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 $\mathbf{P} = \mathbf{NP}$.

The **traveling salesperson problem** or **TSP** may be the most famous **NP-**hard optimization problem. We have a weighted complete graph G representing n cities, a solution is a **Hamilton circuit** of G, and our goal is to find a solution of minimum total weight.

It is **NP-**complete to decide whether an ordinary undirected graph has a Hamilton cycle. The proof of this is rather nice – given a 3-SAT instance, you construct a graph where the only way to have a cycle reaching all vertices is to pick a setting of the n variables that satisfies the formula. (Sipser has a nice presentation of this in *Introduction to the Theory of Computation*.) It stands to reason that comparing Hamilton paths by score would be at least as hard as finding one, and we can convert this intuition to a formal proof.

Let G be an undirected graph, and define a weighted complete undirected graph H with the same set of n vertices. Define the weight in H of (u, v) to be 2 if (u, v) is not an edge in G and 1 if it is. Now a Hamilton cycle in G corresponds to a Hamilton cycle in H of weight n, and any other Hamilton cycle in H has weight more than n. So the pair (H, n) is an achievable goal for TSP iff G is in the **NP**-complete language HAM-CYCLE.

We can use this same construction to show that TSP is difficult to approximate, even within a factor of, say, n. Given G as before, we form H with weight 1 for edges of G and some weight z for non-edges of G. Then Hamilton cycles of G have weight n in H, and other Hamilton cycles of H have weight at least $z + n - 1$. If we let $z = n^2$, any approximation algorithm with ratio n would have to distinguish these two cases and thus decide HAM-CYCLE for G. How bad could we make the approximation ratio? It is **NP**-hard to get within z/n, where z is anything we can even *write down* in polynomial time.

However, it's natural to consider the special case of TSP where the weight function is somehow "reasonable". The **triangle inequality** is a restriction on a binary distance function d, saying that for any three vertices x, y, and z,

$$d(x, z) \leq d(x, y) + d(y, z).$$

This is the defining property of a **metric**, so the restriction of TSP to weight functions that satisfy the triangle inequality is called **Triangle-TSP** or **Metric TSP** or **MTSP**. Note that our first reduction, with weight 1 for edges and 2 for non-edges, gives us a weight function on H that satisfies the triangle inequality. Thus MTSP is still an **NP-**hard optimization problem.

But it is possible to approximate MTSP to within a constant factor, using a technique based on **Euler circuits** rather than Hamilton circuits. Remember that an Euler circuit is one that visits all the *edges* in a graph. In an undirected graph, an Euler circuit exists iff the graph is connected and every node has even degree. in a directed graph, an Euler circuit exists iff the graph is strongly connected and every vertex has in-degree equal to its out-degree.

For our first approximation algorithm, consider a minimum spanning tree T of the weighted complete graph H. Now view this tree as a directed graph by replacing each edge with a directed edge in each direction. The result is a strongly connected directed graph with equal in-degree and out-degree at every vertex. Let C be an Euler circuit of this graph. Note that the total weight of C is exactly twice that of T, because each edge of T appears in C exactly twice.

We can use this tree T and circuit C to get a relatively low-weight Hamilton circuit in H, and to analyze how close it is to the minimum weight circuit. Of course, C itself is not a Hamilton circuit because it has $2n - 2$ edges, not n. However, we can *shorten* C in terms of edges without increasing its total weight. If x, y, and z are any three consecutive vertices in C, we can replace the two edges (x, y) and (y, z) by the single edge (x, z), and the triangle inequality tells us that the total weight cannot increase. Does the resulting circuit still reach every vertex? Yes, as long as y appears somewhere else on C.

Thus we get a Hamilton circuit by using such shortcuts to bypass any vertex that appears more than once, continuing as long as we can. The process stops when our circuit has only n vertices. Since the weight of C was at most twice that of T, and each bypass at worst keeps the weight the same, we have a Hamilton circuit A of weight at most $w(T)$.

But what of the optimal circuit? It exists somewhere, though we can't find it, and we can reason about it. If we remove any one edge from the optimal circuit O, we get a path, which is among other things a connected acyclic graph, *a spanning tree*, whose total weight is at most $w(O)$. Since T is a minimum spanning tree, we thus know that $w(T) \leq w(O)$. Since $w(A) \leq 2w(T)$, we know that our approximation is within a factor of 2.

A simple improvement of this algorithm, due to Christofides in 1976, gets a $3/2$ approximation rather than 2. Again, let T be a minimum spanning tree of the weighted connected graph H. Let D be the set of nodes that have odd degree in T – there may be as few as 2 or as many as n, but there must be an even number (why?). Let M be a minimum-weight matching on D, and note that $T \cup M$ is now a connected undirected graph with even degree at every vertex. (Actually, $T \cup M$ is a *multigraph* in general, because there is nothing stopping an edge from being in both T and M – in this case we keep two parallel copies of the edge in $T \cup M$).

To get our approximation A, we take $T \cup M$ and (if necessary) apply shortcuts to bypass vertices that occur more than once, as long as we can. This gets us to a Hamilton circuit, with n nodes and n edges.

How good an approximation is A to the optimal TSP tour O? As before, we know that $w(T)$ is at most $w(O)$, because deleting an edge from O gives a spanning tree. What about M? M is a minimal matching on D, an even-size set. Consider the optimal TSP tour on D. Using the triangle inequality, it's easy to show that this tour has weight at most $w(O)$. We can break this tour into sets odd-numbered and even-numbered edges, and each of these sets is a *matching* on D. One of them must have weight at most $w(O)/2$, and thus the *minimum* weight matching on D, M, has at most this weight.

This means that the weight of $T \cup M$ is at most $w(T) + w(M) \leq w(O) + w(O)/2 = 3w(O)/2$. As before, the weight of A can be no greater than this, so A is at worst a 3/2 approximation to the minimum weight Hamilton circuit in H.

Can this be improved farther? Checking the archive again, we find no indication that it has been. The MIN-MTSP problem is listed as being "APX-complete", which means that there exists some positive constant ϵ such that it is **NP-**hard to approximate it within $1 + \epsilon$. There is reference to an explicit ϵ for which this has been proved, but since no one seems to want to talk about its specific value it would seem to be very small.

When people think about the TSP, and when they construct test instances of it, they often consider the **Euclidean** TSP problem, where the nodes are points in the Euclidean plane and the weight of an edge is the Euclidean distance between the two nodes. It turns out that this version of the TSP decision problem is still **NP-**hard (though technicalities arise when trying to prove it **NP-**complete, involving adding square roots).

For Euclidean TSP, there is a **polynomial time approximation scheme** due to Arora. For any positive constant ϵ, there is a polynomial-time algorithm f_ϵ that gets a $1+\epsilon$ approximation. However, the running time of f_ϵ depends *exponentially* on $1/\epsilon$, so this is not a **fully polynomial time approximation scheme**.