CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2019

Max Flow Recap

Max-Flow Problem, Min Cut Problem
- Definition of a s-t flow f(e) 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 we have $v(f) \le cap(A,B)$ (i.e., capacity of minimum cut is upper bound on max-flow)

Finding a Max-Flow:
- Greedy algorithm fails!
- Residual Graph
- Ford-Fulkerson Algorithm
  - Repeatedly find augmenting path in residual graph
  - Proof of Correctness
  - Max-Flow Min-Cut Equivalence

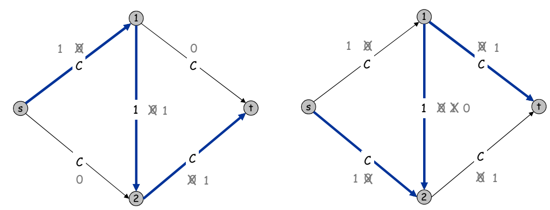
7.3 Choosing Good Augmenting Paths

Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

m, n, and log C

A. No. If max capacity is C, then algorithm can take C iterations.



Choosing Good Augmenting Paths

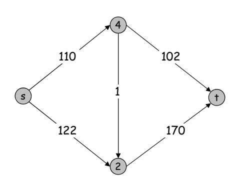
Use care when selecting augmenting paths.
- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:
- Can find augmenting paths efficiently.
- Few iterations.

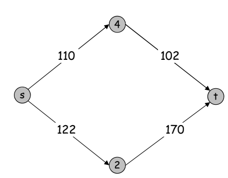
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]
- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

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 Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.



G_f



$G_f(100)$

Copyright 2000, Kevin Wayne

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

7

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.
Pf.
- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

8

Capacity Scaling: Running Time

Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.
Pf. Initially $C \le \Delta < 2C$. Δ decreases by a factor of 2 each iteration. ▪

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

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 $\Rightarrow$ $v(f^*) \le v(f) + m\,(2\Delta)$.
- Each augmentation in a Δ-phase increases v(f) by at least Δ. ▪

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. ▪

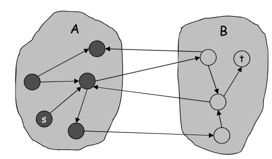
9

Capacity Scaling: Running Time

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.
Pf. (almost identical to proof of max-flow min-cut theorem)
- We show that at the end of a Δ-phase, there exists a cut (A, B) such that $cap(A, B) \le v(f) + m\,\Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) \\
&\ge \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\
&= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
&\ge cap(A,B) - m\Delta
\end{aligned}
$$

▪

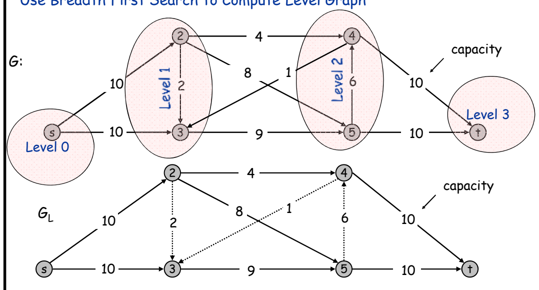


original network

10

Dinic's Max Flow Min-Cut Algorithm

Use Breadth First Search to Compute Level Graph

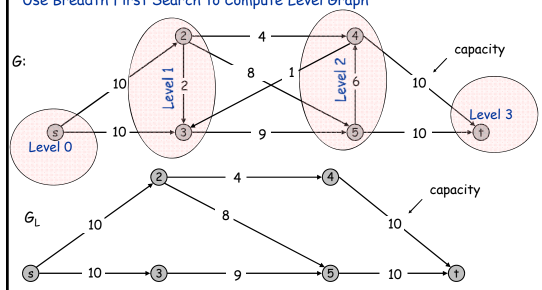


Discard cross-layer edges

11

Dinic's Max Flow Min-Cut Algorithm

Use Breadth First Search to Compute Level Graph

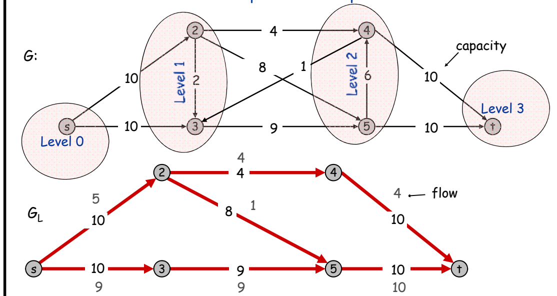


Discard cross-layer edges

12

Dinic's Max Flow Min-Cut Algorithm

Use Breadth First Search to Compute Level Graph

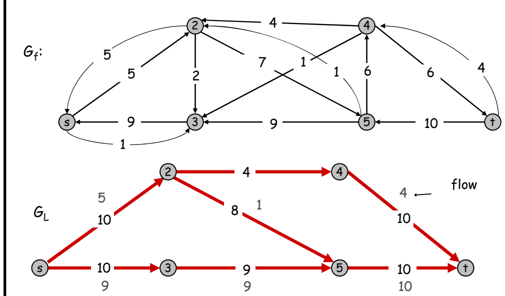


Discard cross-layer edges
Find Blocking Flow

13

Dinic's Max Flow Min-Cut Algorithm

Create Residual Graph G_f

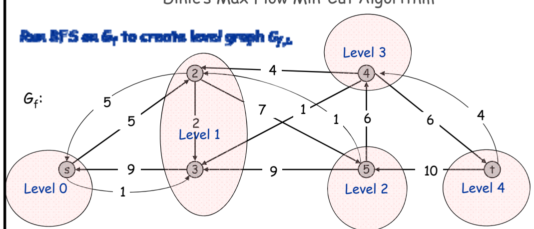


Total Flow: 14

14

Dinic's Max Flow Min-Cut Algorithm

Run BFS on G_f to create level graph $G_{f,L}$

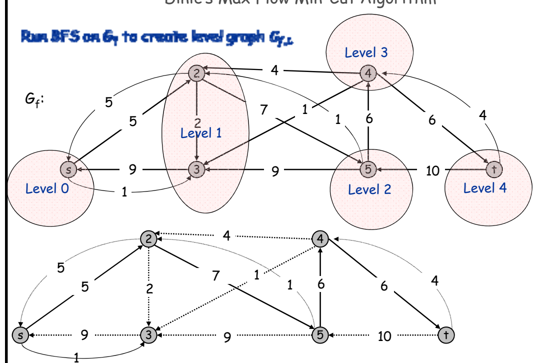


Remark: Number of levels increased. This is not a coincidence!

15

Dinic's Max Flow Min-Cut Algorithm

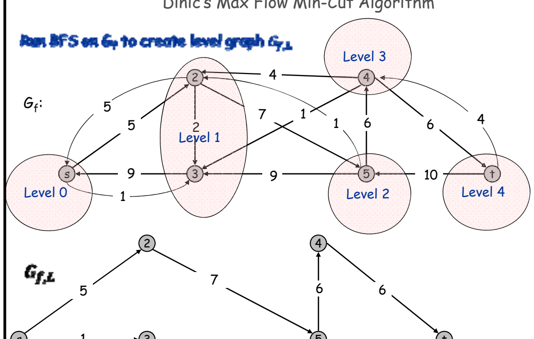
Run BFS on G_f to create level graph $G_{f,L}$



16

Dinic's Max Flow Min-Cut Algorithm

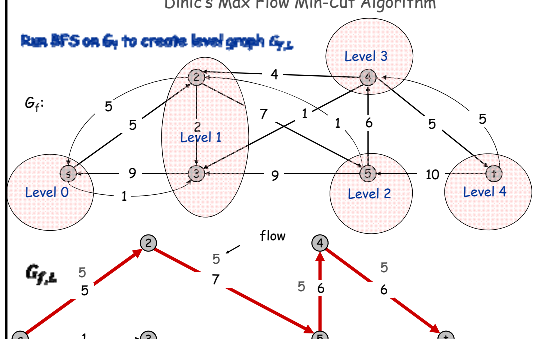
Run BFS on G_f to create level graph $G_{f,L}$



17

Dinic's Max Flow Min-Cut Algorithm

Run BFS on G_f to create level graph $G_{f,L}$



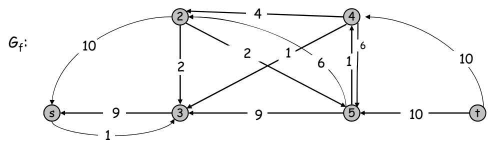
Total Extra Flow: 5

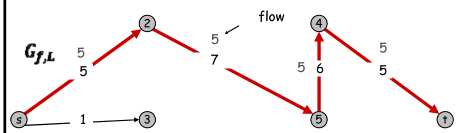
Blocking Flow for level graph $G_{f,L}$

18

Dinic's Max Flow Min-Cut Algorithm

New Residual Graph G_f





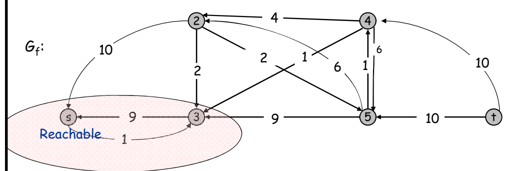
Total Extra Flow: 5

Blocking Flow for level graph $G_{f,L}$

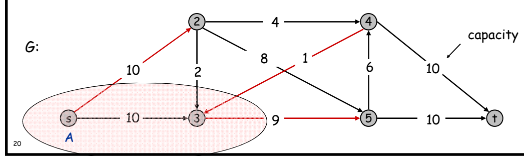
19

Dinic's Max Flow Min-Cut Algorithm

New Residual Graph G_f



Breadth First Search: Yields minimum s-t cut!→ We are done!



20

Finding a Blocking Flow in $G_{f,L}$

Definition: $C_{f,L}(e)$ denotes the capacity of the edge e in $G_{f,L}$

Definition: *Given an augmenting flow f' for the level graph $G_{f,L}$ and a path P in $G_{f,L}$ we define $B(P,f') = \min_{e\in P}\{C_{f,L}(e) - f'(e)\}$*

FindBlockingFlow($G_{f,L}$)

- **Initialize:**
  - RemCap(e) $= C_{f,L}(e)$ and $f'(e) = 0$ for each edge e in $G_{f,L}$
- **While there is a path P with $B(P,f') > 0$**
  - Update $f'(e) = f'(e) + B(P,f')$ for each edge $e \in P$
  - Update RemCap(e) = RemCap(e) $- B(P,f')$ for each edge $e \in P$

Analysis: Each iteration of the "while" loop eliminates an edge

Implication: Terminates after O(m) iterations of while loop.

21

Finding a Blocking Flow in $G_{f,L}$

Definition: $C_{f,L}(e)$ denotes the capacity of the edge e in $G_{f,L}$

Definition: *Given an augmenting flow f' for the level graph $G_{f,L}$ and a path P in $G_{f,L}$ we define $B(P,f') = \min_{e\in P}\{C_{f,L}(e) - f'(e)\}$*

FindBlockingFlow($G_{f,L}$)

- **Initialize:**
  - RemCap(e) $= C_{f,L}(e)$ and $f'(e) = 0$ for each edge e in $G_{f,L}$
- **While there is a path P with $B(P,f') > 0$**
  - Update $f'(e) = f'(e) + B(P,f')$ for each edge $e \in P$
  - Update RemCap(e) = RemCap(e) $- B(P,f')$ for each edge $e \in P$

Analysis: Each iteration of the "while" loop eliminates an edge

Implication: Terminates after O(m) iterations of while loop.
Naïve Running Time Analysis: O(m(m+n))

22

Finding a Blocking Flow in $G_{f,L}$

Definition: We let $C_{f,L}(e)$ denote the capacity of an edge e in $G_{f,L}$
Definition: *Given an augmenting flow f' for $G_{f,L}$ and a s-t path P we define $B(P) = \min_{e\in P} C_{f,L}(e)$*

FindBlockingFlow($G_{f,L}$)

- Initialize RemCap(e) = $C_{f,L}(e)$
- While there exists a path P with $B(P) > 0$
  - Set $f'(e) = f'(e) + B(P)$ for each edge $e \in P$
  - Set RemCap(e) = RemCap(e) $- B(P)$ for each edge $e \in P$

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

Implication: Terminates after at most m rounds.

Naïve Running Time: O((m+n)m)
Amortization: *Can enumerate paths in amortized time O(n) per path*

23

Dinic's Algorithm

1. Start with empty flow f
2. Construct G_f
3. Repeat until s and t are disconnected (no augmenting path)
   1. (Level Graph) Run BFS on G_f to build $G_{f,L}$
   2. (Blocking Flow) Find blocking flow f' in $G_{f,L}$
   3. (Augment) Let f=f+f' and Construct G_f
4. Output f

Analysis:
Claim: Each time we iterate the loop we increase the depth of G_f

Implication: Must terminate in at most n iterations!

Time Per Iteration: O(mn) to find blocking flow f'

Total Time: O(n²m)

24

Dinic's Algorithm: Correctness and Running Time

Correctness follows directly from Augmenting Path Theorem.

Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

Running Time Analysis: Let f_i denote residual graph after iteration i ($G_{f_0} = G$)

Definition: $\text{depth}(G_{f_i})$ = length of the shortest directed path from s to t).

Key Claim: $\text{depth}(G_{f_{i+1}}) > \text{depth}(G_{f_i})$ (depth always increases)

25

Dinic's Algorithm: Correctness and Running Time

Running Time Analysis: Let f_i denote residual graph after iteration i ($G_{f_0} = G$)

Definition: $\text{depth}(G_{f_i})$ = length of the shortest directed path from s to t).

Key Claim: $\text{depth}(G_{f_{i+1}}) > \text{depth}(G_{f_i})$ (depth always increases)
Proof: Suppose (for contradiction) that $\text{depth}(G_{f_{i+1}}) \le \text{depth}(G_{f_i})$.
- Then $G_{f_{i+1}}$ contains an s-t path of length $\le \text{depth}(G_{f_i})$.
- This path corresponds to an augmenting path for the flow $f' = f_{i+1} - f_i$ in G_{f_i}.
- But since the augmenting path has length $\text{depth}(G_{f_i})$ it is also an augmenting path in the level graph $G_{f_i,L}$.
- This contradicts the claim that f' is a blocking flow in $G_{f_i,L}$!

26

Dinic's Algorithm: Correctness and Running Time

Running Time Analysis: Let f_i denote residual graph after iteration i ($G_{f_0} = G$)

Definition: $\text{depth}(G_{f_i})$ = length of the shortest directed path from s to t).

Key Claim: $\text{depth}(G_{f_{i+1}}) > \text{depth}(G_{f_i})$ (depth always increases)

Implication: #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: $O(mn \log n)$ with dynamic trees or $O(mn^2)$ without.

27

7.7 Extensions to Max Flow

Circulation with Demands

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

demand if d(v) > 0; supply if d(v) < 0; transshipment if d(v) = 0

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

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

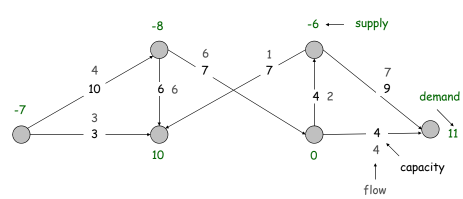
29

Circulation with Demands

Necessary condition: sum of supplies = sum of demands.

$$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$

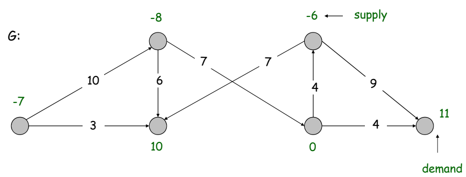
Pf. Sum conservation constraints for every demand node v.



30

Circulation with Demands

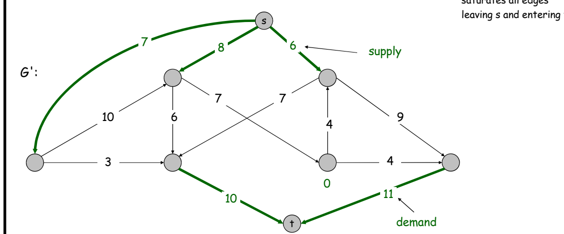
Max flow formulation.



Circulation with Demands

Max flow formulation.
- Add new source s and sink t.
- For each v with d(v) < 0, add edge (s, v) with capacity -d(v).
- For each v with d(v) > 0, add edge (v, t) with capacity d(v).
- Claim: G has circulation iff G' has max flow of value D.

saturates all edges leaving s and entering t



Circulation with Demands

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 $\Sigma_{v \in B} d_v > cap(A, B)$

demand by nodes in B exceeds supply of nodes in B plus max capacity of edges going from A to B

Pf idea. Look at min cut in G'.

Circulation with Demands and Lower Bounds

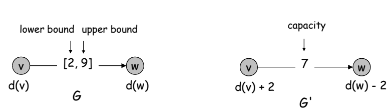
Feasible circulation.
- Directed graph G = (V, E).
- Edge capacities c(e) and lower bounds ℓ (e), e $\in$ E.
- Node supply and demands d(v), v $\in$ V.

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

Circulation problem with lower bounds. Given (V, E, ℓ, c, d), does there exists a a circulation?

Circulation with Demands and Lower Bounds

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



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) - ℓ(e) is a circulation in G'.

7.8 Survey Design

Copyright 2000, Kevin Wayne

Survey Design

one survey question per product

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.

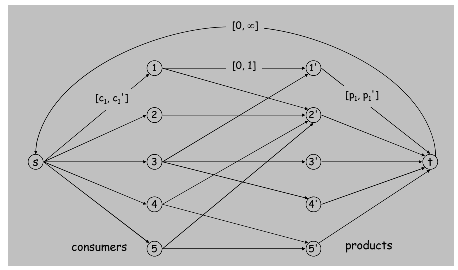
Bipartite perfect matching. Special case when $c_i = c_i' = p_i = p_i' = 1$.

37

Survey Design

Algorithm. Formulate as a circulation problem with lower bounds.

- Include an edge (i, j) if consumer j owns product i.
- Integer circulation $\Leftrightarrow$ feasible survey design.



38