CSC 2401 Introduction To Complexity—Fall 2001
Lecture #13, 24 October 2001
Lecturer: Allan Borodin
Scribe: Tristan Miller

Notational aside It should be noted that while the textbook uses the notation EXP to refer to $\bigcup_c DTIME(c^n)$, the standard practice is to use the notation E. Henceforth (and if we correct the previous lectures), we will use EXP to denote $\bigcup_k DTIME(2^{n^k})$ and E to denote $\bigcup_c DTIME(c^n)$.

13.1 KP in polynomial time

Recall from the previous lecture the knapsack problem (KP) which, in its optimization form, is as follows: Given a set

$$x = \{(w_1, v_1), (w_2, v_2), ..., (w_n, v_n), B\},$$

a feasible solution is a subset $I \subseteq \{1, 2, ..., n\} : \sum_{i \in I} w_i \leq B$ and gthe goal is to find a feasible solution of maximum value. The problem can also be stated as a decision problem, where x contains an additional member V, and we impose on the solution the further stipulation that $\sum_{i \in I} v_i \geq V$.

Claim 13.1.1 *If either B or* $\max\{v_i\}$ *is small (*i.e. *poly in n), then we can compute KP in poly time.*

Proof (a dynamic programming solution):

(a) ***B* is small.**

Without loss of generality, say all the w_i are small. Let $v = [i, t] =$ maximum value of the subset $I \subseteq \{1, 2, ..., n\}$ with $\sum_{i \in I} w_i \leq t$. Then $v^*(x) = v[n, B]$. We now perform induction on i, with our base case $\tilde{v}[0, t] = 0$. Let

$$\tilde{v}[i, t] = \max \begin{cases} \tilde{v}[i-1, t] & \text{if we don't use the } i\text{th element;} \\ \tilde{v}[i-1, t-w_i] + v_i & \text{otherwise.} \end{cases}$$

Then we show that v and $\tilde{v}$ are the same.

The complexity of an algorithm based on this recurrence is approximately $O(n \cdot B)$.

(b) **$\max\{v_i\}$ is small** (*i.e.* maximum profit $\leq \sum_{i\in I} v_i$.)

Let $w[i,v]=$ the least weight of a subset $I \subseteq \{1,2,...,n\}$ that achieves profit v, or ∞ if no such I exists. Then $v^*(x)=\max v : w[n,v] \neq \infty$. Again, we compute inductively on i, starting with $w[0,0]=0$ and $w[0,v]=\infty$ if $v>0$.

So why does this lead to a good approximation algorithm? And what's the general concept of an NP-hard/complete optimization problem?

13.2 General combinatorial optimization problem

Given input x, there is some feasible set of solutions $F(x)$. Also, we have a cost/profit function $v(x,y(x))$ where $y(x) \in F(x)$ and $v(x,y(x)) \in \mathbb{R}^{\geq 0}$. For strict optimization, given x, find either

$$y^*(x) : v^*(x) = v(x,y^*(x)) \leq v(x,y(x)) \ \ \forall y(x) \in F(x) \quad \text{(for cost) or}$$
$$y^*(x) : v^*(x) = v(x,y^*(x)) \geq v(x,y(x)) \ \ \forall y(x) \in F(x) \quad \text{(for profit).}$$

13.3 r-approximation algorithm

Let $r \geq 1$. Given x, find $\tilde{y}(x)$ such that $\forall y(x) \in F(x)$

$$\begin{aligned} v(x,\tilde{y}(x)) &\leq r\cdot v(x,y(x)) \quad \text{(cost) or} \\ \cdot v(x,\tilde{y}(x)) &\geq \frac{v(x,y(x))}{r} \qquad \text{(profit).} \end{aligned}$$

Consider the bin-packing problem: given a set of items of size ≤ 1, how many bins are needed to pack all items? Because we can transform SS into bin-packing, it is NP-hard to decide if we need 2 or 3 bins. Therefore $P \neq NP \Rightarrow r \geq \frac{3}{2}$—any poly-time algorithm is at best an $r \geq \frac{3}{2}$-approximation algorithm.

BUT can we get a minimization $\tilde{v}(x) \leq v^*(x)+1$? This is an open problem. However, it is known that there exists an algorithm that gets $\tilde{v}(x) \leq v^*(x) + o(v^*(x))$.[1]

13.3.1 Asymptotic r-approximation algorithm

In this case, we add a constant C:

$$\begin{aligned} v(x,\tilde{y}(x)) &\leq r\cdot v(x,y(x)) + C \quad \text{(cost) or} \\ \cdot v(x,\tilde{y}(x)) + C &\geq \frac{v(x,y(x))}{r} \qquad \text{(profit).} \end{aligned}$$

[1]Here o(f) indicates growth is asymptotically less than f.

13.4 NP Optimization problmes

For an NP optimization problem we have $\{(x,y)|y \in F(x)\} \in P$, poly-time in $|x|$ and hence the feasible solutions are polynomially bounded. Furthermore, the cost/profit function v is poly-computable. For an NP optimization problem Π, if an NP-hard decision problem (an in particular if the underlying decision problem related to Π) can be reduced to Π, the we abuse treminology and say that Π is an NP-complete optimization problem.

13.5 Major categories of approximation problems

Even though all NP-complete decision problems/languages are in practice, and in a certain theoretical sense, usually isomorphic to each other (*i.e.* they all "look the same"; they're all transformable to each other), NP-complete optimization problems can and do behave very differently (always assuming, say, $P \neq NP$).

(a) $\nexists$ a constant r such that an optimization problem can be solved within a factor of r. That is, there is no r-approximation algorithm, even in the asymptotic sense.

Examples: TSP, CLIQUE, graph-colouring

(b) $\exists$ constants $r_1, r_2 : r_2 < r_1$ and $\exists$ a poly-time r_1-approximation algorithm such that $\nexists$ a poly-time r_2-approximation algorithm.

(c) for every $r = 1 + \frac{1}{k} > 0$ (*i.e.* $\forall k \geq 1$), $\exists$ a poly-time r-approximation algorithm when k is viewed as a constant (might be EXP, $DOUBLE$ EXP, *etc.* in k). This is known as a PTAS (a poly-time approximation scheme).

(d) $\exists$ a poly-time algorithm $A(x,k)$ yielding a $(1 + \frac{1}{k})$-approximation algorithm whose complexity is polynomial on $|x|$ and k. This is known as FPTAS (fully poly-time approximation scheme).