One way to get around the fact that we don’t have efficient algorithms for NP-complete problems is to use heuristics. Heuristics can be useful in practice, but sometimes we would like to have guarantees. Approximation algorithms give guarantees. It is worth keeping in mind that sometimes approximation algorithms do not always perform as well as heuristic-based algorithms. Other times they provide insight into the problem, so they can help determine good heuristics.

Often when we talk about an approximation algorithm, we give an approximation ratio. The approximation ratio gives the ratio between our solution and the actual solution. The goal is to obtain an approximation ratio as close to 1 as possible. If the problem involves a minimization, the approximation ratio will be greater than 1; if it involves a maximization, the approximation ratio will be less than 1.

**Vertex Cover Approximations**

In the Vertex Cover problem, we wish to find a set of vertices of minimal size such that every edge is adjacent to some vertex in the cover. That is, given an undirected graph \( G = (V, E) \), we wish to find \( U \subseteq V \) such that every edge \( e \in E \) has an endpoint in \( U \). We have seen that Vertex Cover is NP-complete.

A natural greedy algorithm for Vertex Cover is to repeatedly choose a vertex with the highest degree, and put it into the cover. When we put the vertex in the cover, we remove the vertex and all its adjacent edges from the graph, and continue. Unfortunately, in this case the greedy algorithm gives us a rather poor approximation, as can be seen with the following example:

![Figure 18.1: A bad greedy example.](image)

In the example, all edges are connected to the base level which has \( m \) vertices; there are \( m/2 \) vertices at the next level, \( m/3 \) vertices at the next level, and so on. Each vertex at the base level is connected to one vertex at each other
level, and the connections are spread as evenly as possible at each level. A greedy algorithm could always choose a rightmost vertex, whereas the optimal cover consists of the leftmost vertices. This example shows that, in general, the greedy approach could be off by a factor of $\Omega(\log n)$, where $n$ is the number of vertices.

A better algorithm for vertex cover is the following: repeatedly choose an edge, and throw both of its endpoints into the cover. Throw the vertices and its adjacent edges out of the graph, and continue.

It is easy to show that this second algorithm uses at most twice as many vertices as the optimal vertex cover. This is because each edge that gets chosen during the course of the algorithm must have one of its endpoints in the cover; hence we have merely always thrown two vertices in where we might have gotten away with throwing in 1.

Somewhat surprisingly, this simple algorithm is still the best known approximation algorithm for the vertex cover problem. That is, no algorithm has been proven to do better than within a factor of 2.

**Maximum Cut Approximation**

We will provide both a randomized and a deterministic approximation algorithm for the MAX CUT problem. The MAX CUT problem is to divide the vertices in a graph into two disjoint sets so that the numbers of edges between vertices in different sets is maximized. This problem is NP-hard. Notice that the MIN CUT problem can be solved in polynomial time by using a max flow algorithm as discussed in an earlier lecture.

The randomized version of the algorithm is as follows: we divide the vertices into two sets, HEADS and TAILS. We decide where each vertex goes by flipping a (fair) coin.

What is the probability an edge crosses between the sets of the cut? This will happen only if its two endpoints lie on different sides, which happens 1/2 of the time. (There are 4 possibilities for the two endpoints – HH,HT,TT,TH – and two of these put the vertices on different sides.) So, on average, we expect 1/2 the edges in the graph to cross the cut. Since the most we could have is for all the edges to cross the cut, this random assignment will, on average, be within a factor of 2 of optimal.

We now examine a deterministic algorithm with the same “approximation ratio”. (In fact, the two algorithms are intrinsically related– but this is not so easy to see!) We will split the vertices into sets $S_1$ and $S_2$. Start with all vertices on one side of the cut. Now, if you can switch a vertex to a different side so that it increases the number of edges across the cut, do so. Repeat this action until the cut can no longer be improved by this simple switch.

We switch vertices at most $|E|$ times (since each time, the number of edges across the cut increases). Moreover, when the process finishes we are within a factor of 2 of the optimal, as we shall now show. In fact, when the process
finishes, at least $|E|/2$ edges lie in the cut.

We can count the edges in the cut in the following way: consider any vertex $v \in S_1$. For every vertex $w$ in $S_2$ that it is connected to by an edge, we add $1/2$ to a running sum. We do the same for each vertex in $S_2$. Note that each edge crossing the cut contributes 1 to the sum—$1/2$ for each vertex of the edge.

Hence the cut $C$ satisfies

$$C = \frac{1}{2} \left( \sum_{v \in S_1} |\{w: (v, w) \in E, w \in S_2\}| + \sum_{v \in S_2} |\{w: (v, w) \in E, w \in S_1\}| \right).$$

Using this algorithm, at least half the edges from any vertex $v$ must lie in the set opposite from $v$; otherwise, we could switch what side vertex $v$ is on, and improve the cut! Hence, if vertex $v$ has degree $\delta(v)$, then

$$C \geq \frac{1}{2} \left( \sum_{v \in S_1} \frac{\delta(v)}{2} + \sum_{v \in S_2} \frac{\delta(v)}{2} \right)$$

$$= \frac{1}{4} \sum_{v \in V} \delta(v)$$

$$= \frac{1}{2} |E|,$$

where the last equality follows from the fact that if we sum the degree of all vertices, we obtain twice the number of edges, since we have counted each edge twice.

**Euclidean Travelling Salesperson Problem**

In the Euclidean Travelling Salesman Problem, we are given $n$ points (cities) in the $x - y$ plane, and we seek the tour (cycle) of minimum length that travels through all the cities. This problem is NP-hard (we won’t show this in class). The decision problem of whether there is a tour of total length at most $k$, however, is not known to be in NP! The reason is that the natural certificate would of course be the tour itself, but to compute the length of the tour and compare it to $k$, we would have to be able to compare the sum of some number of square roots (lengths along the tour) to $k$. However, it is unknown that this can be done in polynomial precision! \(^1\)

Our approximation algorithm involves the following steps:

1. Find a minimum spanning tree $T$ for the points.

\(^1\)See for example the open problem at [http://cs.smith.edu/~orourke/TOPP/P33.html](http://cs.smith.edu/~orourke/TOPP/P33.html). If copy/pasting the URL, you may need to manually fix the tilde before “orourke”.
2. Create a *pseudo tour* by walking around the tree. The pseudo tour may visit some vertices twice.

3. Remove repeats from the tour by *short-cutting* through the repeated vertices. (See Figure 18.2.)

![Figure 18.2: Building an approximate tour. Start at X, move in the direction shown, short-cutting repeated vertices.](image)

We now show the following inequalities:

\[
\text{length of tour} \leq \text{length of pseudo tour} \\
\leq 2(\text{size of T}) \\
\leq 2(\text{length of optimal tour})
\]

Short-cutting edges can only decrease the length of the tour, so the tour given by the algorithm is at most the length of the pseudo tour. The length of our pseudo tour is at most twice the size of the spanning tree, since this pseudo tour consists of walking through each edge of the tree at most twice. Finally, the length of the optimal tour is at least the size of the minimum spanning tree, since any tour contains a spanning tree (plus an edge!).

Using a similar idea, one can come up with an approximation algorithm that returns a tour that is within a factor of 3/2 of the optimal. Also, note that this algorithm will work in any setting where short-cutting is effective. More specifically, it will work for any instance of the travelling salesperson problem that satisfies the *triangle inequality* for distances: that is, if \(d(x,y)\) represents the distance between vertices \(x\) and \(y\), and \(d(x,z) \leq d(x,y) + d(y,z)\) for all \(x,y\) and \(z\).