MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

Introduction

Everywhere we look in our daily lives, networks are apparent. Some examples are the electrical and power networks, the telephone networks, the national highway system, the rail networks, and the airline service networks.
In all of these problem domains, and in many more, we wish to move some entity from one point to another in the underlying network and to do so as efficiently as possible, both to provide good service to the users of the network and to use the underlying transmission facilities effectively.

The topic of this assignment is the "*Minimum cost flow problem"*:
if we incur a cost per unit flow on a network with arc capacities and we need to send units of a good that reside at one or more points in the network to one or more points, how can we send the material at minimum possible cost?

Mathematical programming formulation

Let $G = (N, A)$ be a *directed network* defined by a set *N* of *n nodes* and a set *A* of *m directed arcs*. Each arc $(i, j) \in A$ has an associated cost c_{ij} that denotes the cost per unit flow on that arc. We assume that the flow cost varies linearly with the amount of flow. We also associate with each arc $(i, j) \in A$ a *capacity* u_{ij} that denotes the maximum amount that can flow on the arc and a *lower bound* l_{ij} that denotes the minimum amount that must flow on the arc.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

We associate with each node $i \in N$ an integer number $b(i)$ representing its supply/demand. If $b(i) > 0$, node i is a *supply node*; if $b(i) < 0$, node i is a *demand node* with a demand of $-b(i)$, and if $b(i) = 0$, node i is a *transshipment node*.
The decision variables in the minimum cost flow problem are arc flows and we represent the flow on the arc $(i, j) \in A$ by x_{ij}.

The minimum cost flow problem is an optimization model formulated as follows:

Minimize $\sum_{(i,j)\in A} c_{ij} x_{ij}$

subject to $\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \quad for\ all\ i \in N$ (1)

$l_{ij} \leq x_{ij} \leq u_{ij} \quad for\ all\ (i, j) \in A$

where $\sum_{i=1}^{n} b(i) = 0$

We refer to the constraints in (1) as *mass balance constraint*. The first term represents the total *outflow* of the node and the second term represents the total *inflow* of the node.
The total outflow minus the total inflow must be equal to the supply/demand of the node.
In most applications, the lower bounds on arc flows are zero. The flow bounds typically model physical capacities or restrictions.

Let $U = max\{u_{ij}: (i, j) \in A\}$: the largest magnitude of any supply/demand or finite arc capacity.
Let $C = max\{c_{ij}: (i, j) \in A$: the largest magnitude of any arc cost.

Assumptions

1. All data (cost, supply/demand, and capacity) are integral;
2. The network is directed;
3. $\sum_{i=1}^{n} b(i) = 0$;
4. The minimum cost flow problem has a feasible solution;
5. All arc costs are nonnegative;
6. We assume that network *G* contains an uncapacitated directed path between every pair of nodes. We impose this condition, if necessary, by adding artificial arcs $(1, j)$ and $(j, 1)$ and assigning large cost and infinite capacity to each of these arcs. No such arc would appear in a minimum-cost solution unless the problem contains no feasible solution without artificial arcs.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

Residual Network

The residual network *G(x)* corresponding to a flow *x* is defined as follows. We replace each arc $(i,j) \in A$ with two arcs (i, j) and (j, i). The arc (i, j) has cost c_{ij} and *residual capacity* $r_{ij} = u_{ij} - x_{ij}$, and the arc (j, i) has cost c_{ij} and residual capacity $r_{ij} = x_{ij}$. The residual network consists only of arcs with positive residual capacity.

The potential of a node

We associate a real number $\Pi(i)$, unrestricted in sign, with each node $i \in N$.
We refer to $\Pi(i)$ as the potential of node i. As said before $\Pi(i)$ is the dual variable corresponding to the mass balance constraint of node i.
For a given set of node potentials Π, we define the reduced costs of an arc (i,j) as:
$c_{ij} = c_{ij} - \Pi(i) + \Pi(j)$.

Optimality Conditions

1. *Negative Cycle Optimality Conditions*:

   A feasible solution x^* is an optimal solution of the minimum cost flow problem if and only if satisfies the negative cycle optimality conditions: namely, the residual network $G(x^*)$ contains no negative cost (directed) cycle.
2. *Reduced Cost Optimality Conditions*:

   A feasible solution x^* is an optimal solution of the minimum cost flow problem if and only if some set of node potentials Π satisfy the following reduced cost optimality conditions: $c_{ij} = c_{ij} - \Pi_i + \Pi_j \geq 0$ for every arc $(i, j)\ in\ G(x^*)$.
3. *Complementary Slackness Optimality Conditions*:

   A feasible solution x^* is an optimal solution of the minimum cost flow problem if and only if for some set of node potentials Π, the reduced costs and flow values satisfy the following complementary slackness optimality conditions for every arc $(i, j) \in A$:
   - If $c_{ij}^{\Pi} > 0$ then $x_{ij}^* = 0$;
   - If $c_{ij}^{\Pi} = 0$ the n $0 \leq x_{ij}^* \leq u_{ij}$;
   - If $c_{ij}^{\Pi} < 0$ then $x_{ij}^* = u_{ij}$.

Minimum cost flow duality

We associate a *dual variable* with every constraint of the primal except for the nonnegativity restriction on arc flows.
We associate the variable $\Pi(i)$ with the *mass balance constraint* of the node i and the variable a_{ij} with the capacity constraint of $arc(i,j)$.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

The dual minimum cost flow problem can be stated as follows:

Maximize $w(\Pi, \alpha) = \sum_{i \in N} b(i)\Pi(i) - \sum_{(i,j) \in A} u_{ij} a_{ij}$

subject to $\Pi(i) - \Pi(j) - \alpha_{ij} \leq c_{ij}$ $for\ all\ (i,j) \in A$

$\alpha_{ij} \geq 0$ $for\ all\ (i,j) \in A$

Π_i $unrestricted\ for\ all\ i \in N$

Algorithms

- V: the number of nodes;
- E: the number of edges;
- U: the maximum capacity;
- C: the maximum absolute value of cost.

Cycle Cancelling Algorithm

The negative cycle optimality conditions suggest one simple algorithmic approach for solving the minimum cost flow problem, which we call the cycle-canceling algorithm. This algorithm maintains a feasible solution and at every iteration attempts to improve its objective function value. The algorithm first establishes a feasible flow x in the network by solving a maximum flow problem. Then it iteratively finds negative cost-directed cycles in the residual network and augments flow on these cycles. The algorithm terminates when the residual network contains no negative cost-directed cycle.

```
algorithm cycle-canceling;
begin
    establish a feasible flow x in the network;
    while G(x) contains a negative cycle do
    begin
        use some algorithm to identify a negative cycle W;
        δ := min{r_ij : (i, j) ∈ W};
        augment δ units of flow in the cycle W and update G(x);
    end;
end;
```

Figure 9.7 Cycle canceling algorithm.

My implementation uses the Edmonds-Karp (it calculates the augmenting paths of minimum cardinality) algorithm to find the maximum flow and the Bellman-Ford algorithm to detect negative cycles.

The time complexity of the algorithm is $O(V \times E^2 \times C \times U)$.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

Successive Shortest Path Algorithm

The cycle-canceling algorithm maintains the feasibility of the solution at every step and attempts to achieve optimality. In contrast, the successive shortest path algorithm maintains the optimality of the solution at every step and strives to attain feasibility. It maintains a solution x that satisfies the nonnegativity and capacity constraints but violates the mass balance constraints of the nodes. At each step, the algorithm selects a node s with excess supply and a node t with unfulfilled demand and sends flow from s to t along the shortest path in the residual network. The algorithm terminates when the current solution satisfies all the mass balance constraints.

The solution x will be admissible only when the algorithm terminates so, in the middle phases, x is defined as a *pseudo flow*:

$$e_i = b_i - \sum_{j \in \Gamma_i} x_{ij} + \sum_{j \in \Gamma_i^{-1}} .$$

```
algorithm successive shortest path;
begin
    x := 0 and π := 0;
    e(i) := b(i) for all i ∈ N;
    initialize the sets E := {i : e(i) > 0} and D := {i : e(i) < 0};
    while E ≠ ∅ do
    begin
        select a node k ∈ E and a node l ∈ D;
        determine shortest path distances d(j) from node s to all
            other nodes in G(x) with respect to the reduced costs c^π_ij;
        let P denote a shortest path from node k to node l;
        update π := π − d;
        δ := min[e(k), − e(l), min{r_ij : (i, j) ∈ P}];
        augment δ units of flow along the path P;
        update x, G(x), E, D, and the reduced costs;
    end;
end;
```

My implementation uses a simple implementation of Dijkstra with time complexity $O(V^2)$.
Since performances was not the key point of the assignment I decided to keep the implementation simple using a simple queue. It can be improved with a Fibonacci Heap reaching time complexity of $O(E + Vlog(V))$.

The total time complexity is $O(V^3 \times U)$ and it can be improved to $O(V \times U \times (E + Vlog(V)))$.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

Primal-Dual Algorithm

The primal-dual algorithm for the minimum cost flow problem is similar to the successive shortest path algorithm in the sense that it also maintains a pseudo flow that satisfies the reduced cost optimality conditions and gradually converts it into a flow by augmenting flows along shortest paths. In contrast, instead of sending flow along one shortest path at a time, it solves a maximum flow problem that sends flow along all shortest paths.
The primal-dual algorithm generally transforms the minimum cost flow problem into a problem with a single excess node and a single deficit node. We transform the problem into this form by introducing a source node s and a sink node t. For each node i with $b(i) > 0$, we add a zero cost arc (s, i) with capacity $b(i)$, and for each node i with $b(i) < 0$, we add a zero cost arc (i, t) with capacity $-b(i)$.

The primal-dual algorithm solves a maximum flow problem on a subgraph of the residual network $G(x)$, called the admissible network, which we represent as $G^{\circ}(x)$. We define the admissible network $G^{\circ}(x)$ with respect to a pseudo flow x that satisfies the reduced cost optimality conditions for some node potentials $\Pi(i)$; the admissible network contains only those arcs in $G(x)$ with a zero reduced cost. The residual capacity of an arc in $G^{\circ}(x)$ is the same as that in $G(x)$.

```
algorithm primal–dual;
begin
    x := 0 and π := 0;
    e(s) := b(s) and e(t) := b(t);
    while e(s) > 0 do
    begin
        determine shortest path distances d(·) from node s to all other nodes in G(x) with
            respect to the reduced costs c̄ij;
        update π := π − d;
        define the admissible network G°(x);
        establish a maximum flow from node s to node t in G°(x);
        update e(s), e(t), and G(x);
    end;
end;
```

My implementation uses Edmonds-Karp to establish the maximum flow and Dijkstra to determine the shortest path distances.
As said before, the Dikstra implementation is not the optimal one so the total time complexity is: $O(min\{V \times U, V \times C\})(V^2 + V \times E^2)$ but it can be improved to $O(min\{V \times U, V \times C\})(E + Vlog(V) + V^2 \times \sqrt{E})$ using Dijkstra implemented using *Fibonacci Heap* and using the *Highest-Label Preflow-Push* algorithm to obtain the maximum flow.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

Command Line Tool

It is a simple command line tool using C++ which permits solving the following problems:

- Maximum flow;
- Minimum cost maximum flow.

The algorithms implemented are the following:

- Base Algorithms:
  - BFS;
  - Bellman-Ford (very useful to find negative cycles);
  - Dijkstra: basic implementation using a queue $O(V^2)$, a future improvement can be the usage of a Fibonacci Heap ($O(E + Vlog(V))$).
- Maximum Flow:
  - Edmonds-Karp: similar to the famous Ford-Fulkerson but instead of a DFS it's used a BFS to find the augmenting paths of minimum cardinality. As a future improvement, it can be implemented a faster algorithm such as Highest-label preflow push.
- Minimum cost flow:
  - Cycle cancelling;
  - Successive shortest path;
  - Primal-dual.

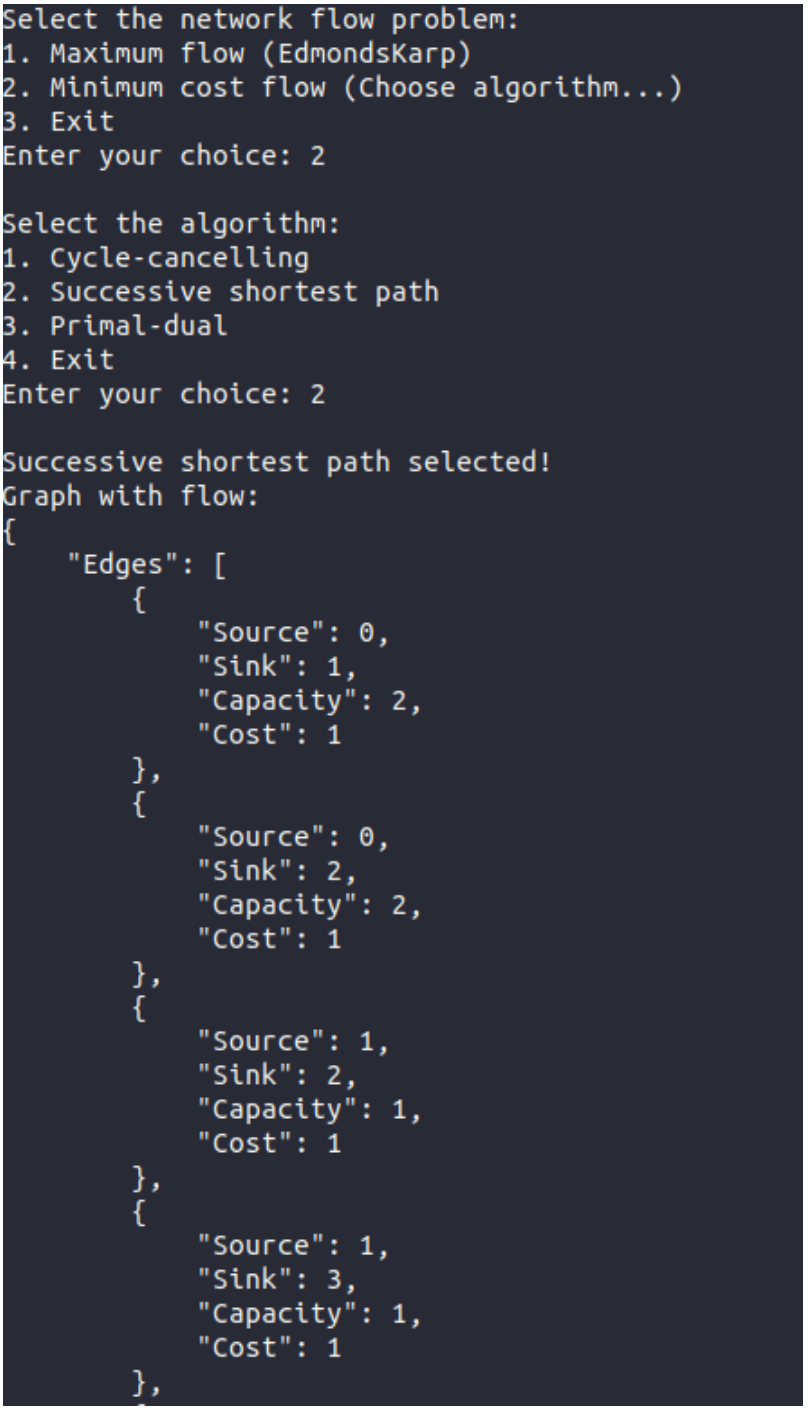
To see more details and how to use the tool see the repo's README.
Repo link.

MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

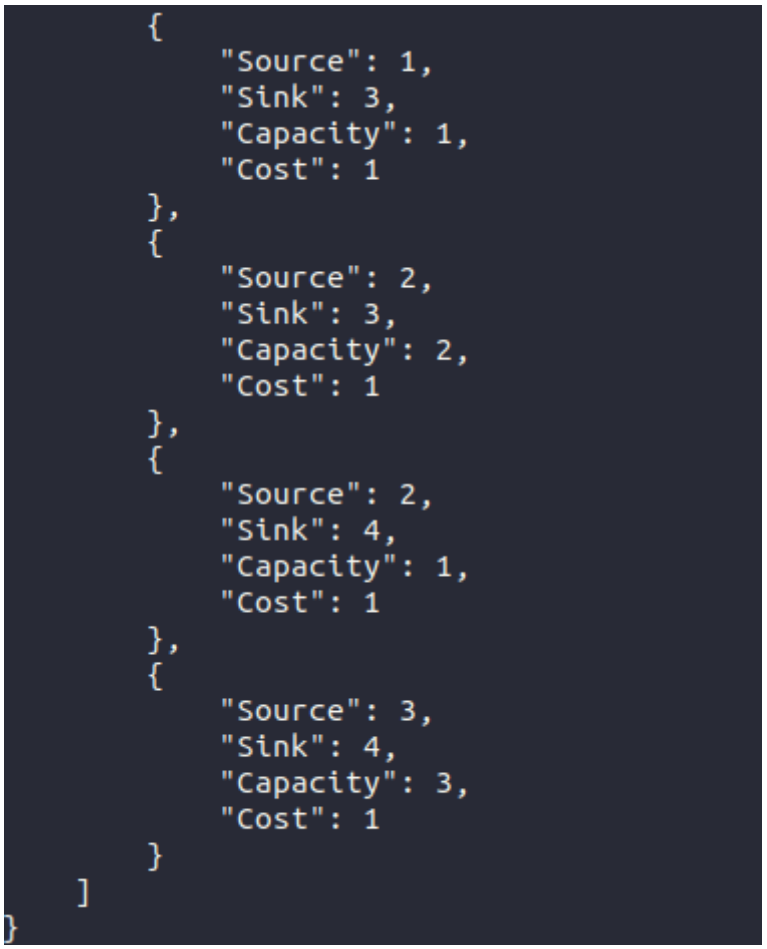
Execution demo

Here are some screenshots of the execution:



MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871



As you can see it's possible to execute the tool by the command line both by passing the file name as an argument or inserting it as the first step of the execution. Then it's possible to choose the type of problem you want to solve and, in case of the minimum cost flow, you can also choose the algorithm to use.
As a result, you will obtain the flow and the graph with the flow passing on each edge inside the "capacity" value.

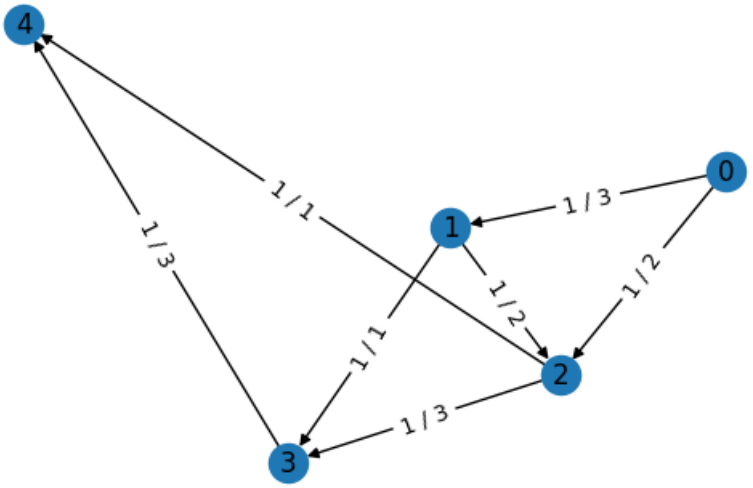
MINIMUM COST FLOW PROBLEM

Drudi Lorenzo - 0000969871

Furthermore, the repo contains a simple solver implemented in Python using **matplotlib** and the networkx library. It permits to:

- solve the maximum flow problem
- solve the minimum cost flow problem
- draw the graph.

Here a screenshot:




```
Maximum flow graph:
{0: {1: 2, 2: 2}, 1: {2: 2, 3: 0}, 2: {3: 3, 4: 1}, 3: {4: 3}, 4: {}}

Maximum flow value:
4

Minimum cost flow graph:
{0: {1: 2, 2: 2}, 1: {2: 1, 3: 1}, 2: {3: 2, 4: 1}, 3: {4: 3}, 4: {}}

Minimum cost flow cost:
12
```