CSE 548: (*Design and*) Analysis of Algorithms

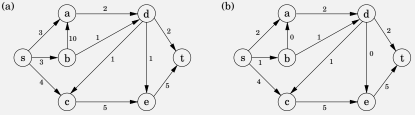
Flows in Networks

R. Sekar

Overview

- Network flows model important real-world problems
  - Oil pipelines, water and sewage networks, ...
  - Electricity grids
  - Communication networks
- In addition, several graph problems can be solved using maxflow algorithms
  - Bipartite matching, weighted bipartite matching, assignment problems,...
- Can be solved using linear programming
  - But we will study more efficient algorithms

Example 1: Maximizing Oil Flow



A pipeline network (a) and an assignment of flows (b)

- Edge capacities cannot be exceeded: $0 \leq f_e \leq c_e$
- Except for the source and sink nodes, incoming oil = outgoing oil:

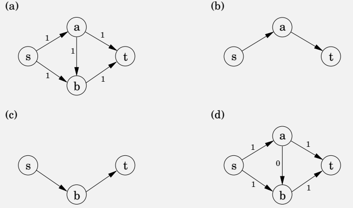
$$\sum_{(w,u)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$$

- *Maximize flow from s to t subject to these constraints.*

Solving Oil Flow

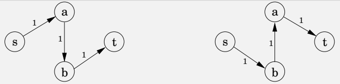
- *Can be posed as an LP problem:*
  - Objective: maximize the sum of flows on edges out of s
  - One variable per edge, with capacity constraint
  - Conservation conditions become equality constraints
- *Advantage of studying a powerful technique:*
  - Even in situations where it may not most efficient, we can use it to solve many problems
  - By studying this solution, we can gain insight that enable us to develop a direct algorithm that is more efficient.
- *So, how does Simplex solve flow problems?*
  - Start at the origin, i.e., zero flow
  - move to next corner: push max flow through one s—t path
  - repeat until no more paths can be added.

Simplex in Action



A pipeline network (a), steps taken by Simplex (b), (c), and the final assignment of flows (d)

But what happens if you pick the wrong path?



Incorrect path selected: left or right

- It seems we are stuck! What does Simplex do?
  - Simplex can increase a variable, but decrease later, so not stuck!
  - Will pick (left) and then (right), thus getting to maxflow
  - Flows in opposite directions in the middle edge cancel out
- Can we model this directly in a graph algorithm?
  - Construct a *residual graph*, with edges representing *positive or negative changes* that can be made to the current assignment.

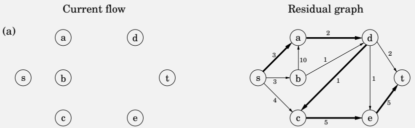
Augmented Graph G_f

Residual vertices: Same as G

Residual Edges: Edges representing left over capacities c^f

- If an edge e is not at full capacity in G, then $c_f = c_e - f_e$
- There is also an edge in opposite direction to each edge with a capacity f_e
  - Represents the fact we can cut back current flow to zero.

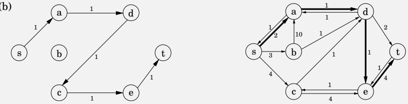
Maxflow Algorithm Illustration (I)



- Initial assignment is zero flows on all edges
- So, the residual graph G_f is exactly the same as G
- Thick edges show a possible new path P for additional flow
  - The algorithm sends a flow of $min_{e \in P}(c_e^f)$ on this path

Maxflow Algorithm Illustration (2)

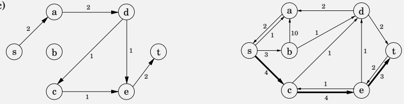
(b)



- Note addition of back edges in G_f on the right for each forward edge given a flow (see left)
- Capacity of a forward edge shrunk by amount of current flow
  - Full forward edges disappear, e.g., (d, c)
- Thick edges show the next possible path P for additional flow
  - The algorithm sends a flow of $min_{e \in P}(c_e^f)$ on this path

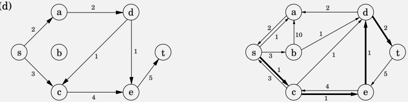
Maxflow Algorithm Illustration (3)

(c)



Maxflow Algorithm Illustration (4)

(d)

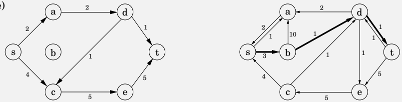


Maxflow Algorithm Illustration (5)

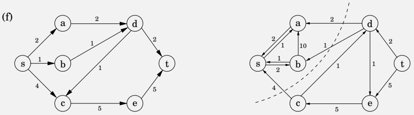
Current Flow

Residual Graph

(e)

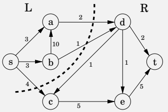


Maxflow Algorithm Illustration (6)



- No path from s to t in G_f: means we are done.
- Graph highlights a cut-set to show
  - G_f is disconnected, so no more flow can be sent
  - The very same (but inverted) edges in original graph form a *minimal cut-set* that proves we have maximized the flow

Max-flow min-cut theorem



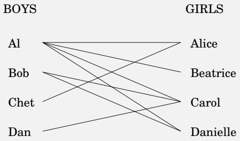
Theorem: The size of maximum flow in a network equals the capacity of the smallest $(s.t)$-cut.

- The dual of maximizing flow: finding a minimum cut-set
- A solution to dual problem is an optimality proof of primal
- Exercise: Find the cutset efficiently in the final G_f.

Runtime of Max-flow Algorithm

- Each path-finding step takes $O(E)$, say, using DFS or BFS
- G_f can be recomputed in the same amount of time
- Each iteration adds at least one unit of flow
- Total runtime: $O(C|E|)$ where C is the maximum flow computed.
  - Note that C can be large.
  - Unfortunately, this worst-case behavior can arise in some graphs if paths are chosen without care
  - If paths are chosen carefully, say, using *BFS*, number of iterations is $O(|V| \cdot |E|)$

Bipartite Matching



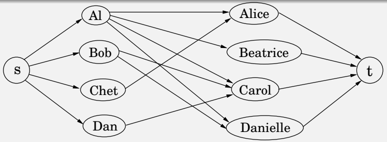
Bipartite: Two disjoint vertex sets, no edges within each set

Matching: Pair each vertex on left with one on right.

Maximal matching: Pairs as many vertices as possible

Exercise: Find an efficient algorithm for this problem

Bipartite Matching and Max-flow



Integral solutions are a must for bipartite matching, but not a real issue for max-flow in general

- As it turns out, Max-flow algorithm does guarantee to produce integral solutions when capacities are integers
- But in general integer optimization problems are much harder then non-integral versions