For the next two lectures we'll be seeing examples of approximation algorithms for interesting **NP**-hard problems. Today we consider MAX-CUT, which we proved to be **NP**-hard in Lecture 18. Our goal is to divide the vertices of an undirected graph G into two sets A and B, so as to maximize the number of edges that have one edge in A and the other in B.

Here is a somewhat greedy algorithm that does reasonably well at approximating the maximum cut. Given any two sets A and B, look at an arbitrary vertex v. Suppose for the moment that v is in A. Of the edges incident to v, some go to other vertices in A and the others go to B. If *more* vertices go to A then B, consider what happens if we move v into B. Our score goes up one for each edge to an element of A, and goes down one for each edge to B, so all in all it goes up.

Thus our algorithm – starting with $A = V$ and $B = \emptyset$, keep switching any vertex v from A to B or vice versa as long as it will increase the number of edges from A to B. If we reach a position where there is no such v, return that cut as the output.

Note that there is no reason a single v might not go back and forth from A to B several times. But the algorithm must terminate, because the count of edges across the cut starts at 0, increases by at least 1 with each switch, and ends at some value that is at most e. This analysis also gives us a bound on the running time – we have at most e rounds during which we might have to check all n vertices and examine all e edges, so the total time is $O(e(e+n))$.

How closely does this algorithm approximate the maximum? Consider any A and B such that each vertex has at least as many edges to the other set as it does to its own set. That is, the fraction of cross edges at each vertex is at least $1/2$. The fraction for the whole graph is a weighted average of the fractions for each edge (where the weight of each vertex is its degree), and so *it* must be at least $1/2$. Thus we have a 2-approximation to the maximum, because the fraction of edges across the maximum cut can't be any greater than 1.

Can we do better? In 1995, using a different method ("semidefinite programming"), Goemans and Williamson found a poly-time way to approximate MAX-CUT within 1.1383. Can we get a poly-time approximation scheme?

There is a lovely archive of known results (as of about 2000) on approximation algorithms for **NP**-hard problems, located at:

`www.nada.kth.se/~viggo/wwwcompendium`

There (at `node85.html`) we find that MAX-CUT is **NP**-hard to approximate within 1.0684. Like VERTEX-COVER, it is approximable to within one constant in polynomial time but not to within another constant, unless $\mathbf{P} = \mathbf{NP}$.