


Network Flow: A Better Ford-Fulkerson and Bipartite Matching

CMSC 451, Summer 2009



Some slides by Kevin Wayne
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

Reminders

- Homework 3 due Monday
 - 7.2, 7.5, 7.9, 7.15 (7.9 and 7.15 b should follow full algorithm write-up guidelines)
- Midterm on Friday
- Project Checkpoint 2 Monday – pass all public tests (2 are up, 1 will be added)

Augmenting Path Algorithm

```

Augment(f, c, P) {
  b ← bottleneck(P)
  foreach e ∈ P {
    if (e ∈ E) f(e) ← f(e) + b
    else f(erev) ← f(erev) - b
  }
  return f
}
    
```

the minimum cost of any edge along the path
forward edge
reverse edge
decrease the flow in the original graph along the edge by b if our path in the residual graph uses a reverse edge

```

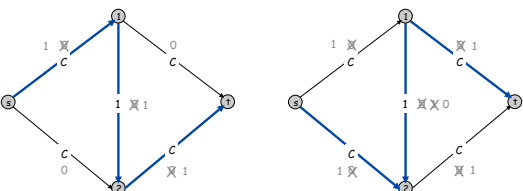
Ford-Fulkerson(G, s, t, c) {
  foreach e ∈ E f(e) ← 0
  Gr ← residual graph
  while (there exists augmenting path P) {
    f ← Augment(f, c, P)
    update Gr
  }
  return f
}
    
```

7.3 Choosing Good Augmenting Paths

Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?
m, n, and log C

A. No. If max capacity is C, then algorithm can take C iterations.



Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ .
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ .

G_f $G_f(100)$

Capacity Scaling

```

Scaling-Max-Flow(G, s, t, c) {
  foreach e ∈ E f(e) ← 0
  Δ ← smallest power of 2 greater than or equal to C
  G_r ← residual graph

  while (Δ ≥ 1) {
    G_Δ(Δ) ← Δ-residual graph
    while (there exists augmenting path P in G_Δ(Δ)) {
      f ← augment(f, c, P)
      update G_r(Δ)
    }
    Δ ← Δ / 2
  }
  return f
}
    
```

Capacity Scaling: Correctness

Assumption. All edge capacities are integers between 1 and C.

Integrity invariant. All flow and residual capacity values are integral.

Correctness. If the algorithm terminates, then f is a max flow.

Pf.

- By integrity invariant, when $\Delta = 1 \Rightarrow G_r(\Delta) = G_r$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths. •

Capacity Scaling: Running Time

Lemma 1. The outer while loop (while $\Delta \geq 1$) repeats $1 + \lceil \log_2 C \rceil$ times.
Pf. Initially $C \leq \Delta < 2C$. Δ decreases by a factor of 2 each iteration. •

Lemma 2. Let f be the flow at the end of a Δ -scaling phase. Then the value of the maximum flow is at most $v(f) + m\Delta$. — proof on next slide

Lemma 3. There are at most $2m$ augmentations per scaling phase.

- Let f be the flow at the end of the previous scaling phase.
- Lemma 2 $\Rightarrow v(f_{opt}) \leq v(f) + m(2\Delta)$.
- Each augmentation in a Δ -phase increases $v(f)$ by at least Δ . •

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. Each augmentation can take time $O(m)$. The algorithm can be implemented to run in $O(m^2 \log C)$ time. •

Capacity Scaling: Running Time

Lemma 2. Let f be the flow at the end of a Δ -scaling phase. Then value of the maximum flow is at most $v(f) + m\Delta$.

Pf. (almost identical to proof of max-flow min-cut theorem)

- We show that at the end of a Δ -phase, there exists a cut (A, B) such that $\text{cap}(A, B) \leq v(f) + m\Delta$.
- Choose A to be the set of nodes reachable from s in $G_r(\Delta)$.
- By definition of A , $s \in A$.
- By definition of f , $t \notin A$.

$$\begin{aligned}
 v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\
 &= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
 &\geq \text{cap}(A, B) - m\Delta \quad \blacksquare
 \end{aligned}$$

Intuition: f is max flow in $G_r(\Delta)$. Adding back arcs with capacity less than Δ can only increase capacity of cut by at most $m\Delta$.

7.5 Bipartite Matching

Matching

Matching.

- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .
- Max matching: find a max cardinality matching.

Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .
- Max matching: find a max cardinality matching.

matching
1-2', 3-1', 4-5'

Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .
- Max matching: find a max cardinality matching.

max matching
1-1', 2-2', 3-3', 4-4'

Bipartite Matching

Max flow formulation.

- Create directed graph $G' = (L \cup R \cup \{s, t\}, E')$.
- Direct all edges from L to R, and assign infinite (or unit) capacity.
- Add source s , and unit capacity edges from s to each node in L.
- Add sink t , and unit capacity edges from each node in R to t .

Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in $G =$ value of max flow in G' .

Pf. \leq

- Given max matching M of cardinality k .
- Consider flow f that sends 1 unit along each of k paths.
- f is a flow, and has cardinality k . ■

Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in $G =$ value of max flow in G' .

Pf. \geq

- Let f be a max flow in G' of value k .
- Integrality theorem $\Rightarrow k$ is integral and can assume f is 0 or 1.
- Consider $M =$ set of edges from L to R with $f(e) = 1$.
 - each node in L and R participates in at most one edge in M
 - $|M| = k$: consider cut $(L \cup s, R \cup t)$ ■

Perfect Matching

Def. A matching $M \subseteq E$ is **perfect** if each node appears in exactly one edge in M .

Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.

- Clearly we must have $|L| = |R|$.
- What other conditions are necessary?
- What conditions are sufficient?

19

Perfect Matching

Notation. Let S be a subset of nodes, and let $N(S)$ be the set of nodes adjacent to nodes in S (or the *neighbors* of S).

Observation. If a bipartite graph $G = (L \cup R, E)$, has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. Each node in S has to be matched to a different node in $N(S)$.

20

Marriage Theorem

Marriage Theorem. [Frobenius 1917, Hall 1935] Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then, G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. \Rightarrow This was the previous observation.

21

Proof of Marriage Theorem

Pf. \Leftarrow Suppose G does not have a perfect matching. (Contrapositive)

- Formulate as a max flow problem and let (A, B) be min cut in G' .
- By max-flow min-cut, $\text{cap}(A, B) < |L|$.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$.
- $\text{cap}(A, B) = |L_B| + |R_A|$.
- Since min cut can't use ∞ edges: $N(L_A) \subseteq R_A \cup \{s\}$.
- $|N(L_A)| \leq |R_A| = \text{cap}(A, B) - |L_B| < |L| - |L_B| = |L_A|$.
- Choose $S = L_A$.

22

Bipartite Matching: Running Time

Which max flow algorithm to use for bipartite matching?

- Generic augmenting path: $O(m \text{val}(f^*)) = O(mn)$.
- Capacity scaling: $O(m^2 \log C) = O(m^2)$.
- Shortest augmenting path: $O(m n^{1/2})$.

Non-bipartite matching.

- Structure of non-bipartite graphs is more complicated, but well-understood. [Tutte-Berge, Edmonds-Galai]
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m n^{1/2})$. [Micali-Vazirani 1980]

23