12 Bin Packing

12.1: $\mathcal{NP}$-hardness • 12.2: 3/2-Approximation • 12.3: Asymptotic $(1+\varepsilon)$-Approximation Scheme

BIN PACKING: Given a set $I = \{1, \dots, n\}$ of *items* with *sizes* $s_i \in (0, 1]$, and a set $B = \{1, \dots, n\}$ of *bins* with capacity 1, put all the items in as few bins as possible.
More formally, find an assignment $a : I \to B$ such that for each $b \in B$ we have $\sum_{i:a(i)=b} s_i \leq 1$ and $a(I)$ is minimal.

12.1 $\mathcal{NP}$-hardness

We first have to show that BIN PACKING is $\mathcal{NP}$-hard. We'll do this by reducing VERTEX COVER to SUBSET SUM, then reducing SUBSET SUM to PARTITION, and finally PARTITION to BIN PACKING. I don't know of a shorter way.

SUBSET SUM: Given a sequence $a_1, \dots, a_n, t$ of nonnegative integers, decide whether there is a subset $S \subset \{1, \dots, n\}$ such that

$$\sum_{i \in S} a_i = t.$$

PARTITION: Given a sequence $a_1, \dots, a_n$ of integers, decide whether there is a subset $S \subset \{1, \dots, n\}$ such that

$$\sum_{i \in S} a_i = \sum_{j \notin S} a_j.$$

Theorem 12.1. SUBSET SUM *and* PARTITION *are* $\mathcal{NP}$*-complete.*

Theorem 12.2. BIN PACKING *is* $\mathcal{NP}$*-hard. It has no* k*-approximation algorithm with* $k < 3/2$ *(unless* $\mathcal{P} = \mathcal{NP}$*).*

Proof. We reduce PARTITION to BIN PACKING. Given an instance $a_1, \dots, a_n$ of PARTITION, we let $A = \sum a_i$ and consider the instance of BIN PACKING with $s_i = 2a_i/A$. It is easy to see that the answer to the PARTITION instance is YES if and only if the minimum number of bins for the BIN PACKING instance is 2.
If there was a k-approximation algorithm for BIN PACKING with $k < 3/2$, then when the exact minimum is 2 bins, this algorithm would always find it, since 3 bins would be an approximation with factor $\geq 3/2$. So by the reduction above, this algorithm would exactly solve PARTITION in polynomial time, which would imply $\mathcal{P} = \mathcal{NP}$. □

12.2 3/2-Approximation Algorithm

As we saw above, an approximation algorithm for Bin Packing with approximation factor $< 3/2$ is not possible. Here we'll see something that we haven't seen before in this course, namely an approximation algorithm that exactly matches a lower bound.
However, as we'll see in the next section, this lower bound is a little misleading, because it really only holds for instances that come down to deciding between 2 or 3 bins. If you know that more bins will be needed, better approximations are possible.

First Fit Decreasing Algorithm (FFD)

1. Sort the items in decreasing order and relabel them, so that $s_1 \geq s_2 \geq \cdots \geq s_n$; set $i = 1$;
2. Put i in the first bin that it fits into, i.e.

$$a(i) = \min \left\{ b \in B : \left(\sum_{j:a(j)=b} s_j \right) + s_i \leq 1 \right\};$$

   set $i := i + 1$;
3. If $i < n$ go back to 2, otherwise return a.

First Fit is the same algorithm without the sorting step. It can be proven that First Fit has an approximation factor basically $\frac{17}{10}$, but this is fairly hard. Fortunately, the sorting step makes it easier to prove a better approximation factor.

Theorem 12.3. *First Fit Decreasing is a* $3/2$*-approximation algorithm for* Bin Packing.

Proof. Let a be the assignment found by the algorithm, with $k = |a(I)|$, and let a^* be the minimal assignment, with $k^* = |a^*(I)|$. We want to show that $k \leq \frac{3}{2}k^*$. We will assume here that the items are in decreasing order. We'll write $S = \sum_{i \in I} s_i$; note that we have a trivial bound $k^* \geq S$.
Let $b \leq k$ be an arbitrary bin used by a. We will analyze the following two cases: b contains an item of size $> 1/2$, or it does not.
Suppose b contains an item i of size $s_i > 1/2$. Then the previously considered items $i' < i$ all have $s_{i'} > 1/2$, and each bin $b' < b$ must contain one of these, so we have $\geq b$ items of size $> 1/2$. No two of these can be in the same bin in any packing, so a^* uses at least b bins, i.e. $k^* \geq b$.
Suppose b does not contain an item of size $> 1/2$. Then no used bin $b'' > b$ contains an item of size $> 1/2$, which implies that each of these bins contains ≥ 2 items, except maybe for the last used one (bin k). So the $k - b$ bins $b, b+1, \ldots, k-1$ together contain $\geq 2(k-b)$ items. We know that none of these items would have fit in any bin $b' < b$.
We consider two subcases. If $b \leq 2(k-b)$, then we can imagine adding to every bin $b' < b$ one of these $2(k-b)$ items, which would give us $b-1$ overfilled bins. This implies that $S > b-1$. On the other hand, if $b > 2(k-b)$, then we can imagine adding each of the $2(k-b)$ elements to a different bin $b' < b$, giving us $2(k-b)$ overfilled bins. Then $S > 2(k-b)$.
So for any b we have either $k^* \geq b$ (in the first case or the first subcase of the second case) or $k^* \geq 2(k-b)$ (in the second subcase). Now we choose b so that it will more or less maximize the minimum of b and $2(k-b)$: Equating $b = 2(k-b)$ gives $b = \frac{2}{3}k$, and we take $b = \lceil \frac{2}{3}k \rceil$ to get an integer. Then we have that $k^* \geq \lceil \frac{2}{3}k \rceil \geq \frac{2}{3}k$, or $k^* \geq 2(k - \lceil \frac{2}{3}k \rceil) \geq \frac{2}{3}k$. □

12.3 Asymptotic $(1+\varepsilon)$-Approximation Scheme

We won't define in general what an approximation scheme is, it should be clear from the theorem. The word "asymptotic" refers to the fact that the approximation factor only holds for large instances above some threshold. Non-asymptotic approximation schemes exist for instance for KNAPSACK or EUCLIDEAN TSP.

These schemes are less useful in practice than they may sound, because the running times tend to be huge (although polynomial). Another drawback in this case is that the running time is polynomial in n, but not in ε. So in practice, fast approximation algorithms like the one above might be more useful.

Theorem 12.4. *For any $0 < \varepsilon \leq 1/2$ there is an algorithm that is polynomial in n and finds an assignment having at most $k \leq (1+\varepsilon)k^*$ bins, whenever $k^* \geq K(\varepsilon)$.*

The proof requires the following two lemmas.

Lemma 12.5. *Let $\varepsilon > 0$ and $d \in \mathbb{N}$ be constants. There is a polynomial algorithm that exactly solves any instance of* BIN PACKING *with all $s_i \geq \varepsilon$ and $|\{s_1, \ldots, s_n\}| \leq d$.*

Proof. This can be done simply by enumerating all possibilities and checking each one.
The number of items in a bin is at most $L = \lfloor 1/\varepsilon \rfloor$. Therefore the number of ways of putting items in a bin (disregarding the ordering) is less than $M = \binom{L+d}{L}$ (using a standard formula from combinatorics/probability theory). The number of feasible assignments to n bins is then $N = \binom{n+M}{M}$ (by the same formula).
Now $N \leq c \cdot n^M$, and M is independent of M, so the number of cases to check is polynomial in n. Checking an assignment takes constant time, so this gives an exact polynomial algorithm. (But note that the running time is not polynomial in $1/\varepsilon$ or d.) $\square$

Lemma 12.6. *Let $\varepsilon > 0$ be a constant. There is a polynomial algorithm that gives a $(1+\varepsilon)$-approximation for any instance of* BIN PACKING *that has all $s_i \geq \varepsilon$.*

Proof. Let I be the list of items. Sort them by increasing size, and partition them into $P+1$ groups of at most $Q = \lfloor n/P \rfloor$ items (so the smallest Q items in a group, then the next Q items, etc., and the last group may have fewer than Q items). We will choose the integer P later.
We construct two new lists H and J. For H, round down the size of the items in each group to the size of the smallest item of that group. For J, round up to the largest size in each group. Let $k^*(H)$, $k^*(I)$, and $k^*(J)$ be the minimal numbers of bins for each instance.
Then we clearly have

$$k^*(H) \leq k^*(I) \leq k^*(J),$$

since a minimal assignment for one list will also work for a list that is smaller entry-by-entry. On the other hand, we have

$$k^*(J) \leq k^*(H) + Q \leq k^*(I) + Q,$$

because, given an assignment a_H for H, we get an assigment a_J for J as follows. The items of group $i+1$ in H will be larger than (or equal to) the items of group i in J, so we can assign the items of group i in J to the bins of the items in group $i+1$ in H. This leaves the $\leq Q$ items of the largest group in J, which we assign to the Q extra bins. Note that we've ignored the smallest group of H.
Now if we choose P so that $Q \leq \varepsilon \cdot k^*(I)$, then we have $k^*(J) \leq (1+\varepsilon) \cdot k^*(I)$. We can choose $P = \lceil 1/\varepsilon^2 \rceil$, since then $Q = \lfloor \varepsilon^2 n \rfloor \leq \varepsilon \cdot k^*(I)$, using that $k^*(I) \geq \sum s_i \geq \varepsilon \cdot n$.
Now we have what we want, since we can apply Lemma 12.5 to J, with ε and $d = P+1$. The resulting assignment for J will also work for I. We do not need to assume $\varepsilon \leq 1/2$, because we already have a 3/2-approximation algorithm. $\square$

Proof of Theorem 12.4. Given a list of items I, remove all items of size $< \delta$ to get a list I' (with δ to be chosen later). We have $k^*(I') \leq k^*(I)$. Lemma 12.6 gives an assignment a' of I' using $k(I') \leq (1+\delta) \cdot k^*(I')$ bins.
We now assign the removed small items to the bins used by a' as far as possible, using First Fit; we may have to use some new bins. If we do not need new bins, we are done since then we have an assignment of I with $k(I) = k(I') \leq (1+\delta) \cdot k^*(I') \leq (1+\delta) \cdot k^*(I)$ bins.
If we do need new bins, then we know that all $k(I) - 1$ bins have remaining capacity less than ε. This means that

$$k^*(I) \geq \sum_{i \in I} s_i \geq (k(I) - 1)(1 - \delta),$$

so

$$k(I) \leq \frac{k^*(I)}{1-\delta} + 1 \leq (1 + 2\delta) \cdot k^*(I) + 1,$$

using the inequality $\frac{1}{1-x} \leq 1 + 2x$, which holds for $0 < x \leq 1/2$.
Now choose $\delta = \varepsilon/4$ and take $K(\varepsilon) = 2/\varepsilon$. Then when $k^*(I) \geq K(\varepsilon)$ we have $1 \leq \varepsilon \cdot k^*(I)/2$, so

$$k(I) \leq (1 + \frac{\varepsilon}{2}) \cdot k^*(I) + 1 \leq (1 + \frac{\varepsilon}{2}) \cdot k^*(I) + \frac{\varepsilon}{2} \cdot k^*(I) \leq (1+\varepsilon) \cdot k^*(I).$$

□

The algorithm would now look as follows. As mentioned above, it is not an algorithm that would really be implemented, its interest is more theoretical.

Asymptotic $(1+\varepsilon)$-Approximation Algorithm for Bin Packing

1. Order I by increasing size;
2. Remove items of size $< \varepsilon$ to get new list I';
3. Partition I' into $P = \lceil 1/\varepsilon^2 \rceil$ groups;
4. Round the sizes up to the largest size in each group, call this list J';
5. Find the minimal assignment for J' using enumeration;
6. Make this into an assignment for I by assigning the removed items using First Fit;
7. Return this assignment.