We begin today with the problem we didn't get to at the end of last lecture – the SUBSET-SUM problem, which we also saw back in Lecture 8. The input to SUBSET-SUM is a set of numbers $\{a_1, \ldots, a_n\}$ and a target number t, and we ask whether there is a subset of the numbers that add exactly to t. Using dynamic programming, we showed that we could decide this language in time that is polynomial in n and s, the sum of all the a_i.

Now we allow the numbers to get larger, so that they now might be n bits long. The problem is still in **NP**, because we can guess a subset by guessing a bitvector, add the numbers in the set, and verify that we get t. But it's no longer clear that we are in **P**, and in fact we will now see that the general problem is **NP**-complete.

We reduce 3-SAT to SUBSET-SUM (with large numbers). We first assume that every clause in our input formula has exactly three literals – we can just repeat literals in the same clause to make this true. Our numbers will be represented in decimal notation, with a column for each of the v variables and a column for each clause in the formula.

We'll create an item a_i for each of the $2v$ literals. This item will have a 1 in the column for its variable, a 1 in the column of each clause where the literal appears, and zeroes everywhere else. We also have two items for each clause, each with a 1 in the column for that clause and zeroes everywhere else. The target number has a 1 for each variable column and a 3 for each clause column.

We now have to prove that there is a subset summing to the target iff the formula is satisfiable. If there is a satisfying assignment, we choose the item for each literal in that assignment. This has one 1 in each variable column, and somewhere from one to three 1's in each clause column. Using extra items as needed, we can reach the target.

Conversely, if we reach the target we *must* have chosen one item with a 1 in each variable column, so we have picked v variables forming an assignment. Since we have three 1's in each clause column and at most two came from the extra items, we must have at least one 1 in each clause column from our assignment, making it a satisfying assignment.

Given a problem with numerical parameters, we say that it is **pseudopolynomial** if it becomes polynomial when those parameters are given in unary. If it is **NP-**complete with parameters given in unary, we say that it is **strongly NP-complete**. The SUBSET-SUM problem is pseudopolynomial, but all our graph problems are strongly **NP-**complete.

Recall that the KNAPSACK problem is similar to SUBSET-SUM but has a **value** for each item as well as its **weight**. We are asked to find whether a set of at least a given value exists with at most a given weight. Since SUBSET-SUM is an identifiable special case of KNAPSACK (where weight and value are both equal), we know that SUBSET-SUM $\leq_p$ KNAPSACK. Since KNAPSACK (as a decision problem) is in **NP**, it is **NP-**complete. The associated optimization problem is thus **NP-**hard.

Earlier we looked at the problem of taking an undirected graph and finding the **minimum cut** in it – a partition of the vertices into two sets such that the number of edges with one endpoint in each set was as small as possible. We solved this in polynomial time in two different ways – using the network flow algorithm to find the minimum cut between one vertex s and each other vertex t, then taking the minimum over all t, and Karger's randomized algorithm.

What about the problem of finding the **maximum cut**? For example, each edge might represent a difference between two objects, and we might want to divide the objects in two so as to maximize the number of differnences. Despite the obvious similarity to the min-cut problem, finding the maximum cut (as an optimization problem) is **NP-**hard. In particular, the language MAX-CUT, the set of all pairs (G, k) where G has a cut of size *at least* k, is **NP-**complete.

Once again it is clear that this decision problem is in **NP**. We can guess a partition (as a bitvector of length n), count the edges between the two sets, and accept if the number is at least k. To prove MAX-CUT is **NP**-complete, we will reduce from NAE-3-SAT, one of the problems we proved to be **NP**-complete last lecture.

Remember that NAE-3-SAT is the set of 3-CNF formulas φ for which there exists a setting of the variables giving each clause *either one or two* true literals, not either zero or four. (The initials stand for "not all equal 3-SAT".) So given such a φ, we need to produce a graph G and a number k such that G has a cut of size k iff φ is NAE-satisfiable.

Before we begin, we make two modifications to φ if necessary:

- If any clause contains both a literal ℓ and its negation $\neg\ell$, we delete it – it was guaranteed to be satisfied anyway.
- If any two clauses share two of there three literals, we replace them by four clauses, introducing two new variables, that don't have this property and are NAE-satisfiable iff the original two clauses were. In particular, if $(a \vee b \vee c)$ and $(a \vee b \vee d)$ are both clauses, we replace them by $(a \vee b \vee c)$, $(a \vee x \vee d)$, $(b \vee \neg x \vee y)$, and $(\neg b \vee x \vee y)$. Whichever way y is set, NAE-satisfying the last two clauses forces x to be set the same way as b.

Now we can construct our graph. Say that φ, as modified, has v variables and m clauses. G will have $2v$ vertices, one for each literal. It will have $v + 3m$ edges:

- An edge between ℓ and $\neg\ell$ for each literal ℓ (v edges), and
- A triangle of edges connecting the three literals of each clause.

Our conditions on φ mean that no edge is added both ways, and no two of the triangles share an edge. Finally, we set k to be $v + 2m$.

We must prove that φ is NAE-satisfiable iff G has a cut of size at least k. First assume that there is a setting that NAE-satisfies φ. We look at the cut that separates the true literals of this setting from the false literals. Every one of the v negation edges goes across this cut. Every triangle in G consists of either two true literals and one false one, or two false and one true. Thus exactly two of the three edges of the triangle cross the cut. So the number of edges crossing the cut is exactly $v + 2m$, or k.

Now assume that we have a partition of the $2v$ vertices into two sets such that there are k edges across the cut. We must prove that there is a setting of the variables that NAE-satisfies all the clauses. Look at the two categories of edges:

- The only way that all of the v negation edges cross the cut is if each pair of literals is divided between the two sets, so that each set has v literals from different variables and thus represents a setting of the variables.
- Each of the m triangles has either zero edges crossing the cut, if all three nodes are in the same set, or two edges crossing it, if the three nodes are split two and one.

Thus the maximum possible number of edges crossing the cut is $v+2m$, and this is achieved only if the cut represents a setting, and this setting contains either one or two literals from each clause. This is exactly the definition of NAE-satisfying φ. (Note that if a setting NAE-satisfies φ, so does the setting obtained by negating all its literals.) We have completed the reduction and shown that NAE-3-SAT $\leq_p$ MAX-CUT and therefore that MAX-CUT is **NP**-complete.

On HW#4 you will consider the related MAX-BISECTION and MIN-BISECTION problems, which are similar to MAX-CUT and MIN-CUT except that in the bisection problems the cut is required to have an equal number of vertices in each set.

We now turn to an important **NP-**complete problem, **THREE-DIMENSIONAL-MATCHING (3DM)**. This problem is useful as a base for reductions, is an identifiable special case of a number of other problems, and has an interesting **NP-**completeness proof.

Let X, Y, and Z be disjoint sets of n elements each, and let T be a set of triples in $X \times Y \times Z$. The 3DM problem is the set of tuples (X, Y, Z, T) such that there is a set of n triples in T that together include each item in X, Y, and Z exactly once.

The 3DM problem generalizes the bipartite perfect matching problem, which we know to be in **P**. There we have two sets X and Y of size n and a set of edges that can be viewed as a subset of $X \times Y$, and we want a set of n edges that together include every vertex.

Clearly 3DM is in **NP**. We will prove it to be **NP-**complete by reducing 3-SAT to it.

Let φ be a 3-CNF formula with m clauses and v variables. We'll set up a set S of $6mv$ elements and a bunch of three-element subsets of this set, such that we can make S the union of exactly $2mv$ 3-sets iff φ is satsifiable. This is actually an instance of the related EXACT-COVER-BY-3-SETS problem – later we'll divide S into X, Y, and Z and make our 3-sets into triples.

The elements of S are as follows:

- $4m$ elements for each variable, a **core** of $2mv$ elements and $2mv$ others called **tips**. As I'll show on the board, we'll think of the core as arranged in a circle, with the tips in another circle outside it.
- A pair of **clause elements** for each clause, $2m$ in all.
- $m(v-1)$ pairs of **cleanup elements**, making $2m(v-1)$ in all.

Now to define the 3-sets in T:

- Two sets of m 3-sets for each variable. One set covers the core elements and the *odd-numbered* tips, the other the core elements and the *even-numbered tips*. There is no way to cover the whole core without taking either one set or the other. Covering the odds will correspond to setting that variable true, covering the evens to setting it false.
- For each clause $C_i = (\ell_1 \vee \ell_2 \vee \ell_3)$, we have three 3-set, each one covering the two clause elements and a tip corresponding to one of the ℓ's. If ℓ_j is the variable x_k, for example, the corresponding tip is the $2i$'th tip for variable x_k. If ℓ_j is $\neg x_k$, the corresponding tip is the $2i + 1$'st.
- For each cleanup clause and each tip, we have a 3-set containing those three elements.

We'll sketch the proof that a successful cover of all the elements by 3-sets in T corresponds to an assignment that satisfies φ. Suppose we have the asignment. We choose sets to cover the core elements for each variable as described above, leaving *either* all the odd tips or all the even tips uncovered. For each clause, we pick a literal that is assigned to be true and choose the 3-set that covers the two clause elements and the corresponding tip (which is uncovered because of the assignment). Then we cover the other tips and the cleanup elements with the 3-sets that include the cleanup elements.

Conversely, suppose we have a successful covering. To cover all the core elements, we must in effect pick an assignment to the variables. To cover the clause elements, we must also cover n tips, in such a way that the tips chosen correspond to literals that are set to true. This gives us a satisfying assignment. The clause and core elements can be covered in no other way, so the assignment must exist if the covering does.

Finally, we split S into X, Y, and Z to make these 3-sets into triples. Z is exactly the set of tips. The clause element and cleanup element pairs each have one element of X and one of Y. The core elements are alternatively from X and from Y, as I'll show on the board.

Here are some of the related problems that can be shown to be **NP**-complete by reduction from 3DM:

- SET-COVER: Given S and a collection of subsets of S, and a number k, is there a collection of k of the subsets that union together to give S?
- SET-PACKING: Given S, a collection of subsets, and k, is there a pairwise disjoint collection of k of the subsets?
- EXACT-COVER-BY-3-SETS: As above.
- PARTITION-INTO-TRIANGLES: Given an undirected graph with $3n$ vertices, is there a set of n disjoint triangles in the graph that include all the vertices?
- PARTITION-INTO-2-PATHS: Given an undirected graph with $3n$ vertices, are there n vertex disjoint paths of length 2 that together include all the vertices?
- STEINER-TREE: Given a weighted graph, a division of the vertices into "required" and "optional", and a target cost, is there a tree that includes all the required vertices and has total cost less than the target?

Clearly all these problems are in **NP**. For many of them problems, 3DM is an identifiable special case and so the reduction just uses the identity map. In the other cases, except for STEINER-TREE, the reduction is simple.

STEINER-TREE is an extension of MST – MST is the identifiable special case where there are no optional vertices. The Steiner tree problem is extensively studied as a geometric problem, where the edge weights are distances in the plane. It is also **NP-**complete in the geometric version, with either Euclidean or Manhattan distances.

We can arrange known **NP**-complete functions in a tree, based on the reductions used to prove them **NP**-complete. At the end of Chapter 7, the Adler notes have such a tree (unreadable in the original version of the notes). We'll repeat it here and discuss some of the problems:

- 3-SAT $\leq_p$ NAE-3-SAT $\leq_p$ MAX-CUT $\leq_p$ MIN-BISECTION $\leq_p$ MAX-BISECTION
- 3-SAT$\leq_p$ CLIQUE $\leq_p$ IND-SET $\leq_p$ VERTEX-COVER $\leq_p$ HAM-CYCLE $\leq_p$ LONGEST-PATH
- HAM-CYCLE $\leq_p$ TSP
- 3-SAT $\leq_p$ SUBSET-SUM $\leq_p$ KNAPSACK
- 3-SAT $\leq_p$ 3DM $\leq_p$ STEINER-TREE

Of course there *exists* a reduction from any **NP**-complete problem to any other, so a tree like this only represents a choice as to which reductions are easier.

In the rest of this lecture, we'll look at some of the problems on this list. In many cases, we have pairs of similar problems, one of which is in **P** and the other of which is **NP**-complete.

We've mentioned MAX-CUT (**NP**-complete) and MIN-CUT (in **P**).

The HAM-CYCLE problem asks whether there is a cycle that contains all the vertices. This contrasts with EULER-CYCLE, where the cycle (in a multigraph) must contain all edges. The latter is in **P**.

The problem of finding a LONGEST-PATH from s to t in an undirected graph is easily proved to be **NP**-complete by reduction from HAM-CYCLE. By contrast, we can find a SHORTEST-PATH in **P** by breadth-first or matrix powering.

The famous TRAVELING-SALESPERSON (TSP) problem is easily proved **NP**-complete (or **NP**-hard as an optimization problem) by reduction from HAM-CYCLE.

Sometimes the parameter of a problem is very important. While 3-SAT and 3DM are **NP**-complete, we have seen that 2DM is in **P** and 2-SAT is easily proved so.

EDGE-COVER, where we ask for a minimum-sized set of edges to cover every vertex, is in **P**, in contrast with VERTEX-COVER. Finally, MST generalizes to the **NP**-complete STEINER-TREE problem.