in the graph.

The residual graph is not a network, it is a graph with labelled edges. The residual graph is always created from the input network **and the feasible flow** (it is parametrized by the feasible flow).

For simplicity, we assume that there are no parallel edges in the input network. The algorithm can work with them, but the residual graph would be a multigraph.

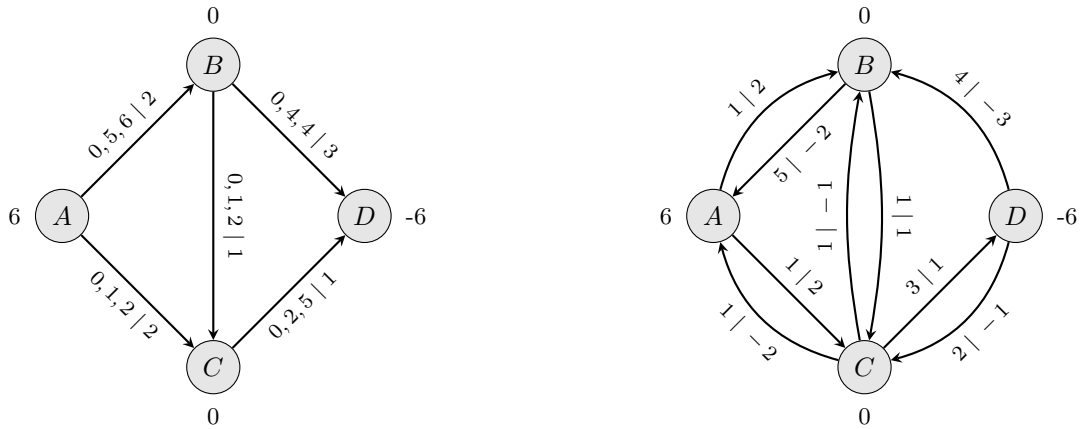
Residual graph: Residual graph G_f with respect to flow f is 4-tuple (V, E_f, u_f, c_f), where

$$
\begin{aligned}
&E_f = \{(i,j) \mid (i,j) \in E \cup E^{-1}, u_f((i,j)) > 0\},\\
&u_f : E \cup E^{-1} \to \mathbb{R}_0^+,\\
&c_f : E \cup E^{-1} \to \mathbb{R},
\end{aligned}
$$

$$
\forall (i,j) \in E(G) : \begin{cases} u_f((i,j)) = u((i,j)) - f((i,j)), \\ u_f((j,i)) = f((i,j)) - l((i,j)), \\ c_f((i,j) = c((i,j)), \\ c_f((j,i) = -c((i,j)). \end{cases}
$$

Set E contains the oriented edges of the original graph, whereas set E^{-1} contains the reversed edges of E, i. e., $E^{-1} = \{(j,i) \,|\, (i,j) \in E\}$.

Example (residual graph): Now, try to create a residual graph to the graph illustrated in Fig. 5a).



a) Original graph G with flow f (edges labelled by $l, f, u|c$) b) Residual graph G_f (edges labelled by $u_f|c_f$)

Figure 5: Creation of residual graph

Now, we can find negative cycles in G_f, with respect to c_f. One of them is B-C-D-B with value $1 + 1 - 3 = -1$. Increasing the flow on the edges of the negative cycle leads to improvement of the objective value, while balances of the vertices stay the same. Try it!

Of course, we update the flow associated with the edges on cycle C by maximal possible value, which is $\delta(C) = \min_{e \in C} u_f(e)$ (e.g. δ(B-C-D-E) = 1). If the edge of the cycle corresponds to the 'forward' edge, the flow on that edge is increased, otherwise it is decreased.

The CC algorithm terminates when there is no negative cycle.

Do you know how to find a negative cycle in G_f?

How to find negative cycle in G_f: Add a dummy source vertex s connected to all other vertices $v \in V(G_f)$ by oriented edges (s, v) with zero cost. Then you can run modified Bellman-Ford algorithm, which tests

$$\forall (i,j) \in E(G_f) : l(j) \leq l(i) + c((i,j)),$$

if it does not hold for some edge, then there is a negative cycle. To get the cycle from the solution of BF, just iterate through the predecessor vector.

Notes on the cycle cancelling: The algorithm is very similar to Ford-Fulkerson – instead of augmenting paths, we try to find negative cycles. If there is no more, the algorithm terminates. Both algorithms need an initial flow to start. In fact, the maximal flow problem can be solved by finding augmenting paths on the residual graphs (any path between s and t will be augmenting).

Complexity: assuming that $|C|$ is the maximum cost and $|U|$ is the maximum upper bound on the flow, $(|E| \cdot |C| \cdot |U|)$ is an upper bound on the initial flow cost, while $-(|E| \cdot |C| \cdot |U|)$ is a lower bound. Each iteration of the algorithm changes the objective function value by some $\delta < 0$. Since we assume that all data are integral, the algorithm terminates in $\mathcal{O}(|E| \cdot |C| \cdot |U|)$ iterations. During each iteration, we need to find a negative cycle in the residual graph, which can be done by Bellman-Ford algorithm, which runs in $\mathcal{O}(|E| \cdot |V|)$. Therefore the complexity of the CC algorithm is $\mathcal{O}(|V| \cdot |E|^2 \cdot |C| \cdot |U|)$.

Note that this complexity is only pseudopolynomial (because of $|C|$ and $|U|$). However, the algorithm can be made polynomial if minimum mean cost negative cycles are being found, which can be done in $\mathcal{O}(|V| \cdot |E|)$. This is similar to Ford-Fulkerson, which can be improved to Edmonds-Karp algorithm by finding the best augmenting paths.

Summary

After going through this lab, you should learn about the min-cost flow problem. You should be able to formulate simple problems as flows. Also, you should understand the basic concepts of Ford-Fulkerson and Cycle-cancelling algorithms.