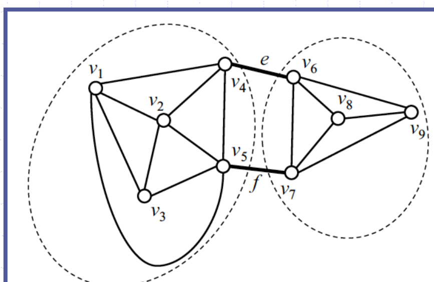
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Minimum Cuts

Trees with snow on branches, "Half Dome, Apple Orchard, Yosemite," 1933. Ansel Adams. U.S. government image. U.S. National Archives and Records Administration.

Definition of Minimum Cut

- A cut, C, of a connected graph, G, is a subset of the edges of G whose removal disconnects G.
- That is, after removing all the edges of C, we can partition the vertices of G into two subsets, A, and B such that there are no edges between a vertex in A and a vertex in B.
- A **minimum cut** of G is a cut of smallest size among all cuts of G.



Applications

- In several applications, it is important to determine the size of a smallest cut of a graph.
  - For example, in a communications network, the failures of the edges of a cut prevents the communication between the nodes on the two sides of a cut.
  - Thus, the size of a minimum cut and the number of such cuts give an idea of the vulnerability of the network to edge failures.
- Small cuts are also important for the automatic classification of web content.
  - Namely, consider a collection of web pages and model them as a graph, where vertices correspond to pages and edges to links between pages.
  - The size of a minimum cut provides a measure of how much groups of pages have related content. Also, we can use minimum cuts to recursively partition the collection into clusters of related documents.

Relationship to Max Flow

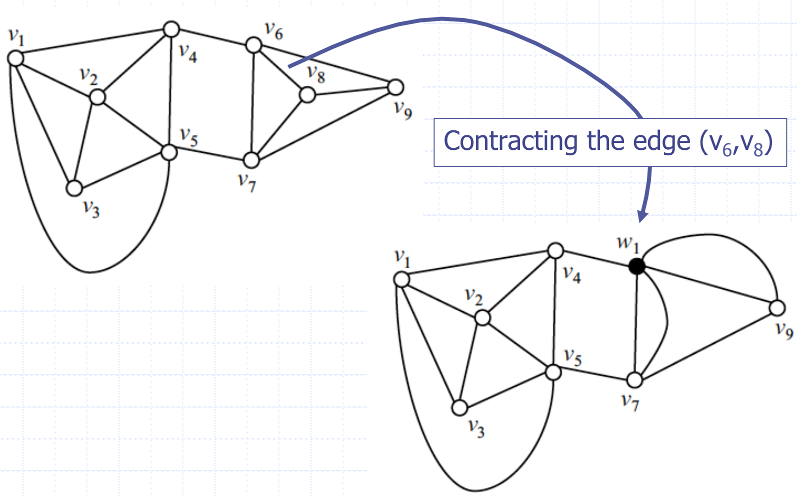
- The min-cut/max-flow theorem states that, given a pair of vertices, s and t, we can compute a minimum cut in polynomial time such that s is on one side of the cut and t is on the other.
- In this case, however, we want the minimum cut over all possible cuts.
- Nevertheless, we can compute such an overall minimum cut by O(n) calls to an (s,t)-min-cut-max-flow algorithm. How? (See Exercise C-19.9.)
- Here, we show how to design an simple, efficient randomized algorithm that succeeds with high probability without using min-cut-max-flow.

Contracting Edges

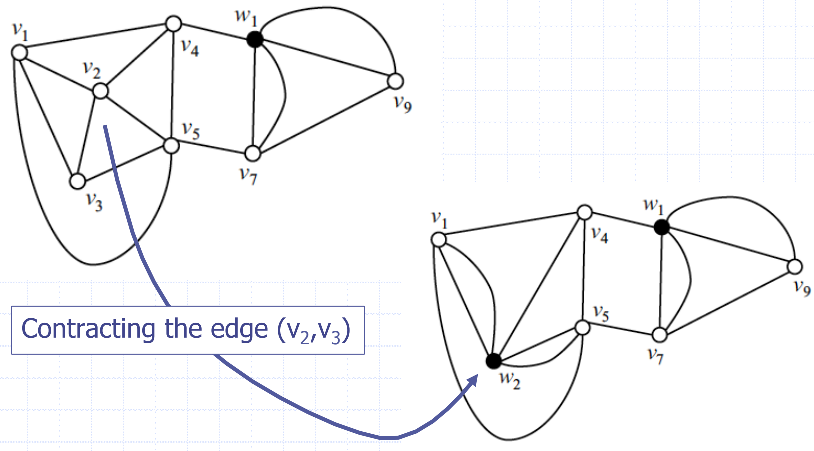
- The simple randomized algorithm algorithm repeatedly performs contraction operations on the graph.
- Let G be a graph with n vertices, where we allow G to have parallel edges. We denote with (v,w) any edge with endpoints v and w.
- The contraction of an edge e of G with endpoints u and v consists of the following steps that yield a new graph with n − 1 vertices, denoted G/e:

1. Remove edge e and any other edge between its endpoints, u and v.
2. Create a new vertex, w.
3. For every edge, f, incident on u, detach f from u and attach it to w. Formally speaking, let z be the other endpoint of f. Change the endpoints of f to be z and w.
4. For every edge, f, incident on v, detach f from v and attach it to w.

Contracting Edges, An Example

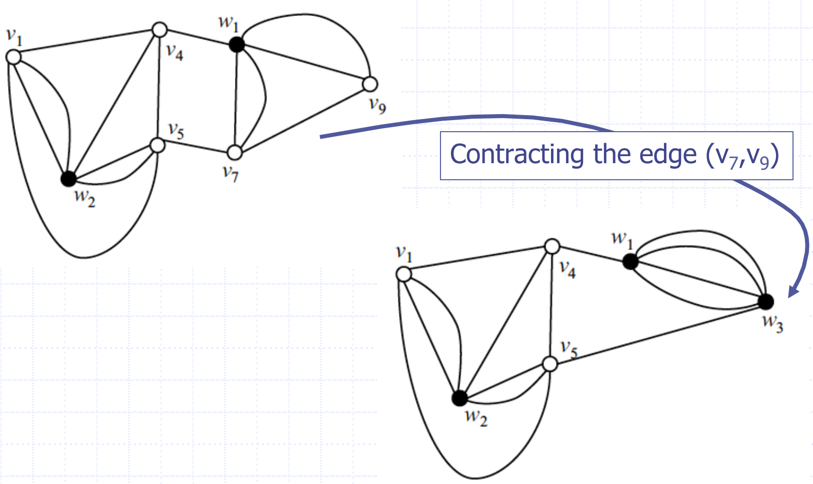


Contracting Edges, An Example



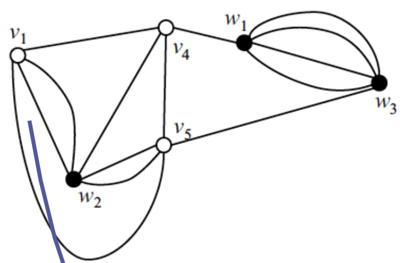
 Minimum Cuts 7

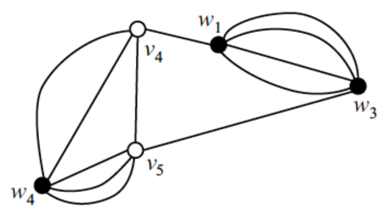
Contracting Edges, An Example



 Minimum Cuts 8

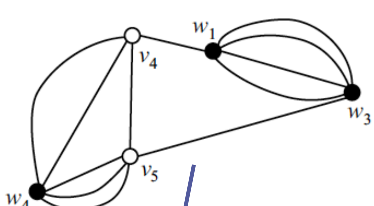
Contracting Edges, An Example





Contracting the edge (v_1,w_2)

Contracting Edges, An Example



Contracting the edge (v_4,v_5), yielding w_5, contracting (w_4,w_5), yielding w_6, and contracting (w_1,w_3), yielding w_7.

This final graph has the same minimum cut as the original graph.

Karger's Algorithm

- A simple min-cut algorithm, which succeeds with high probability is to repeat the following procedure multiple times, keeping track of the smallest cut that it ever finds:

Algorithm ContractGraph(G):
Input: An undirected graph, G, with n vertices
Output: A cut of G that has minimum size with probability at least $\frac{2}{n(n-1)}$

```
while G has more than 2 edges do
    pick a random edge, e, of G
    contract edge e
    G ← G/e
return the edges of G
```


Analysis

- Let G be a graph with n vertices and m edges, and let C be a given minimum cut of G. We will evaluate the probability that the algorithm returns the cut C.
- Since G may have other minimum cuts, this probability is a lower bound on the success probability of the algorithm.
- Let G_i be the graph obtained after i contractions performed by the algorithm and let m_i be the number of edges of G_i. Assume that G_{i-1} contains all the edges of C. The probability that G_i also contains all the edges of C is equal to $1 - k/m_{i-1}$ since we contract any given edge of C with probability $1/m_{i-1}$ and C has k edges.
- Thus, the probability, P, that the algorithm returns cut C is

$$P = \prod_{i=0,\ldots,n-3} \left(1 - \frac{k}{m_i}\right).$$

Analysis, part 2

- Since k is the size of the minimum cut of each graph G_i, we have that each vertex of G_i has degree at least k.
- Thus, we obtain the following lower bound on m_i, the number of edges of G_i:

$$m_i \geq \frac{k(n-i)}{2}, \quad \text{for } i = 0, 1, \ldots, n-3 .$$

- We can then use these bounds to derive a lower bound for P.

Analysis, part 3

- The following bound implies that P is at least proportional to $1/n^2$:

$$\begin{aligned}
P &= \prod_{i=0}^{n-3}\left(1-\frac{k}{m_i}\right) \\
&\geq \prod_{i=0}^{n-3}\left(1-\frac{2k}{k(n-i)}\right) \\
&= \prod_{i=0}^{n-3}\left(\frac{n-i-2}{n-i}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\left(\frac{n-5}{n-3}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} \\
&= \frac{1}{\binom{n}{2}}
\end{aligned}$$

Analysis, part 4

- We can boost the probability by running the algorithm multiple times. In particular, if we run the
- algorithm for t*n-choose-2 rounds, where t is a positive integer, we have that at least one round returns cut C with probability

$$P(t) = 1 - \left(1 - \frac{1}{\binom{n}{2}}\right)^{t\binom{n}{2}}.$$

- By a well-known property (Theorem A.4) of the mathematical constant **e**, the base of the natural logarithm, ln, we obtain $P(t) > 1 - 1/\mathbf{e}^t$.
- If we choose $t = c \ln n$, where c is a constant, then the success probability is at least $1 - 1/n^c$.

Running Time Analysis

- A contraction operation can be executed in O(n) time, if the graph is represented using an adjacency list.
- Thus, the running time of one round is $O(n^2)$.
- We repeat the algorithm $O(n^2 \log n)$ times.
- Thus, the total running time is $O(n^4 \log n)$.
- This can be improved to $O(n^2 \log^3 n)$. (See book)