Mathematics for Decision Making: An Introduction

Lecture 13

Matthias Köppe

UC Davis, Mathematics

February 17, 2009

Reminder: Flows in networks

General structure: Flows in networks

- In general, consider a **digraph** (directed graph) (V, A), where
  - V is a set of vertices,
  - A is a set of (directed arcs), which are (ordered) pairs $(a, b) \in V \times V$ with $a \neq b$ (no loops!)
- We call two designated vertices $r, s \in V$ the **source** and the **sink**
- An r-s **flow x** is a vector of real values $x_{a,b}$ for every arc $(a, b) \in A$ such that, in every vertex (except for source r and sink s), **flow conservation constraints** are satisfied:

$$f_{\mathbf{x}}(b) := \sum_{\substack{a \in V: \\ (a,b) \in A}} x_{a,b} - \sum_{\substack{c \in V: \\ (b,c) \in A}} x_{b,c} = 0 \quad \text{for } b \in V,\ b \neq r, s.$$

- We call $f_{\mathbf{x}}(b)$ the **excess** of the flow **x** at vertex b.
- The excess $f_{\mathbf{x}}(s)$ at the sink is called the **value** of the flow **x**; note $f_{\mathbf{x}}(s) = -f_{\mathbf{x}}(r)$.
- Often, **capacities** $u_{(a,b)}$ are given, i.e., upper bounds on the flow values $x_{(a,b)}$. We call a flow **x feasible** if it is non-negative and respects the upper bounds (if given).

Reminder: Flows in networks, II

Maximum flow problem

Given a digraph (V, A), source r, sink s, arc capacities $u_{a,b}$:
Find a feasible flow $\mathbf{x}$ of maximum value $f_{\mathbf{x}}(s)$.

Minimum-cost r-s flow problem

Given a digraph (V, A), source r, sink s, arc capacities $u_{a,b}$, per-unit costs $c_{a,b}$, and a flow value ϕ:
Find a feasible flow $\mathbf{x}$ of value $f_{\mathbf{x}}(s) = \phi$ that has minimum total flow costs $\sum c_{a,b} x_{a,b}$.

Note that it is essential to use the excess function $f_{\mathbf{x}}$ at the sink s if we want to measure or prescribe the flow value. If we just determine the amount of incoming flow, the flow model is not correct! Disconnected solutions are possible.

We now turn to study the maximum flow problem. We will discuss an optimality criterion (strong duality theory): "max-flow min-cut", and combinatorial algorithms to compute the maximum flow.

Flows: Decomposition into Paths and Cycles

We have interpreted an integer r-s flow of value 1 as a path from r to s (plus circulations), when we modeled the shortest-path problem as a minimum-cost flow.

Lemma (Integer flows as path packings)

There exists a family $(P_1, \ldots, P_k)$ of directed paths from r to s in (V, A) with

$$\left|\{i : (a,b) \in P_i\}\right| \leq u_{a,b} \quad \textit{for all } (a,b) \in A$$

***if and only if** there exists a feasible integral r-s flow $\mathbf{x}$ of value k.*

Proof.

When we add up unit **path flows** along the P_i, we clearly get a feasible integral r-s flow of value k. It remains to show the converse.

- If there is any directed circuit C with $x_{a,b} > 0$ for all arcs $(a,b) \in C$, then construct a **new feasible flow $\mathbf{x}'$** by reducing the arc flows along this cycle by 1 (or by $\min\{x_{a,b} : (a,b) \in C\}$). Continue until no such directed circuit exists.
- If $k > 0$, follow a path starting from r, through arcs (a,b) with $x_{a,b} > 0$. We never get stuck due to flow conservation, and finally reach s. Call this (simple) path P_k. Subtract the path flow from $\mathbf{x}$, and continue. $\square$

Flows: Decomposition into Paths and Cycles, II

For general flows we have:

Theorem

Every r-s flow of nonnegative value is the sum of at most $m = |A|$ flows, each of which is a path flow or a circuit flow.

These decomposition results give us a natural idea how to construct a maximum flow:

- Start with the zero flow $\mathbf{x} = \mathbf{0}$
- Send flow along a path from source to sink
- Repeat until there is no path with positive “bottleneck capacity”

Unfortunately, this fails. While we **can decompose** flows into paths, by using arbitrary paths, one at a time, we **cannot construct** a maximum flow by packing arbitrary paths. We either need to “look ahead”, or use a different idea that allows us to continue if we get stuck.

Augmenting Paths

- An important idea is to consider also paths P from r to s that include arcs **in the wrong direction** (so these paths P are **not directed paths**).
- We call these arcs **reverse arcs** (as opposed to **forward arcs**).
- For a given flow $\mathbf{x}$, we shall call a path **x-incrementing** if
  - for every forward arc $(a, b) \in P$, we have $x_{a,b} < u_{a,b}$, and
  - for every reverse arc $(a, b) \in P$, we have $0 < x_{a,b}$.

  An $\mathbf{x}$-incrementing path from r to s is called **x-augmenting**.
- Why are these paths useful? We can:
  - increase flow by some $\varepsilon > 0$ on all forward arcs, and
  - decrease flow by this ε on backward arcs.

  The resulting, **augmented flow**
  - respects the flow conservation constraints;
  - for ε small enough, is a **feasible** flow (capacities and nonnegativity hold);
  - Has a **larger flow value**.

We now claim:

Theorem

A feasible flow $\mathbf{x}$ *is maximum* ***if and only if*** *there is no* $\mathbf{x}$*-augmenting path.*

Optimality Criteria

A theorem like this is easiest to prove if we have an optimality criterion that is based on a certificate of optimality.

- In the case of shortest paths, feasible potentials served as optimality certificates:
  - Any feasible potential **y** provided a **lower bound** for the length of any path from r to v.
  - For an optimal solution of this **minimization problem** (minimum-cost paths P_v from r to v) we could construct a feasible potential (namely, setting y_v to be the cost of P_v), for which this lower bound was **attained**.

  So the lower bound, matching the value of the feasible solution, provided the optimality certificate.
- In the case of maximum flow, we should be looking for useful **upper bounds** with similarly strong properties.

Useful Upper Bounds: Cut Capacities

- Let $R \subseteq V$ be a set of vertices. Then we call the arc set

$$\delta(R) = \{ (a,b) \in A : a \in R, b \notin R \}$$

  the **cut induced by** R. Any arc set that can be obtained this way is called a **cut**.
- An r**-**s **cut** is a cut induced by a set R with $r \in R$ and $s \notin R$.

Lemma

For any r-s cut $\delta(R)$ and any r-s flow $\mathbf{x}$,

$$\mathbf{x}(\delta(R)) - \mathbf{x}(\delta(V \setminus R)) = f_{\mathbf{x}}(s).$$

Proof.

Add up flow conservation constraints. □

Corollary

For any r-s cut $\delta(R)$ and any feasible r-s flow $\mathbf{x}$,

$$f_{\mathbf{x}}(s) \leq \mathbf{u}(\delta(R)).$$

The Max-Flow Min-Cut Theorem

So any r-s cut provides an upper bound. But we have something much stronger:

Theorem (Max-Flow Min-Cut; Ford–Fulkerson [1956], Kotzig [1956])

If there is a maximum r-s flow, then

$$\max\{\, f_{\mathbf{x}}(s) : \mathbf{x} \textit{ is a feasible } r\text{-}s \textit{ flow} \,\} = \min\{\, \mathbf{u}(\delta(R)) : \delta(R) \textit{ is an } r\text{-}s \textit{ cut} \,\}.$$

So, for an optimal solution of the maximization problem, there always exists an r-s cut, for which the provided upper bound is **attained**.

Proof of the Max-Flow Min-Cut Theorem, and the theorem on slide 6.

- By the Corollary, $\leq$ holds. Only need to find one pair $\mathbf{x}$, $\delta(R)$ for which $=$ holds.
- Let $\mathbf{x}$ be a maximum r-s flow [or, a flow without $\mathbf{x}$-augmenting path]
- We define $R = \{\, v \in V$: there is an $\mathbf{x}$-incrementing path from r to $v \,\}$.
- Then $r \in R$ (trivial) and $s \notin R$ (otherwise $\mathbf{x}$-augmenting path, so $\mathbf{x}$ not maximum).
- For $(a,b) \in \delta(R)$, we have $x_{a,b} = u_{a,b}$; for $(a,b) \in \delta(V \setminus R)$, we have $x_{a,b} = 0$ (Otherwise, there would be an $\mathbf{x}$-incrementing path to b.)
- By the Lemma, $f_{\mathbf{x}}(s) = \mathbf{x}(\delta(R)) - \mathbf{x}(\delta(V \setminus R)) = \mathbf{u}(\delta(R))$. □

Consequences from Max-Flow Min-Cut, I

Corollary

If $\mathbf{u}$ *is integral and there exists a maximum flow, there also exists a maximum flow that is integral.*

Thus integer flow problems are as easy to solve as continuous flow problems! This is in contrast to the general situation of linear programs vs integer programs: the latter are much harder to solve. This integrality property of network flows is one reason why network flow formulations are useful as a modeling tool.

Proof.

- Among all integral flows, pick one of maximum value, call it $\mathbf{x}$.
- If there existed an $\mathbf{x}$-augmenting path, then since $\mathbf{u}$ is also integral, we could increase the flow by an integer amount.
- So $\mathbf{x}$ is already a maximum flow. $\square$

Consequences from Max-Flow Min-Cut, II

Corollary (Complementary slackness)

Let $\mathbf{x}$ *be a feasible* r*-*s *flow and* $\delta(R)$ *be an* r*-*s *cut. Then:* $\mathbf{x}$ *is a maximum* r*-*s *flow* ***if and only if***

$$
\begin{aligned}
x_{a,b} &= u_{a,b} && \text{for } (a,b) \in \delta(R) \\
x_{a,b} &= 0 && \text{for } (a,b) \in \delta(V \setminus R)
\end{aligned}
$$

Auxiliary directed graphs

- An algorithm for constructing maximum flows thus repeatedly needs to find $\mathbf{x}$-augmenting paths.
- We already know algorithms for constructing (shortest) paths, so we might want to use them.
- However, reverse arcs complicate the picure.
- So let us construct an **auxiliary digraph** $G(\mathbf{x})$ such that finding a **directed path** from r to s is the same as finding an $\mathbf{x}$-augmenting path in G.

Constructing the auxiliary digraph $G(\mathbf{x})$ from $G = (V, A)$

- For any arc $(a, b) \in A$ with $x_{a,b} < u_{a,b}$, introduce an arc (a, b) in $G(\mathbf{x})$.
- For any arc $(b, a) \in A$ with $x_{b,a} > 0$, introduce an arc (a, b) in $G(\mathbf{x})$.

(We allow to introduce parallel arcs.)

The Ford–Fulkerson Algorithm

Ford–Fulkerson Maximum Flow Algorithm

Input: A digraph $G = (V, A)$ with arc capacities $\mathbf{u}$, vertices r and s.
Output: A maximum flow $\mathbf{x}$ and a set $R \subseteq V$ inducing a minimum cut $\delta(R)$.

- Set $\mathbf{x} := \mathbf{0}$.
- While we find a directed r-s path P in the auxiliary graph $G(\mathbf{x})$:
  Determine the **x-width** of P:
  $$\varepsilon := \min\left\{ \begin{array}{l} \min\{\, u_{a,b} - x_{a,b} : (a,b) \text{ forward in } P\,\}, \\ \min\{\, x_{a,b} : (a,b) \text{ reverse in } P\,\} \end{array} \right\}$$
  Augment $\mathbf{x}$ along P by ε.
- Set R to the set of vertices that can be reached by paths from r in $G(\mathbf{x})$.

The Ford–Fulkerson Algorithm: Termination, Efficiency

Theorem (Termination of the Algorithm)

If $\mathbf{u}$ *is integral and there is a maximum flow (of value* K*), then the Ford–Fulkerson Maximum Flow Algorithm terminates after at most* K *augmentations.*

Proof.

Each of the augmentations increases the flow value by an integer amount. □

- This also establishes that the Ford–Fulkerson Algorithm is a **pseudo-polynomial algorithm** (for inputs with integer data that have a maximum flow).
  (By the Max-Flow Min-Cut Theorem, the flow value is the same as some cut capacity, so it is at most $\sum u_{ab}$, a quantity that is polynomial in the given data.)
- Examples that really take K augmentations (with a specific choice of a sequence of augmenting paths) can be easily constructed.
- Moreover, if there is no maximum flow, the procedure might fail to terminate.
- So, we are **not completely happy** with this basic algorithm.
- A scaling approach (with data $\mathbf{u}/2^k$, for k decreasing to 0) leads to a polynomial algorithm; we omit the details.
- Even better, it turns out that a **specific choice** of **x**-augmenting paths (which is currently unspecified) will lead to a strongly polynomial algorithm.

A Strongly Polynomial Time Variant

Theorem (Dinits [1970], Edmonds–Karp [1972])

*If each augmentation is along a **shortest** (i.e., minimum number of arcs) **augmenting path**, then the algorithm terminates after at most $nm = |V| \cdot |A|$ augmentations.*

- To prepare the proof, consider an augmentation along a (shortest) augmenting path $P = (v_0, \ldots, v_k)$ of length k, leading from flow $\mathbf{x}$ to flow $\mathbf{x}'$.
- Denote by $d_{\mathbf{x}}(v, w)$ the least number of arcs in a directed path from v to w in the auxiliary digraph $G(\mathbf{x})$; we set $d_{\mathbf{x}}(v, w) = +\infty$ if no such directed path exists.
- Since subpaths of shortest paths are shortest, we have $d_{\mathbf{x}}(r, v_i) = i$ and $d_{\mathbf{x}}(v_i, s) = k - i$.

A Strongly Polynomial Time Variant, II

Lemma

Shortest-augmenting-path augmentations ***never decrease the length of shortest directed paths in the auxiliary digraph*** *from the source r to any node v and from any node v to the sink s:*

$$d_{\mathbf{x}'}(r,v) \geq d_{\mathbf{x}}(r,v) \quad \text{and} \quad d_{\mathbf{x}'}(v,s) \geq d_{\mathbf{x}}(v,s).$$

In particular, they never decrease the length of a shortest augmenting path:

$$d_{\mathbf{x}'}(r,s) \geq d_{\mathbf{x}}(r,s)$$

This lemma implies that shortest-augmenting-path augmentations proceed in **stages**, during which augmenting paths of **constant length** are used:

- Augmentations along paths of length 1 (possibly none)
- Augmentations along paths of length 2 (possibly none)

  ⋮
- Augmentations along paths of length $n-1$ (possibly none).

It now suffices to bound the number of augmentations of each stage in a strongly polynomial way.

A Strongly Polynomial Time Variant, III

Let $\tilde{A}(\mathbf{x})$ be the set of arcs $(a, b) \in A$ that appear in a shortest $\mathbf{x}$-augmenting path.

Lemma

If a shortest-augmenting-path augmentation does not increase the length of a shortest augmenting path, i.e., $d_{\mathbf{x}'}(r, s) = d_{\mathbf{x}}(r, s)$*, then* $\tilde{A}(\mathbf{x}')$ *is a proper subset of* $\tilde{A}(\mathbf{x})$*.*

Proof of the theorem.

From the second lemma, in each stage, there are at most $m = |A|$ augmentations per stage.
From the first lemma, there are at most $n - 1$ stages.
So, in total at most nm augmentations. $\square$

Programming Project

The programming project is due **Tuesday, March 17**.

- Implement the Bellman–Ford and the Dijkstra algorithms in a general-purpose programming language of your choice.
- Verify that your implementations are correct, by comparing (for small examples) with an optimization model in ZIMPL/SCIP.
- Document the running time of your implementations for all test problems (that can be solved within reasonable time).
- Your program should read data files in the format of the example data files available at: `http://www.math.ucdavis.edu/~mkoeppe/teaching/2009w/mat-180-1/graphs/`
- Note that these data files describe the edges (with costs) of undirected graphs; it is understood that the shortest path algorithms should run on the directed graph obtained by replacing each edge by two arcs.
- Bonus points, towards homework, can be earned by making these implementations fast enough that the large sparse examples (`bgfh-*`) work within seconds.