CSE347 Lecture 25

Approximation Algorithms III: Polynomial-Time Approximation Schemes

1. Polynomial-time approximation schemes (PTAS)
2. A PTAS for Knapsack

We've seen a variety of approximation algorithms, some of them strongly believed to achieve the best possible approximation ratios. And yet, even the very best of these were 2-approximations — *quite* dissapointing if one had hoped to pay, for example, only 5% more than an optimal solution. (From this perspective, we pay 100% more!) But, there are some rare examples where it is possible to achieve such strong approximations.

Generally speaking, if the problem is NP-hard and the objective function is integer-valued and of polynomially bounded digits, then if we are getting a $(1\pm\varepsilon)$-approximation, we <u>cannot</u> hope to have a running time that depends polynomially on the number of digits in ε. Why?

If the maximum value is M and the values are integers (suppose we have cleared denominators if we had rational-valued weights) then taking $\varepsilon = \frac{1}{2M}$ gives that a $(1\pm\varepsilon)$-approximate solution has (integer) value within $(1-\frac{1}{2M})OPT \geq OPT - \frac{1}{2}$ and $(1+\frac{1}{2M})OPT \leq OPT + \frac{1}{2}$. But, OPT is the only such integer, so we must find OPT. And, if the time depends polynomially on

$\log \frac{1}{\varepsilon}$, this is $\log M + 1$, which we assumed polynomial in the input size. Thus, we'd have a polynomial-time algorithm for the NP-complete optimization problem...

So, the best we can hope for is an algorithm that for any fixed $\varepsilon > 0$ runs in polynomial time, i.e., it may have worse-than-polynomial dependence on the length of ε, but must have a polynomial bound in terms of the size of the rest of the input, and gives a $(1+\varepsilon)$-approximation (or a $(1-\varepsilon)$-approximation). Such an algorithm is called a polynomial-time approximation scheme (PTAS).

Note the similarity between a PTAS and a pseudopolynomial time algorithm; the difference is mainly that a pseudopolynomial time algorithm allows an exponential dependence on the lengths of input numbers, whereas a PTAS allows such a dependence on the approximation parameter. But, as we'll see, pseudopolynomial time algorithms often can be converted into a PTAS by rounding the values and rescaling them into a smaller range, in which the pseudopolynomial-time algorithm is efficient (in terms of the input size). Our one example of a problem with pseudopolynomial time algorithms is Knapsack, so that will be our focus.

Actually, for Knapsack, the natural approximation formulation treats the weight bound as strict (you really

can't carry more weight than W) and seeks to approximate the optimum value we get within that bound. But our strategy is then to round the <u>values</u>, not the weights. So, we want a pseudopolynomial time algorithm that has exponential dependence on the <u>values</u> instead of the $O(n W)$ bound that we actually got. This is possible, but we'll defer discussing it to later. If $V = \max_{i=1,\dots,n} v_i$, then we'll get a $O(n^2 V)$-time algorithm.

So, supposing we have this $O(n^2 V)$-time algorithm, here is the idea: for some "scale" parameter b (that we will set depending on our approximation tolerance ε), we round each of our values to the nearest multiple of b: the value of each i^{th} item, v_i, is rounded to $\tilde{v}_i = \lceil \frac{v_i}{b} \rceil \cdot b$. Since $\frac{v_i}{b} \le \lceil \frac{v_i}{b} \rceil \le \frac{v_i}{b} + 1$, $v_i \le \tilde{v}_i \le v_i + b$.

Now, observe that for any set of items S, our rounded values satisfy $\sum_{i \in S} \tilde{v}_i = \sum_{i \in S} \lceil \frac{v_i}{b} \rceil \cdot b = b \cdot \sum_{i \in S} \lceil \frac{v_i}{b} \rceil$. So, we can re-scale all of our values by b, and solve an equivalent problem in which each i^{th} item has value $v_i' = \lceil \frac{v_i}{b} \rceil$. The value of S in this problem is $\sum_{i \in S} \lceil \frac{v_i}{b} \rceil$, so any knapsack S^* of greatest value using the rescaled values $v_1', \dots, v_n'$ is also a knapsack of greatest value using the rounded values $\tilde{v}_1, \dots, \tilde{v}_n$ and vice-versa. [Why? We can obtain the value of S for the other problem by multiplying or dividing by $b > 0$, and $x > y$ if and only if $bx > by$.)

Notice also that $v_1', \dots, v_n'$ are integers, so this is another in-

stance of the Knapsack problem. But, now our maximum value is $\lceil \frac{V}{b} \rceil$, which may be much smaller than V.

In particular, we assume without loss of generality that every item has weight $w_i \leq W$. (We can filter out such items in a preprocessing step since they cannot be included in any feasible solution.) We also assume $\frac{1}{\varepsilon}$ is an integer — just take $\varepsilon' = 1/\lceil \frac{1}{\varepsilon} \rceil < \varepsilon$.

Now, we take $b = \frac{\varepsilon}{2n} V$, and solve the corresponding Knapsack problem with values $v_1', \ldots, v_n'$, and use its solution. Observe: since $\max_i v_i' = \lceil \frac{V}{\frac{\varepsilon}{2n} V} \rceil = \lceil \frac{2n}{\varepsilon} \rceil$, our algorithm runs in time $O(n^3 \frac{1}{\varepsilon})$, which is $O(n^3)$ for any fixed $\varepsilon > 0$, as needed for a PTAS.

Moreover, since we use the same weights $w_1, \ldots, w_n$ and the returned set satisfies $\sum_{i \in S} w_i \leq W$, it is guaranteed to be a feasible solution to the original Knapsack problem.

We now come to the key issue: the approximation quality

Theorem: For any set S^* satisfying $\sum_{i \in S^*} w_i \leq W$, our returned solution S satisfies $(1+\varepsilon) \sum_{i \in S} v_i \geq \sum_{i \in S^*} v_i$.

Proof: Since S is an optimal solution to the problem with the rounded values and S^* is a feasible solution, $\sum_{i \in S} v_i' \geq \sum_{i \in S^*} v_i'$

Because the rounded values satisfy $v_i \leq b \cdot v_i' \leq v_i + b$,

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} b \cdot v_i' \leq \sum_{i \in S} b \cdot v_i' \leq \sum_{i \in S} (v_i + b) \leq n \cdot b + \sum_{i \in S} v_i \quad (|S| \leq n).$$

Now, since we assumed that each item could fit in the

knapsack (has $w_i \le W$), the item of maximum value V surely fits. Therefore, our optimum solution S has value at least $\lceil \frac{V}{b} \rceil b = \frac{2n}{\varepsilon} b$ by our choice of b. For the set consisting of just this item, we get $\sum_{i \in S} v_i + nb \ge \frac{2n}{\varepsilon} b$ by the above. Since $\frac{1}{\varepsilon}$ is an integer, $nb \le \frac{1}{\varepsilon} nb$, and thus $\sum_{i \in S} v_i \ge \frac{1}{\varepsilon} nb$. We therefore find, again by the above, $\sum_{i \in S'} v_i \le \sum_{i \in S} v_i + \varepsilon \sum_{i \in S} v_i$

$= (1+\varepsilon) \sum_{i \in S} v_i$ ▨

So, given the $O(n^2 V)$-time algorithm for Knapsack, we have our PTAS. We now return to this last detail. We can think of Knapsack as optimizing two criteria: we'd like to maximize our value and minimize the total weight. Of course, we had set a target bound on the weight of W. But, we could as well have asked: what is the smallest weight knapsack w^* that obtains some target value of at least v^*? Given solutions for various choices of v^*, we can find the largest v^* for which $w^* \le W$. Observe, then, any S of value larger than v^* has weight larger than W, so v^* is the desired optimum value of a feasible Knapsack solution (of weight at most W). It thus suffices to solve the problem parameterized by target value v^*, and hence similarly, subproblems parameterized by such a target value v^*. Notice, since each item has value at most V and there are n items, the largest v^* can be for any feasible subproblem is nV, so we only need to consider $v^* = 0, 1, \ldots, nV$.

In summary, our subproblems will be $OPT(i, v^*)$: the minimum weight w^* of a set $O \subseteq \{1, \dots, i\}$ of value $\geq v^*$.

What recurrence relation can we use for a dynamic programming algorithm? We first have $OPT(i, 0) = 0$ for all i since the empty O achieves this. For $v^* > 0$,

$$OPT(i, v^*) = \min\{\underbrace{OPT(i-1, v^*)}_{\text{if } i \notin O}, \underbrace{w_i + OPT(i-1, \max\{0, v^* - v_i\})}_{\text{if } i \in O}\}$$

... assuming there is such a solution of value v^*.

Note that if $\sum_{j=1}^{i-1} v_j < v^*$, then any feasible solution must contain i, so $OPT(i, v^*) = w_i + OPT(i-1, \max\{0, v^* - v_i\})$

Notice that this gives $n \cdot v^* \leq n^2 V$ subproblems.

Our algorithm is:

Initialize $M[0 \cdots n, 0 \cdots \sum_{i=1}^{n} v_i]$ with $M[i, 0] = 0 \ \forall i$, $M[0, v^*] = 0 \ \forall v^*$

For $i = 1, \dots, n$

  For $v^* = 1, \dots, \sum_{j=1}^{n} v_j$

    If $v^* > \sum_{j=1}^{i-1} v_j$ Put $M[i, v^*] \leftarrow w_i + M[i-1, \max\{0, v^* - v_i\}]$

    Else Put $M[i, v^*] = \min\{M[i-1, v^*], w_i + M[i-1, \max\{0, v^* - v_i\}]\}$

Return $\max\{v : M[n, v] \leq W\}$, M

We observe that this algorithm runs in time $O(n^2 V)$, and likewise, we can trace back through M starting from $[n, v^*]$ for the returned v^* to find S, also in time $O(n^2 V)$. Correctness follows by induction on i: for all v^*, if the subproblem (i, v^*) is feasible, $M[i, v^*]$ contains the smallest weight of $O \subseteq \{1, \dots, i\}$ of value $\geq v^*$. If $(i-1, v^*)$ is not feasible but (i, v^*) is, i must be in O, so

$OPT(i,v^*) = OPT(i-1, \max\{0, v^*-v_i\}) + w_i$. By IH, since the sub-problem must be feasible, $OPT(i,v^*) = w_i + M[i-1, \underbrace{\max\{0, v^*-v_i\}}_{= v^*-v_i > 0}]$

$= M[i, v^*]$ ✓

Likewise, if $(i-1, v^*)$ was also feasible, $OPT(i,v^*)$ is the minimum of $OPT(i-1,v^*)$ $(= M[i-1,v^*]$ by IH) and $w_i + OPT(i-1, \max\{0,v^*-v_i\})$ $(= M[i-1, \max\{0,v^*-v_i\}]$ also by IH again), so also $M[i,v^*] = OPT(i,v^*)$

Now, we observe that for $v^* = 0, \ldots, \sum_{i=1}^{n} v_i$, (n,v^*) is feasible since $\{1,\ldots,n\}$ achieves enough value, so these subproblems all contain $OPT(n,v^*)$. Also, $OPT(n,0) = 0 \leq W$, so we must return a solution.