The Max-Flow Problem

Themistoklis Charalambous

6 February 2014

Introduction

The Max-Flow Problem:

We have a graph $(\mathcal{N}, \mathcal{A})$ with flow bounds $x_{ij} \in [b_{ij}, c_{ij}]$ for each arc (i, j) and two special nodes: the *source*, denoted by s, and the *sink*, denoted by t. We want to find a flow vector that makes the divergence of all nodes other than s and t equal to 0, while maximizing the divergence of s.

Roughly, the objective is to move as much flow as possible from s to t, while observing the capacity constraints.

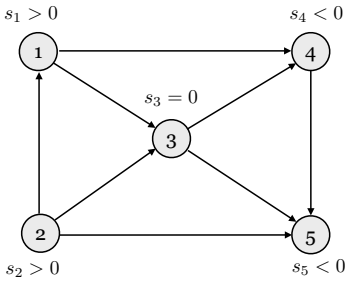
The structures of the *shortest path* and *max-flow* problems are seen as complementary. **Why?**

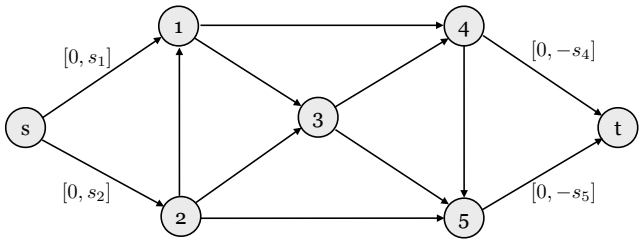
Historical facts

- Formulated in 1954 as a simplified model of Soviet railway traffic flow.
- In 1955, Ford & Fulkerson created the first known algorithm, the FordFulkerson algorithm.
- Over the years, various improved solutions to the maximum flow problem were discovered, e.g.,
  - the shortest augmenting path algorithm of Edmonds and Karp
  - blocking flow algorithm of Dinitz

Why is it useful?

- Arises in many practical contexts:
  - calculate the throughput of a highway system;
  - calculate the throughput of a communication network;
- Arises as a subproblem in more complicated problems; e.g.,
  - *feasibility* of a minimum cost flow problem $\rightarrow$ max-flow problem.

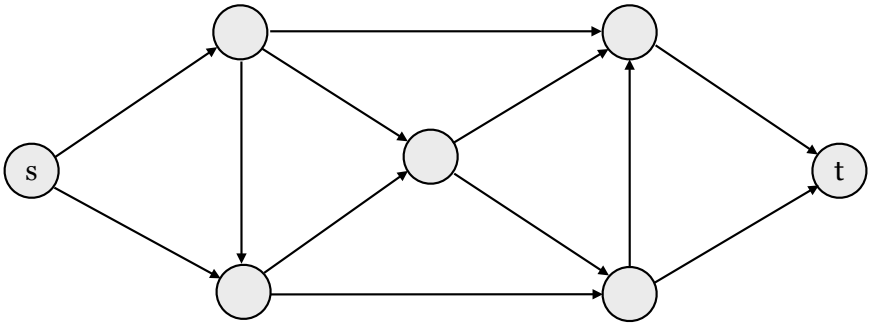




Formulation - I

The problem becomes a special case of the minimum cost flow problem:

- by assigning cost 0 to all arcs, and
- by introducing an artificial arc (t, s) with cost -1.



Cuts in a graph

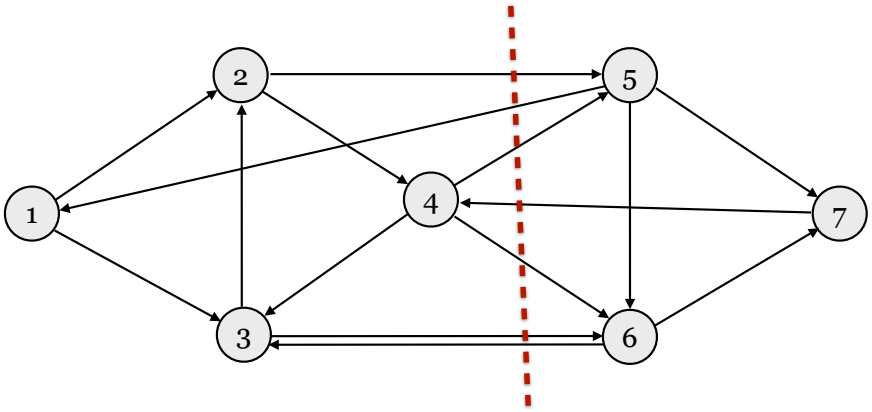
- A cut Q in a graph $(\mathcal{N}, \mathcal{A})$ is a partition of the node set $\mathcal{N}$ into two sets: $\mathcal{S}$ and $\mathcal{N} - \mathcal{S}$. Denoted by

$$Q = [\mathcal{S}, \mathcal{N} - \mathcal{S}] \ .$$

- Partition is *ordered*, i.e., $[\mathcal{S}, \mathcal{N} - \mathcal{S}] \neq [\mathcal{N} - \mathcal{S}, \mathcal{S}]$.
- Q^+ and Q^- are the sets of forward and backward arcs of the cut, respectively; i.e.,

$$Q^+ = \{(i,j) \in \mathcal{A} | i \in \mathcal{S}, j \notin \mathcal{S}\} \ ,$$
$$Q^- = \{(i,j) \in \mathcal{A} | i \notin \mathcal{S}, j \in \mathcal{S}\} \ .$$

- Q is *nonempty* if $Q^+ \cup Q^- \neq \emptyset$; otherwise, Q is *empty*.



- A flux across a nonempty set Q is the total net flow coming out of $\mathcal{S}$, i.e.,

$$F(Q) \triangleq \sum_{(i,j)\in Q^+} x_{ij} - \sum_{(i,j)\in Q^-} x_{ij}$$

- $F(Q) = \sum_{i\in\mathcal{S}} y_i$, i.e., the sum of the divergences y_i of the nodes in $\mathcal{S}$. **Why?**

- Given $b_{ij} \leq x_{ij} \leq c_{ij}$ for all $(i,j) \in \mathcal{A}$, the capacity of a nonempty cut Q is

$$C(Q) \triangleq \sum_{(i,j)\in Q^+} c_{ij} - \sum_{(i,j)\in Q^-} b_{ij} \ .$$

- The flux of Q is less than or equal to the capacity of Q, i.e.,

$$F(Q) \leq C(Q).$$

- If $F(Q) = C(Q)$, then Q is said to be saturated cut with respect to x.

Proposition 1

Let x be a *capacity feasible* flow vector, and let s and t be two nodes. Then, one of the two holds:

(1) there exists a simple path from s to t that is unblocked with respect to x;

(2) there exists a saturated cut that separates s from t.

Unblocked Path Search Algorithm

- For $k = 0, 1, 2, \ldots$, given T_k,
  - terminate if either T_k is empty or $t \in T_k$
  - otherwise, set

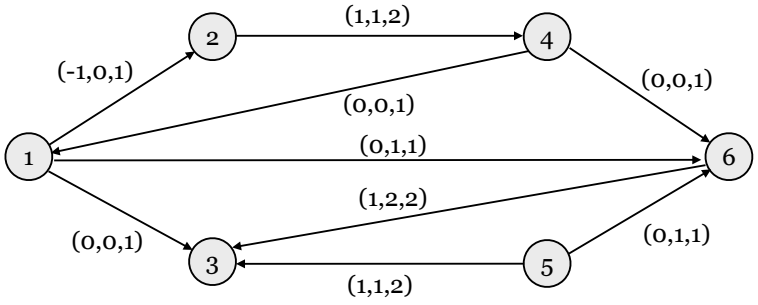
$$T_{k+1} = \{n \notin \cup_{i=0}^{k} T_i | \exists m \in T_k \text{ and either an arc } (m, n) \text{ with } x_{mn} < c_{mn}, \text{ or, an arc } (n, m) \text{ with } b_{nm} < x_{nm}\}$$

- mark each $n \in T_{k+1}$ with the label "(m, n)" or "(n, m)", accordingly.

Note: A special case of *Breadth-First Search* (BFS) algorithm.

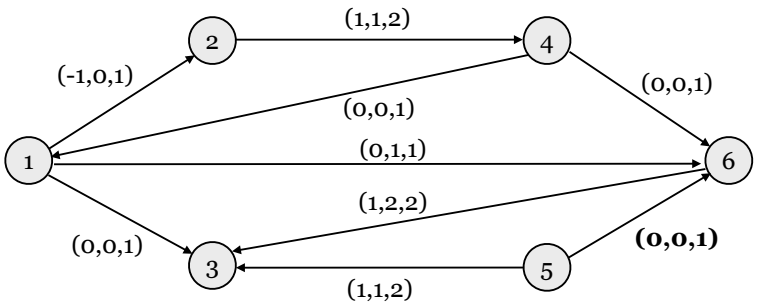
Example 1:

Use the Unblocked Path Search Algorithm to find either a simple path or a saturated cut from nodes 1 to 5.



Example 2:

Use the Unblocked Path Search Algorithm to find either a simple path or a saturated cut from nodes 1 to 6.



The Max-Flow/Min-Cut Theorem

Theorem

(a) If x^* is an optimal solution of the max-flow problem, then the divergence out of s corresponding to x^* is equal to the minimum cut capacity over all cuts separating s from t.

(b) If all lower arc flow bounds are zero, the max-flow problem has an optimal solution, and the maximal divergence out of s is equal to the minimum cut capacity over all cuts separating s from t.

Proof. The proof relies on Proposition 1 (Proposition 3.1 in Network Optimization book).

The Maximal and Minimal Saturated Cuts

- There may exist more than one saturated cuts Q, separating s and t.
- Maximal saturated cut $\overline{Q} \triangleq [\overline{\mathcal{S}}, \mathcal{N} - \overline{\mathcal{S}}]$, where $\overline{\mathcal{S}}$ corresponds to *the union* of all node sets $\mathcal{S}$ such that $[\mathcal{S}, \mathcal{N} - \mathcal{S}]$ is a saturated cut.
- $\overline{Q}$ also separates s and t; therefore, for $(i,j) \in \overline{Q}^+$, then $x^*_{ij} = c_{ij}$.
- Similarly for the minimal saturated cut but we use *the intersection* of the node sets $\mathcal{S}$ such that $[\mathcal{S}, \mathcal{N} - \mathcal{S}]$ is a saturated cut.

The Ford-Fulkerson Algorithm

- The algorithm starts with a *feasible flow vector* x.
- Given a path P (augmenting path) from s to t, unblocked with respect to x (this can be found using the Unblocked Path Search Algorithm) we can
  - increase the flow of all *forward* arcs $(i,j) \in P^+$, and
  - decrease the flow of all *backward* arcs $(i,j) \in P^-$
- The maximum increment of flow change is

$$\delta = \min\left\{\{c_{ij} - x_{ij} | (i,j) \in P^+\}, \{x_{ij} - b_{ij} | (i,j) \in P^-\}\right\}.$$

- The resulting flow vector $\overline{x}$ (flow augmentation along P) is

Iteration of the Ford-Fulkerson Algorithm

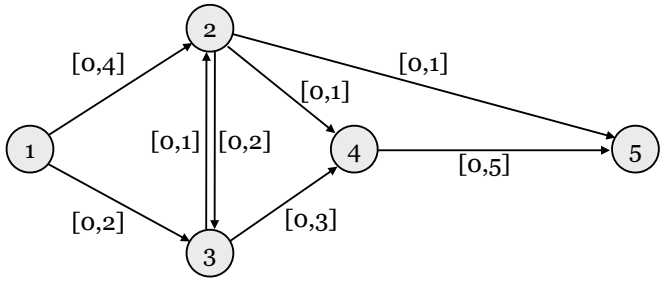
- Use the Unblocked Path Search Algorithm to either
  (1) find a saturated cut separating s from t, or
  (2) find an unblocked path P with respect to x starting from s and ending at t
- **In case (1):** terminate the algorithm; current flow vector solves the max-flow problem.
- **In case (2):** perform an augmenting along P and go to the next iteration.

Remarks:

- If the flow bounds are integer, the flow vector is integer and δ is a positive integer at each iteration, then the algorithm terminates.
- If the flow bounds are rational, the flow vector is rational, they can be multiplied by a large integer number and problem remains unaffected.
- If the problem data are irrational
  - the problem terminates if a specific unblocked path search algorithm is used, augmenting paths with as few arcs as possible
  - the algorithm need not be terminated if unblocked algorithms are constructed using a different method.

Example 3:

Use the Ford-Fulkerson algorithm to find a maximum flow from $s = 1$ to $t = 5$.



Decomposition of Infeasible Network Problems

Consider the minimization of a *separable cost function*

$$\sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}), \qquad (\text{e.g., } f_{ij}(x_{ij}) = a_{ij}x_{ij})$$

subject to *conservation of flow constraints*

$$\sum_{j|(i,j)\in\mathcal{A}} x_{ij} - \sum_{j|(j,i)\in\mathcal{A}} x_{ji} = s_i, \qquad \forall i \in \mathcal{N},$$

and *capacity constraints*

$$b_{ij} \leq x_{ij} \leq c_{ij}, \qquad \forall (i,j) \in \mathcal{A}.$$

We assume s_i given and $\sum_{i\in\mathcal{N}} s_i = 0$, but capacities c_{ij} not sufficiently large.

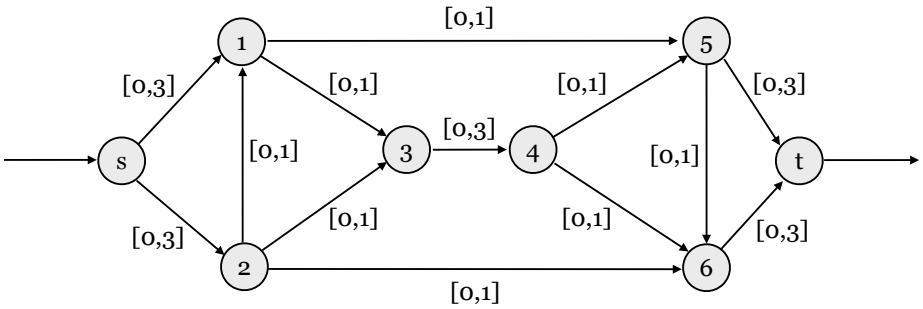
- In order to fulfill as much demand as possible $\Rightarrow$ minimize the cost function over the set of all *maximally feasible flows*, i.e.,

$$\text{minimize} \sum_{i\in\mathcal{N}} |s_i - y_i|$$

- We can find a *maximally feasible flow* x^* by adding s and t and solving the max-flow problem.
- The vector x^* defines corresponding unique minimal $[\mathcal{S}_{\min}, \mathcal{N} - \mathcal{S}_{\min}]$ and maximal $[\mathcal{S}_{\max}, \mathcal{N} - \mathcal{S}_{\max}]$ saturated cuts.
- Decompose the problem into 2 (if $\mathcal{S}_{\min} = \mathcal{S}_{\max}$) or 3 feasible and independent subproblems, separating the source node set from the demand node set.

Example 4:

In the following graph, the problem is to send 6 units of flow from node s and t, while satisfying constraints $[0, c_{ij}]$ and minimizing a cost function $\sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij})$. The problem is infeasible (**why?**), so let's optimize over all maximally feasible solutions.



Thank You!



Questions?

For more questions:

themisc@kth.se

chatw@kth.se

carlofi@kth.se

michael.rabbat@mcgill.ca