CMPSCI611: **Lower Bounds on Approximation** Lecture 22

We can now divide the approximation problems we've seen into a number of categories:

- Problems with an FPTAS, such as KNAPSACK
- Problems with a PTAS but not a known FPTAS, such as EUCLIDEAN-TSP
- Problems aproximable to within one constant but NP-hard to approximate within another, such as VERTEX-COVER
- Problems approximable to within $O(\log n)$ ($\Theta(\log n)$ unless $\mathbf{P} = \mathbf{NP}$), such as SET-COVER
- Problems NP-hard to approximate even within some polynomial factor (even n^{ϵ})

It is natural to try to use **reductions** to classify approximation problems into these or other classes. But the normal Karp reductions we used to define **NP**-completeness are not suitable here, because they don't respect numerical values for the score of an optimization problem. A more sensible reduction is one that maps settings to settings in a way that preserves validity and preserves scores.

Consider our reduction from 3-SAT to 3-CLIQUE, where we took a 3-CNF formula and constructed a graph where a clique represented a set of literals that were mutually compatible and occurred in different clauses of the formula. This mapping from formulas to graphs forms a transformation from the MAX-3-SAT problem to the MAX-CLIQUE problem, in which there exists a setting satisfying k or more clauses if and only if there exists a clique of size k in the graph.

Using this reduction, we know that if we have a poly-time algorithm to approximate MAX-CLIQUE to within any given factor, we have a poly-time algorithm to approximate MAX-3-SAT within the same factor. But this works only one way – later in this lecture we will give an 8/7 approximation for MAX-3-SAT, but it is **NP-**hard to approximate MAX-CLIQUE to within any constant.

It's more useful to have reductions that can operate where there is not an exact correspondence between scores. One theory defines the class **APX** of optimization problems to be those that have any constant-factor poly-time approximation algorithm (the first three classes in our hierarchy). We can define an **AP-**reduction to be a pair of mappings between two problems P_1 and P_2. The details are a bit messy, but P_2 is in **APX** and the reduction exists, we can take an instance of P_1, map it to an instance of P_2, get an approximate solution, and map this solution back to get an approximate solution to the P_1 instance.

Since an **AP-**reduction is parametrized by how much worse the P_1 approximation may be than the P_2 approximation, it is possible to find optimization problems that are **APX-**complete – in **APX** and having a **AP-**reduction from every problem in **APX**. If one of these problems had a PTAS, then the reductions would give PTAS's for every problem in **APX**. MAX-3-SAT is an example of an **APX-**complete problem.

This theory was useful primarily *before* the early 1990's, when it was proved directly that MAX-3-SAT is **NP-**hard to approximate within a particular constant. This result follows from the **PCP theorem**, which we discuss briefly in CMPSCI 601 and basically not at all here. But we'll look at some of its implications.

The PCP Theorem gives us the **NP**-hardness of a particular **gap problem**. Given a maximization problem where the scores range from 0 to x, and given two real numbers α and β with $0 \leq \alpha < \beta \leq 1$, the (α, β) gap problem is to distinguish inputs where the maximum possible score is less than or equal to αx from inputs where it is greater than or equal to βx.

From the PCP Theorem, one can prove that the $(1 - \epsilon, 1)$ gap problem for MAX-3-SAT is **NP**-hard, where ϵ is a particular small positive number. ($1/36864$, by the proof in *Algorithmics for Hard Problems* by Hromkovic.) This implies that MAX-3-SAT cannot have a PTAS unless $\mathbf{P} = \mathbf{NP}$. Later work has shown that the best achievable ratio is $8/7$ (the approximation satisfies at least $7/8$ as many clauses as the optimum).

One then defines **gap reductions**, which are simply functions from instances of one problem to instances of another that preserve particular gaps. With these reductions it's possible to prove various other gap problems **NP**-hard and thus prove that it is **NP**-hard to achieve particular approximation ratios.

Even without the PCP Theorem, we can show that "most" problems do not have an FPTAS. Remember that a decision problem is **strongly NP-complete** if it remains **NP**-complete when its integer inputs are given in unary. An optimization problem is **strongly NP-hard** if the decision problem of whether a given score is achievable is strongly **NP**-complete.

Suppose we have a strongly **NP**-hard optimization problem where the scores are integers that are polynomial in the input size n and the largest number in the input. We'll show that if the problem has an FPTAS, then $\mathbf{P} = \mathbf{NP}$.

(Assume we have a maximization problem.) We need to show that the hypothetical FPTAS allows us to solve the decision problem in polynomial time, given the inputs in unary. So we have an input size n, our inputs are polynomial in n, and the scores are polynomial in n. If $U(I)$ is the maximum possible score, run the FPTAS with ϵ equal to $1/(U(I)+1)$. The error in determining the maximum possible score is at most $\epsilon U(I) < 1$, so the integer value returned by the FPTAS is the exact correct answer. Since $1/\epsilon$ is polynomial in the input size, so is the running time of the FPTAS.

We now turn to what we *can* accomplish in approximating MAX-3-SAT. Remember that our goal is to find a setting that satisfies as many clauses of a 3-CNF formula as possible. To be precise, our formulas will have *exactly* three literals per clause, and aren't allowed to repeat a literal.

A *random* assignment to the n variables ought to do pretty well. If the three literals in a clause are each independently set true or false, there is a $7/8$ chance that the clause will be satisfied. Because expected values add, the expected number of satisfied clauses is m (the number of clauses) times $7/8$. And a setting that does this well *must* be an $8/7$ approximation to the optimum, because the optimum can't satisfy any more than m clauses as that's all there are.

This suggests a sort of Las Vegas algorithm for the approximation – keep choosing random settings until you get one that satisfies at least $7m/8$ clauses. With enough trials, the probability of failure should get small (or should it?). In any case we'd like a *deterministic* approximation algorithm.

We'll give two deterministic versions of the randomized algorithm. The first is similar to the "self-reducibility" problems on HW#4. If we have any 3-CNF formula, even one with clauses of fewer than three literals, and 0 and 1 constants, we can compute the expected number of clauses satisfied by a uniform random assignment. We add up 1 for each 1-clause, $1/2$ for each one-literal clause, $3/4$ for each two-literal clause, and $7/8$ for each three-literal clause.

If we set some of the variables in our original formula, we may satisfy some clauses, make some unsatisfiable, and shorten others. But for any given setting we can determine the expected number of clauses satisfied by a uniform random setting of the *other* variables. This gives us a deterministic algorithm as follows.

We first look at setting x_1. We compute the expected number of satisfied clauses if we set x_1 to 0, and then the number if we set x_1 to 1. Since these two numbers average to $7m/8$, at least one of them is $\geq 7m/8$.

We pick that setting of x_1 and then look at the two ways to set x_2, with that setting of x_1. Again at least one of the options must be at least $7m/8$. In this way we set $x_3, x_4, \ldots$ all the way to x_n, each time either increasing the expected number of clauses we will satisfy or keeping it the same. At the end the expected value is the number of clauses we *have* satisfied with the setting we have picked, and this must be at least $7m/8$.

The other method of derandomizing uses a **small sample space**. When we computed the expected number of satisfied clauses, we didn't really need the fact that the settings of the n variables were *independent* random variables. What we needed was that the three variables in each clause were independent, so that the probability of satisfying the clause would be $7/8$. As long as our n random variables are **3-wise independent**, then, the expected number satisfied is still $7m/8$.

We'll show that we can find a set of n^3 particular settings of the n variables that is generic in the following sense – if we pick one of these settings uniformly at random, the probability of each variable being set true is exactly $1/2$ and the settings of the variables are 3-wise independent. Then we can be sure that one of these settings satisfies at least $7m/8$ clauses, since it can't be that all the settings score below average.

To get these n^3 settings we need a bit of abstract algebra. By adding dummy variables, let n be a power of two and let F be a **finite field** of order n. Number the variables with the elements of F and arbitrarily pick a set T of $n/2$ field elements.

If a, b, and c are any three elements of F, not necessarily distinct, we define the setting S_{abc} as follows. For any field element i, S_{abc} sets x_i to be true iff the field element $ai^2 + bi + c$ is in the set T.

If a, b, and c are any three elements of F, not necessarily distinct, we define the setting S_{abc} as follows. For any field element i, S_{abc} sets x_i to be true iff the field element $ai^2 + bi + c$ is in the set T.

We need to show that for each i, exactly half of the settings set x_i to true. This is easy – if we fix a and b and vary c, $ai^2 + bi + c$ takes on every value in F, so it is in T for exactly half the possible c.

Beyond that, we have to show that the random variables are 3-wise independent. For any triple of variables x_i, x_j, and x_k, there are n^3 triples of values that the field values i, j, and k might be mapped to by the polynomial ax^2+bx+c. In a field, three function values are sufficient to determine the coefficients of a quadratic function. So the n^3 settings each take i, j, and k to a different triple of values, and exactly $7/8$ of the settings take them to a triple with at least one component in T.

As we mentioned, it can be proved from the PCP Theorem that it is **NP**-hard to approximate MAX-3-SAT to a factor better than $8/7$. You would think that the *best* of the n^3 settings in our small sample space ought to do better than average. Perhaps, but the theorem says it can't be an $8/7 - \epsilon$ approximation for any positive real number ϵ.

Our reduction from MAX-3-SAT to MAX-CLIQUE preserves the score exactly, so it follows immediately from the **NP**-hardness of approximating MAX-3-SAT to within $8/7$ that it is **NP**-hard to approximate MAX-CLIQUE to within $8/7$. But much more is known about CLIQUE. You'll show on HW#5 that if *any* constant-factor approximation exists for MAX-CLIQUE, then an FPTAS exists, which it doesn't unless $\mathbf{P} = \mathbf{NP}$.

With the PCP Theorem, and further work, it has been shown that it is **NP**-hard to approximate MAX-CLIQUE within a factor of $n^{\frac{1}{2}-\epsilon}$ for any positive constant ϵ. The stronger hypothesis $\mathbf{NP} \neq \mathbf{ZPP}$ is known to imply that a factor of $n^{1-\epsilon}$ is impossible. The best known approximation algorithm comes within $\frac{n}{\log^2 n}$. That is, there might be a clique of size almost n, and this algorithm would find one of size $\log^2 n$.

We showed earlier that SET-COVER can be approximated to within a factor of $\ln n$. It is known that it is **NP**-hard to approximate SET-COVER to within $c \log n$ for some small constant c, and that if $\mathbf{NP}$ is not contained in $\mathbf{DTIME}(n^{\log \log n})$, it can't be approximated to within $(1 - \epsilon) \ln n$.

We've observed that it is **NP**-hard to approximate the minimization problem GRAPH-COLOR to any factor smaller than $4/3$. Further work has shown that it is **NP**-hard to achieve a factor of $n^{1/7-\epsilon}$, and that if $\mathbf{NP} \neq \mathbf{ZPP}$ it is impossible to achieve a factor of $n^{1-\epsilon}$. The best known approximation factor is $O(\frac{n(\log \log n)^2}{(\log n)^3})$.

Telling whether a **planar** graph is 3-colorable is **NP**-complete, even though it must be 4-colorable. To 4-color a 3-colorable graph is an **NP**-hard problem. As far as my casual web search indicates, it's not known whether it is **NP**-hard to 5-color a 3-colorable graph. On HW#5 you'll work through a proof that you can color a 3-colorable graph with $O(\sqrt{n})$ colors.