CSE 347 Analysis of Algorithms March 4, 2019

Lecture 13 Summary

Instructor: Brendan Juba

1. 3SAT is NP-complete (and therefore so are Independent Set and Vertex Cover)
   (a) We reduce from Circuit SAT, the only NP-complete problem we have so far. By our lemma from last time, this is enough.
   (b) We express the circuit as a set of constraints on the values the wires can take.
   (c) If each gate has two wires entering and one wire exiting, we can write these constraints as clauses on at most three literals each. We also add a clause indicating that the output wire must carry value 1.
   (d) The values these wires take on an input that satisfies the circuit are a satisfying assignment to the 3CNF, and conversely, any satisfying assignment to the 3CNF must place values on the wires that are then consistent with the computation done at each gate. So, the reduction is correct
   (e) This formula is easy to compute in linear time.
2. To show NP-completeness of a new problem, it's useful to reduce from an NP-complete problem that "looks similar to" the problem we want to show to be hard. (See the example of Vertex Cover and Independent Set.) So we will try to expand the variety of NP-complete problems available to us.
3. Subset Sum and Knapsack are NP-complete (problems with numbers)
   (a) Subset sum is the problem: given a set of positive integers $w_1, \ldots, w_n$ and a target W, is there a subset of the integers that sums to W? (Note that this is a special case of Knapsack where weights and values are the same.)
   (b) Subset sum is in NP because we can take a vector indicating the subset, and sum up those numbers and check if it equals W. Such a vector exists if and only if there is a subset that sums to W.
   (c) We now reduce 3SAT to Subset Sum, which is enough by our lemma.
      i. We use $n+m$ digit numbers if there are n variables and m clauses: the first n digits track which variables are assigned, and the final m variables check that each clause has a variable setting that satisfies it.
      ii. We create a number for each possible assignment of each variable in the formula: it has a 1 in the position corresponding to the variable being assigned, and a 1 in the positions for each clause that variable setting satisfies, and 0s everywhere else.
      iii. We now add $|C_j| - 1$ numbers for the clause C_j that are 1 in the position for the jth clause and 0 everywhere else.
      iv. The target W has 1 in the first n positions, and $|C_j|$ in the position corresponding to clause C_j.

v. Note that there are no carries in the sum of any subset. So we sum to W only if we include one number for each ith variable. Moreover, since we only have $|C_j| - 1$ "padding numbers" for the jth clause, to hit $|C_j|$, some variable must satisfy the jth clause.

vi. But conversely, the subset of numbers corresponding to a satisfying assignment together with the appropriate number of these padding numbers indeed hits W.

4. NP-completeness is usually not permission to give up, but it tells us that we need to lower our expectations for the solutions we find

   (a) We can relax the various criteria we demanded of our algorithms in the first part of the course to yield different kinds of algorithms.

   (b) We can give up on worst-case polynomial running time and try to just obtain a polynomial *average-case* running time.

      i. Actually, this turns out to not be very easy to use to solve NP-complete problems in practice

      ii. But, the ideas will be useful to us to obtain faster and simpler *randomized algorithms*

   (c) We can give up on optimal solutions, and obtain *approximate* solutions (later in the course).