Advanced Algorithms	25 September 2018

Lecture 2: Approximation Algorithms II

Lecturer: Mohsen Ghaffari — *Scribe: Davin Choo*

1 Approximation schemes

Previously, we described simple greedy algorithms that approximate the optimum for minimum set cover, maximal matching and vertex cover. We now formalize the notion of efficient $(1+\epsilon)$-approximation algorithms for minimization problems, a la [Vaz13].

Let I be an instance from the problem class of interest (e.g. minimum set cover). Denote $|I|$ as the size of the problem (in bits), and $|I_u|$ as the size of the problem (in unary). For example, if the input is just a number x (of at most n bits), then $|I| = \log_2(x) = \mathcal{O}(n)$ while $|I_u| = \mathcal{O}(2^n)$. This distinction of "size of input" will be important later when we discuss the knapsack problem.

Definition 1 (Polynomial time approximation algorithm (PTAS))**.** *For cost metric c, an algorithm $\mathcal{A}$ is a PTAS if for each fixed $\epsilon > 0$, $c(\mathcal{A}(I)) \leq (1+\epsilon) \cdot c(OPT(I))$ and $\mathcal{A}$ runs in* $\text{poly}(|I|)$.

By definition, the runtime for PTAS may depend arbitrarily on ϵ. A stricter related definition is that of fully polynomial time approximation algorithms (FPTAS). Assuming $\mathbb{P} \neq \mathbb{NP}$, FPTAS is the best one can hope for on $\mathbb{NP}$-hard optimization problems.

Definition 2 (Fully polynomial time approximation algorithm (FPTAS))**.** *For cost metric c, an algorithm $\mathcal{A}$ is a FPTAS if for each fixed $\epsilon > 0$, $c(\mathcal{A}(I)) \leq (1+\epsilon) \cdot c(OPT(I))$ and $\mathcal{A}$ runs in* $\text{poly}(|I|, \frac{1}{\epsilon})$.

As before, $(1-\epsilon)$-approximation, PTAS and FPTAS for maximization problems are defined similarly.

2 Knapsack

Definition 3 (Knapsack problem)**.** *Consider a set $\mathcal{S}$ with n items. Each item i has $size(i) \in \mathbb{Z}^+$ and $profit(i) \in \mathbb{Z}^+$. Given a budget B, find a subset $S^* \subseteq S$ such that:*

(i) (Fits budget): $\sum_{i \in S^*} size(i) \leq B$

(ii) (Maximum value): $\sum_{i \in S^*} profit(i)$ is maximized.

Let us denote $p_{max} = \max_{i \in \{1,\ldots,n\}} profit(i)$. Further assume, without loss of generality, that $size(i) \leq B, \forall i \in \{1,\ldots,n\}$. As these items cannot be chosen in S^*, we can remove them, and relabel, in $\mathcal{O}(n)$ time without affecting the correctness of the result. Thus, observe that $p_{max} \leq profit(OPT(I))$ because we can always pick at least one item, namely the highest valued one.

Example Denote the size and profit of each item by a pair $i : (size(i), profit(i))$. Consider an instance where budget $B = 10$ and $\mathcal{S} = \{1 : (10, 130), 2 : (7, 103), 3 : (6, 91), 4 : (4, 40), 5 : (3, 38)\}$. One can verify that the best subset $S^* \subseteq S$ is $\{2 : (7, 103), 5 : (3, 38)\}$, yielding a total profit of $103 + 38 = 141$.

2.1 An exact algorithm in poly(np_{max}) via dynamic programming (DP)

Observe that the maximum achievable profit is at most np_{max}, where $S^* = S$. Using dynamic programming (DP), we can form a n-by-(np_{max}) matrix M where $M[i,p]$ is the smallest total sized subset from $\{1,\ldots,i\}$ such that the total profit equals p. Trivially, set $M[1, profit(1)] = size(1)$ and $M[1,p] = \infty$ for $p \neq profit(1)$. To handle boundaries, we also define $M[i,j] = \infty$ for $j \leq 0$. Then,

$$M[i+1,p] = \begin{cases} M[i,p] & \text{if } profit(i+1) > p \text{ (Cannot pick)} \\ \min\{M[i,p], size(i+1) + M(i, p - profit(i+1))\} & \text{if } profit(i+1) \leq p \text{ (May pick)} \end{cases}$$

Since each cell can be computed in $\mathcal{O}(1)$ using the DP via the above recurrence, matrix M can be filled in $\mathcal{O}(n^2 p_{max})$ and S^* may be extracted by back-tracing from $M[n, np_{max}]$.

Remark This dynamic programming algorithm is *not* a PTAS because $\mathcal{O}(n^2 p_{max})$ is exponential in input problem size $|I|$. This is because the value p_{max} is just a single number, hence representing it only requires $\log_2(p_{max})$ bits. As such, we call this DP algorithm a *pseudo-polynomial time algorithm.*

2.2 FPTAS for the knapsack problem via profit rounding

Algorithm 1 FPTAS-KNAPSACK($\mathcal{S}, B, \epsilon$)

$k \leftarrow \max\{1, \lfloor \frac{\epsilon p_{max}}{n} \rfloor\}$ ▷ Choice of k to be justified later
for $i \in \{1, \ldots, n\}$ **do**
  $profit'(i) = \lfloor \frac{profit(i)}{k} \rfloor$ ▷ Round the profits
end for
Use DP described in Section 2.1 with same sizes and same budget B but re-scaled profits.
return Answer from DP

Algorithm 1 pre-processes the problem input and calls the dynamic programming algorithm described in Section 2.1. Since we scaled down the profits, the new maximum profit is $\frac{p_{max}}{k}$, hence the DP now runs in $\mathcal{O}(\frac{n^2 p_{max}}{k})$. To obtain a FPTAS for Knapsack, we pick k such that Algorithm 1 is a $(1-\epsilon)$-approximation algorithm and runs in $\text{poly}(n, \frac{1}{\epsilon})$.

Theorem 4. *For any $\epsilon > 0$ and knapsack instance $I = (\mathcal{S}, B)$, then Algorithm 1 ($\mathcal{A}$) is a FPTAS.*

Proof. Let $loss(i)$ denote the decrease in value by using rounded $profit'(i)$ for item i. By the profit rounding definition, for each item i, $loss(i) = profit(i) - k\lfloor \frac{profit(i)}{k} \rfloor \leq k$. Then, over all n items,

$$\begin{array}{rcll} \sum_{i=1}^{n} loss(i) & \leq & nk & \\ & < & \epsilon \cdot p_{max} & \text{Since } k = \lfloor \frac{\epsilon p_{max}}{n} \rfloor \\ & \leq & \epsilon \cdot profit(OPT(I)) & \text{Since } p_{max} \leq profit(OPT(I)) \end{array}$$

Thus, $profit(\mathcal{A}(I)) \geq (1-\epsilon) \cdot profit(OPT(I))$.
Furthermore, the $\mathcal{A}(I)$ runs in $\mathcal{O}(\frac{n^2 p_{max}}{k}) = \mathcal{O}(\frac{n^3}{\epsilon}) \in \text{poly}(n, \frac{1}{\epsilon})$. □

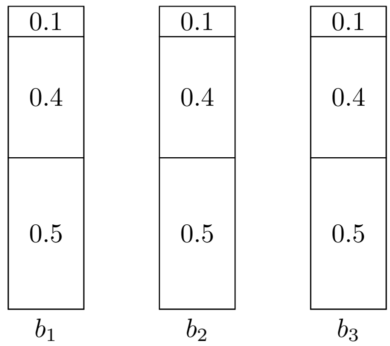
Example Recall the earlier example where budget $B = 10$ and $\mathcal{S} = \{1 : (10, 130), 2 : (7, 103), 3 : (6, 91), 4 : (4, 40), 5 : (3, 38)\}$. For $\epsilon = \frac{1}{2}$, one would set $k = \max\{1, \lfloor \frac{\epsilon p_{max}}{n} \rfloor\} = \max\{1, \lfloor \frac{\frac{1}{2} \cdot 130}{5} \rfloor\} = 13$. After rounding, we have $\mathcal{S}' = \{1 : (10, 10), 2 : (7, 7), 3 : (6, 7), 4 : (4, 3), 5 : (3, 2)\}$. The optimum subset from $\mathcal{S}'$ is $\{3 : (6, 7), 4 : (4, 3)\}$ which translates to a total profit of $91 + 40 = 131$ in the original problem. As expected, $131 = profit(\text{FPTAS-KNAPSACK}(I)) \geq (1 - \frac{1}{2}) \cdot profit(OPT(I)) = 70.5$.

3 Bin packing

Definition 5 (Bin packing problem)**.** *Given a set $\mathcal{S}$ with n items where each item i has $size(i) \in (0, 1]$, find the minimum number of unit-sized (size 1) bins that can hold all n items.*

For any problem instance I, let $OPT(I)$ be a optimum bin assignment and $|OPT(I)|$ be the corresponding minimum number of bins required. One can see that $\sum_{i=1}^{n} size(i) \leq |OPT(I)|$.

Example Consider an instance where $\mathcal{S} = \{0.5, 0.1, 0.1, 0.1, 0.5, 0.4, 0.5, 0.4, 0.4\}$, where $|\mathcal{S}| = n = 9$. Since $\sum_{i=1}^{n} size(i) = 3$, at least 3 bins are needed. One can verify that 3 bins suffices: $b_1 = b_2 = b_3 = \{0.5, 0.4, 0.1\}$. Hence, $|OPT(\mathcal{S})| = 3$.



3.1 First-fit: A 2-approximation algorithm for bin packing

Algorithm 2 FIRSTFIT($\mathcal{S}$)

B $\rightarrow \emptyset$ ▷ Collection of bins
for $i \in \{1, \ldots, n\}$ **do**
  if $size(i) \leq size(b)$ for some bin $b \in B$ **then**
    $size(b) \leftarrow size(b) - size(i)$ ▷ Put item i to existing bin b
  else
    $B \leftarrow B \cup \{b'\}$, where $size(b') = 1 - size(x_i)$ ▷ Put item i into a fresh bin b'
  end if
end for
return B

Algorithm 2 shows the First-Fit algorithm which processes items one-by-one, creating new bins if an item cannot fit into existing bins.

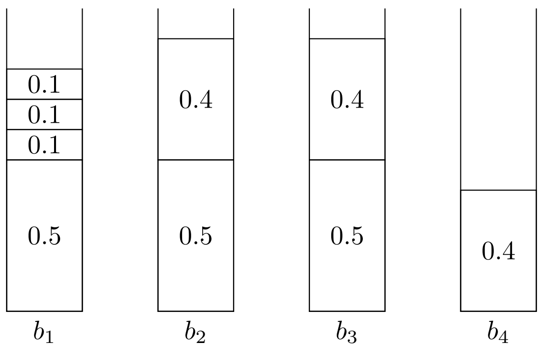
Lemma 6. *Using First-Fit, at most one bin is less than half-full. That is,* $|\{b \in B : size(b) \leq \frac{1}{2}\}| \leq 1$.

Proof. Suppose, for a contradiction, that there are two bins b_i and b_j such that $i < j$, $size(i) \leq \frac{1}{2}$ and $size(j) \leq \frac{1}{2}$. Then, First-Fit could have put all items in b_j into b_i, and not create b_j. Contradiction. □

Theorem 7. *First-Fit is a 2-approximation algorithm for bin packing.*

Proof. Suppose First-Fit terminates with $|B| = m$ bins. By lemma above, $\sum_{i=1}^{n} size(i) > \frac{m-1}{2}$. Since $\sum_{i=1}^{n} size(i) \leq |OPT(I)|$, we have $m - 1 < 2\sum_{i=1}^{n} size(i) \leq 2 \cdot |OPT(I)|$. That is, $m \leq 2 \cdot |OPT(I)|$. □

Recall the example where $\mathcal{S} = \{0.5, 0.1, 0.1, 0.1, 0.5, 0.4, 0.5, 0.4, 0.4\}$. First-Fit will use 4 bins: $b_1 = \{0.5, 0.1, 0.1, 0.1\}, b_2 = b_3 = \{0.5, 0.4\}, b_4 = \{0.4\}$. As expected, $4 = |\textsc{FirstFit}(\mathcal{S})| \leq 2 \cdot |OPT(\mathcal{S})| = 6$.



Remark If we first sort the item weights in non-increasing order, then one can show that running First-Fit on non-increasing ordering of item weights will yield a $\frac{3}{2}$-approximation algorithm for bin packing. See footnote for details[1].

[1] Curious readers may want to read the following lecture notes for proof on First-Fit-Decreasing:
http://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/bin_packing.pdf
https://dcg.epfl.ch/files/content/sites/dcg/files/courses/2012%20-%20Combinatorial%20Optimization/12-BinPacking.pdf

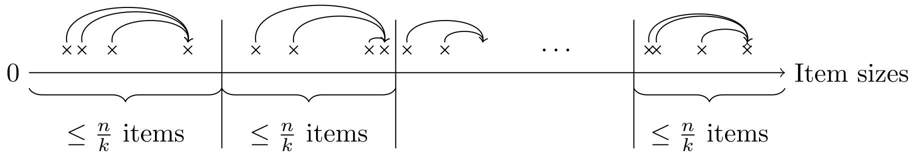


Figure 1: Partition items into k groups, then round sizes up to the maximum size in each group.

It is natural to wonder whether we can do better than a $\frac{3}{2}$-approximation. Unfortunately, unless $\mathbb{P} = \mathbb{NP}$, we cannot do so efficiently. To prove this, we show that if we can efficiently derive a $(\frac{3}{2} - \epsilon)$-approximation for bin packing, then the partition problem (which is $\mathbb{NP}$-hard) can be solved efficiently.

Definition 8 (Partition problem)**.** *Given a multiset $\mathcal{S}$ of (possibly repeated) positive integers $x_1, \ldots, x_n$, is there a way to partition $\mathcal{S}$ into $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\sum_{x \in \mathcal{S}_1} x = \sum_{x \in \mathcal{S}_2} x$?*

Theorem 9. *Solving bin packing with $(\frac{3}{2} - \epsilon)$-approximation for $\epsilon \in (0, \frac{1}{2}]$ is $\mathbb{NP}$-hard.*

Proof. Suppose algorithm $\mathcal{A}$ solves bin packing with $(\frac{3}{2} - \epsilon)$-approximation for $\epsilon > 0$. Given an instance of the partition problem with $\mathcal{S} = \{x_1, \ldots, x_n\}$, let $X = \sum_{i=1}^{n} x_i$. Define set $\mathcal{S}' = \{\frac{2x_1}{X}, \ldots, \frac{2x_n}{X}\}$ and run $\mathcal{A}(\mathcal{S}')$. Since $\sum_{x \in \mathcal{S}'} x = 2$, at least two bins are required. By construction, one can bi-partition $\mathcal{S}$ if and only if only two bins are required to pack $\mathcal{S}'$. Since $\mathcal{A}$ gives a $(\frac{3}{2} - \epsilon)$-approximation, if the $OPT(I)$ returns 2 bins, then $\mathcal{A}(I)$ will return $\lfloor (\frac{3}{2} - \epsilon)(2) \rfloor = 2$ bins. As $\mathcal{A}$ can solve the partition problem, solving bin packing with $(\frac{3}{2} - \epsilon)$-approximation for $\epsilon \in (0, \frac{1}{2}]$ is $\mathbb{NP}$-hard. $\square$

3.2 Special case where items have sizes larger than ϵ, for some $\epsilon > 0$

In this section, we describe a PTAS algorithm that solves the special case of bin packing assuming all items have at least size $\epsilon > 0$. We first describe an exact algorithm that further assumes another condition. Then, we show how we round the item weights and make use of the exact algorithm, as a black box, to yield a PTAS. Note that the final algorithm we describe is *not* a FPTAS because it will run in time exponential in $\frac{1}{\epsilon}$.

3.2.1 Exact solving via $\mathcal{A}_\epsilon$

Let us describe an exact algorithm for a special case of bin packing with two assumptions:

1. All items have at least size ϵ
2. There are only k different possible sizes (for some constant k)

Let $M = \lceil \frac{1}{\epsilon} \rceil$ and x_i be the number of items of the i^{th} possible size. Let R be the number of weight configurations, or possible item configurations (multiset of item weights) in a bin. By assumption 1, each bin can only contain $\leq M$ items. By assumption 2, there are at most $R = \binom{M+k}{M}$. Then, the total number of bin configurations is at most $\binom{n+R}{R}$. Since k is a constant, one can enumerate over all possible bin configurations (denote this algorithm as $\mathcal{A}_\epsilon$) to *exactly* solve bin packing in this special case in $\mathcal{O}(n^R) \in \text{poly}(n)$ since R is a constant (with respect to constants ϵ and k).

Remark 1 Number of configurations are computed by solving combinatorics problems of the following form: How many non-negative integer solutions are there to $x_1 + \cdots + x_n \leq k$?[2]

Remark 2 The number of bin configurations is computed out of n bins (i.e. 1 bin for each item). One may use less than n bins, but this upper bound suffices for our purposes.

3.2.2 PTAS for special case

Algorithm 3 pre-processes the sizes of a given input instance, then calls the exact algorithm $\mathcal{A}_\epsilon$ to solve the modified instance. Since we only round up sizes, $\mathcal{A}_\epsilon(J)$ will yield a satisfying bin assignment for instance I, with spare "slack". We will prove the following claim in the next lecture.

Claim 10. $|OPT(J)| \leq |OPT(I)| + n\epsilon^2$

[2]See slides 22 and 23 of http://www.cs.ucr.edu/~neal/2006/cs260/piyush.pdf for illustration of $\binom{M+k}{M}$ and $\binom{n+R}{R}$.

Algorithm 3 PTAS-BinPacking($I = \mathcal{S}, \epsilon$)

$k \leftarrow \lceil \frac{1}{\epsilon^2} \rceil$
Partition n items into k non-overlapping groups, each with at most $\frac{n}{k}$ items ▷ See Figure 1
for $i \in \{1, \ldots, k\}$ **do**
  $k_{max} \leftarrow \max_{\text{item } j \text{ in group } i} size(j)$
  for item j in group i **do**
    $size(j) \leftarrow k_{max}$
  end for
end for
Denote the modified instance as J
return $\mathcal{A}_\epsilon(J)$

References

[Vaz13] Vijay V Vazirani. *Approximation algorithms.* Springer Science & Business Media, 2013.