Chapter 13

Network Flow III – Applications

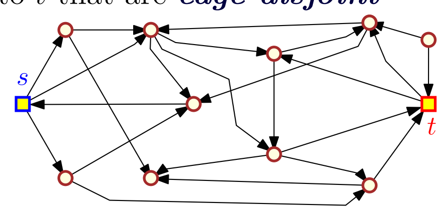
CS 573: Algorithms, Fall 2014
October 9, 2014

13.1 Edge disjoint paths

13.1.1 Edge-disjoint paths in a directed graphs

13.1.1.1 Edge disjoint paths

questionG: graph (dir/undir). s, t: vertices. k: parameter.
Task: Compute k paths from s to t that are ***edge disjoint***



(A) Convert G into network flow H.
(B) Capacities 1.
(C) Compute max flow in H.
(D) Value of flow = # of edge disjoint paths.

13.1.1.2 Edge Disjoint paths lemma

Lemma 13.1.1. *$\exists k$ edge disjoint s-t paths in* G
$\implies$ *max flow value H is at least k.*

Proof: Given k such edge disjoint paths, push one unit of flow along each such path. The resulting flow is legal in h and it has value k. ■

Definition 13.1.2 (0/1-flow). A flow f is a 0/1-***flow*** if every edge has either no flow on it, or one unit of flow.

13.1.1.3 0/1 flow

Lemma 13.1.3. *f: 0/1 flow in H with flow value μ. Then there are μ edge disjoint paths between s and t in H.*

proof

(A) Induction on # edges $\in H$ with 1 unit of flow on them. If $\mu = 0$...
(B) Otherwise... Travel from s on edges with flow 1. Extract path. Repeat.
(C) If reached t. Take path π. Reduce flow along π.
H'/f': new network/flow
$|f'| = \mu - 1$, H' has less edges,
(D) By induction: has $\mu - 1$ edge disjoint paths in H' between s and t. With π this forms μ such paths.

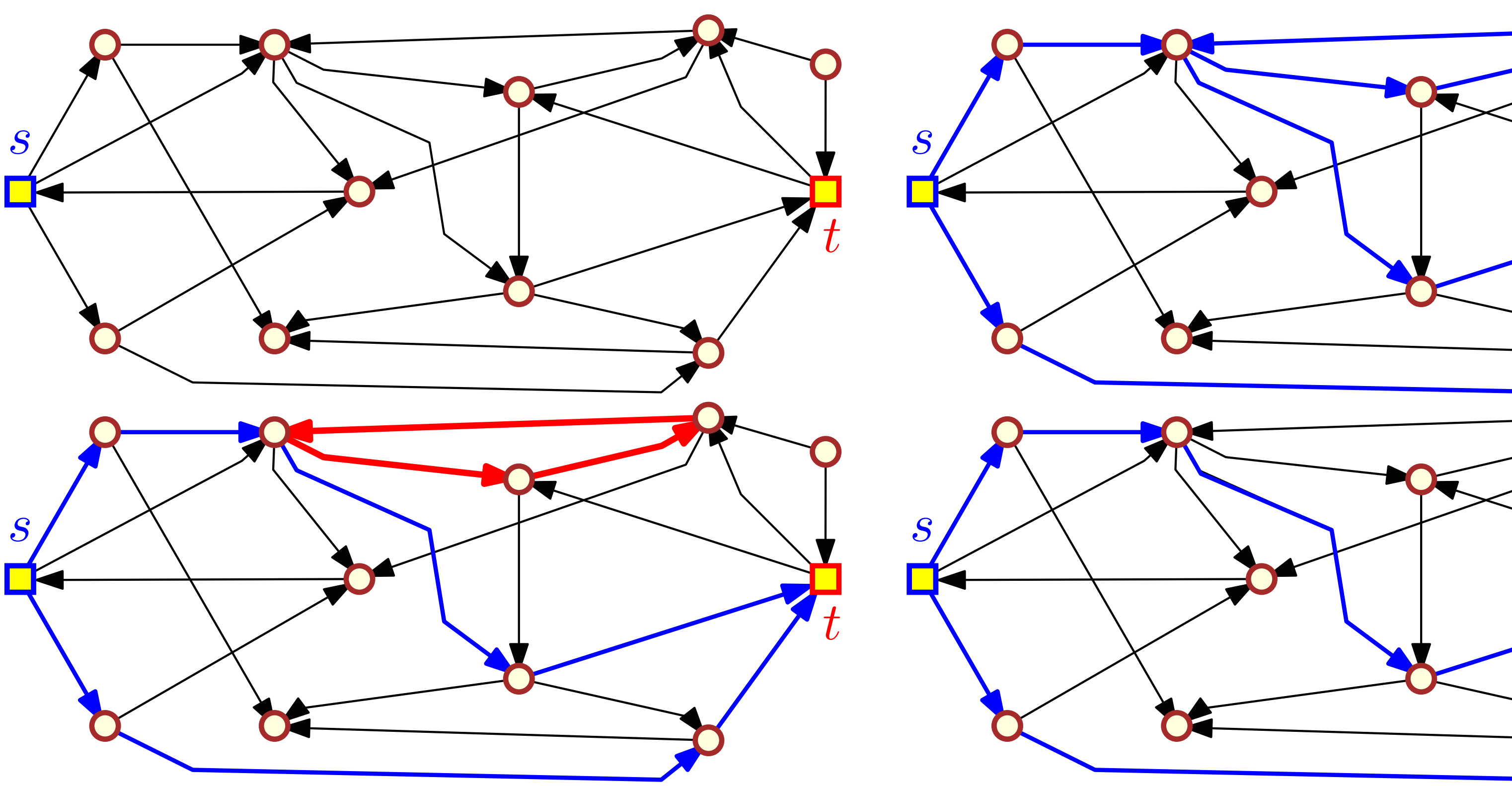
13.1.1.4 0/1 flow proof continued

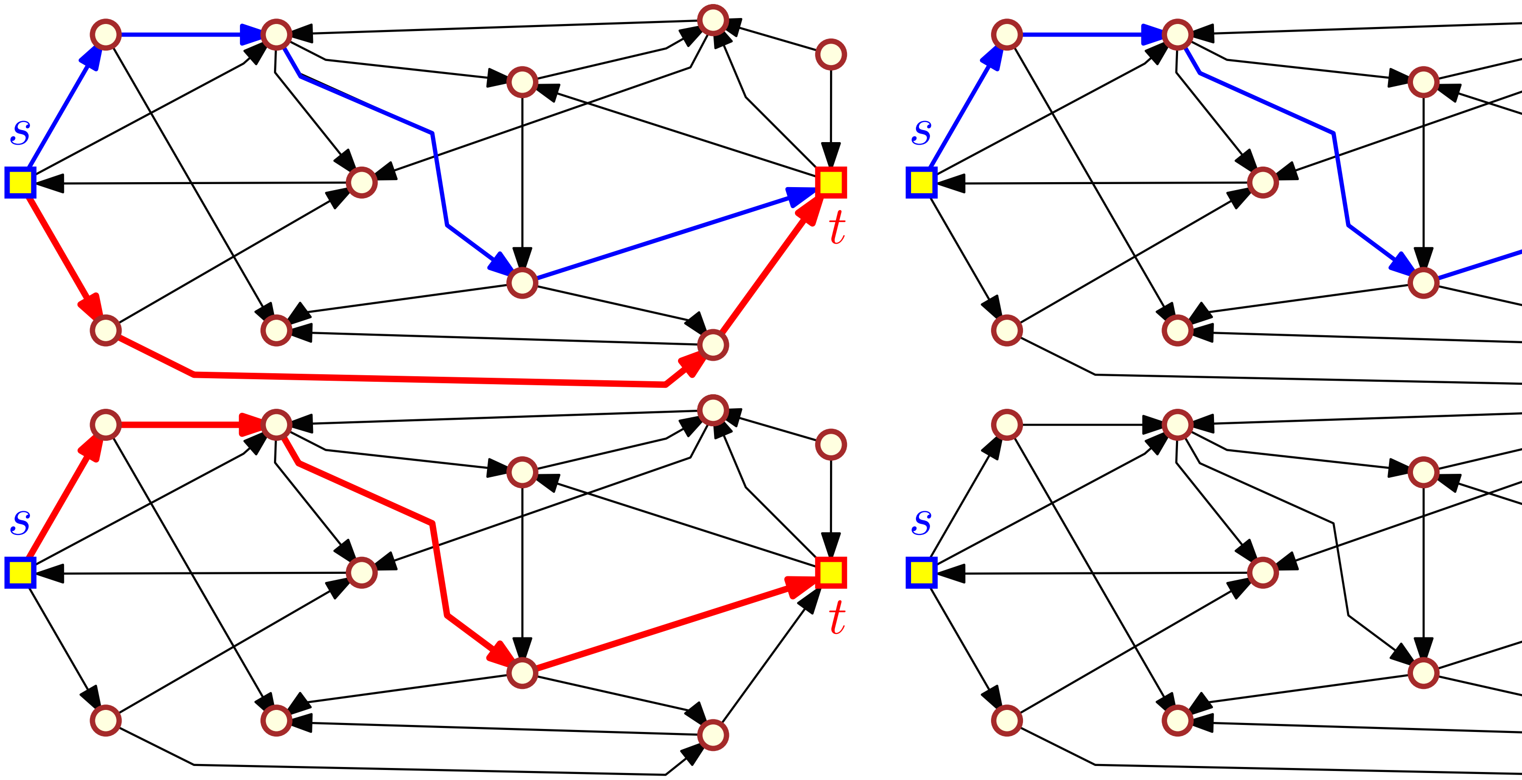
Lemma 13.1.4. *f: 0/1 flow in H with flow value μ. Then there are μ edge disjoint paths between s and t in H.*

Proof continued

(A) If visit a vertex v for the second time (while extracting π).
(B) Traversal contains a cycle C.
(C) C edges in H have flow 1 on them.
(D) Set flow along edges of C to 0.
(E) Induction on the remaining graph.
(F) Value of f did not change by removing C.
(G) By induction $\exists \mu$ edge disjoint paths $s \rightsquigarrow t$ in H. ■

13.1.1.5 Example: Extracting paths





13.1.1.6 Extracting paths

(A) G is simple
(B) $\implies \leq n = |\mathsf{V}(\mathsf{H})|$ edges leaving s.
(C) max flow in H is $\leq n$.
(D) Ford-Fulkerson takes $O(mn)$ time.
(E) Extraction of paths takes linear time (by proof).

Theorem 13.1.5. G*: directed graph,* n *vertices,* m *edges,* s, t *vertices.*
Compute max # edge disjoint paths from s *to* t *in* $O(mn)$ *time.*

13.1.2 # edge disjoint paths

13.1.2.1 Max-flow min-cut theorem strikes again!

Lemma 13.1.6. G, s, t *as above.*

$$\boxed{\text{Max \# edge disjoint } s - t \text{ paths}} = \boxed{\text{min \# edges whose removal separates } s \text{ from } t.}$$

Proof: (A) U: set of edge-disjoint paths from s to t.
(B) F: set of edges removing them separates s from t
(C) every path in U contains edge of F. $\implies |U| \leq |F|$.
(D) F: form a cut in G between s and t.
(E) F minimal $= s - t$ min cut.
(F) max-flow mincut theorem $|F| =$ max flow in G
(G) $\implies \exists \quad |F|$ disjoint paths in G. $\implies |F| \leq |U|$.

13.1.3 Edge-disjoint paths in undirected graphs

13.1.3.1 Edge-disjoint paths in undirected graphs

Problem 13.1.7. G: undirected graph G, s and t, find max # edge-disjoint paths between s and t.

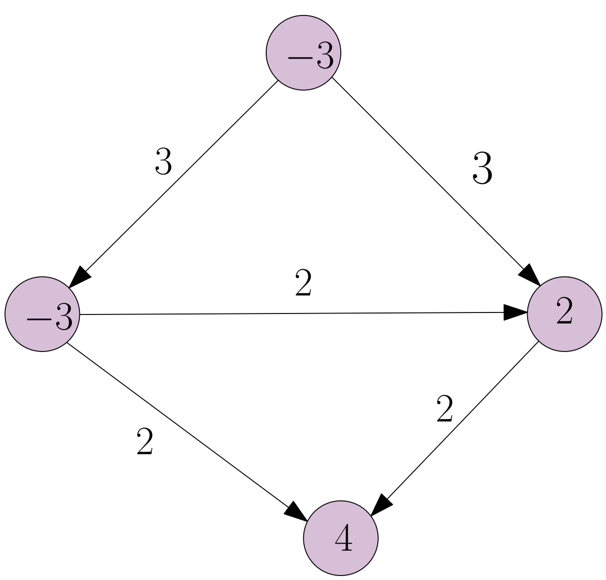
(A) Duplicate every edge in G $\implies$ directed graph H, apply algorithm for directed case.
(B) Problem: flow f might use simultaneously edge in both directions: $(u \to v)$ and $(v \to u)$.
(C) Solution: Remove 2-cycle! Repeat.
(D) Then use algorithm for directed case...

Lemma 13.1.8. *$\exists$ k edge-disjoint $s - t$ paths in undirected G $\iff$ max flow in (directed) graph is at least k.*

Paths in G computed in $O(mn)$ time (Ford-Fulkerson).

13.2 Circulations with demands

13.2.0.2 Circulations with demands



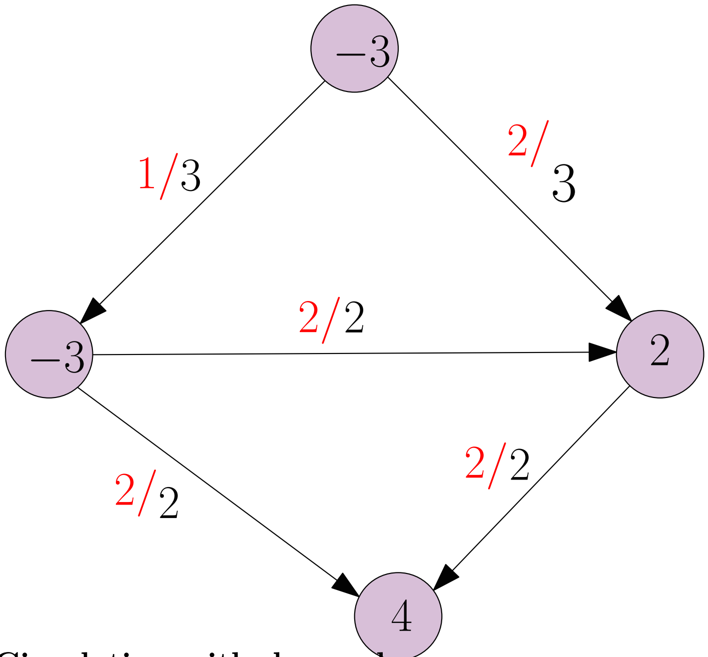
$\mathsf{G} = (\mathsf{V}, \mathsf{E})$.
$\forall v \in \mathsf{V}$ there is a ***demand*** d_v:
(A) $d_v > 0$: sink requiring d_v flow into this node.
(B) $d_v < 0$: source with $-d_v$ units of flow leaving it.
(C) $d_v = 0$: regular node.

S set of source vertices

T: set of sink vertices.

13.2.1 A circulation with demands: example

13.2.1.1 A valid circulation for the given instance



13.2.1.2 Definition: Circulation with demands

Definition 13.2.1. ***circulation*** with demands $\{d_v\}$ is a function $f : \mathsf{E}(\mathsf{G}) \rightarrow \mathbb{R}^+$:

- Capacity condition: $\forall e \in E$: $f(e) \leq c(e)$.
- Conservation condition: $\forall v \in V$: $f^{in}(v) - f^{out}(v) = d_v$.

Where:
(A) $f^{in}(v)$ flow into v.
(B) $f^{out}(v)$: flow out of v.

Problem 13.2.2. Is there a circulation that comply with the demand requirements?

13.2.1.3 Feasible circulation lemma

Lemma 13.2.3. *If there is a feasible circulation with demands* $\{d_v\}$*, then* $\sum_v d_v = 0$.

Proof: (A) Circulation $\implies \forall v \quad d_v = f^{in}(v) - f^{out}(v)$.
(B) $\sum_{v \in \mathsf{V}} d_v = \sum_v f^{in}(v) - \sum_v f^{out}(v)$
(C) Flow on every edge is summed twice, one with positive sign, one with negative sign.
(D) $\implies \displaystyle\sum_v d_v = \sum_v f^{in}(v) - \sum_v f^{out}(v) = 0,$

13.2.1.4 Computing circulations

$\exists$ feasible circulation only if

$$D = \sum_{v, d_v > 0} d_v = \sum_{v, d_v < 0} -d_v.$$

13.2.2 The algorithm for computing a circulation

Algorithm for computing circulation

(A) $\mathsf{G} = (\mathsf{V}, \mathsf{E})$: input network with demands on vertices.
(B) Check $D = \sum_{v, d_v > 0} d_v = \sum_{v, d_v < 0} -d_v$.
(C) Create super source s. Connect to all v with $d_v < 0$. Set capacity $(s \to v)$ to $-d_v$.
(D) Create super sink t. Connect to all vertices u with $d_u > 0$. Set capacity $(u \to t)$ to d_u.
(E) H: new network flow. Compute max-flow f in H from s to t.
(F) If $|f| = D \implies \exists$ valid circulation. Easy to recover.

13.2.2.1 Result: Circulations with demands

Theorem 13.2.4. *$\exists$ feasible circulation with demands $\{d_v\}$ in G $\iff$ max-flow in H has value D.*

Integrality*: If all capacities and demands in G are integers, and there is a feasible circulation, then there is a feasible circulation that is integer valued.*

13.3 Circulations with demands and lower bounds

13.3.0.2 Circulations with demands and lower bounds

(A) circulation and demands + for each edge a lower bound on flow.
(B) $\forall e \in E(\mathsf{G})$: $\ell(e) \leq c(e)$.
(C) Compute f such that $\forall e \quad \ell(e) \leq f(e) \leq c(e)$.
(D) Be stupid! Consider flow: $\forall e \quad f_0(e) = \ell(e)$.
(E) f_0 violates conservation of flow!

$$L_v = f_0^{in}(v) - f_0^{out}(v) = \sum_{e \text{ } into \text{ } v} \ell(e) - \sum_{e \text{ } out \text{ } of \text{ } v} \ell(e).$$

(F) If $L_v = d_v$, then no problem.
(G) Fix-up demand: $\forall v \quad d'_v = d_v - L_v$.
Fix-up capacity: $c'(e) = c(e) - \ell(e)$.
(H) G': new network w. new demands/capacities (no lower bounds!)
(I) Compute circulation f' on G'.
$\implies$ The flow $f = f_0 + f'$, is a legal circulation,

13.3.0.3 Circulations with demands and lower bounds

Lemma 13.3.1. *$\exists$ feasible circulation in G $\iff$ $\exists$ feasible circulation in G'.*
Integrality*: If all numbers are integers $\implies$ $\exists$ integral feasible circulation.*

Proof: Let f' be a circulation in G'. Let $f(e) = f_0(e) + f'(e)$. Clearly, f satisfies the capacity condition in G, and the lower bounds.
$f^{in}(v) - f^{out}(v) = \sum_{e \text{ } into \text{ } v} (\ell(e) + f'(e)) - \sum_{e \text{ } out \text{ } of \text{ } v} (\ell(e) + f'(e)) = L_v + (d_v - L_v) = d_v$.
f: valid circulation in G. Then $f'(e) = f(e) - \ell(e)$ is a valid circulation for G'. ■

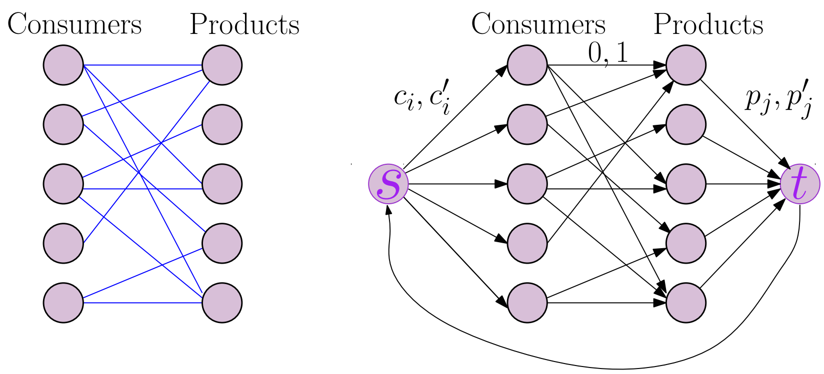
13.4 Applications

13.4.1 Survey design

13.4.1.1 Survey design

(A) Ask "Consumer i: what did you think of product j?"
(B) ith consumer willing to answer between c_i to c'_i questions.
(C) For each product j: at least p_j opinions, no more than p'_j opinions.
(D) Full knowledge which consumers can be asked on which products.
(E) Problem: How to assign questions to consumers?

13.4.1.2 Survey design...



13.4.1.3 Result...

Lemma 13.4.1. *Given n consumers and u products with their constraints $c_1, c'_1, c_2, c'_2, \ldots, c_n, c'_n$, $p_1, p'_1, \ldots, p_u, p'_u$ and a list of length m of which products where used by which consumers. An algorithm can compute a valid survey under these constraints, if such a survey exists, in time $O((n + u)m^2)$.*