CSC 373: Algorithm Design and Analysis
Lecture 12

Allan Borodin

February 4, 2013
Announcements
- Term test 1 in tutorials. Need to use only two rooms due to sickness and conference. We will be using BA 2145 and BA 2155.

Today’s outline
- Different ways to choose an augmenting path so as to ensure polynomial time termination
- Immediate applications of the max-flow problem
A consequence of the max-flow min-cut theorem

**Corollary**
If all capacities are integral (or rational), then any implementation of the Ford-Fulkerson algorithm will terminate with an optimal integral max flow.

**Rational capacities**
Why does the claim about integral capacities imply the same for rational capacities?
The runtime of Ford-Fulkerson

**Observation**

Each augmenting path has residual capacity at least one.

- The max-flow min-cut theorem along with the above observation ensures that with integral capacities, Ford-Fulkerson must always terminate and the number of iterations is at most:
  \[ C = \text{the sum of edge capacities leaving } s. \]
- Hence complexity is \( O(m + nC). \)

**Notes**

- There are bad ways to choose augmenting paths such that with irrational capacities, the Ford-Fulkerson algorithm will not terminate.
- However, even with integral capacities, there are bad ways to choose augmenting paths so that the Ford-Fulkerson algorithm does not terminate in polynomial time.
Bad example for naive Ford-Fulkerson

Figure: The numbers denote the capacities of the edges.

- The max-flow is clearly $2X$.
- A naive Ford-Fulkerson algorithm could alternate between
  - pushing a 1 unit flow along the augmenting path $s \to a \to b \to t$
  - pushing a 1 unit flow along the augmenting path $s \to b \to a \to t$
- This leads to a runtime of $\Omega(X)$, which is exponential if say $X$ is given in binary.
Some ways to achieve polynomial time

- **Choose an augmenting path having shortest distance:** This is the Edmonds-Karp method and can be found in CLRS. It has running time $O(nm^2)$, where $n = |V|$ and $m = |E|$.

- **There is a “weakly polynomial time” algorithm in KT**
  - Here the number of arithmetic operations depends on the length of the integral capacities.
  - It follows that always choosing the largest capacity augmenting path is at least weakly polynomial time.

- There is a history of max flow algorithms leading to a recent $O(mn)$ time algorithm (see http://tinyurl.com/bczkdfz).

- The method I like to present (although not the fastest) is Dinitz’s algorithm which has runtime $O(n^2m)$.
  - A shortest augmenting-path method based on the concept of a blocking flow in the leveled graph.
  - Has some additional advantages beyond the somewhat better running time of Edmonds-Karp.
Dinitz’s algorithm

Definition

- Define \( \text{level}(u) = \) length of shortest path from \( s \) to \( u \) in \( G_f \).
- Let the “leveled graph” w.r.t the residual graph \( G_f \) be the graph \( L_f = (\hat{V}, \hat{E}) \) where
  - \( \hat{V} = \{v \mid v \text{ reachable from } s\} \)
  - \( (u, v) \in \hat{E} \text{ if and only if } \text{level}(v) = \text{level}(u) + 1 \text{ in } G_f \).
- A blocking flow \( \tilde{f} \) is a flow such that every \( s-t \) path in \( L_f \) has a saturated edge (i.e. an edge \( e \) such that \( \tilde{f}(e) = c_f(e) \)).

Dinitz’s algorithm

1: Initialize \( f(e) = 0 \) for all \( e \in E \).
2: while \( t \) is reachable from \( s \) in \( G_f \) do
3: \hspace{0.5cm} Construct \( L_f \) from \( G_f \)
4: \hspace{0.5cm} Find a blocking flow \( \tilde{f} \) w.r.t. \( L_f \) and set \( f := f + \tilde{f} \)
5: end while
6: % There’s no more augmenting path
Proof Sketch

**Claims**

1. The algorithm halts in at most \( n - 2 \) iterations.
2. A blocking flow in the leveled graph can be found in time \( O(mn) \).

**Proof.**

Let \( f \) be a flow. Let \( f' \) be the updated flow after one iteration of Dinitz's algorithm, and let \( \text{level}' \) be the updated level w.r.t. the graph \( G_{f'} \).

1. The proof of this claim rests on two facts:
   - For every node \( v \in L_{f'} \), \( \text{level}'(v) \geq \text{level}(v) \) since every edge in \( L_{f'} \) is either an edge in \( G_f \) or the reverse of an edge in \( L_f \).
   - Since \( f' \) was a blocking flow, \( \text{level}'(t) > \text{level}(t) \).
Proof Sketch

Claims

1. The algorithm halts in at most \(n - 2\) iterations.
2. A blocking flow in the leveled graph can be found in time \(O(mn)\).

Proof.

Let \(f\) be a flow. Let \(f'\) be the updated flow after one iteration of Dinitz's algorithm, and let \(\text{level}'\) be the updated level w.r.t. the graph \(G_{f'}\).

1. The proof of this claim rests on two facts:
   - For every node \(v \in L_{f'}\), \(\text{level}'(v) \geq \text{level}(v)\) since every edge in \(L_{f'}\) is either an edge in \(G_f\) or the reverse of an edge in \(L_f\).
   - Since \(f'\) was a blocking flow, \(\text{level}'(t) > \text{level}(t)\).

2. The leveled graph can be computed in \(O(m)\). And using depth first search we can compute a blocking path in time \(O(mn)\).
An application of max-flow: the maximum bipartite matching problem

The maximum bipartite matching problem

- Given a bipartite graph \( G = (V, E) \) where
  - \( V = V_1 \cup V_2 \) and \( V_1 \cap V_2 = \emptyset \)
  - \( E \subseteq V_1 \times V_2 \)
- **Goal:** Find a maximum size matching.

- We do not know any efficient DP or greedy optimal algorithm for solving this problem.
- But we can efficiently reduce this problem to the max-flow problem.
The reduction

Figure: Assign every edge of the network flow a capacity 1.
The reduction preserves solutions

**Claims**

1. Every matching $M$ in $G$ gives rise to an integral flow $f_M$ in the newly constructed network flow $F_G$ with $\text{val}(f_M) = |M|$

2. Conversely every integral flow $f$ in $F_G$ gives rise to a matching $M_f$ in $G$ with $|M_f| = \text{val}(f)$.

Time complexity for bipartite matching using this reduction.

- $O(mn)$ using any Ford Fulkerson algorithm since the max flow is at most $n$ and all capacities are integral.

- Dinitz’s algorithm can be used to obtain a runtime $O(m\sqrt{n})$. 
A few more comments on this reduction

- When we get to our next big topic (NP completeness), we will be focusing on decision problems and as a decision problem we have $|M| \geq k$ iff $\text{val}(f_M) \geq k$.
- The reduction we are using is very efficient (linear time in the representation of the graph) and it is a special type of polynomial time reduction which we will call a polynomial time transformation.

Alternating and augmenting paths in graphs

There is a graph theoretic concept of an augmenting path relative to a matching (in an arbitrary graph).

- An alternating path $\pi$ relative to a matching $M$ is one whose edges alternate between edges in $M$ and edges not in $M$.
- An augmenting path is an alternating path that starts and ends with nodes not in $M$.

The reduction provides a 1-1 correspondence between augmenting paths in the bipartite $G$ w.r.t. $M_f$ and augmenting paths in $G_{f_M}$. 
Can this be extended to maximum weighted bipartite matching?

- In the **weighted bipartite matching problem** we are given an edge weighted bipartite graph \( G = (V, E) \) with \( V = V_1 \cup V_2 \) and say integral weights \( w : E \to \mathbb{N} \).
- **Goal:** compute a matching \( M \) so as to maximize \( \sum_{e \in M} w(e) \).
- A more or less obvious idea now is to form a flow network with new distinguished source and terminal nodes \( s \) and \( t \).
- We would then set the capacity of the directed edge \((x, y)\) to be \( c(x, y) = w(x, y) \) for all \((x, y) \in E\).
- For edges leaving \( s \) and entering \( t \) we set
  
  \[
  c(s, x) = \max_y \{w(x, y) : (x, y) \in E\},
  \]
  
  \[
  c(y, t) = \max_x \{w(x, y) : (x, y) \in E\}.
  \]

  Why doesn’t this work?
Disjoint paths: Another similar application of max flow

- A natural problem of interest in fault tolerant networks is to ensure that there are sufficiently many edge disjoint paths.
- Given a directed graph $G = (V, E)$ with a distinguished source node $s$ and terminal node $t$.
- **Goal:** compute the maximum number of edge disjoint paths from $s$ to $t$.
- Similar to the bipartite matching transformation, we view $G$ as a flow network $\mathcal{F}_G$ by setting the capacity of all edges equal to 1.
- Once again, because of integrality and unit capacities, we can argue that there are $k$ edge disjoint paths in $G$ iff $\mathcal{F}_G$ has (integral) flow $k$.
- And hence we can deduce Menger’s theorem which states that the maximum number of edge-disjoint $s$-$t$ paths in a directed graph is equal to the minimum number of edges in an $s$-$t$ cut.
- The same theorem holds for undirected graphs.
The \( \{0,1\} \) metric labeling problem

- We now wish to consider one more application of max flow-min cut. Namely, we will consider the \( \{0,1\} \) metric labeling problem as discussed in §12.6 and §7.10 of Kleinberg and Tardos.

- This is in fact a special case of a more general metric labeling problem defined as follows:
  - The input is an edge weighted graph \( G = (V, E) \), a set of labels \( L = \{a_1, \ldots, a_r\} \) in a metric space with distance metric \( d \), and functions \( w : E \to \mathbb{R}^\geq 0 \) and \( c : V \times L \to \mathbb{R}^\geq 0 \).
  - \( c(u, a_j) \) is the cost of giving label \( a_j \) to node \( u \).
  - **Goal:** find a labeling \( \lambda \) of the nodes \( \lambda : V \to L \) so as to minimize
    \[
    \sum_u c(u, \lambda(u)) + \sum_{(u, v) \in E} w(u, v) \cdot d(\lambda(u), \lambda(v))
    \]

- For example, the nodes might represent documents, the labels are topics and the edges are links between documents weighted by the importance of the link.

- When there are 3 or more labels the problem is NP-hard even for the case of the \( \{0,1\} \) metric \( d \) for which \( d(a_i, a_j) = 1 \) for \( a_i \neq a_j \) (and \( d(a, a) = 0 \) by the definition of a metric).
The labeling problem with 2 labels

- When there are only 2 labels, the only metric is the \( \{0, 1\} \) metric.
- While the labeling problem is NP-hard for 3 or more labels, it is solvable in polynomial time for 2 labels by reducing the problem to the min cut problem.
- There is also a 2-approximation algorithm for the \( \{0, 1\} \) metric and 3 or more labels by another reduction to min cut. (And there are other non-constant approximation algorithms for arbitrary metrics.)
- Informally, the idea is that we can construct a flow network such that the nodes on the side of the source node \( s \) will correspond to say nodes labeled \( a \) and the node on the side of the terminal node \( t \) will correspond to the nodes labeled \( b \).
- We will place capacities between the source \( s \) and other nodes to reflect the cost of a mislabel and similarly for the terminal \( t \).
- The min cut will then correspond to a min cost labeling.