The Knapsack Problem and Fully Polynomial Time Approximation Schemes (FPTAS)

Katherine Lai
18.434: Seminar in Theoretical Computer Science
Prof. M. X. Goemans

March 10, 2006

1 The Knapsack Problem

In the knapsack problem, you are given a knapsack of size $B \in \mathbb{Z}^+$ and a set $S = \{a_1, \ldots, a_n\}$ of objects with corresponding sizes and profits $s(a_i) \in \mathbb{Z}^+$ and $p(a_i) \in \mathbb{Z}^+$. The goal is to find the optimal subset of objects whose total size is bounded by B and has the maximum possible total profit. This problem is also sometimes called the *0/1 knapsack problem* because each object must be either in the knapsack completely or not at all. There are other variations as well, notably the *multiple knapsack problem*, in which you have more than one knapsack to fill.

The obvious greedy algorithm would sort the objects in decreasing order using the objects' ratio of profit to size, or profit density, and then pick objects in that order until no more objects will fit into the knapsack. The problem with this is that we can make this algorithm perform arbitrarily bad.

2 Approximation Schemes

Let Π be an **NP**-hard optimization problem with objective function f_Π. The algorithm $\mathcal{A}$ is an *approximation scheme* for Π if on input $I, \epsilon)$, where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

- $f_\Pi(I, s) \leq (1 + \epsilon) \cdot \text{OPT}$ if Π is a minimization problem.
- $f_\Pi(I, s) \geq (1 - \epsilon) \cdot \text{OPT}$ if Π is a maximization problem.

The approximation scheme $\mathcal{A}$ is said to be a *polynomial time approximation scheme*, or PTAS, if for each fixed $\epsilon > 0$, its running time is bounded by a polynomial in the size of instance I. This, however, means that it could be exponential with respect to $1/\epsilon$, in which case getting closer to the optimal solution is incredibly difficult.

Thus the *fully polynomial time approximation scheme*, or FPTAS, is an approximation scheme for which the algorithm is bounded polynomially in both the size of the instance I and by $1/\epsilon$.

3 PTAS for Knapsack

A smarter approach to the knapsack problem involves brute-forcing part of the solution and then using the greedy algorithm to finish up the rest [1]. In particular, consider all $O(kn^k)$ possible subsets of objects that have up to k objects, where k is some fixed constant [1]. Then for each subset, use the greedy algorithm to fill up the rest of the knapsack in $O(n)$ time. Pick the most profitable subset A. The total running time of this algorithm is thus $O(kn^{k+1})$. If O is the optimal subset, then the resulting approximation $P(A)$ achieves

$$P(O) \leq P(A)\left(1 + \frac{1}{k}\right)$$

Theorem 1 *Let $P(A)$ denote the profit achieved by this algorithm and $P(O)$ be the profit achieved by the optimal set O.*

$$P(O) \leq P(A)\left(1 + \frac{1}{k}\right)$$

Proof: If the optimal set O has size less than or equal to k, then this algorithm returns the optimal solution because O will be considered in the brute-force step. Otherwise, let $H = \{a_1, \ldots, a_k\}$ be the set of k most profitable items in O. Since all subsets of size k items are considered in the brute-force step, H must be one of them. For this subset, filling in the rest of the knapsack using the greedy algorithm will yield a profit that satisfies the desired bound.

Let $L_1 = O \setminus H = \{a_{k+1}, \ldots, a_x\}$, the remaining items in O in decreasing order of ratio of profit density. Let m be the index of the first item in L_1 which is not picked by the greedy algorithm step after picking H in the brute-force step. The reason the item a_m is not picked must be because its size is larger than the remaining empty space B_e. This also means that the greedy algorithm step has only picked items that have profit density of at least $p(a_m)/s(a_m)$ because it picks items in decreasing order of profit density. At this point, the knapsack contains $H, a_{k+1}, \ldots, a_{m-1}$, and some items not in O.

Let G be the items packed by the greedy algorithm step. As mentioned before, all items in set G have profit density of at least $p(a_m)/s(a_m)$. The items in $G \setminus O$, or the items in G that are not in the optimal set O, have total size

$$\Delta = B - \left(B_e + \sum_{i=1}^{m-1} s(a_i)\right)$$

Since all these items have profit density of at least $p(a_m)/s(a_m)$, we can bound the profit of G:

$$P(G) \geq \sum_{i=k+1}^{m-1} p(a_i) + \Delta \frac{p(a_m)}{s(a_m)}$$

We can write the total profit of the items in O as:

$$\begin{aligned} P(O) &= \sum_{i=1}^{k} p(a_i) + \sum_{i=k+1}^{m-1} p(a_i) + \sum_{i=m}^{|O|} p(a_i) \\ &\leq P(H) + \left(P(G) - \Delta \frac{p(a_m)}{s(a_m)} \right) + \left(B - \sum_{i=1}^{m-1} s(a_i) \right) \frac{p(a_m)}{s(a_m)} \\ &= P(H) + P(G) + B_e \frac{p(a_m)}{s(a_m)} < P(H \cup G) + p(a_m) \end{aligned}$$

The best found subset A returned after both the brute-force and the greedy algorithm steps will have at least as much profit as that of H and G combined since we know that the union of H and G must have been one of the considered subsets. Since $P(O) < P(H \cup G) + p(a_m)$ and $P(A) \geq P(H \cup G)$, then $P(O) - P(A) < p(a_m)$. Because the k items in H all have profit at least as large as that of a_m, we have that $s(a_m) \leq S(O)/(k+1)$, and this gives the approximation ratio. □

To obtain the polynomial time approximation scheme or PTAS, we have a $(1-\epsilon)$ approximation where $1/\epsilon = k + 1$. The resulting running time is $O(\frac{1}{\epsilon} n^{1/\epsilon})$, so the approximation scheme is polynomial in n but not $1/\epsilon$.

4 A Pseudo-polynomial Time Algorithm for Knapsack

The instance I of an optimization problem Π consists of the objects and numbers needed to describe the problem. First of all, an algorithm is said to run in *polynomial time* if its running time is polynomial in the size of the instance $|I|$, the number of bits needed to write I. It is said to run in *pseudo-polynomial time* if its running time is polynomial in $|I_u|$, where $|I_u|$ is the size of the instance when all the numbers are written in unary.

Knapsack is **NP**-hard, so we don't know a polynomial time algorithm for it. However, it does have a pseudo-polynomial time algorithm that we can use to create an FPTAS for knapsack. This algorithm uses dynamic programming to find the optimal solution. The algorithm is as follows:

Let P be the profit of the most profitable object, i.e. $P = \max_{a \in S} p(a)$. From this, we can upper bound the profit that can be achieved as nP for the n objects. For each $i \in \{1, \ldots, n\}$ and $p \in \{1, \ldots, nP\}$, let $S_{i,p}$ denote a subset of $\{a_1, \ldots, a_i\}$ that has a total profit of exactly p and takes up the least amount of space possible. Let $A(i, p)$ be the size of the set $S_{i,p}$, with a value of ∞ to denote no such subset. For $A(i, p)$, we have the base cases $A(1, p)$ where $A(1, p(a_1))$ is $s(a_1)$ and all other values are ∞. We can use the following recurrence

to calculate all values for $A(i,p)$:

$$A(i+1,p) = \begin{cases} \min\{A(i,p), s(a_{i+1}) + A(i,p-p(a_{i+1}))\} & \text{if } p(a_{i+1}) \leq p \\ A(i,p) & \text{otherwise} \end{cases}$$

The optimal subset then corresponds with the set $S_{n,p}$ for which p is maximized and $A(n,p) \leq B$. Since this iterates through at most n different values to calculate each $A(i,p)$, we get a total running time of $O(n^2P)$ and thus a pseudo-polynomial algorithm for knapsack.

5 FPTAS for Knapsack

From the pseudo-polynomial time algorithm, we see that if the profits of the objects were all small numbers, i.e. polynomially bounded in n, then we would have a regular polynomial time algorithm. We will use this to obtain an FPTAS for the knapsack problem. In particular, we can scale the profits down enough such that the profits of all the objects are polynomially bounded in n, use dynamic programming on the new instance, and return the resulting subset. By scaling with respect to the desired ϵ, we will be able to get a solution that is at least $(1-\epsilon) \cdot \text{OPT}$ in polynomial time with respect to both n and $1/\epsilon$, giving a FPTAS.

The algorithm is as follows:

1. Given $\epsilon > 0$, let $K = \frac{\epsilon P}{n}$.
2. For each object a_i, define $p'(a_i) = \left\lfloor \frac{p(a_i)}{K} \right\rfloor$.
3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set, say S'.
4. Output S'.

Lemma 2 *The set, S', output by the algorithm satisfies:*

$$P(S') \geq (1-\epsilon) \cdot \text{OPT}$$

Proof: Let O be the optimal set returning the maximum profit possible. Because we scaled down by K and then rounded down, any object a will have $K \cdot p'(a) \leq p(a)$ with the difference at most K. Thus the most that the profit of the optimal set O can decrease is at most nK for K per possible member of the set, or

$$P(O) - K \cdot P'(O) \leq nK$$

After the dynamic programming step, we get a set that is optimal for the scaled instance and therefore must be at least as good as choosing the set O with the smaller profits. From

that, we can see that

$$\begin{aligned} P(S') &\geq K \cdot P'(O) \\ &\geq P(O) - nK = \text{OPT} - \epsilon P \\ &\geq (1-\epsilon) \cdot \text{OPT} \end{aligned}$$

since OPT $\geq$ P. □

Theorem 3 *This algorithm is a fully polynomial approximation scheme for knapsack.*

Proof: As shown by Lemma 2, the solution of this algorithm falls within a $(1-\epsilon)$ factor of OPT. The running time of the algorithm is $O(n^2\lfloor\frac{P}{K}\rfloor)$ or $O(n^2\lfloor\frac{n}{\epsilon}\rfloor)$, which is polynomial in both n and $1/\epsilon$. □

6 Strong NP-hardness and FPTAS's

A problem Π is said to be *strongly* **NP**-*hard* if every problem in **NP** can be polynomially reduced to Π such that the numbers in this new reduction are all written in unary. A strongly **NP**-hard problem cannot have a pseudo-polynomial time algorithm, assuming **P**$\neq$**NP**.

Theorem 4 *Let p be a polynomial and Π be an* **NP**-*hard minimization problem such that the objective function f_Π is integer valued and on any instance I, $OPT(I) < p(|I_u|)$. If Π admits an FPTAS, then it also admits a pseudo-polynomial time algorithm.*

Proof: Suppose there is an FPTAS for Π whose running time on instance I and error parameter ϵ is $q(|I|, 1/\epsilon)$, where q is a polynomial. If we let $\epsilon = 1/p(|I_u|)$, running the FPTAS will give us a solution that is at most:

$$(1+\epsilon)\text{OPT}(I) < \text{OPT}(I) + \epsilon p(|I_u|) = \text{OPT}(I) + 1$$

since by assumption $\text{OPT}(I) < p(|I_u|)$. This result implies that the FPTAS will have to yield the optimal solution. The running time of the FPTAS is then $q(|I|, 1/\epsilon) = q(|I|, p(|I_u|))$, a polynomial in $|I_u|$, yielding a pseudo-polynomial time algorithm for Π. □

Corollary 5 *Let Π be an* **NP**-*hard optimization problem satisfying the restrictions of Theorem 4. If Π is strongly* **NP**-*hard, then* Π *does not admit an FPTAS, assuming* **P**$\neq$**NP**.

Proof: If Π admits an FPTAS, then it also has a pseudo-polynomial time algorithm by Theorem 4. This in turn means that it is not strongly **NP**-hard, assuming **P**$\neq$**NP**, which is a contradiction. □

If we let $\text{OPT}(I) < p(|I|)$ in Theorem 4, we would then be able to find a polynomial time algorithm, but this restriction is too strict for most problems.

6.1 Effectiveness of PTAS and FPTAS

While PTAS's and FPTAS's run in polynomial time, it is often the case that the running time is still prohibitive for reasonable choices for n and ϵ. Most of the approximation schemes use dynamic programming to exhaustively search a polynomial number of possibilities. It is unclear whether this is the best we can hope for in terms of tackling **NP**-hard problems.

References

[1] H. Shachnai and T. Tamir, "Polynomial Time Approximation Schemes - A Survey".

[2] V. Vazirani, "Approximation Algorithms," Springer, pp. 68-72, 2003.