CS 142 & Math 167
Fall 2008
Homework 4
Due Wednesday, October 1

1. **[25 Points] Finishing Up the Bin Packing Approximation Scheme!** In class we examined a polynomial time approximation scheme for Bin Packing. We showed that if all elements have size at least ϵ then in polynomial time we can find solutions which are at most $1 + \epsilon$ times optimal.

   Now we turn to those elements of size less than ϵ. We try to place them in the existing bins using the First Fit heuristic. If they all fit in the existing bins, we're done.

   The remaining case is that there are some items of size less than ϵ and they don't all fit in existing bins when using the First Fit heuristic. In this case, the First Fit heuristic was required to start some new bins. Your job is to show that in this case, the number of bins used by the algorithm is at most $(1 + 2\epsilon)OPT + 1$, assuming $\epsilon \leq 1/2$.

   Our analysis here will be entirely from scratch!

   (a) Let M be the number of bins used by this algorithm. Observe that with the exception of the last bin, all other bins are full to the extent $1 - \epsilon$ or more. Explain briefly why this is so.

   (b) Observe that the sum of the items sizes is at least $(M - 1)(1 - \epsilon)$. Briefly explain why.

   (c) Show the mathematical fact that if $\epsilon \leq \frac{1}{2}$ then

   $$\frac{1}{1 - \epsilon} \leq (1 + 2\epsilon)$$

   (d) Explain why the algorithm uses a number of bins at most $(1+2\epsilon)OPT+1$.

   (e) Finally, write out the ENTIRE algorithm (including what we did in class) in either English or pseudo-code.

2. **[25 Points] Taking 3SAT to the Max!** 3SAT is a decision problem. How can we turn it into an optimization problem? Here's how! MAX 3SAT takes an instance of 3SAT (a boolean expression in 3-CNF form) and asks for the valuation which maximizes the number of satisfied clauses.

Consider the following approximation algorithm for MAX 3SAT: Choose a literal (a "literal" is a variable or the negation of a variable) which appears the most number of times in the given expression (breaking ties arbitrarily). Make that literal true, thereby satisfying some clauses. Remove all of these satisfied clauses. In the remaining clauses, every occurrence of the negation of the literal is crossed-off, thereby reducing the number of "live" literals in some clauses. Repeat until there are no clauses with "live" literals remaining. Notice that in this way, the number of "live" literals in each clause starts off at 3 but can eventually drop to 0, at which point the clause is "dead" and thus unsatisfied!

(a) Prove that this is an approximation algorithm with ratio 3/4. That is, it satisfies at least 3/4 of the number of clauses satisfied by the optimal algorithm. (*Hint:* Recall that the first thing we normally do in such an analysis is to obtain a bound on the value of the optimal solution. In this case, use the obvious bound that the optimal solution can at most satisfy all of the clauses. This weak bound will suffice!)

(b) Briefly explain why this algorithm runs in polynomial time.

(c) Now generalize this algorithm for an approximation algorithm for MAX κ-SAT, the variant in which each clause contains exactly k literals. Your approximation ratio will depend on k and should approach 1 as k approaches infinity!

(d) You may be saying to yourself: "That's very cool, but we should be able to do even better!" You're right! We can do MUCH better. Consider the following algorithm for the MAX κ-SAT problem: Give each clause of the C clauses a weight 2^{-k}. Now, repeatedly choose some variable which has not been assigned a value yet. Let x be this variable. Let C_x be the set of clauses containing x and let $C_{\overline{x}}$ be the set of clauses containing $\overline{x}$. (These sets are disjoint, since otherwise the clauses in the intersection are trivially satisfiable.) If the total weight of the clauses in C_x is at least as large as the total weight of the clauses in $C_{\overline{x}}$, then make x true. Otherwise make x false. In the former case, remove all clauses containing x and, for each clause containing $\overline{x}$, remove $\overline{x}$ from the clause and double that clause's weight. Do the analogous thing if x is made false. Prove that this algorithm is a polynomial-time approximation algorithm with approximation ratio of $\frac{2^k-1}{2^k}$. YOWZA! *Hint:* Keep track of the total weight of all clauses. What happens to the total weight after reach round of the algorithm? What is the weight of a clause when it "dies" ?

3. **[25 Points] The Amazing Relationship between FPTAS and Strong NP-completeness!** Recall that our FPTAS for the Knapsack Problem relied on the fact that Knapsack, although NP-complete, is not NP-complete in the strong sense. In this problem you'll prove something rather remarkable; in essence you'll show that a problem that is NP-complete in the strong sense cannot have a FPTAS! We'll do this in two parts.

   First, let Π be a number problem and let I denote an instance of the problem. As always, we let $n = |I|$. Let $\max(I)$ denote the value of the maximum number in this instance. Recall that we said that a problem is NP-complete in the strong sense if it is NP-complete even when $\max(I)$ is bounded by some polynomial in n. We also observed that if a problem has a pseudo-polynomial time algorithm then it is not NP-complete in the strong sense, assuming $P \neq NP$.

   (a) Let Π be a NP-complete minimization problem (this would work for maximization problems very similarly, but you don't need to worry about that here) and let p be a polynomial. Assume that Π is such that on an input instance I of length n, the optimal solution to the instance is a number which is upper-bounded by $p(\max(I))$. Prove that if Π has a FPTAS then it also has a pseudo-polynomial time algorithm.

   (b) Now show that if Π is NP-complete and satisfies the restrictions in part (a) above, then if Π is NP-complete in the strong sense then Π has no FPTAS, assuming that $P \neq NP$.

4. **[25 Points] Something Funky about the Clique Problem!** Recall that the Maximum Clique problem (written "CLIQUE") takes an undirected graph G as input and determines the size of the maximum clique (maximum fully connected subgraph) in G.

   (a) Prove that CLIQUE is NP-complete via a reduction from 3SAT. (You may have seen a reduction from Vertex Cover in another course, but you're asked here to describe a different reduction.) In the interest of brevity, describe your reduction very precisely but keep the "iff" part of the proof brief – one short paragraph for each direction should suffice.

   (b) Now we'll show something rather remarkable. Namely, we'll show that if there exists some absolute approximation algorithm for CLIQUE then there exists a PTAS for CLIQUE. We'll do this in two steps:

i. First, let $G = (V, E)$ be an undirected graph. Let G^2 denote the graph constructed as follows: For each vertex v of G we replace that vertex by an entire copy of G. Let G_u and G_v be two such copies of G, corresponding to vertices u and v of G. If $(u, v) \in E$ then every vertex of G_u is connected to every vertex of G_v. Prove that G has a clique of size k if and only if G^2 has a clique of size k^2.

ii. Use the fact that you proved above to show that if there exists an absolute approximation algorithm for CLIQUE with any positive constant ratio bound, then there is a PTAS for CLIQUE. Amazing, but true!