# Network Flows: Reductions and Applications

#### Admin

- Assignment 6 due tomorrow evening
- Help on slack or in office hours
- Today may give practice that will help with problem
  2. (It's not a network flow problem, but it is (another) reduction problem.)

#### Ford-Fulkerson Algorithm

- Start with f(e) = 0 for each edge  $e \in E$
- Find an  $s \sim t$  path P in the residual network  $G_f$
- Augment flow along path P
- Repeat until you get stuck

```
FORD-FULKERSON(G)
```

```
FOREACH edge e \in E : f(e) \leftarrow 0.
```

 $G_f \leftarrow$  residual network of *G* with respect to flow *f*.

WHILE (there exists an s $\neg$ t path *P* in *G*<sub>*f*</sub>)

```
f \leftarrow \text{AUGMENT}(f, P).
```

Update  $G_f$ .

```
RETURN f.
```

# Ford-Fulkerson Algorithm Running Time

#### Ford-Fulkerson Performance

```
FORD-FULKERSON(G)
```

```
FOREACH edge e \in E: f(e) \leftarrow 0.

G_f \leftarrow residual network of G with respect to flow f.

WHILE (there exists an s\negt path P in G_f)

f \leftarrow AUGMENT(f, P).

Update G_f.

RETURN f.
```

- Does the algorithm terminate?
- Can we bound the number of iterations it does?
- Running time?

### Ford-Fulkerson Running Time

- Recall we proved that with each call to AUGMENT, we increase value of flow by  $b = \text{bottleneck}(G_f, P)$
- Assumption. Suppose all capacities c(e) are integers.
- Integrality invariant. Throughout Ford–Fulkerson, every edge flow f(e) and corresponding residual capacity is an integer. Thus  $b \ge 1$ .
- Let  $C = \max_{u} c(s \rightarrow u)$  be the maximum capacity among edges leaving the source *s*.
- It must be that  $v(f) \le (n-1)C = O(nC)$
- Since, v(f) increases by  $b \ge 1$  in each iteration, it follows that FF algorithm terminates in at most v(f) = O(nC) iterations.

### Ford-Fulkerson Running Time

- **Claim.** Ford-Fulkerson can be implemented to run in time O(nmC), where  $m = |E| \ge n 1$  and  $C = \max_{u} c(s \rightarrow u)$ .
- **Proof**. We know algorithm terminates in at most C iterations. Each iteration takes O(m) time:
  - We need to find an augmenting path in  $G_f$
  - $G_f$  has at most 2m edges, using BFS/DFS takes O(m + n) = O(m) time
  - Augmenting flow in P takes O(n) time
  - Given new flow, we can build new residual graph in O(m) time

# [Digging Deeper] Polynomial time?

- Does the Ford-Fulkerson algorithm run in time polynomial in the input size?
- Running time is O(nmC), where  $C = \max_{u} c(s \rightarrow u)$ What is the input size?
- Let's take an example

# [Digging Deeper] Polynomial time?

- Question. Does the Ford-Fulkerson algorithm run in polynomial-time in the size of the input? <----- ~ m, n, and log C</li>
- Answer. No. if max capacity is C, the algorithm can take  $\geq C$  iterations. Consider the following example.



# [Digger Deeper] Pseudo-Polynomial

- Input graph has n nodes and  $m = O(n^2)$  edges, each with capacity  $c_e$
- $C = \max_{e \in E} c(e)$ , then c(e) takes  $O(\log C)$  bits to represent
- Input size:  $\Omega(n \log n + m \log n + m \log C)$  bits
- Running time:  $O(nmC) = O(nm2^{\log C})$ , exponential in the size of C
- Such algorithms are called **pseudo-polynomial** 
  - If the running time is polynomial in the **magnitude** but **not size** of an input parameter.
  - We saw this for knapsack as well!

#### Non-Integral Capacities?

- If the capacities are rational, can just multiply to obtain a large integer (massively increases running time)
- If capacities are irrational, Ford-Fulkerson can run infinitely!
  - Idea: amount of flow sent decreases by a constant factor each loop

Network Flow: Beyond Ford Fulkerson

### Edmond and Karp's Algorithms

- Ford and Fulkerson's algorithm does not specify which path in the residual graph to augment
- Poor worst-case behavior of the algorithm can be blamed on bad choices on augmenting path
- Better choice of augmenting paths. In 1970s, Jack Edmonds and Richard Karp published two natural rules for choosing augmenting paths
  - Fattest augmenting paths first
  - Shortest (in terms of edges) augmenting paths first (Dinitz independently discovered & analyzed this rule)
  - Can result in  $O(n^2m)$  time

#### **Progress on Network Flows**

1951	$O(m n^2 C)$	Dantzig
1955	$O(m \ n \ C)$	Ford–Fulkerson
1970	$O(m n^2)$	Edmonds-Karp, Dinitz
1974	$O(n^3)$	Karzanov
1983	$O(m \ n \log n)$	Sleator-Tarjan
1985	$O(m n \log C)$	Gabow
1988	$O(m n \log (n^2 / m))$	Goldberg–Tarjan
1998	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg-Rao
2013	O(m n)	Orlin
2014	$\tilde{O}(m n^{1/2} \log C)$	Lee–Sidford
2016	$ ilde{O}(m^{10/7} \ C^{1/7})$	Mądry
		For unit capacity networks

#### **Progress on Network Flows**

- Best known: O(nm)
- Best lower bound?
  - None known. (Needs  $\Omega(n+m)$  just to look at the network, but that's it)
- Some of these algorithms do REALLY well in "practice;" basically O(n + m)

• Well-known open problem

#### Applications of Network Flow: Solving Problems by Reduction to Network Flows

#### Max-Flow Min-Cut Applications

- Data mining
- Bipartite matching
- Network reliability
- Image segmentation
- Baseball elimination
- Network connectivity
- Markov random fields
- Distributed computing
- Network intrusion detection
- Many, many, more.

#### Anatomy of Problem Reductions

- At a high level, a problem X reduces to a problem Y if an algorithm for Y can be used to solve X
- **Reduction.** Convert an arbitrary instance *x* of *X* to a special instance *y* of *Y* such that there is a 1-1 correspondence between them



#### Anatomy of Problem Reductions

- Claim. *x* satisfies a property iff *y* satisfies a corresponding property
- Proving a reduction is correct: prove both directions
- x has a property (e.g. has matching of size k)  $\implies$  y has a corresponding property (e.g. has a flow of value k)
- *x* does not have a property (e.g. does not have matching of size *k*) ⇒ *y* does not have a corresponding property (e.g. does not have a flow of value *k*)
- Or equivalently (and this is often easier to prove):
  - y has a property (e.g. has flow of value k)  $\implies$  x has a corresponding property (e.g. has a matching of value k)

#### Plan for Today

- I'll show you one (classic) network flow reduction
- Then you'll attempt one; we'll go over the answer together

# Max-Cardinality Bipartite Matching

#### Review: Matching in Graphs

Definition. Given an undirected graph
 G = (V, E), a matching M ⊆ E of G is a subset
 of edges such that no two edges in M are
 incident on the same vertex.



#### Review: Matching in Graphs

- Definition. Given an undirected graph
   G = (V, E), a matching M ⊆ E of G is a subset
   of edges such that no two edges in M are
   incident on the same vertex.
- **Max matching problem.** Find a matching of maximum cardinality for a given graph, that is, a matching with maximum number of edges

#### **Review: Bipartite Graphs**

- A graph is **bipartite** if its vertices can be partitioned into two subsets X, Y such that every edge e = (u, v)connects  $u \in X$  and  $v \in Y$
- **Bipartite matching problem.** Given a bipartite graph  $G = (X \cup Y, E)$  find a maximum matching.



#### **Bipartite Matching Example**

- Suppose A is a set of students, B as a set of dorms
- Each student lists a set of dorms they'd like to live in, each dorm lists students it is willing to accommodate
- **Goal.** Find the largest matching (student, dorm) pairs that satisfies their requirements
- **Bipartite matching instance.** V = (A, B) and  $e \in E$  if student and dorm are mutually acceptable, goal is to find maximum matching
- Note. This is a different problem than the one we studied for Gale-Shapely matching!

#### Reduction to Max Flow

- Given arbitrary instance x of bipartite matching problem
   (X): A, B and edges E between A and B
- **Goal.** Create a special instance y of a max-flow problem (Y): flow network: G(V, E, c), source s, sink  $t \in V$  s.t.
- **1-1 correspondence.** There exists a matching of size k iff there is a flow of value k



#### Reduction to Max Flow

- Create a new directed graph  $G' = (A \cup B \cup \{s, t\}, E', c)$
- Add edge  $s \to a$  to E' for all nodes  $a \in A$
- Add edge  $b \to t$  to E' for all nodes  $b \in B$
- Direct edge  $a \rightarrow b$  in E' if  $(a, b) \in E$
- Set capacity of all edges in E' to 1



• Claim  $(\Rightarrow)$ .

If the bipartite graph (A, B, E) has matching M of size k then flow-network G' has an integral flow of value k.



#### • Claim $(\Rightarrow)$ .

If the bipartite graph (A, B, E) has matching M of size k then flow-network G' has an integral flow of value k.

#### • Proof.

- For every edge  $e = (a, b) \in M$ , let f be the flow resulting from sending 1 unit of flow along the path  $s \to a \to b \to t$
- *f* is a feasible flow (satisfies capacity and conservation) and integral

• 
$$v(f) = k$$

• Claim  $(\Rightarrow)$ .

If the bipartite graph (A, B, E) has matching M of size k then flow-network G' has an integral flow of value k.



• Claim (  $\Leftarrow$  ).

If flow-network G' has an integral flow of value k, then the bipartite graph (A, B, E) has matching M of size k.



• Claim (  $\Leftarrow$  ).

If flow-network G' has an integral flow of value k, then the bipartite graph (A, B, E) has matching M of size k.

- Proof.
  - Let M = set of edges from A to B with f(e) = 1.
  - No two edges in *M* share a vertex, why?
  - $\bullet \quad |M| = k$ 
    - $v(f) = f_{out}(S) f_{in}(S)$  for any (S, V S) cut
    - Let  $S = A \cup \{s\}$

• Claim (  $\Leftarrow$  ).

If flow-network G' has an integral flow of value k, then the bipartite graph (A, B, E) has matching M of size k.



# Summary & Running Time

- Proved matching of size k iff flow of value k
- Thus, max-flow iff max matching
- Running time of algorithm overall:
  - Running time of reduction + running time of solving the flow problem (dominates)
- What is running time of Ford–Fulkerson algorithm for a flow network with all unit capacities?
  - *O*(*nm*)
- Overall running time of finding max-cardinality bipartite matching: O(nm)

## **Disjoint Paths Problem**

### **Disjoint Paths Problem**

- **Definition.** Two paths are edge-disjoint if they do not have an edge in common.
- Edge-disjoint paths problem.

Given a directed graph with two nodes *s* and *t*, find the max number of edge-disjoint  $s \sim t$  paths.



#### **Towards Reduction**

- Given: arbitrary instance x of disjoint paths problem (X): directed graph G, with source s and sink t
- **Goal**. create a special instance y of a max-flow problem (Y): flow network G'(V', E', c) with s', t' s.t.
- **1-1 correspondence.** Input graph has *k* edge-disjoint paths iff flow network has a flow of value *k*



#### Reduction to Max Flow

- Reduction. G': same as G with unit capacity assigned to every edge
- Claim [Correctness of reduction]. G has k edge disjoint
   s → t paths iff G' has an integral flow of value k.
- Proof. (  $\Rightarrow$  )
- Set f(e) = 1 if e in some disjoint  $s \sim t, f(e) = 0$  otherwise.
- We have v(f) = k since paths are edge disjoint.
- (  $\Leftarrow$  ) Need to show: If G' has a flow of value k then there are k edge-disjoint  $s \thicksim t$  paths in G

#### **Correction of Reduction**

- Claim. ( ⇐ ) If f is a 0-1 flow of value k in G', then the set of edges where f(e) = 1 contains a set of k edge-disjoint s ~ t paths in G.
- **Proof** [By induction on the # of edges k' with f(e) = 1]
- If k' = 0, no edges carry flow, nothing to prove
- IH: Assume claim holds for all flows that use < k' edges
- Consider an edge  $s \to u$  with  $f(s \to u) = 1$
- By flow conservation, there exists an edge  $u \rightarrow v$  with  $f(u \rightarrow v) = 1$ , continue "tracing out the path" until
- Case (a) reach *t*, Case (b) visit a vertex *v* for a 2nd time

#### **Correction of Reduction**

- **Case (a)** We reach *t*, then we found a  $s \sim t$  path *P* 
  - f': Decrease the flow on edges of P by 1

• 
$$v(f') = v(f) - 1 = k - 1$$

- Number of edges that carry flow now < k': can apply IH and find k 1 other  $s \sim t$  disjoint paths
- **Case (b)** visit a vertex *v* for a 2nd time: consider cycle *C* of edges visited btw 1st and 2nd visit to *v* 
  - f': decrease flow values on edges in C to zero
  - v(f') = v(f) but # of edges in f' that carry flow < k', can now apply IH to get k edge disjoint paths

# Summary & Running Time

- Proved k edge-disjoint paths iff flow of value k
- Thus, max-flow iff max # of edge-disjoint  $s \sim t$  paths
- Running time of algorithm overall:
  - Running time of reduction + running time of solving the max-flow problem (dominates)
- What is running time of Ford–Fulkerson algorithm for a flow network with all unit capacities?
  - *O*(*nm*)
- Overall running time of finding max # of edge-disjoint  $s \sim t$  paths: O(nm)

#### [Take-home Exercise] Reduction to Think About

### **Room Scheduling**

- Williams College is holding a big gala and has hired you to write an algorithm to schedule rooms for all the different parties happening as part of it.
- There are n parties and the ith party has  $p_i$  invitees.
- There are r different rooms and the jth room can fit  $r_j$  people in it.
- Thus, party *i* can be held in room *j* iff  $p_i \leq r_j$ .
- Describe and analyze an efficient algorithm to assign a room to each party (or report correctly that no such assignment is possible).

#### Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (<u>https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsl.pdf</u>)
  - Jeff Erickson's Algorithms Book (<u>http://jeffe.cs.illinois.edu/</u> <u>teaching/algorithms/book/Algorithms-JeffE.pdf</u>)