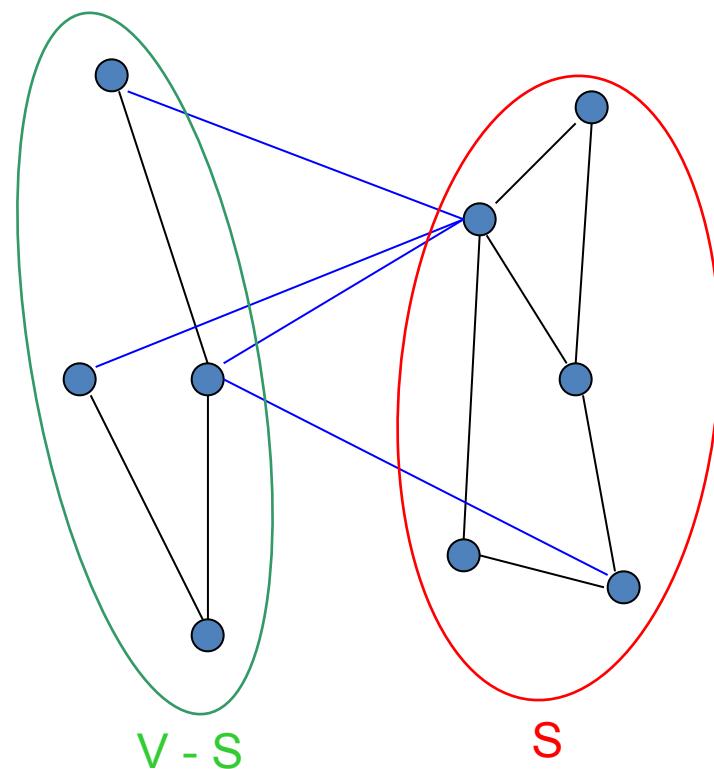


Min-cut for Undirected Graphs

Given an undirected graph, a global min-cut is a cut $(S, V-S)$ minimizing the number of crossing edges, where a crossing edge is an edge (u, v) s.t. $u \in S$ and $v \in V-S$.

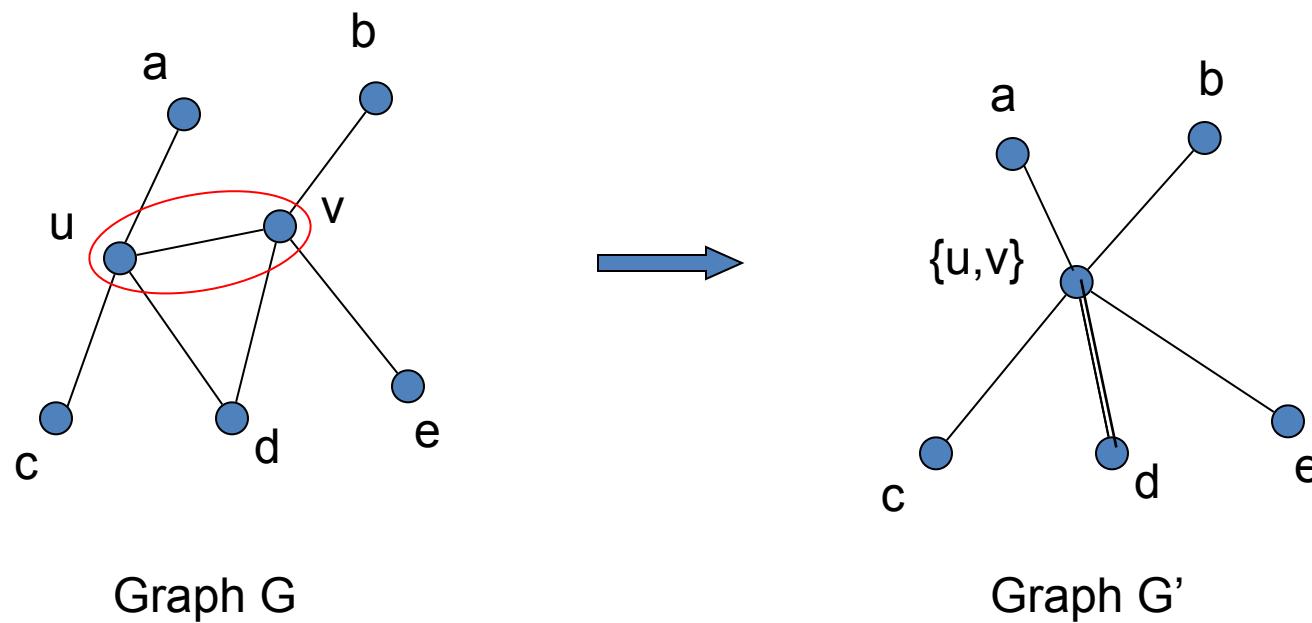


Graph Contraction

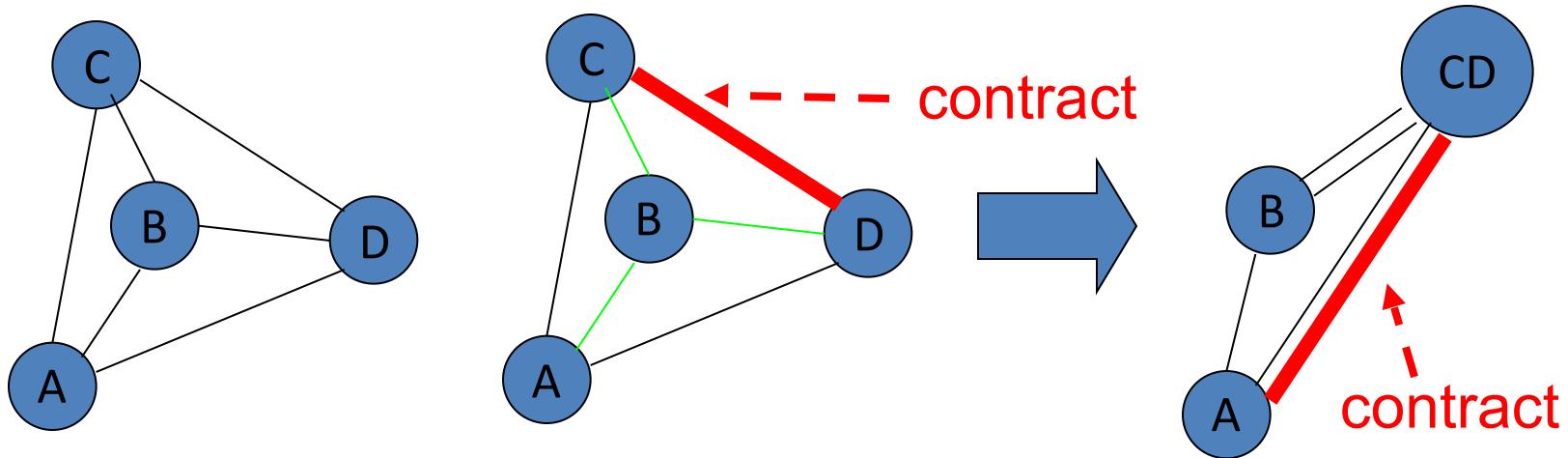
For an undirected graph G , we can construct a new graph G' by contracting two vertices u, v in G as follows:

- u and v become one vertex $\{u,v\}$ and the edge (u,v) is removed;
- the other edges incident to u or v in G are now incident on the new vertex $\{u,v\}$ in G' ;

Note: There may be multi-edges between two vertices. We just keep them.

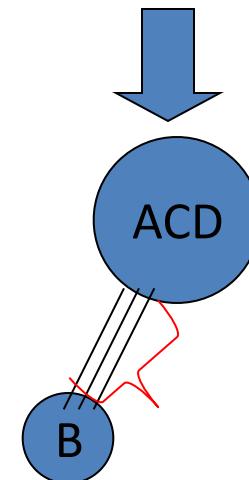


Karger's Min-cut Algorithm



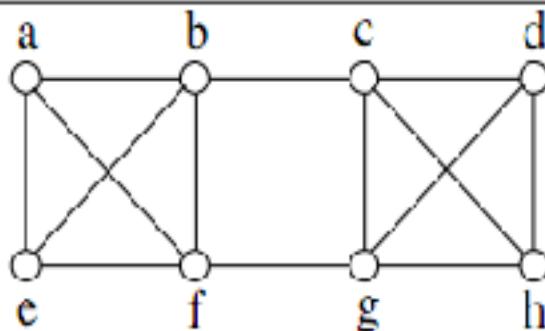
(i) Graph G (ii) Contract nodes C and D (iii) contract nodes A and CD

Note: C is a cut but not necessarily a min-cut.

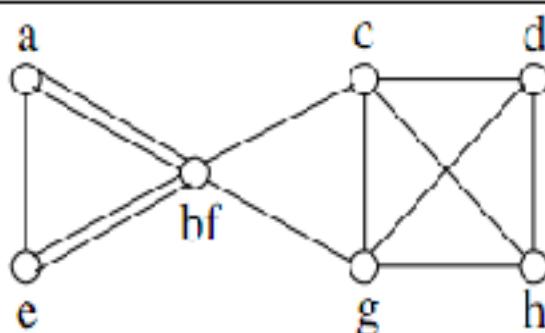


(iv) Cut $C = \{(A, B), (B, C), (B, D)\}$

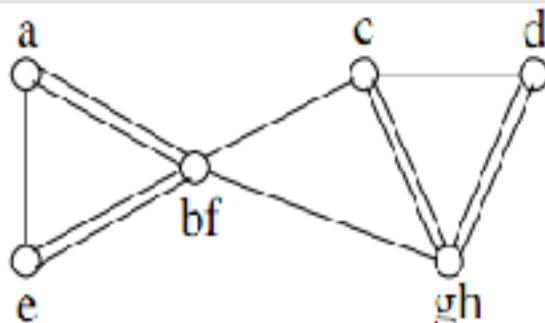
Example



14 edges to choose from
Pick $b - f$ (probability $1/14$)

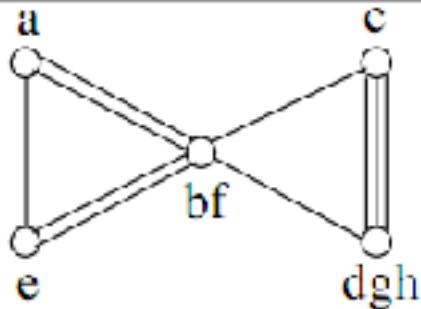


13 edges to choose from
Pick $g - h$ (probability $1/13$)

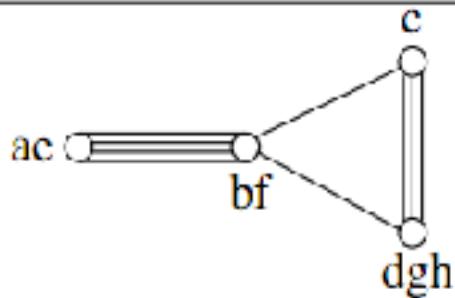


12 edges to choose from
Pick $d - gh$ (probability $1/6$)

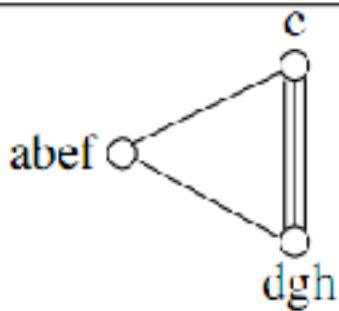
Example Cont.



10 edges to choose from
Pick $a - e$ (probability 1/10)



9 edges to choose from
Pick $ab - ef$ (probability 4/9)



5 edges to choose from
Pick $c - dgh$ (probability 3/5)



Done: just two nodes remain

Karger's Min-cut Algorithm

Karger Min-Cut

input : A graph $G = (V, E)$

output: A cut $C \subseteq E$

begin

repeat

 Choose a random edge e ;

 Contract e and merge its endpoints into a single vertex ;

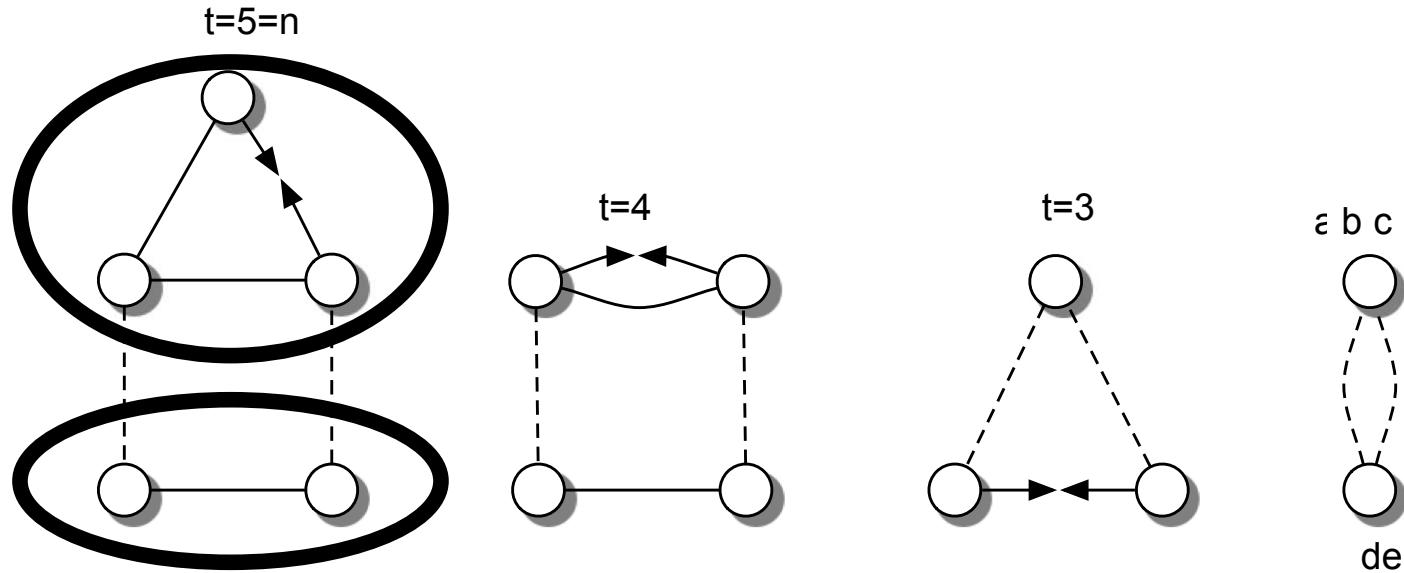
until there are only two vertices a, b left ;

 Let C be the set of edges between a and b ;

return C ;

end

Note that as the graph shrinks, some pairs of vertices will have multiple edges between them, and we give each such edge an equal probability of being chosen for contraction. For instance, if u and v have three edges between them and there are a total of m edges left, then u and v will be merged at the next step with probability $3/m$.



After three contractions, a particular minimum cut C_{\min} consisting of the two dashed edges survives. Note that, in the second step, there are two edges connecting the upper two vertices, so the probability that they will be merged is $2/5$.

We repeat this process until only two vertices are left. At that point, let C be the set of surviving edges connecting these two vertices.

Clearly C is a cut, since it separates G into the two pieces that contracted to form a and b respectively. What is surprising is that, with reasonably high probability, C is as small as possible.

To see this, let C_{\min} be the minimum cut, or one of the minimum cuts if it is not unique, and suppose that C_{\min} has size k .

Then $C = C_{\min}$ if and only if these k edges survive to the end of the algorithm without being contracted.

For instance, in Fig. 10.4 a particular C_{\min} of size 2 survives until only two vertices remain.

Now consider the step of the algorithm when there are t vertices left.

Each vertex must have at least k edges, since otherwise cutting its edges would give a cut of size less than k .

Thus there are at least $t k / 2$ edges left, and the probability that none of the k edges in C_{\min} are chosen for contraction on that step is

$$\begin{array}{l} P=1/(\# \text{ edges}) \text{ uniform} \\ \# \text{ edges} \geq t k / 2 \end{array} \quad 1 - \frac{k}{\# \text{ edges}} \geq 1 - \frac{k}{t k / 2} = \frac{t-2}{t} .$$

If G has n vertices to start with, the probability that C_{\min} survives until the end is the product of this probability over all $n - 2$ steps. This gives the lower bound

product over all steps
of the probability the C_{\min}
survives at each step

$$p \geq \prod_{t=3}^n \frac{t-2}{t} = \frac{1 \cdot 2 \cdot 3 \cdots (n-2)}{3 \cdot 4 \cdot 5 \cdots n} = \frac{2}{n(n-1)} .$$

Thus the probability of success is $\Omega(1/n^2)$.

Note that if the minimum cut is not unique, this lower bound applies to the probability that *any particular one* survives.

Succeeding with probability $\Omega(1/n^2)$ may not seem like a great achievement.

However, we can increase the probability of success by trying the algorithm multiple times.

Since each attempt is independent, if we make $1/p = O(n^2)$ attempts, then the probability that none of them finds C_{\min} is

$$(1-p)^{1/p} \leq 1/e,$$

prob. that at least one succeeds is

$$1 - (1-p)^{1/p} \geq 1 - 1/e = 0.63$$

where we used the inequality $1 - x \leq e^{-x}$.

So, the probability that at least one of these $1/p$ attempts succeeds is at least $1 - 1/e \approx 0.63$.

This gives an algorithm that succeeds with constant probability, i.e., with probability $\Omega(1)$.

If we want an algorithm that succeeds with *high* probability, i.e., with probability

$$1 - o(1),$$

we can make a somewhat larger number of attempts.

If we try the algorithm

$$(1/p) \ln n = O(n^2 \log n)$$

times, say, then the probability that every attempt fails is

$$(1-p)^{(1/p)\ln n} \leq e^{-\ln n} = 1/n = o(1).$$

Thus we can raise the probability of success from $\Omega(1)$ to $1 - o(1)$ with just a $\log n$ increase in the running time.

Moreover, the total running time of this algorithm is competitive with the best known deterministic algorithms.

This “boosting” technique is a simple but important theme in randomized algorithms.

If each attempt succeeds with probability p , the average number of attempts before we succeed is $1/p$, and we can succeed with high probability by making slightly more than $1/p$ attempts.

If each attempt takes polynomial time and $p = 1/\text{poly}(n)$, then the total running time is $\text{poly}(n)$.