COMPSCI 532: Design and Analysis of Algorithms October 28, 2015

Lecture 16

Lecturer: Debmalya Panigrahi *Scribe: Allen Xiao*

1 Overview

In this lecture, we introduce *strong* and *weak* NP-hardness, and the implications for approximation. In particular, we introduce the idea of *polynomial time approximation schemes* and give an example using Knapsack.

2 Polynomial Time Approximation Schemes

So far, we have seen problems with $\Omega(1)$ and $\Omega(\log n)$ approximations. There exist, however, problems which are hard to approximate within a polynomial factor ($\Omega(\text{poly}(n))$), like independent set and k-coloring. In this class, we will examine an approximation scheme which gives arbitrarily good approximations at the cost of runtime. For the following definitions, we will use n to represent the combinatorial size of an instance of an optimization problem.

Definition 1. *A* ***polynomial time approximation scheme*** *(PTAS) for an optimization problem is a set of algorithms such that, given $\varepsilon > 0$, it contains a $(1 \pm \varepsilon)$ approximation algorithm with running time polynomial in n when ε is fixed.*

Intuitively, a PTAS is a box which gives us polynomial approximation algorithms of arbitrary accuracy, but may take super-polynomial time in terms of $1/\varepsilon$. The vital language here is the "fixing" of ε before checking polynomial time. As an example, all of the following running times are acceptable for a PTAS:

- $O(n^2/\varepsilon)$
- $O(n^{100}2^{1/\varepsilon})$
- $O(n^{2^{2^{1/\varepsilon}}})$

In fact, in many cases PTAS are not used in practice because they exhibit super-polynomial dependence on $1/\varepsilon$. A stronger notion, where we require the algorithm to run in time polynomial in $1/\varepsilon$ as well, is more useful.

Definition 2. *A* ***fully polynomial time approximation scheme*** *(FPTAS) for a minimization (resp. maximization) optimization problem is a set of algorithms such that, given $\varepsilon > 0$, it contains a $(1+\varepsilon)$ (resp. $((1-\varepsilon))$) approximation algorithm with running time polynomial in both n and $1/\varepsilon$.*

3 Knapsack FPTAS

As an example, we will construct an FPTAS for the knapsack problem. First, we will give the classical exact algorithm for knapsack using dynamic programming.

Definition 3. *Suppose we have n items (indexed by i) with weights $w_i \in \mathbb{Z}_{\geq 0}$ and values $v_i \in \mathbb{Z}_{\geq 0}$, along with a weight budget W. The **0-1 knapsack** optimization problem is to find the subset with sum of weight is less than W, such that the sum of values is maximized.*

Knapsack is one of Karp's 21 NP-hard problems, by reduction from 3SAT. The "0-1" distinguishes this problem from the version where we are allowed to take fractions of items. An exact dynamic programming algorithm is as follows:

1. The dynamic programming table will be $W(i,v)$, where $W(i,v)$ is the minimum weight of a combination of items $1,\dots,i$ with value at least v.
2. Without loss of generality, all $w_i \leq W$ (we can remove these items otherwise; they cannot fit). We can initialize:
   - $W(0,0) = 0$
   - $W(0,v) = \infty$ for all $v > 0$.
3. The dynamic programming update:

$$W(i,v) = \min \begin{cases} W(i-1,v) \\ w_i + W(i-1, v-v_i) \end{cases}$$

4. The solution is in the cell with the maximum v where $W(n,v) < W$, which we can find by scanning all cells with $i = n$.

The dynamic programming table has size $n \cdot \sum_i v_i$. The sum of all v_i is loosely bounded by $n \cdot V$, where $V = \max(v_i)$. Since we must compute every cell in the table, the running time for this exact algorithm is $O(n^2 V)$, which is pseudo-polynomial time.

To construct the FPTAS, we will get rid of the dependence on V by reducing the number of distinct value categories (columns). We will essentially bucket values into $\text{poly}(n)/\varepsilon$ value categories by scaling and rounding each v_i. This way, we have $\text{poly}(n, 1/\varepsilon)$ table dimension and runtime, at the cost of accuracy.

1. Given $\varepsilon > 0$, we choose $k = \varepsilon V/n$ as a scaling parameter, then replace all item values with v'_i:

$$v'_i = \left\lceil \frac{v_i}{k} \right\rceil k$$

   The number of columns (different values of v') is now no more than:

$$n \frac{V}{k} = \frac{n^2}{\varepsilon}$$

2. Applying the earlier dynamic programming algorithm with values v', the running time is $O(n^3/\varepsilon) = O(\text{poly}(n, 1/\varepsilon)$.

We conclude by proving the approximation ratio of this scheme. Let S be the set of items produced by the PTAS algorithm, and S^* the optimal solution for the original instance.

$$\begin{aligned} \text{ALGO} &= \sum_{i \in S} v_i \\ \text{OPT} &= \sum_{i \in S^*} v_i \end{aligned}$$

The rounding procedure could have increased each v_i by less than k, since ceiling increases the value of the fraction by less than 1.

$$\begin{aligned}
\text{ALGO} &= \sum_{i\in S} v_i \\
&\geq \sum_{i\in S} v_i' - k \\
&= \sum_{i\in S} v_i' - |S|k \\
&= \sum_{i\in S} v_i' - |S| \cdot \frac{\varepsilon V}{n} \\
&\geq \sum_{i\in S} v_i' - \varepsilon V
\end{aligned}$$

Again due to the ceiling operation, it must also be that:

$$\sum_{i\in S} v_i' \geq \sum_{i\in S^*} v_i$$

And we can write:

$$\begin{aligned}
\text{ALGO} &\geq \sum_{i\in S} v_i' - \varepsilon V \\
&\geq \sum_{i\in S^*} v_i - \varepsilon V
\end{aligned}$$

Finally, since $V \leq \sum_{i\in S^*} v_i$ (choosing only the V set is a valid solution):

$$\begin{aligned}
\text{ALGO} &\geq \sum_{i\in S^*} v_i - \varepsilon V \\
&\geq (1-\varepsilon) \sum_{i\in S^*} v_i \\
&= (1-\varepsilon)\text{OPT}
\end{aligned}$$

Thus, we have a $(1-\varepsilon)$-approximation for knapsack using $O(\text{poly}(n, 1/\varepsilon)$ time – an FPTAS.

4 Strong and Weak NP-hardness

As it turns out, very few NP-hard problems have FPTAS. In this final section, we state a theorem which gives some quantification of the class of problems which do admit FPTAS. In some sense, these are the problems which are easiest to approximate. The following presentation is based on lecture notes from a class by Gupta [GK05] at CMU. First, we define a refinement of NP-hardness.

Definition 4. *A problem is **strongly** NP-hard if it is NP-hard to design a pseudo-polynomial algorithm. If there exist pseudo-polynomial algorithms (without assuming* $\mathsf{P} = \mathsf{NP}$*), the problem is **weakly** NP-hard.*

Equivalently, a problem is strongly NP-hard if every problem in NP is polynomial-time reducible to it, in a way that all reduced numbers are written in unary.

Most NP-hard problems we know are strongly NP-hard (e.g. 3SAT, bin packing). The following theorem by Garey and Johnson [GJ78] relates pseudo-polynomial algorithms (for a common style of optimization problems) to FPTAS:

Theorem 1. *Let P be an integral-valued NP-hard problem, and let B be a polynomial bound on the unary size of P. Furthermore, suppose for all instances of P the optimal solution is no more than B. Then if P has an FPTAS, P also has a pseudo-polynomial algorithm.*

Proof. Assume that P satisfies these conditions and has an FPTAS. Without loss of generality, we will use a minimization problem P. Choose $\varepsilon = 1/B$; the FPTAS gives us a $(1+\varepsilon)$-approximation in $O(\text{poly}(n, 1/\varepsilon))$ time. Then, we see that the approximation is:

$$(1+\varepsilon)\text{OPT} < \text{OPT} + \varepsilon B = \text{OPT} + 1$$

Since P is integral, the solution returned by the FPTAS is exact. Furthermore, the running time is:

$$O(\text{poly}(n, 1/\varepsilon)) = O(\text{poly}(n, B))$$

Since B is polynomial in the unary representation of the input, this is a pseudo-polynomial running time. We conclude that the FPTAS algorithm using $\varepsilon = 1/B$ is an exact, pseudo-polynomial time algorithm for P. $\square$

Corollary 2. *If an NP-hard problem fitting the criteria in Theorem 1 is strongly NP-hard, then it does not admit an FPTAS (assuming $\mathsf{P} \neq \mathsf{NP}$).*

References

[GJ78] Michael R Garey and David S Johnson. “strong”np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[GK05] Anupam Gupta and Mihir Kedia. 15-854: Approximation algorithms, lecture 10 scribe notes. URL: `http://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec10.pdf`. School of Computer Science, Carnegie Mellon University, October 2005.