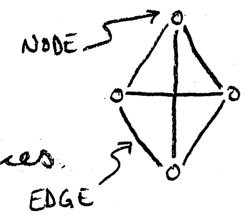
Graphs

A way to represent pairwise relationships among a set of objects.

Consists of:

Vertices (aka Nodes), V: An object

Edges, E: The relationship between two vertices.

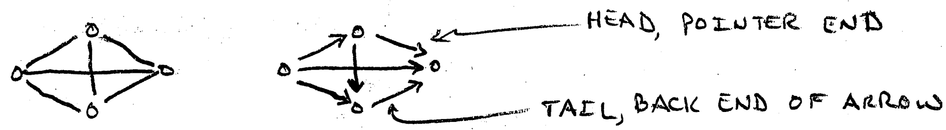


Two Types of Graphs

Undirected: The edges of the graph do not have order in their relationships with the vertices.

Directed: The edges of the graph have an order in their relationships with the vertices, for example, consider a one-way street. One can travel from street A to street B on a one-way street, but one cannot travel back to street A on the one-way street.

Undirected Directed

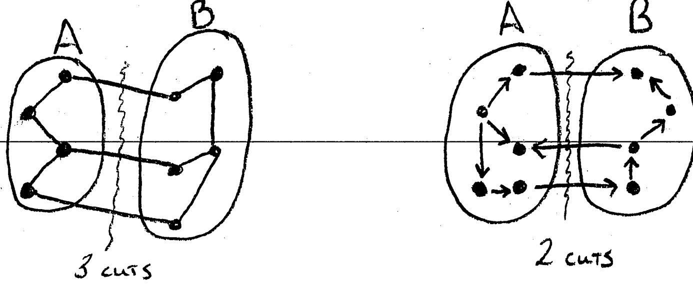


Used in: road networks, internet (webpage = node, hyperlink = edge), social networks (individual = vertex, relationships = edges)

Cuts of Graphs

A cut of a graph (V, E) is a partition of V into two non-empty sets, A and B.

Undirected Directed



Cuts depend on directionality of the relationship.

Generally, tail in A, head in B

All cuts count

(Con't on next sheet)

GRAPHS (con't)

CUTS OF GRAPHS (con't)

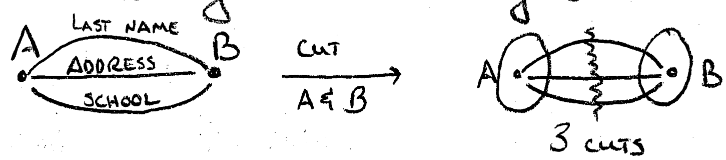
The maximum number of possible cuts a graph of n vertices can have is 2^n. This is because there are two choices in which a vertex can reside, set A or set B.

MINIMUM CUT (MINCUT) PROBLEM

GIVEN: An undirected graph $G = (V, E)$

FIND: The cut to the graph that will cross the fewest

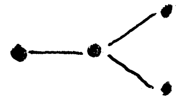
* Note: Parallel edges, edges that define different characteristics of relationships between two nodes all count. For example, two siblings might share the same last name, the same house, and go to the same school; there are three parallel edges between sibling A and sibling B.



BOUNDS ON EDGES OF UNDIRECTED GRAPH

MINIMUM BOUND, SPARSE GRAPH

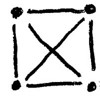
To have a fully-connected graph (all nodes are connected) the minimum number of edges is $n-1$, $n = \#$ of vertices.



$n = 4$
$\text{edges} = n-1 = 4-1 = 3$, aka $\Omega(n)$

MAXIMUM BOUND, DENSE GRAPH

$\frac{n(n-1)}{2}$



All nodes are connected to every other node.

$n = 4$
$\text{edges} = \frac{4(4-1)}{2} = 6$, aka $O(n^2)$
not counting parallel edges

(con't on next sheet)

Graphs (con't)

Representations

Adjacency Matrix

Represents a graph, G, with an $n \times n$ matrix A consisting of Ø's and 1s where an element in the i^{th} row and j^{th} column position

$A_{ij} = 1$ if and only if G has an i-j edge

Variants

Parallel edges – A_{ij} = # of i-j edges (Ø, 1, 2, …)

Weight assigned to an edge – A_{ij} = weight of edge i-j $(-\infty, \infty)$

Directionality of Edge $\begin{cases} +1 & \text{if } i \rightarrow j \\ -1 & \text{if } i \leftarrow j \end{cases}$

Space complexity for an $n \times n$ matrix is $\Theta(n^2)$.
Very wasteful for sparse graphs, good for dense graphs.

Adjacency Lists

Composed with:

- Array (or list) of vertices
- Array (or list) of edges
- Pointers to an edge's endpoints (nodes)
  Directed graph also tracks which pointer is the head and which is the tail.
- Pointers to a node's edges
  This can be done in a directed graph by keeping track of all nodes that serve as tails and/or all nodes that are the heads for a given edge.

Let m = # of edges

Space complexity is $\Theta(m+n)$ or $O(n)$
Good for graph search and sparse graphs.

RANDOM CONTRACTION ALGORITHM

Designed to solve the minimum cut problem (pg. 2)

Due to random nature of algorithm it can give an incorrect answer is $\frac{1}{n}$ where n is the number of vertices.

To overcome this limitation run the algorithm more than n times will allow for a more probable confident answer. That is, the correct answer will be the one that shows up the most in greater than n trials.

We can do better than this, however, this algorithm is not complex or complicated to implement.

PSEUDOCODE FOR KARGER'S ALGORITHM

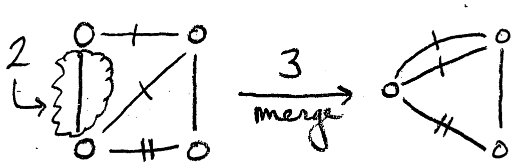
Let u and v each represent distinct nodes, that is, they cannot have identical values or properties.

Let (u,v) represent the relationship between (edge) vertices u and v.

```
1 While there are more than 2 vertices:
2    Pick a remaining edge (u,v) uniformly at random
3    Merge ("contract") u and v into a single vertex
4     Remove self-loops
5 Return final cut
```

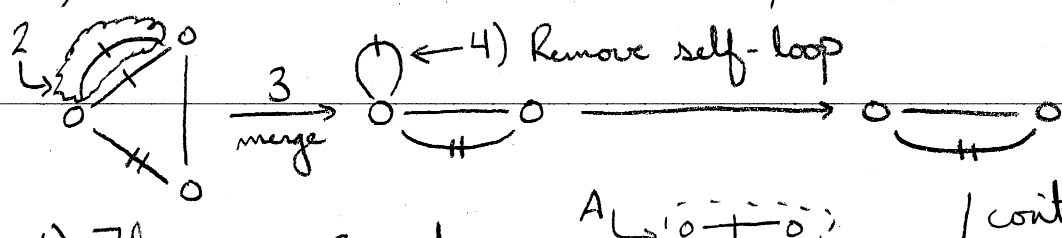
EXAMPLE

1) There are more than 2 vertices, $n = 4$



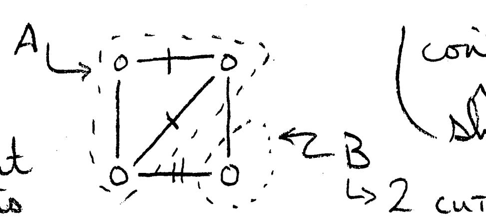
4) No self loops

1) There are more than 2 vertices, $n = 3$



4) Remove self-loop

1) There are 2 vertices
5) Return final cut
↳ On original problem cut would look like this

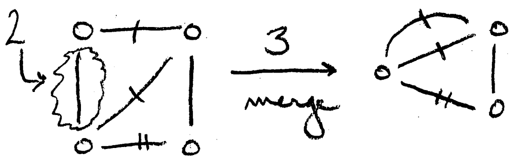


(con't on next sheet)

Random Contraction Algorithm (con't)

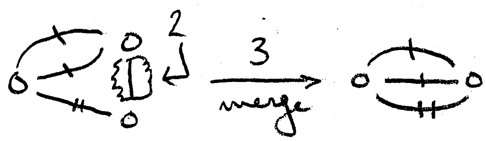
Example showing incorrect solution due to randomness

1) There are more than 2 vertices, $n=4$



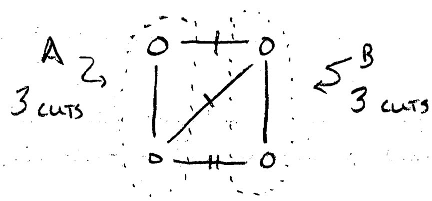
4) No self-loops

1) There are more than 2 vertices, $n=3$



4) No self-loops

1) There are 2 vertices remaining, $n=2$
5) Return final cut
↳ On original problem cut would look



From previous example this is not the minimum number of cuts. See page 4 for how to resolve the problem of getting the incorrect number of minimum cuts.