CS 97SI: INTRODUCTION TO PROGRAMMING CONTESTS

Jaehyun Park
Last Lecture on Graph Algorithms

- Network Flow Problems
  - Maximum Flow
  - Minimum Cut
- Ford-Fulkerson Algorithm
- Application: Bipartite Matching
- Min-cost Max-flow Algorithm
Network Flow Problems

- A type of network optimization problem
- Arise in many different contexts (CS 261):
  - Networks: routing as many packets as possible on a given network
  - Transportation: sending as many trucks as possible, where roads have limits on the number of trucks per unit time
  - Bridges: destroying (?!?) some bridges to disconnect $s$ from $t$, while minimizing the cost of destroying the bridges
Network Flow Problems

- **Settings**: Given a directed graph \( G = (V, E) \), where each edge \( e \) is associated with its capacity \( c(e) > 0 \). Two special nodes source \( s \) and sink \( t \) are given \((s \neq t)\)

- **Problem**: Maximize the total amount of flow from \( s \) to \( t \) subject to two constraints
  - Flow on edge \( e \) doesn’t exceed \( c(e) \)
  - For every node \( v \neq s, t \), incoming flow is equal to outgoing flow
Network Flow Example (from CLRS)

- Capacities

- Maximum Flow (of 23 units)
Alternate Formulation: Minimum Cut

- We want to remove some edges from the graph such that after removing the edges, there is no path from $s$ to $t$
- The cost of removing $e$ is equal to its capacity $c(e)$
- The minimum cut problem is to find a cut with minimum total cost

- Theorem: $(\text{maximum flow}) = (\text{minimum cut})$
  - Take CS 261 if you want to see the proof 😊
Minimum Cut Example

- Capacities (costs)

  - Minimum Cut (red edges are removed)
Flow Decomposition

- Any valid flow can be decomposed into flow paths and circulations

- $s \rightarrow a \rightarrow b \rightarrow t$: 11
- $s \rightarrow c \rightarrow a \rightarrow b \rightarrow t$: 1
- $s \rightarrow c \rightarrow d \rightarrow b \rightarrow t$: 7
- $s \rightarrow c \rightarrow d \rightarrow t$: 4
Ford-Fulkerson Algorithm

- A simple and practical max-flow algorithm
- Main idea: find valid flow paths until there is none left, and add them up
- How do we know if this gives a maximum flow?
  - Proof sketch: Suppose not. Take a maximum flow $f^*$ and subtract our flow $f$. It is a valid flow of positive total flow. By the flow decomposition, it can be decomposed into flow paths and circulations. These must have been found by Ford-Fulkerson. Contradiction.
Back Edges

- We don’t need to maintain the amount of flow on each edge but work with capacity values directly.
- If $f$ amount of flow goes through $u \rightarrow v$, then:
  - Decrease $c(u \rightarrow v)$ by $f$
  - Increase $c(v \rightarrow u)$ by $f$
- Why do we need to do this?
  - Sending flow to both directions is equivalent to canceling flow.
Set $f_{total} = 0$

Repeat until there is no path from $s$ to $t$:

- Run DFS from $s$ to find a flow path to $t$
- Let $f$ be the minimum capacity value on the path
- Add $f$ to $f_{total}$
- For each edge $u \rightarrow v$ on the path:
  - Decrease $c(u \rightarrow v)$ by $f$
  - Increase $c(v \rightarrow u)$ by $f$
Analysis

- Assumption: capacities are integer-valued
- Finding a flow path takes $\Theta(n + m)$ time
- We send at least 1 unit of flow through the path
- If the max-flow is $f^*$, the time complexity is $O((n + m)f^*)$
  - “Bad” in that it depends on the output of the algorithm
  - Nonetheless, easy to code and works well in practice
Computing the Min-Cut

- We know that max-flow is equal to min-cut
- And we now know how to find the max-flow

- Question: how do we find the min-cut?
- Answer: use the residual graph
Computing the Min-Cut

“Subtract” the max-flow from the original graph

Only the topology of the residual graph is shown.
Don’t forget to add the back edges!
Computing the Min-Cut

- Mark all nodes reachable from $S$
  - Call the set of reachable nodes $A$

- Now separate these nodes from the others
  - Edges go from $A$ to $V - A$ are cut
Computing the Min-Cut

- Look at the original graph and find the cut:

- Why isn’t $b \to c$ cut?
Bipartite Matching

- **Settings:**
  - $n$ students and $d$ dorms
  - Each student wants to live in one of the dorms of his choice
  - Each dorm can accommodate at most one student (?!)
    - Fine, we will fix this later…

- **Problem:** find an assignment that maximizes the number of students who get a housing
Flow Network Construction

- Add source and sink
- Make edges between students and dorms
  - All the edge weights are 1
Flow Network Construction

- Find the max-flow
- Find the optimal assignment from the chosen edges
Related Problems

- A more reasonable variant of the previous problem: dorm $j$ can accommodate $c_j$ students
  - Make an edge with capacity $c_j$ from dorm $j$ to the sink
- Decomposing a DAG into nonintersecting paths
  - Split each vertex $v$ into $v_{\text{left}}$ and $v_{\text{right}}$
  - For each edge $u \rightarrow v$ in the DAG, make an edge from $u_{\text{left}}$ to $v_{\text{right}}$
- And many others…
**Min-Cost Max-Flow**

- A variant of the max-flow problem
- Each edge $e$ has capacity $c(e)$ and cost $\text{cost}(e)$
- You have to pay $\text{cost}(e)$ amount of money per unit flow flowing through $e$
- Problem: find the maximum flow that has the minimum total cost
- A lot harder than the regular max-flow
  - But there is an easy algorithm that works for small graphs
Simple (?) Min-Cost Max-Flow

- Forget about the costs and just find a max-flow

- Repeat:
  - Take the residual graph
  - Find a negative-cost cycle using Bellman-Ford
    - If there is none, finish
  - Circulate flow through the cycle to decrease the total cost, until one of the edges is saturated
    - The total amount of flow doesn’t change!

- Time complexity: very slow
Notes on Max-Flow Problems

- Remember different formulations of the max-flow problem
  - Again, (maximum flow) = (minimum cut)!
- Often the crucial part is to construct the flow network
- We didn’t cover fast max-flow algorithms
  - Refer to the Stanford Team notebook for efficient flow algorithms