Maximum Flow Min Cut

CO 351: Network Flow Theory

David Duan, 2019 Fall

Contents

1 Maximum Flow Problem

1.1 Maximum Flow Problem

Def. 1.1.1 Given a digraph $D = (N, A)$, constraints $c \in \mathbb{R}^A$, nodes $s, t \in N$ as source and sink, we wish to maximize the total flow from s to t.

$$\begin{array}{lll} \max & x(\delta(s)) - x(\delta(\bar{s})) & \\ s.t. & x(\delta(\bar{v})) - x(\delta(v)) = 0 & \forall v \in N \setminus \{s, t\} \\ & 0 \leq x_e \leq c_e & \forall e \in A \end{array}$$

Remark. We may think of the maximum flow problem as the problem of sending liquid across a system of pipes. The flow on arc uv is the amount of liquid flowing through pipe uv during one unit of time. The capacity of the corresponding pipe is c_{uv}. The flow conservation constraints indicate there is no loss of liquid, i.e., what comes in comes out (except for the source and the sink).

Remark. Note that we can turn this into an MCFP by adding an arc from sink t to source s, which turns the digraph into a closed network.

1.2 Residual Digraph

Def. 1.2.1 Let P be an s,t-path of D.

- An arc $uv \in P$ is a *forward arc* if uv is directed from s to t; otherwise, it is a *backward arc*.
- The *residual capacity* $\gamma(P)$ of the s,t-dipath P is

$$\min(\{c_{uv} - x_{uv} : uv \text{ is a forward arc of } P\} \cup \{x_{uv} : uv \text{ is a backward of } P\}.$$

- A path P is an *augmenting path* of P is an s,t-dipath and $\gamma(P) > 0$.

Def. 1.2.2 Let x be an s,t-flow and P be an augmenting path. We say that $x' \in \mathbb{R}^A$ is obtained from x by *pushing flow along* P if x' is defined as follows, for all $uv \in A$,

$$x'_{uv} = \begin{cases} x_{uv} + \gamma(P) & \text{if } uv \text{ is a forward arc of } P \\ x_{uv} - \gamma(P) & \text{if } uv \text{ is a backward arc of } P \\ x_{uv} & \text{otherwise} \end{cases}$$

Lemma. 1.2.3 Let x be an s,t-flow and let x' be obtained by pushing flow along an augmenting path P. Then x' is an s,t-flow and $x'(t) = x(t) + \gamma(P)$. In particular, x is not maximum.

Proof. Trivial. □

Def. 1.2.4 Given D, capacities c and flow x, its corresponding *residual digraph* D' is define as

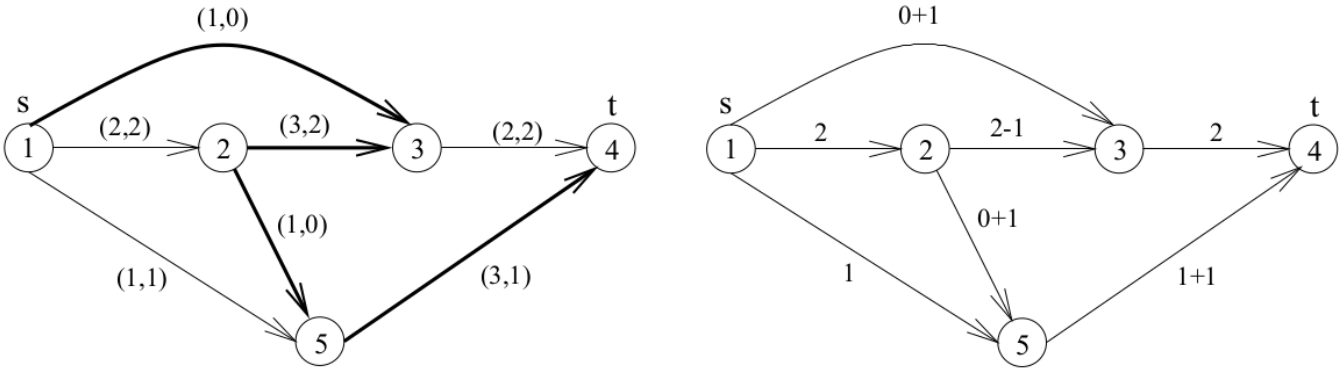
- $N(D') = N(D)$.
- For each arc $uv \in A$,

- If $c_{uv} > x_{uv}$, then add uv with residual $c_{uv} - x_{uv}$ (forward arcs).
- If $x_{uv} > 0$, then add vu with residual x_{uv} (backward arc).

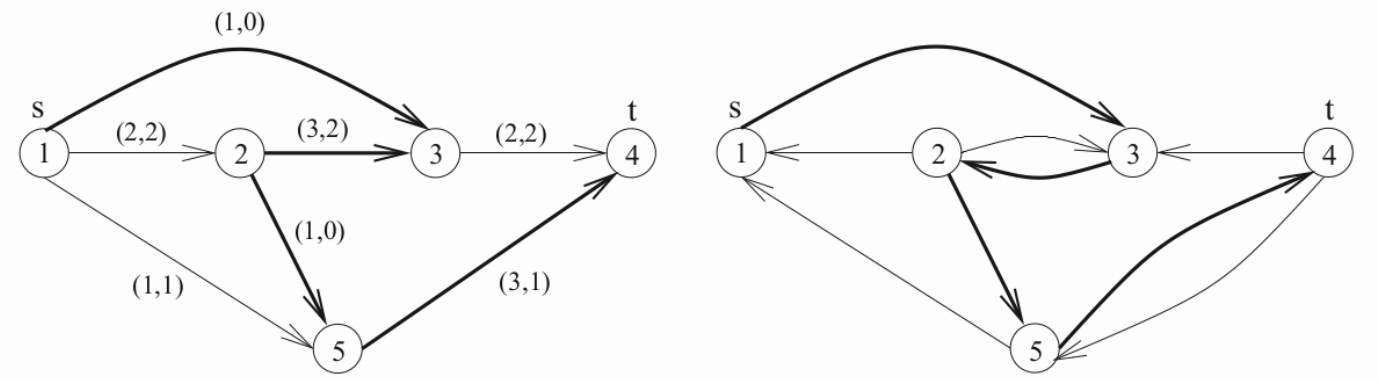
Remark. Some intuition on arcs in the residual digraph:

- $c_{uv} > x_{uv} \implies uv \in D' \wedge r_{uv} = c_{uv} - x_{uv}$: we can push $c_{uv} - x_{uv}$ units on this arc.
- $x_{uv} > 0 \implies vu \in D' \wedge r_{uv} = x_{uv}$: we can reduce x_{uv} units on this arc.

Ex. 1.2.5 The left figure represents a flow for a digraph D with arc labels (c_{uv}, x_{uv}). Consider the path $P = 13, 23, 25, 54$. Since $\gamma(P) = 1$, path P is an augmenting path. We can push one unit of flow to obtain the x' represented on the right figure.



The right figure is the residual digraph D' for D on the left. Observe D' has an s,t-dipath $P' = 13, 32, 25, 54$ and D has a corresponding augmenting path $P = 13, 23, 25, 54$.



Note there is a one-to-one correspondence between incrementing paths P of D and s,t-dipaths P' of the residual digraph D'. In particular, D has an augmenting path iff D' has an s,t-dipath.

1.3 Ford-Fulkerson

Algorithm. 1.3.1 (Ford-Fulkerson)

0. *Initialization.* Let $x_e = 0$ for all $e \in A$.
1. *Residual Graph.* Construct the residual graph D' for the current flow.
2. *Push.* If D' contains an s,t-dipath P, push additional flow $\gamma(P)$ along P. Go back to 1.

3. *Termination.* Let S be the set of nodes reachable from s in D'. STOP. x is the maximum flow and $\delta(S)$ is the minimum cut.

Remark. Will FF terminate?

- If c is integral, then at each step we increase flow by at least 1. By induction, it terminates.
- If c is rational, we can multiply c by the GCD of all denominators and make it integral.
- If c is irrational, then it is possible that it does not terminate.

Remark. The runtime for FF is *pseudo-polynomial*, i.e., it is a polynomial in the numeric values of your input.

1.4 Edmonds-Karp

Algorithm. 1.4.1 (Edmonds-Karp)

0. *Initialization.* Let $x_e = 0$ for all $e \in A$.
1. *Residual Graph.* Construct the residual graph D' for the current flow.
2. *Push.* Pick an augmenting path P in D' with the fewest number of arcs, push additional flow $\gamma(P)$ along P. Go back to 1.
3. *Termination.* Let S be the set of nodes reachable from s in D'. STOP. x is the maximum flow and $\delta(S)$ is the minimum cut.

Remark. Since we pick an augmenting path in D' with the fewest number of arcs, the algorithm is guaranteed to terminate with at most $|N||A|$ iterations.

Remark. The algorithm terminates when there is no augmenting path in D', i.e., there exists an s,t-cut $\delta_{D'}(S)$ that is empty in D'. For arcs out of $\delta_D(S)$, there are no forward version in D', so the flow is at capacity; for arcs into $\delta_D(S)$, there are no backward version in D', so the flow is 0. We cannot do any better than this because an s,t-cut limits the amount of flows by the sum of the capacities of the leaving arcs. Thus, FF/EK terminates with a maximum flow. We will provide a formal proof in the next section.

1.5 Max-Flow Min-Cut Theorem

Prop. 1.5.1 Given $D = (N, A)$, nodes s, t, capacities c, the value of any s,t-flow is at most the capacity of any s,t-cut.

Proof. For any s,t-flow x and any s,t-cut $\delta(S)$, the net flow of S is equal to the net flow of s (as the net flows on other nodes are 0 by the problem setting). The value of flow x is

$$x(\delta(s)) - x(\delta(\bar{s})) = x(\delta(S)) - x(\delta(\bar{S})) \leq x(\delta(S)) \leq c(\delta(S)).$$

Thus the value of any s,t-flow is at most the capacity of $\delta(S)$. □

Thm. 1.5.2 (Max-Flow Min-Cut) Given $D = (N, A)$, nodes s, t, capacities c, the max value of an s, t-flow is equal to the minimum capacity of an s, t-cut.

Proof. Let x be a maximum s, t-flow. Then there is no s, t-dipath in the residual digraph D' (by termination of FF). This implies there is an empty s, t-cut $\delta_{D'}(S)$ in D'.

- For $uv \in \delta_D(S)$, uv is not an arc in D', so $x_{uv} = c_{uv}$.
- For $uv \in \delta_D(\bar{S})$, vu is not an arc in D', so $x_{uv} = 0$.

Therefore, the value of s, t-flow x is $x(\delta(S)) - x(\delta(\bar{S})) = c(\delta(S)) - 0 = c(\delta(S))$. By the previous proposition, x is a max s, t-flow, and $\delta(S)$ is a minimum capacity s, t-cut. □

Cor. 1.5.3 FF does give a max s, t-flow given it terminates.

Proof. Since FF terminated, there is no s, t-dipath in the residual digraph D', so our flow have used up the capacity of some cut $\delta(S)$ in D. By Max-Flow Min-Cut, the capacity of the min cut equals the max flow, so the resulting flow is indeed optimal. □

1.6 Flows with Lower Bounds

We now introduce non-negative lower bounds $\ell_e \in \mathbb{R}^A$ for the arcs. That is, a flow must satisfy $\ell_e \leq x_e \leq c_e$ for all $e \in A$.

Algorithm. 1.6.1 (Modified FF) Given a feasible flow, we form the residual digraph D', where for the backward arcs, we put residual $x_e - \ell_e$ instead of x_e because we want to limit how much flow we could reduce. The rest of the algorithm is exactly the same.

Prop. 1.6.2 For any s, t-flow x and any s, t-cut $\delta(S)$, the value of x is at most $c(\delta(S)) - \ell(\delta(\bar{S}))$.

Proof. For any s, t-flow x and any s, t-cut $\delta(S)$, we have $x(\delta(\bar{S})) \geq \ell(\delta(\bar{S}))$, so

$$x(\delta(s)) - x(\delta(\bar{s})) = x(\delta(S)) - x(\delta(\bar{S})) \leq x(\delta(S)) - \ell(\delta(\bar{S})) \leq c(\delta(S)) - \ell(\delta(\bar{S})).$$

Thus the value of any s, t-flow is at most $c(\delta(S)) - \ell(\delta(\bar{S}))$. □

Cor. 1.6.3 **Algorithm 1.6.1** finds a maximum s, t-flow for the maximum flow problem with lower bounds.

Proof. At the end of FF, there is no s, t-dipath in D', so there is an empty s, t-cut $\delta_{D'}(S)$. The net outflow of x at termination is $c(\delta(S)) - \ell(\delta(\bar{S}))$. By **Prop. 1.6.2**, FF finds a maximum s, t-flow. □

Thm. 1.6.3 (Generalized Max-Flow Min-Cut) Given $D = (N, A)$, c, ℓ, and nodes s, t, there is a maximum s, t-flow x whose value is the same as the minimum value of $c(\delta(S)) - \ell(\delta(\bar{S}))$ provided there is a feasible solution.

Prop. 1.6.4 (Feasibility Characterization) The network is infeasible if and only if there is an s,t-cut $\delta(S)$ such that $c(\delta(S)) - \ell(\delta(\bar{S})) < 0$.

Proof. If $c(\delta(S)) - \ell(\delta(\bar{S})) < 0$, by **Prop. 1.6.2**, the maximum flow x is negative, a contradiction. Now suppose the network is feasible. Then we cannot have s,t-cut $\delta(S)$ with $c(\delta(S)) - \ell(\delta(\bar{S})) < 0$ because that sets an upper bound for the flow, again a contradiction. □

2 Applications: Max-Flow Min-Cut

2.1 Flow Decomposition

Prop. 1.6.1 (Flow Decomposition) Given $D = (N, A)$, s, t, and integer capacities, if x is an s, t-flow of value k and x is integral, then x is the sum of characteristic vectors of k s, t-dipaths and any number of dicycles.

Proof. When $k = 0$ (i.e., x is a *circulation*), then x is the sum of characteristic vectors of dicycles. If $k > 0$, then there is an s, t-dipath P. (Proof: see shortest dipath notes.) $x - x^P$ is an s, t-flow of value $k - 1$. We are done by induction on k. □

2.2 Menger's Theorem

Def. 2.2.1 $A' \subseteq A$ *disconnects* s from t is there is no s, t-dipath in $D' = (N, A - A')$.

Given a digraph $D = (N, A)$, s, t, how many arcs do we need to remove to disconnect s from t?

Thm. 2.2.2 (Menger's Theorem, Arc-Disjoint Version) Given $D = (N, A)$, nodes s, t, the maximum number of arc-disjoint s, t-dipaths is equal to the minimum number of arcs that disconnects s from t.

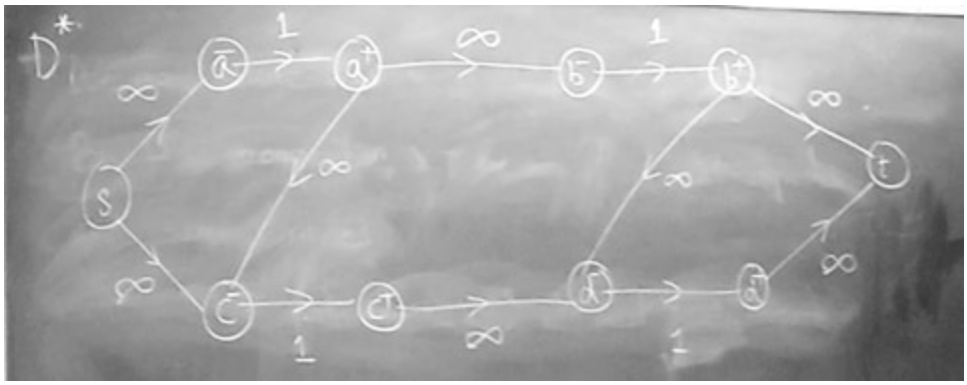
Proof. If there are k arc-disjoint s, t-dipaths, then we must remove at least one arc from each dipath. The inequality $\max \leq \min$ is thus trivial. We want to prove equality.

We apply max-flow min-cut with capacity $c = \mathbf{1}$. By max-flow min-cut, there exists an s, t-flow with the same value as the capacity of an s, t-cut $\delta(S)$, say k. We may assume that x is integral as c is integral. By the flow decomposition, x is the sum of k s, t-dipaths and some dicycles. Since $c = 1$, each arc is used on at most 1 s, t-dipath. Hence, these s, t-dipaths are all arc disjoint. If we remove all k arcs in $\delta(S)$, then S is disconnected from t. □

Assume $st \notin A(G)$, how many nodes do we have to remove to disconnect s from t?

Thm. 2.2.3 (Menger's Theorem, Node-Disjoint Version) If st is not an arc, then the maximum numbers of node-disjoint s, t-dipaths is equal to the minimum number of node whose removal disconnect s from t. We call this an s, t-*separating set* of nodes.

Proof. We transform the node-disjoint case into arc-disjoint case. For every node v, generate two nodes v^+ and v^- with a single edge between them. Arcs of the form uv maps to u^+v^- with $c_{u^+v^-} = 1$ and all other arcs have capacity ∞.



Then we can simply apply Menger's Theorem for arc-disjoint paths and conclude the proof: min-cut cannot use arcs with ∞ and thus only newly created edges, which correspond to an s,t-separator. $\square$

2.3 Konig's Theorem

Thm. 2.3.1 (Konig) In a bipartite graph, $\nu(G) = \tau(G)$.

Proof. Observe that $\nu(G) \leq \tau(G)$ holds trivially since we can take one vertex per edge in the matching to obtain a lower bound. Now, let A, B be a bipartition of V and create a, b with edges aA, Bb. Each newly created edge has capacity 1. We then orient the edges from a to b; original edges have capacity ∞.

Apply Menger's Theorem, the minimum $\delta(S)$ cannot use ∞ arcs, so $(A \setminus S) \cup (B \cap S)$. $\square$

3 Application: Maximum Flow

3.1 Matrix Rounding

Suppose we want to round the matrix $M \in \mathbb{R}^{m,n}$ elements either by taking the ceiling/floor so that the sum of the columns α_i, $1 \leq i \leq m$, and sum of rows β_j, $1 \leq j \leq n$, are floored/ceilinged.

We can describe this as an instance of an s,t-flow problem.

Algorithm 3.1.1 To transform this into a maximum flow problem:

1. Let s,t be dummy nodes and create nodes r_i, c_j for $1 \leq i \leq m, 1 \leq j \leq n$.
2. Add an arc from r_i to c_j with $\ell_{r_i c_j} = \lfloor a_{ij} \rfloor$ and $c_{r_i c_j} = \lceil a_{ij} \rceil$.
3. All arcs sr_i have lower bound 0 and capacity floor/ceiling α_i.
4. All arcs $c_j t$ have lower bound 0 and capacity floor/ceiling β_j.

We can solve this problem using maximum flow algorithm.

3.2 Maximum Closure

Def. 3.2.1 A *closure* is a set of nodes S such that $\delta(S) = \varnothing$.

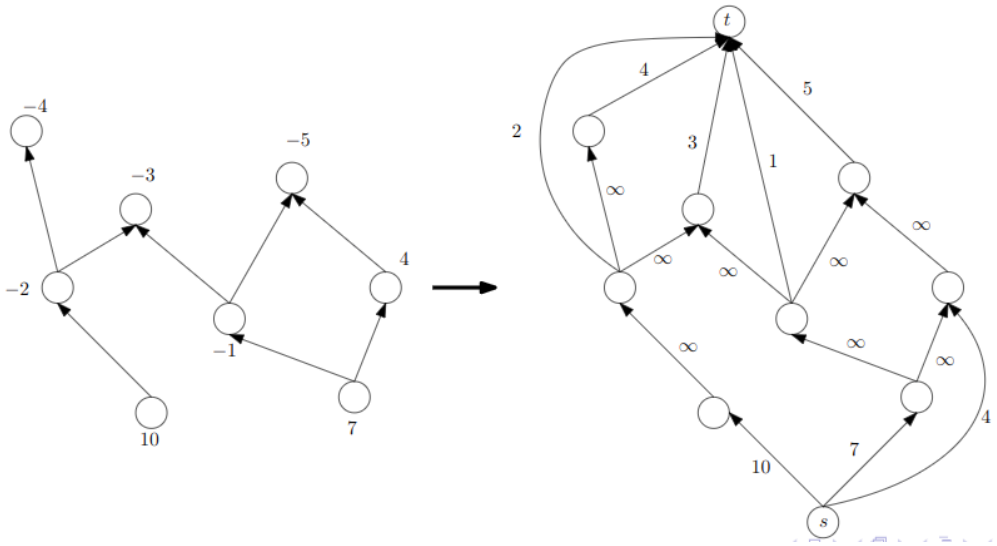
Suppose we are given a set of tasks, each with some benefit (positive or negative). However, to perform some task, you need to also perform other tasks dependent on it.

Form a digraph $D = (N, A)$ with node weights $w \in \mathbb{R}^N$; an arc uv represents "if we take u we must also take v". Our goal is to find a closure with maximum total weight.

Algorithm 3.2.2 To transform this into a maximum flow problem:

1. Add nodes s and t.
2. Add arcs su with capacity $w(u)$ if $w(u) > 0$.
3. Add arcs vt with capacity $-w(v)$ if $w(v) < 0$.
4. Original arcs have capacity ∞.

Ex. 3.2.3 Applying **Algorithm 3.2.2.** to the left digraph:

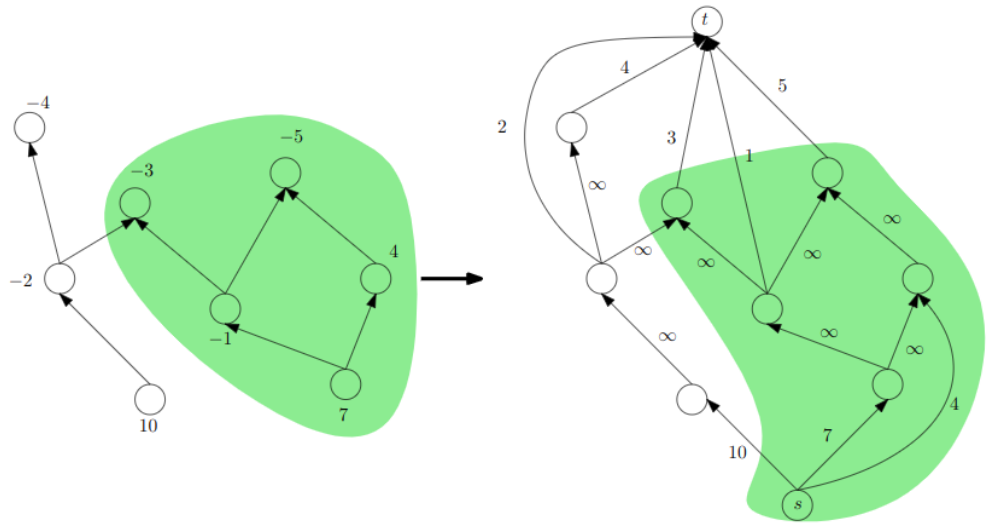


We have the following observations.

Lemma. 3.2.4 An s,t-cut $\delta(S)$ in the new graph has finite capacity if and only if $S \setminus \{s\}$ is a closure in the original.

Proof. There exists an arc in $\delta(S)$ with infinite capacity if and only if such an arc existed in the original graph. □

Lemma. 3.2.5 The weight of the closure is $c(\delta(s)) - c(\delta(S))$.



Proof. There is an arc from s to every u with $b_u > 0$ and an arc from every v with $b_v < 0$ to t. Note that nodes not in the closure (e.g., 10) are included in both and thus gets cancelled. □

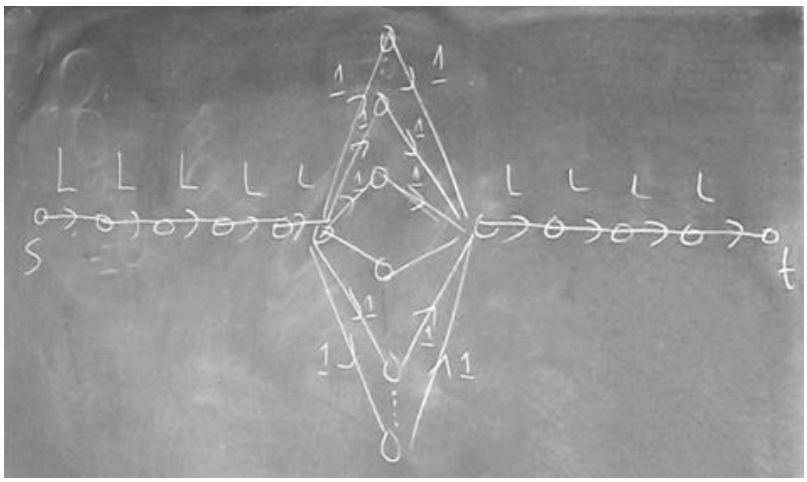
Lemma 3.2.6 $c(\delta(s))$ is a constant, and $c(\delta(S))$ is the capacity of an s,t-cut.

Proof. $c(\delta(s))$ is the sum of positive weights; $c(\delta(S))$ is a cut by inspection . □

Thus, to maximize the weight of closure $c(\delta(s)) - c(\delta(S))$, we minimize $c(\delta(S))$, i.e., find the minimum cut. We can solve this problem using maximum flow.

4 Preflow-Push Algorithm

Recall FF/EK does "global adjustments", i.e., pushes $\gamma(P)$ (the minimum available flow) along an augmenting path, which is sometimes inefficient. For example, in the following case, we can only push 1 unit of flow at a time using FF/EK:



The preflow-push algorithm, on the other hand, makes "local adjustments"; we introduce a different parameter called *height* for each node, and keep the height so that we always push flow downwards but not too steeply.

4.1 Preflow and Residual Graph

An s,t-*preflow* is a flow that (1) *satisfies capacity constraints* and (2) *in-flow is greater than or equal to the out-flow for each intermediate node*. If in-flow is greater than out-flow, we call the extra amount *excess*.

Def. 4.1.1 We call x an s,t-*preflow* if it satisfies $0 \le x \le c$ and $x(\delta(\bar{v})) \ge x(\delta(v))$ for all $v \in N \setminus \{s,t\}$. The *excess* at v is $e_x(v) = x(\delta(\bar{v})) - x(\delta(v))$.

We define the residual graph same as before.

(Def. 1.2.4) Given $D = (N, A)$, c, x, its corresponding *residual digraph* D' is define as

- $N(D') = N(D)$.
- For each arc $uv \in A$,
  - If $c_{uv} > x_{uv}$, then add uv with residual $r_{uv} := c_{uv} - x_{uv}$.
  - If $x_{uv} > 0$, then add vu with residual $r_{uv} := x_{uv}$ (backward arc).

We define a height function $h(v)$ for each node $v \in N$. We say a height is *compatible* when no arc in D' is pointing down too steeply.

Def. 4.1.2 A set of height h is *compatible* with a preflow x if

1. $h(s) = |N|$ - high node; source.

2. $h(t) = 0$ - low node; sink.
3. $h(v) \geq h(u) - 1$ for all $uv \in A(D')$ - for each arc in the residual graph, it head is at most 1 lower than its tail.

You can think of embedding the digraph in $\mathbb{R}^3$ and only allowing changes with "gentle" slopes.

4.2 Preflow Push Algorithm

Algorithm. 4.2.1 (Preflow-Push)

0. *Initialization.*
   a. Set $h(s) = |N|$ and $h(v) = 0$ for every $v \in N \setminus \{s\}$.
   b. Set preflow x with $x_e = c_e$ for every $e \in \delta_D(s)$ and $x_e = 0$ otherwise.
1. While there exists $u \in N \setminus \{s, t\}$ with excess, i.e., $e_u > 0$:
   a. If there exists $uv \in A(D')$ where $h(v) = h(u) - 1$, push $\min\{r_{uv}, e_u\}$ on uv.
   b. Otherwise, increment $h(u)$ by 1. (This is sometimes called a *relabel operation.*)

Remark. Observe that FF/EK always maintains feasibility and works towards optimality; PP starts from an infeasible flow and works towards feasibility while maintaining some optimality conditions.

4.3 Correctness of Preflow Push

Lemma. 4.3.1 If preflow x and height h are compatible, then D' has no s,t-dipath.

Proof. Suppose not, so an s,t-dipath exists. Let $s = v_0, v_1, \ldots, v_k = t$ in D' and $h(v_{i+1}) \geq h(v_i) - 1$. Adding all these inequalities gives $h(v_k) \geq h(v_0) - k$, so $0 \geq |N| - k$ and thus $|N| \leq k$, a contradiction as there are $k + 1$ nodes in the path but only $|N|$ nodes in the digraph. □

Cor. 4.3.2 If x is a feasible flow with compatible heights h, then x is a maximum flow.

Proof. Since x is compatible with h, by **Lemma. 4.3.1**, D' has no s,t-dipath. Since there is no augmenting path in D (no extra flow is available) and x is feasible, it follows that x is an optimal solution. □

Thm. 4.3.3 The algorithm maintains a preflow and a height function that are compatible with each other.

Proof. At initialization, we have a preflow x where $x_e = c_e$ for every $e \in \delta(s)$ and 0 otherwise.

- By construction, $h(s) = |N|$ and $h(t) = 0$.
- Also, $x_{sv} = c_{sv}$ for arcs in D, so only vs is in D'. Observe $h(s) = |N| \geq -1 = h(v) - 1$.

- For arcs x_{uv} where $u \neq s$, $x_{uv} = 0$, so $uv \in D'$. Now $h(v) = 0 \geq -1 = h(u) - 1$.Thus, the preflow and height are compatible at initialization. Now suppose there exists $u \in N \setminus \{s, t\}$ with excess, i.e., $e_u > 0$.

Consider a push operation, i.e., when there exists $uv \in A(D')$ satisfying $h(v) = h(u) - 1$, we push flow $\min\{e_u, r_{uv}\}$ along uv. Since we are pushing the minimum between excess and residual, it is always true that $x(\delta(\bar{u})) \geq x(\delta(u))$ and $x_{uv} \leq c_{uv}$. (Exactly one of these two becomes equality, depending on whether we push e_u or r_{uv}.)

- If we push residual, then $x_{uv} = c_{uv}$. The only possible new arc in D' is vu but $h(u) > h(v)$ so we are fine.
- If we push excess, then $e_u = 0$. Then $x_{uv} > 0$, so the only possible new arc is also vu and as above we are fine

Thus, the preflow and height are compatible after a push operation.

Consider a relabel operation, i.e., when every $uv \in A(D')$ satisfies $h(v) > h(u) - 1$, we increment $h(u)$ by one. Then before relabeling, it must be that every $uv \in A(D')$ satisfies $h(v) \geq h(u)$. When we add 1 to $h(u)$, we have $h(v) \geq h(u) - 1$ for all arcs $uv \in A(D')$. Thus, the preflow and height are compatible after a relabel operation. □

Note that if the algorithm terminates, **Thm. 4.3.3** guarantees the correctness of the algorithm.

4.4 Termination

We now try to bound the number of push and relabel operations, thus showing that the algorithm terminates after a maximum number of iterations.

To bound the number of relabel operations, we bound the maximum value of $h(v)$, so that the number of relabel operations is finite.

Lemma. 4.4.1 If u has excess, i.e., $e(u) > 0$, then there is a u, s-dipath in D'.

Proof. We show the contrapositive: let T be the set of all nodes $u \in N$ with no u, s-dipath in D' and we show that $e(u) = 0$. Suppose $\delta_{D'}(T) \neq \varnothing$, i.e., there exists vz where $v \in T$ and $z \in N \setminus T$. But then v could reach s via z. Thus, $\delta_{D'}(T) = \varnothing$. Then

$$\sum_{v \in T} e(v) = \cdots = 0 - c(\delta(T)) \leq 0. \quad \square$$

Cor. 4.4.2 Throughout the algorithm, $h(u) \leq 2|N| - 1$ for all $u \in N$.

Proof. Suppose that at some point in the algorithm the height $h(u)$ increases to $2|N|$. Since a relabel operation only occurs when there is excess, this implies that $e(u) > 0$. By compatibility, height decreases by at most 1 in each arc resulting in $h(s) > |N|$, a contradiction. □ ??

Cor. 4.4.3 The total number of relabel operations is at most $2|N| \times |N| = 2|N|^2$.

Proof. There are $|N|$ nodes in D, each can have at most $2|N|$ relabel operations (from 0 to $2|N|-1$), so the overall upper bound is $2|N|^2$. $\square$

Def. 4.4.4 We perform a *saturating push* if we push r_{uv} on arc $uv \in A(D')$ and a *non-saturating push* if we push $e(u)$.

Remark. Observe if we perform a saturating push on uv, then $x_{uv} = c_{uv}$ so uv disappears from D' and vu is in D'.

Prop. 4.4.5 The number of saturating pushes throughout the algorithm is at most $2|N||A|$.

Proof. Suppose we have a saturating push on uv, then before the push, $h(u) = h(v) + 1$. Observe uv disappears from D' as $x_{uv} = c_{uv}$. In order to push on uv again, we need to first push flow on vu. To do so, we need to relabel v at least twice to obtain $h(v) = h(u) + 1$. Thus, between two saturated pushes on the same edge (uv), at least two relabel operations need to occur. By **Cor. 4.4.2**, the maximum number of relabel operations on node v cannot exceed $2|N|$, so up to $|N|$ saturating pushes are possible on uv. There are $2|A|$ arcs in D' (counting both forward and backward), so the number of saturating pushes is $2|N||A|$. $\square$

Prop. 4.4.6 The number of non-saturating pushes throughout the algorithm is $\leq 4|N|^2|A|$.

Proof. Define the function $\Phi(x, h)$ which counts the total height for nodes with excess, i.e.,

$$\Phi(x, h) = \sum_{v \in N, e(v) > 0} h(v).$$

At initialization, $\Phi = 0$ (because nodes with excess all have height zero) and the function never becomes negative (as we are adding up non-negative heights). We will determine the effect on $\Phi(x, h)$ for each type of operation.

Each relabel operation is done on a node with excess, so $\Phi(x, h)$ goes up by one. There are $2|N|^2$ relabel operations by **Cor. 4.4.3**, so the maximum increases for relabeling is $2|N|^2$ throughout the algorithm.

Each saturating push (i.e., push r_{uv}) on uv decreases $e(u)$ and increases $e(v)$. If $e(v) = 0$ before the push, we add $h(v)$ to $\Phi(x, h)$ after the push. By **Lemma. 4.4.1**, $h(v) \leq 2|N| - 1$, so we add at most $2|N| - 1$ over the algorithm. By **Prop. 4.4.5**, there are no more than $2|N||A|$ saturating pushes, so the maximum increase from saturating pushes is $2|N||A|(2|N| - 1) \leq 4|N|^2|A|$.

Each non-saturating push (i.e., push $e(u)$) on uv makes $e(v)$ positive but $e(u) = 0$. At worst, we need to add $h(v)$ to $\Phi(x, h)$ and we subtract $h(u)$ from $\Phi(x, h)$. By the choice of uv (i.e., while condition), $h(u) = h(v) + 1$, so $\Phi(x, h)$ decreases by at least 1. We have argued that $\Phi(x, h) \geq 0$ throughout the algorithm, so the number of non-saturating pushes (which decreases $\Phi(x, h)$ by ≥ 1) is therefore at most $2|N|^2 + 2|N||A|(2|N| - 1) = 4|N|^2|A| + 2|N|^2 - 2|N||A| \leq 4|N|^2|A|$. $\square$

Cor. 4.4.7 Preflow-push terminates in $2|N|^2 + 2|N|^2|A| + 4|N|^2|A| = 8|N|^2|A| \approx 8|N|^4$ operations.

Proof. Note that $O(|A|) = O(|N|^4)$. See **Prop. 4.4.6** proof. □

5 Global Minimum Cut

5.1 Global Minimum Cut Problem

Def. 5.1.1 Given $D = (N, A)$ and capacity c, we want to find a global (non-trivial) minimum cut, i.e., partition N into two disjoint sets S and S', where $c(\delta(S)) := \{c_{uv} : u \in S, v \in S'\}$ is minimized.

Remark. (Brute-Force) Run maximum flow algorithm on every possible pair $s, t \in N(D)$, which requires $|N|(|N| - 1) \in O(|N|^2)$ calls to maximum flow.

Remark. (Improvement) Fix $s \in N$. Observe it must reside on either side of a global minimum cut, i.e., either $s \in S$ or $s \in S'$ where $\delta(S)$ is the global minimum cut. Thus, we just need to run maximum flow algorithm on all s, t-cuts and t, s-cuts for all $t \in N \setminus \{s\}$. There are $|N| - 1$ choices of t so this requires $2(|N| - 1)$ calls to maximum flow.

5.2 Generic Algorithm for Minimum s-Cut

The Hao-Orlin algorithm is a modification of preflow-push algorithm and solves the global minimum cut problem efficiently.

Def. 5.2.1 Given $X \subseteq N$ and $t \notin X$, an X, t-cut has the form $\delta(S)$ where $X \subseteq S$ and $t \notin S$. In other words, if $\delta(S)$ is an X, t-cut and $uv \in \delta(S)$, then $u \in X \subseteq S$ and $t \in N \setminus S$.

Def. 5.2.2 Given $s \in N$, an s-cut has the form $\delta(S)$ where $s \in S$ and $S \neq N$. In other words, if $\delta(S)$ is an s-cut and $uv \in \delta(S)$, then $u \in S$ and $v \in N \setminus S$.

We can reduce the global minimum cut problem to minimum s-cut problem. By remark from Section 5.1, if a global minimum cut contains s, then we are good. Otherwise, reverse all arcs and we will find the global minimum cut. Thus, to solve global minimum cut, we just solve minimum s-cut twice for a fixed s.

Algorithm. 5.2.3 (Generic Algorithm for Minimum s-Cut)

1. Initialize $X = \{s\}$.
2. While $X \neq N$, pick $t \notin X$, find a min X, t-cut, add t to X.
3. Output the minimum over all cuts found.

Prop. 5.2.4 The generic algorithm above solves the minimum s-cut problem.

Proof. Let $\delta(S^*)$ be a minimum s-cut. Consider the first time we pick some t not in S^*. Immediately before this step, $X \subseteq S^*$. The algorithm gives us a minimum X, t-cut $\delta(\bar{S})$. But $\delta(S^*)$ is also an X, t-cut and $\delta(\bar{S})$ is also an s-cut, so they have the same capacity. □

5.3 Preparation for Hao-Orlin

We define an X-preflow to be a flow where every node not in X has a non-negative excess (and allow nodes in X to have negative excess).

Def. 5.3.1 For $X \subseteq N$, an X-preflow is a flow where $e(v) = x(\delta(\bar{v})) - x(\delta(v)) \geq 0$ for $v \notin X$.

The definition for compatible height is similar to preflow-push (with (1) and (2) is modified).

Def. 5.3.2 Height h are compatible with an X-preflow if

- $h(v) = |N|$ for all $v \in X$.
- $h(t) \leq |X| - 1$
- $h(v) \geq h(u) - 1$ for all $uv \in A(D')$.

We define two more terms for Hao-Orlin.

Def. 5.3.3 A *level* k, denoted $H(k)$, consists of all nodes with height k.

Def. 5.3.4 A *cut level* is a level k where no arc goes from $H(k)$ to $H(k-1)$ in D'.

Lemma. 5.3.5 If $\delta(S)$ is an X,t-cut with $\delta_{D'}(S) = \varnothing$ and $e(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$ then $\delta(S)$ is a minimum X,t-cut.

Proof. Take any X,t-cut $\delta(S)$. Note the flow out of S is equal to the flow into $N \setminus S$. The net flow out of S is $x(\delta(S)) - x(\delta(\bar{S})) \leq x(\delta(S)) \leq c(\delta(S))$. The net flow into $N \setminus S$ is equal to

$$\sum_{v \in N \setminus S} x(\delta(\bar{v})) - x(\delta(v)) = \sum_{v \in N \setminus S} e(v) \geq e(t).$$

Since $\delta_{D'}(S) = \varnothing$, all out-arcs of S are full and all in-arcs of S are empty, so

$$x(\delta(S)) - x(\delta(\bar{S})) = c(\delta(S)).$$

Next, $e(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$ tells us

$$\sum_{v \in N \setminus S} e(v) = e(t) + \sum_{v \in N \setminus (S \cup \{t\})} e(v) = e(t) + 0 = e(t).$$

Consider the flow f. Since $e(t) \leq f \leq c(\delta(S))$ for every f and we see that for this particular S we have $e(t) = c(\delta(S))$, then S achieves the minimum (lower bound) and hence $\delta(S)$ is a minimum X,t-cut. □

Cor. 5.3.6 If ℓ is a cut level and $e(v) = 0$ for all v with $h(v) < \ell$, except t, then $\{v : h(v) \geq \ell\}$ is a min X,t-cut.

Proof. Consider $S := \{v : h(v) \geq \ell\}$, the set of nodes at or above level ℓ. If ℓ is a cut level, there is no arc going from level ℓ to level $\ell - 1$ in D', so $\delta(S)$ is an X,t-cut with $\delta_{D'}(S) = \varnothing$. Combine this with $e(v) = 0$ for all v with $h(v) < \ell$ except t, we see that $\delta(S)$ is indeed a minimum X,t-cut. □

5.4 Hao-Orlin Algorithm

We run preflow push algorithm, maintaining a cut level ℓ while getting rid of excess on nodes below ℓ. When we succeed, $S := \{v : h(v) \geq \ell\}$ is the desired min X, t-cut by **Cor. 5.3.6**.

Note the algorithm keeps non-empty levels consecutive (except $|N|$) at step (2 a(ii)).

Algorithm. 5.4.1 (Hao-Orlin)

1. *Initialization.*
   a. Initialize $X = \{s\}$ and pick $t \in N \setminus X$.
   b. Initialize $h(s) = |N|$ and $h(v) = 0$ for all other $v \in N \setminus \{s\}$.
   c. Initialize $\ell = |N| - 1$.
   d. Send as much flow out of s as possible.
2. *Loop.* While $X \neq N$,
   a. While there exists low node $u \in N \setminus \{s, t\}$ with excess, i.e., $e_u > 0$ and $h(v) < \ell$:
      i. If there exists $uv \in A(D')$ where $h(v) = h(u) - 1$, push $\min\{r_{uv}, e_u\}$ on uv.
      ii. Otherwise,
         i. If v is the only node with $h(v)$, do not relabel. Instead, set $\ell = h(v)$.
         ii. Otherwise, to maintain a cut level, relabel and reset $\ell = |N| - 1$.
   b. When no node satisfies $e(v) = 0$ and $h(v) < \ell$, store the cut $\{v : h(v) \geq \ell\}$ (this is a min X, t-cut by **Cor. 5.3.6**.)
      i. Add t to X. Set $h(t) = |N|$.
      ii. Send as much flow out of t as possible (so that $\delta_{D'}(t) = \varnothing$).
      iii. Pick t with lowest height.
      iv. Reset cut level by setting $\ell = |N| - 1$ because we want a non-trivial s-cut (if ℓ is too low, you might not be able to find another node so you end up with a trivial cut).
3. Pick the minimum cut amongst the stored cuts. This is your global minimum cut.

5.5 Correctness

Lemma. 5.5.1 The non-empty levels less than $|N|$ are consecutive.

Proof. Initially, $X = \{s\}$ is at level $|N|$ and everything else is at level 0, so the claim is trivially true at initialization. We do not relabel v when v is the only node of height h; this keeps non-empty levels consecutive. Transitioning to a new iteration, we move t with lowest height to X meaning the non-empty levels remain consecutive. □

Prop. 5.5.2 The X-preflow and height h are always compatible.

Proof. $h(v) = |N|$ for all $v \in X$ follows from the algorithm. $h(v) \geq h(u) - 1$ for all $uv \in A(D')$ follows from PFP during an iteration. At the end of each iteration, we move t to X and push all flow out of t. Thus, we have $h(v) \geq h(u) - 1$ for all $uv \in A(D')$.

It remains to show that $h(t) \leq |X| - 1$. Initially, $|X| = 1$ and $h(t) = 0 \leq 0 = |X| - 1$. When we move t to X, the next t, call it t', has height $h(t)$ or $h(t) - 1$, since non-empty levels are consecutive. Originally, $h(t) \leq |X| - 1$. We add 1 to $|X|$ after the move and might add 1 to $h(t)$, so this inequality is true after this move. □

Lemma. 5.5.3 $h(v) \leq |N| - 2$ for all $v \notin X$.

(Idea: Consider the worst case, where we put a single node on each of the levels.)

Proof. Recall $h(t) \leq |X| - 1$. There are $|N| - |X| - 1$ nodes not in $X \cup \{t\}$. Since the non-empty levels are consecutive, the highest level not in X is $\leq (|X| - 1) + (|N| - |X| - 1) = |N| - 2$. □

Lemma. 5.5.4 ℓ is always a cut level.

Proof. When $\ell = |N| - 1$, level ℓ is empty, so it is automatically a cut level. We change ℓ to something else when we want to relabel v, but v is the only node with its height. We want to relabel v because $e(v) > 0$ and no neighbour of v is one level below v in D'. Thus, no arcs goes from $h(v)$ to level $h(v) - 1$. Thus, $h(v)$ is a cut level and we can set $\ell = h(v)$. □

Prop. 5.5.5 The stored cuts in each iteration are minimum X, t-cuts.

Key. It suffices to show that ℓ is always a cut level, because then our corollary from before applies.

Proof. ℓ is always a cut level, so **Cor. 5.3.6** applies. Thus, Hao-Orlin produces a min s-cut. □

5.6 Termination

Relabel operation: $h(v) \leq |N| - 2$ so in total $\leq |N|^2$ operations.

Saturating and non-saturating pushes: same as before.

Level setting operations: at most number of relabel operations $+n$ (number of iterations).

Overall: the runtime is roughly the same as preflow-push.

(Termination of Hao-Orlin is not on the exam.)

6 Global Min Cut in Undirected Graphs

Consider the undirected graph $G = (V, E)$ with edge capacities $c \in \mathbb{R}_+^E$.

6.1 Karger's Algorithm

Idea.

1. Pick one edge at random and contract it.
2. Keep track of vertices each contracted vertex represents.
3. Do this until two vertices remains.
4. Output the cut represented by these two vertices.

In general, edges with small capacities are more likely to be in a minimum cut, so we will try to lower the probability that these edges are selected. Thus, we set probability of an edge gets selected proportional to its capacity.

Algorithm. 6.1.1

- While $|V| > 2$, pick uv with probability $\frac{c_{uv}}{\sum_{e \in E} c_e}$ and contract uv. Repeat.

Thm. 6.1.2 Let $\delta(S^*)$ be a global min cut. The probability that the algorithm produces $\delta(S^*)$ is

$$\geq \frac{1}{\binom{|V|}{2}}$$

Proof. Consider the probability we pick an edge in $\delta(S^*)$ in the first step. The denominator is $\sum_{e \in E} c_e$ and the numerator is $c(\delta(S^*))$.

Consider the cuts of the form $\delta(\{v\})$ for $v \in V$. Each edge uv appears in two such cuts $\delta(\{u\})$ and $\delta(\{v\})$. Since $\delta(S^*)$ is a global minimum cut,

$$\sum_{e \in E} c_e = \frac{1}{2} \sum_{v \in V} c(\delta(\{v\})) \geq \frac{1}{2} \sum_{v \in V} c(\delta(S^*)) = \frac{1}{2}|V| c(\delta(S^*))$$

So the probability that an edge in $\delta(S^*)$ is selected is

$$\frac{c(\delta(S^*))}{\sum_{e \in E} c_e} \leq \frac{c(\delta(S^*))}{\frac{1}{2}|V| c(\delta(S^*))} = \frac{1}{\frac{1}{2}|V|} = \frac{2}{|V|}.$$

The probability that $\delta(S^*)$ survives the first contraction is thus at least $1 - \frac{2}{|V|}$.

Now suppose we have contracted k edges and $\delta(S^*)$ is still intact. We have $|V| - k$ vertices left and say the graph is $G' = (V', E')$. We want to find the probability of selecting an edge in $\delta(S^*)$.

The numerator is $c(\delta(S^*))$; the denominator is $\sum_{e\in E'} c_e$. Now, each node represents several nodes (because of contractions):

$$\sum_{e\in E'} c_e = \frac{1}{2}\sum_{v\in V'} c(\delta(v)) \geq \frac{1}{2}\sum_{v\in V'} c(\delta(S^*)) = \frac{1}{2}(|V|-k)c(\delta(S^*)).$$

So, the probability is at most

$$\frac{c(\delta(S^*))}{\frac{1}{2}(|V|-k)(c(\delta^*))} = \frac{2}{|V|-k}.$$

The probability that $\delta(S^*)$ survives this contraction is at least

$$1-\frac{2}{|V|-k}.$$

The largest possible k is $|V|-3$ (since we finish when we have 2 vertices, so in the last step, we have $3 = |V|-(|V|-3)$ vertices left).

Overall, the probability that $\delta(S^*)$ survives all contractions is

$$\geq \prod_{k=0}^{|V|-3}\left(1-\frac{2}{|V|-k}\right) = \prod_{k=0}^{|V|-3}\frac{|V|-k-2}{|V|-k} = \frac{2}{|V|(|V|-1)} = \frac{1}{\binom{|V|}{2}}. \quad \square$$

We expect the algorithm to produce a minimum cut if it runs $\binom{|V|}{2} \sim |V|^2$ times.

Cor. 6.1.3 The probability that that algorithm produces $\delta(S^*)$ after $k|V|^2$ runs is at least $1-e^{-2k}$ where $k \geq 1$.

Proof. We use $1-x \leq e^{-x}$. The probability of failure is at most

$$\left(1-\frac{1}{\binom{|V|}{2}}\right)^{k|V|^2} \leq \left(1-\frac{1}{|V|^2}\right)^{k|V|^2} \leq \left(e^{\frac{-2}{|V|^2}}\right)^{k|V|^2} = e^{-2k}. \quad \square$$

Ex. 6.1.4 Let $k=5$. Then $1-e^{-10} \approx 0.999955$. So after $5|V|^2$ times we have a 0.999955 chance of getting a global min cut.