CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2018

Trend: I will be attending a conference next week.
Tuesday: Recorded Lecture + return midterms (hopefully)
Thursday: No class
Midterm Regrade? Must be completed within 2 weeks (syllabus). Please e-mail us before then.
Midterm Solutions: Will post on blackboard before Tuesday.

Travel:
I will be attending a conference next week.

Max Flow Recap
- Max-Flow Problem, Min-Cut Problem
  - Definition of a s-t flow f and a s-t cut (A,B)
  - Value of a flow f
  - Capacity of a s-t cut (A,B)
- Weak Duality Lemma: For any flow f and s-t cut (A,B), max(v(f)) ≤ cap(A,B) (A.u., capacity of minimum cut is upper bound on max-flow)
- Finding a max-flow:
  - Greedy algorithm fails
  - Randomized Graph
  - Ford-Fulkerson Algorithm
    - Resiliently find augmenting path in residual graph
    - Proof of Correctness
    - Max-Flow Min-Cut Equivalence

Max-Flow Recap
- Ford-Fulkerson Algorithm
  - Graph G
  - Capacity 10
  - Flow 4

Max-Flow Min-Cut Theorem
- Augmenting path theorem: Flow f is a max flow if there are no augmenting paths.
- Proof of Max-Flow Min-Cut Theorem
  - (i) ⇒ (ii) Let f be a flow with no augmenting paths.
  - (ii) ⇒ (iii) Let A be set of vertices reachable from s in residual graph.
  - (iii) ⇒ (i)

Proof of Max-Flow Min-Cut Theorem
- (i) ⇒ (ii) Let f be a flow with no augmenting paths.
  - By definition of A, s ∈ A.
  - By definition of f, t ∈ A.
  - Let f be a flow with no augmenting paths.
  - Let A be set of vertices reachable from s in residual graph.
  - By definition of A, s ∈ A.
  - By definition of f, t ∈ A.

Max-Flow Min-Cut Theorem
- Running Time
- Assumption. All capacities are integers between 1 and C.
- Invariant. Every flow value f(e) and every residual capacity c_f(e) remains an integer throughout the algorithm.
- Theorem. The algorithm terminates in at most v(f*) ≤ nC iterations.
- Proof. Each augmentation increase value by at least 1.
- Corollary. If C = 1, Ford-Fulkerson runs in O(mn) time.
- Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.
- Proof. Since algorithm terminates, theorem follows from invariant.
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 $\Delta$.
- Let $G_{\Delta}$ be the subgraph of the residual graph consisting of only arcs with capacity at least $\Delta$.

Capacity Scaling: Correctness

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

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

Correctness: If the algorithm terminates, then $f$ is a max flow. By
- By integrality invariant, when $\Delta = 1 \Rightarrow G_1 = G$
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths. *
Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.

Proof (Pf). Initially $C \leq \Delta < 2C$.

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

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

Let $f$ be the flow at the end of the previous scaling phase.

L2 $\implies v(f^*) \leq v(f) + m(2\Delta)$.

Each augmentation in a $\Delta$-phase increases $v(f)$ by at least $\Delta$.

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

Proof on next slide.
Dinic's Max Flow Min-Cut Algorithm

Remark: Number of levels increased. This is not a coincidence!

New Residual Graph \( G_f \)

Breadth First Search: Yields minimum s-t cut! We are done!
Finding a Blocking Flow in $G_{s,t}$

**Definition:** We let $G_{s,t}$ denote the capacity of an edge $e$ in $G_{s,t}$.

**Definition:** Given an augmenting flow $f'$ for $G_{s,t}$ and a $s$-$t$ path $P$ we define $R'(e) = \min_{e' \in P} R'(e')$.

**FindBlockingFlow($G_{s,t}$)**
- Initialize RemCap($e$) = $G_{s,t}(e)$
- While there exists a path $P$ with $R'(e) > 0$
  - Set $f'(e) = f'(e) + R'(e)$ for each edge $e$ in $P$
  - Set RemCap($e$) = RemCap($e$) - $R'(e)$ for each edge $e$ in $P$

**Analysis:** Each iteration of while loop "eliminates" at least one edge.

**Implication:** Terminates after at most $m$ rounds.

**Naive Running Time:** $O(mn)$

**Amortization:** Can enumerate paths in amortized time $O(n)$ per path

**Even Better:** Find blocking flow in time $O(m \log n)$ with dynamic trees.

**Dinic's Algorithm: Correctness and Running Time**

**Correctness follows directly from Augmenting Path Theorem.**

**Augmenting path theorem:** Flow $f$ is a max flow if there are no augmenting paths.

**Running Time Analysis:** Let $G_i$ denote residual graph after iteration $i$ ($G_0 = G$)

- **Definition:** depth($G_i$) = length of the shortest directed path from $s$ to $t$.

- **Key Claim:** depth($G_{i+1}$) > depth($G_i$) (depth always increases)

**Proof:** Suppose (for contradiction) that depth($G_{i+1}$) ≤ depth($G_i$).

- Then $G_{i+1}$ contains an $s$-$t$ path of length ≤ depth($G_i$).
- This path corresponds to an augmenting path for the flow $f' = f_i$.
- But since the augmenting path has length depth($G_i$) it is also an augmenting path in the level graph $G_i$.
- This contradicts the claim that $f'$ is a blocking flow in $G_{s,t}$!

**Implication:** $i$ iterations is at most $n$.

**Time to Compute Blocking Flow in Level Graph:** $O(mn)$

**Using special data structure called dynamic trees $O(m \log n)$**

**Total Time:** $O(m \log n)$ with dynamic trees or $O(m^2)$ without.

---

**7.7 Extensions to Max Flow**
Circulation with Demands

Circulation with demands:
- Directed graph $G = (V, E)$.
- Edge capacities $c(e), e \in E$.
- Node supply and demands $d(v), v \in V$.

Def. A circulation is a function that satisfies:
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \in \in(v)} f(e) - \sum_{e \in \out(v)} f(e) = d(v)$ (conservation)

Circulation problem: given $(V, E, c, d)$, does there exist a circulation?

Max flow formulation.

Integrality theorem. If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Pf. Follows from max flow formulation and integrality theorem for max flow.

Characterization. Given $(V, E, c, d)$, there does not exists a circulation iff there exists a node partition $(A, B)$ such that $\sum_d(v) > \text{cap}(A, B)$


Circulation with Demands and Lower Bounds

Feasible circulation:
- Directed graph $G = (V, E)$.
- Edge capacities $c(e)$ and lower bounds $\lambda(e), e \in E$.
- Node supply and demands $d(v), v \in V$.

Def. A circulation is a function that satisfies:
- For each $e \in E$: $\lambda(e) \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \in \in(v)} f(e) - \sum_{e \in \out(v)} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds: given $(V, E, c, d)$, does there exists a circulation?
Circulation with Demands and Lower Bounds

Idea. Model lower bounds with demands.
- Send \((e)\) units of flow along edge \(e\).
- Update demands of both endpoints.

<table>
<thead>
<tr>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d(v))</td>
<td>(d(w))</td>
<td>(d(v) + 2)</td>
</tr>
</tbody>
</table>

Theorem. There exists a circulation in \(G\) iff there exists a circulation in \(G'\). If all demands, capacities, and lower bounds in \(G\) are integers, then there is a circulation in \(G\) that is integer-valued.

Pf sketch. \(f(e)\) is a circulation in \(G\) iff \(f'(e) = f(e) - \lambda(e)\) is a circulation in \(G'\).

7.8 Survey Design

Survey design.
- Design survey asking \(n_1\) consumers about \(n_2\) products.
- Can only survey consumer \(i\) about product \(j\) if they own it.
- Ask consumer \(i\) between \(c_i\) and \(c_i'\) questions.
- Ask between \(p_j\) and \(p_j'\) consumers about product \(j\).

Goal. Design a survey that meets these specs, if possible.

Bipartite perfect matching. Special case when \(c_i = c_i' = p_j = p_j' = 1\).

Algorithm. Formulate as a circulation problem with lower bounds.
- Include an edge \((i, j)\) if consumer \(i\) owns product \(j\).
- Integer circulation \(\Rightarrow\) feasible survey design.