COMPSCI 632: Approximation Algorithms 09-06-2017

Lecture 4

Lecturer: Debmalya Panigrahi *Scribe: Harsh Parikh*

1 Overview

In this lecture, we give approximation algorithms that use the technique of *data rounding*. The high level idea will be to round the values of an input in such a way so that the number of possible solutions is greatly reduced, which will allow us to use dynamic programming or even a brute-force algorithm to solve the problem efficiently. The goal then will be to show that the solution we obtain from the rounded instance is still a good approximation with respect to the original instance. The two problems we will examine for this technique are the *knapsack* and *bin packing* problems, where we give $1-\varepsilon$ and $1+\varepsilon$ approximations, respectively.

2 The Knapsack Problem

The first problem we examine is the *knapsack problem*, which is defined as follows.

Definition 1. KNAPSACK: *As input, we are given n items, where item i has a specified weight w_i and profit p_i. We are also given a maximum capacity B. The objective is to find the subset of items of maximum total profit such that its total weight does not exceed the maximum capacity B.*

The classic approach to solving this problem is to use dynamic programming. Namely, we define $b(i,p)$ to be the minimum budget needed to obtain a profit of exactly p using only items 1 through i, which can be expressed by the following recurrence.

$$b(i,p) = \begin{cases} \min\{(w_i + b(i-1, p-p_i), w(i-1,p)\} & \text{if } p_i \leq p \\ w(i-1,p) & \text{otherwise} \end{cases}$$

Thus, if we let $P = \max_i p_i$ and use nP as an upper bound on the total possible profit, we compute a table of size $nP \times n$, where each entry can be computed in constant time using the above recurrence. We can then can obtain our solution by looking for the largest value of p such that $b(n,p) = B$. Therefore, this gives us a $O(n^2P)$ time exact algorithm to the knapsack problem.

Observe, however, that since profit values can be represented using $\log_2 P$ bits, this running time is actually exponential in the size of the input (in other words, its only a polynomial-time algorithm if the input is represented in unary). However, what we can do is use this dynamic programming algorithm to obtain a PTAS for the problem. Namely, for given parameter ε, we will round the profits in a way so that the number of entries in the table is now polynomial in n and $1/\varepsilon$ (so now the algorithm runs in polynomial time if we think of ε as a constant). At the same time, we will show that an optimal solution to this rounded instance cannot be too far from an optimal solution to the original problem, i.e., we will show this scheme gives us as $1-\varepsilon$ approximation. We will see how to design such a scheme specifically for the knapsack

problem, but interestingly, it can be shown that for *any* problem that has pseudo-polynomial-time algorithm one can design such a PTAS [2].

Define $k = P\varepsilon/n$. To obtain our rounded instance, we round all profit values p_i down to the nearest multiple of k. Thus, the number of possible profits we need to consider in our table is now $nP/k = n^2/\varepsilon$, which implies the running time for the entire procedure is now $O(n^3/\varepsilon)$. We now show that this algorithm gives us the desired approximation ratio for a PTAS.

Theorem 1. *The rounding algorithm for the knapsack problem is a* $1-\varepsilon$ *approximation.*

Proof. Let ALG_R and ALG_O the value of the algorithm's solution on the rounded instance and on the original instance, respectively; define OPT_O and OPT_R similarly. First observe that through rounding, we can only decrease the value of each item by at most k. Thus, for any solution, its total value in the rounded instance is at most εP less than its value in the original instance. Therefore by the optimality of OPT_R on the rounded instance, we have $\text{OPT}_R \geq \text{OPT}_O - \varepsilon P$.

Clearly $\text{ALG}_R = \text{OPT}_R$ since the algorithm produces an exact solution to the rounded instance via dynamic programming. We also have that $\text{ALG}_O \geq \text{ALG}_R$ since rounding down can only decrease the value of the solution. Combining these observations, coupled with the fact that $\text{OPT}_O \geq P$, we obtain that $\text{ALG}_O \geq (1-\varepsilon)\text{OPT}_O$, as desired. □

3 The Bin Packing Problem

Now, we will examine the bin packing problem, which is defined as follows.

Definition 2. BIN PACKING [1]: *As input, we are given n items, where each item i has as specified size* $s_i \in [0,1]$. *Each item must be assigned to a unique bin, where each bin has a capacity of 1. The objective is to pack all items into the minimum number of bins such that the total size of the items within each bin does not exceed the bin's capacity.*

We can actually obtain a 2-approximation for bin packing using a simple algorithm known as *First Fit*: we scan the items in an arbitrary order, and if the current item fits in a bin we have already opened, then we place the item in that bin. Otherwise, we open an new bin and place the current item in it.

It is straightforward to show that First Fit ensures the following property.

Lemma 2. *At the end of First Fit, there is at most one bin that is less than half full.*

Proof. For sake of contradiction, suppose there are two bins b and b' which are less than half full at the end of the algorithm, and suppose b' was opened after b. Let i' be the first item that was placed in b' (i.e., the item that made us open b' for the first time). Since b' is less than half full at the end of the algorithm, $s_{i'} < 1/2$. However, this means that it must have been possible to fit i' into b (since it must have also been less than half full when we were assigning i'). Thus, i' should have been placed in b, which is a contradiction. □

Let ALG and OPT denote the number of bins opened by First Fit and optimal solution, respectively. Observe that $\text{OPT} \geq \sum_i s_i$ since its impossible to use fewer bins than the total volume and still pack all the items. But based on Lemma 1, we know that $\text{ALG} \leq 2\sum_i s_i + 1$ since doubling the amount of volume in each bin that is at least half full will be more than the total number of bins (plus 1 for the additional bin that may be less than half full). Therefore, for First Fit we have that $\text{ALG} \leq 2\text{OPT}+1$. Note that we are losing an extra additive term of 1 ; however, since this term becomes negligible as the size of the instances we consider goes to infinity, we still consider this a 2 approximation.

We now show how to obtain a $(1+\varepsilon)$-approximate algorithm, which is given formally in Algorithm 1. At a high level, our approach will be to first discard items that are small, i.e., less than $\varepsilon/2$. For the remaining items (call these large items), we will round their sizes so that in the rounded instance there are only polynomial in n number of configurations to consider (if we think ε as a constant). Thus for large items we enumerate all possible solutions to find an optimal solution. The task is then to show that 1) the optimal number of bins used in the rounded solution is not too far from the original instance, and 2) when we add the small jobs back to the solution using First Fit, the solution still remains good.

Algorithm 1 Approximate Bin Packing [1]

1: **procedure** APPROXIMATE PACKING
2: Sort items in increasing order of their sizes
3: Remove items of size less than $\varepsilon/2$ (call these *small items*).
4: Make blocks of $\varepsilon^2 n/2$ items $\implies$ $\#blocks = \frac{2}{\varepsilon^2}$
5: Round all items to the maximum item size in the block
6: Remove all items in block with the largest items; call this the *rounded instance*.
7: Solve exactly for the rounded instance (using brute force).
8: Assign discarded large items each to their own bin
9: Pack remaining small items using First Fit

We note that on line 4, we start by making blocks with the largest items, i.e., the first block contains the $\varepsilon^2 n/2$ largest items, and so on. Therefore, note that the last block containing the smallest items may have less than $\varepsilon^2 n/2$ items.

We first argue that this algorithm runs in polynomial time. It is clear to see that all steps are polynomial time except the brute-force search on line 7. Thus, we establish line 7 takes polynomial time in the following lemma.

Lemma 3. *For a fixed ε and c, consider a bin-packing instance where $\varepsilon/2 \leq s_i \leq 1$ and number of distinct s_i is equal to c. There exist an exact bin-packing algorithm which runs in $O(n \cdot n^{(2/\varepsilon)^c})$ time.*

Proof. Observe that since items are all at least size $\varepsilon/2$, there can be at most $2/\varepsilon$ items per bin. Therefore, since there are c distinct item sizes, the number of ways to configure a single bin is $\binom{2/\varepsilon+c-1}{c} \leq (2/\varepsilon)^c$. Call this value N. Since there can be only n bins in total, a solution corresponds to picking at most n bin configurations. Thus, there are at most $\binom{N+n-1}{N} \leq n^N$ possible solutions to consider. Generating and validating a solution takes $O(n)$ time; thus, the lemma follows. □

As noted on line 4 in Algorithm 1, the number of distinct item sizes is $2/\varepsilon^2$. Thus, we set $c = 2/\varepsilon^2$ in Lemma 3, it follows that a brute-force search on the rounded instance takes polynomial in n time if ε is a constant.

Now we establish that the algorithm is $(1+\varepsilon)$ approximate. We first argue that the solution obtain on large jobs (i.e., before executing line line 9) is $(1+\varepsilon)$ approximate. We first show the following lemma.

Lemma 4. *Let OPT and OPT$'$ denote the optimal solutions for the original instance with only large items and the rounded instance, respectively. Then OPT$' \leq$ OPT.*

Proof. Observe that we can create a 1-to-1 mapping that maps items in the rounded instance to original instance. Namely, each item in a block will be mapped to an item in the next largest block; note that this is possible since each block contains $\varepsilon^2 n/2$ items (or in the case of the last block containing the $h \leq \varepsilon^2 n/2$

smallest items, we can pick an arbitrary subset of h items to map to in the second smallest block). Thus, since each item in the rounded instance is mapped to larger item in the original instances, the packing used by OPT gives us a feasible packing on the rounded instance, and therefore $\text{OPT}' \leq \text{OPT}$. □

Next, we observe that the solution produced by the rounded solution is a feasible packing for the items included in the rounded instances (i.e., all items except those in the largest block) since items are rounded up to larger sizes. The algorithm creates a feasible packing for the large items using $\text{OPT}' + \varepsilon^2 n/2$ bins, which by Lemma 4 is at most $\text{OPT} + \varepsilon^2 n/2$. Furthermore, we know that since we've discarded small items, we have that $\text{OPT} \geq \varepsilon n/2$. Combing these observations, it follows that the algorithms opens at most $(1+\varepsilon)\text{OPT}$ bins for large items.

Finally, we need to show that adding in small items with First Fit does not affect our approximation ratio. This is straight forward to show. If we do not any new bins when packing small items, then clearly the algorithm is still a $(1+\varepsilon)$ approximation. If we do open new bins, then we know that (using the same argument used in Lemma 2), all bins except for one have volume at least $1-\varepsilon/2$. Therefore, if ALG is the number of bins opened by the algorithm, we have $\text{ALG} \leq 1/(1-\varepsilon/2) \cdot \sum_i s_i + 1$. Since, $1/(1-\varepsilon/2) \leq 1+\varepsilon$ and OPT must be at least $\sum_i s_i$, we have that $\text{ALG} \leq (1+\varepsilon)\text{OPT} + 1$, as desired.

References

[1] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within 1 + ε in linear time. *Combinatorica*, 1(4):349–355, Dec 1981.

[2] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.