600.469 / 600.669 Approximation Algorithms **Lecturer:** Michael Dinitz
Topic: Dynamic Programing: Min-Makespan and Bin Packing **Date:** 2/19/15
Scribe: Gabriel Kaptchuk

8.1 Makespan Scheduling

Consider an instance of the Makespan Scheduling Problem

$$\begin{array}{ll} \text{Input:} & \text{Jobs } J \in \{1,2,\ldots,n\} \text{ with processing times } P_i. \\ & \text{Machines } i \in \{1,2,\ldots,m\} \\ \text{Feasable:} & \Phi : J \to M \\ \text{Objective:} & \text{Minimize makespan (max load)} \end{array} \tag{8.1.1}$$

We proved in the previous lecture that there exists a simple greedy algorithm that is a 2-approximation. We will now attempt to find a $(1+\epsilon)$-approximation. To accomplish this, we will divide jobs into large and small.

Let us suppose we have an algorithm which, given some guess T, will either

1. if $T \geq OPT$, return a solution with makespan at most $(1+\epsilon)T$
2. if $T < OPT$, either return false or return a solution with makespan at most $(1+\epsilon)OPT$

Such an algorithm is sometimes called a $(1+\epsilon)$-relaxed decision procedure.

Definition 8.1.1 *Let $P = \sum_{i \in J} P_i$. Clearly $1 \leq OPT \leq P$.*

Since $1 \leq OPT \leq P$, if we have access to a $(1+\epsilon)$-relaxed decision procedure we can use it $O(\log P)$ times to do a binary search on different values of T (between 1 and P) to find a $(1+\epsilon)$-approximation. Since $\log P$ is polynomial in the input size, if the relaxed decision procedure runs in polynomial time then we have a $(1+\epsilon)$-approximation for Makespan Scheduling that runs in polynomial time.

Hence we will now only be concerned with designing a $(1+\epsilon)$-relaxed decision procedure. Since we are now given a guess T, we can define small and large jobs with respect to T.

Definition 8.1.2 *Let $J_{small} = \{j \in J | P_j \leq \epsilon T\}$ and $J_{large} = \{j \in J | P_j > \epsilon T\}$*

Theorem 8.1.3 *Given a schedule for J_{large} with makespan at most $(1+\epsilon)T$, we can find a schedule for all of J with makespan at most $(1+\epsilon)\max(T, OPT)$*

Proof: We greedily place small jobs. More specifically, we start with the schedule for large jobs guaranteed by the theorem. We then consider the small jobs in arbitrary order, and for each job we consider we place it on the least-loaded machine.

Consider a machine i. We break into two cases, depending on whether i was assigned any small jobs.

Case 1: i has no small jobs. Then the jobs on i are exactly the large jobs which were originally scheduled on it. Hence its makespan is at most $(1+\epsilon)T$ because its load is bounded by the given solution to J_{large}.

Case 2: i has one or more small jobs. Then we can repeat the analysis of the greedy algorithm from the last lecture, but now using the fact that the jobs added are all small. In particular, the load on i before the last job was added to it was smaller than the load on any other machine (by the definition of the greedy algorithm), and hence just before the last job was added to i is has load at most $\frac{P}{M} \leq OPT$. The last job added has processing time at most ϵT since it is small, and therefore the final load on i is at most $OPT + \epsilon T \leq (1+\epsilon)\max(T, OPT)$.

Since the above analysis holds for an arbitrary machine i, it holds for every machine and thus the makespan is at most $(1+\epsilon)\max(T, OPT)$. ■

Given the above theorem, from this point forward we will only consider methods for solving Makespan Scheduling for J_{large}

Definition 8.1.4 *Let* $b = \lceil \frac{1}{\epsilon} \rceil$, *so* $\frac{1}{b} \leq \epsilon$.

Definition 8.1.5 *Let* $P'_j = \lfloor \frac{P_j b^2}{T} \rfloor \cdot \frac{T}{b^2}$

We can think of P'_j as rounding P_j down to the nearest multiple of T/b^2. In particular, we get that $P'_j \leq P_j \leq P'_j + \frac{T}{b^2}$. Also, since $j \in J_{large}$ we know that $\epsilon T \leq P_j \leq T$, and hence $P'_j = k \cdot \frac{T}{b^2}$ for some $k \in \{b, b+1, \ldots, b^2\}$. We will call this new instance the *rounded* instance.

Lemma 8.1.6 *If there is a schedule with makespan at most T in the original instance, then there is a schedule with makespan at most T in the rounded instance.*

Proof: Consider the schedule with makespan at most T in the original instance. Since $P'_j \leq P_j$ for all j, this schedule has makespan at most T under the rounded processing times. ■

Theorem 8.1.7 *Any schedule with makespan at most T in the rounded instance has makespan at most $(1+\epsilon)T$ under the original processing times.*

Proof: Since $P'_j \geq \frac{T}{b}$, there are at most b jobs on each machine. Thus under the original processing times, for any machine i the load is at most

$$\begin{aligned}\sum_{j \text{ assigned to } i} P_j &\leq \sum_{j \text{ assigned to } i} (P'_j + \frac{T}{b^2}) \\ &\leq \sum_{j \text{ assigned to } i} P'_j + \sum_{j \text{ assigned to } i} \frac{T}{b^2} \leq T + \sum_{j \text{ assigned to } i} \frac{T}{b^2} \\ &\leq T + b \cdot \frac{T}{b^2} = (1 + \frac{1}{b})T \leq (1+\epsilon)T,\end{aligned}$$

proving the theorem. ■

Thus in order to find a solution with makespan at most $(1+\epsilon)T$ (assuming a schedule of makespan at most T exists), we just need an algorithm that computes a solution to the rounded instance with makespan at most T. Fortunately, since we have bounded the number of possible job lengths, this is a reasonably straightforward using dynamic programming.

Definition 8.1.8 *Let a* configuration *be a tuple* $(a_b, a_{b+1}, \ldots, a_{b^2})$ *with each* $a_i \in \mathbb{Z}^+$*, such that* $\sum_{i=b}^{b^2}(a_i \cdot i \cdot \frac{T}{b^2}) \leq T$*. Let* $C(T)$ *be a set of all configurations. Note that* $|C(T)| \leq (b+1)^{b^2-b} \approx b^{b^2}$

The point of this definition is that in any schedule with makespan at most T in the rounded instance, the jobs assigned to each machine are described by a configuration. In other words, for each machine i, there is some configuration $(a_b, \ldots, a_{b^2})$ such that there are exactly a_k jobs of length $k \cdot \frac{T}{b^2}$ assigned to it for each $k \in \{b, b+1, \ldots, b^2\}$. Thus in order to find a schedule with makespan at most T in the rounded instance we just need to find a configuration for each machine so that every job is assigned to some machine.

Definition 8.1.9 *Let* $f(n_b, n_{b+1}, \ldots, n_{b^2})$ *be the minimum number of machines need to schedule* n_i *jobs of length* $i \cdot \frac{T}{b^2}$*, (for all* $i \in \{b, b+1, \ldots, b^2\}$*) with makespan at most* T*.*

Clearly $f(0, 0, ..., 0) = 0$. Now we can write a recurrence relation for other inputs:

$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{\vec{a} \in C(T), a_i \leq n_i} f(n_b - a_b, n_{b+1} - a_{b+1}, \ldots, n_{b^2} - a_{b^2})$$

This is a correct relation because it essentially tries all possible configurations for one machine, recursing on the rest. Consider a dynamic program for this function. Since each $n_i \leq n$, clearly the number of table entries of the dynamic program is at most n^{b^2}. Since the number of configurations is at most b^{b^2}, computing each table entry given the previous ones takes at most $O(b^{b^2})$ time. Hence the total running time of this dynamic program is at most $O(n^{O(b^2)})$.

With this function f in hand, we are finished. Suppose the rounded instance has n_i jobs of length $i \cdot \frac{T}{b^2}$. Then we can just check whether $f(n_b, n_{b+1}, \ldots, n_{b^2})$ is at most m (the number of machines we have available). If no, then we return false – there is no schedule of makespan at most T in the rounded instance and thus by Lemma 8.1.6 no schedule of makepsan at most T in the original instance. If yes, then by Theorem 8.1.7 this schedule has makespan at most $(1 + \epsilon)$ on the original instance.

8.2 Bin Packing

Bin packing is essentially the converse of Makespan scheduling: rather than minimizing the makespan for a fixed number of machines, it is the problem of minimizing the machines subject to a makespan bound. For historical reasons, though, it is usually phrased somewhat differently.

The Bin Packing problem is defined as follows.

Input:	Items $I \in \{1, 2, \ldots, n\}$ with sizes $0 \leq S_i \leq 1$.
Feasable:	Partition $\{B_j\}_{j \in \{1,2,\ldots,m\}}$ of I with $\sum_{i \in B_j} S_i \leq 1, \forall j \in \{1, 2, \ldots, m\}$
Objective:	Minimize m

Consider the following greedy algorithm, which simply assigns items one at a time in arbitrary order. If there is a bin with enough room in it to add the item, we do so. Otherwise, we create a new bin.

Algorithm 1 Greedy Algorithm for Bin Packing Problem

```
Input: Items $I$ with sizes $S_i$
Output: a partition of those items into $B_j$
  $k \leftarrow 1$
  $B_1 \leftarrow \emptyset$
  while there is an unassigned element $j \in I$ do
    if there exists $a \leq k$ with $S_j + \sum_{i \in B_a} S_i \leq 1$ then
      assign $j$ to $B_a$ (if more than one such $a$ exists pick one arbitrarily)
    else
      $k \leftarrow k + 1$
      assign $j$ to $B_k$
    end if
  end while
  return $B_j, \forall j \in \{1, 2, \ldots, k\}$
```

Theorem 8.2.1 *The above greedy algorithm uses at most* $2 \cdot OPT + 1$ *Bins*

Proof: First note that all bins in this algorithm other than last one are at least half full. To see this, suppose there were two bins with load at most 1/2. Then when the second bin was created, the job added to it could have fit in the first bin. Hence the algorithm would not have created the second bin, giving us a contradiction. Let $s(I) = \sum_{i \in I} S_i$, and suppose that greedy uses a bins. Then since all but the last are at least half full, we know that $s(I) \leq \frac{1}{2}(a-1)$. But clearly since each bin has capacity 1 we also know that $s(I) \leq OPT$. Hence $a \leq 2 \cdot OPT + 1$. ■

Similar to makespan scheduling, we can improve this with rounding and dynamic programming to get the following theorem.

Theorem 8.2.2 *For all* $\epsilon > 0$*, there is an algorithm that finds a solution with at most* $(1+\epsilon)OPT + 1$ *bins with running time* $n^{O(\frac{1}{\epsilon^2})}$.

We begin by splitting into large and small jobs.

Definition 8.2.3 *Let* $I_{large} = \{i \in I | S_i \geq \frac{\epsilon}{2}\}$ *and* $I_{small} = \{i \in I | S_i < \frac{\epsilon}{2}\}$

Theorem 8.2.4 *Given a partition of* I_{large} *into* b *bins, we can find a partition of all of* I *into* $\max(b, (1+\epsilon)OPT + 1)$ *bins.*

Proof: We simply perform greedy on I_{small} starting with the solution from I_{large}. The argument is similar to Makespan Scheduling. If the greedy algorithm does not need to add more bins, then we get a solution with b bins. If it does need to add more bins, then every bin other than the last one must contain items with total size at least $1 - \frac{\epsilon}{2}$. Hence if the greedy algorithm ends up with a bins, we know that $(a-1)(1-\frac{\epsilon}{2}) \leq OPT$ and hence $a \leq \frac{1}{1-\epsilon/2}OPT + 1 \leq (1+\epsilon)OPT + 1$ ■

Given the above theorem, from this point forward we will only consider methods for solving Bin Packing for I_{large}

We want to construct a "rounded" instance as in makespan scheduling, but unfortunately simple rounding does not work as well here. Instead, we need to use a more sophisticated method known as *linear grouping.* We first order the elements of I_{large} by size. WLOG we say $S_1 \geq S_2 \geq \cdots \geq S_l$ for $l = |I_{large}|$. Then for some number k (to be defined later) we partition I_{large} into sets of size k sets by descending order. So we have

$$G_1 = \{S_1, S_2, \ldots, S_k\}$$
$$G_2 = \{S_{k+1}, S_{k+2}, \ldots, S_{2k}\}$$
$$\vdots$$
$$G_r = \{S_{(r-1)k+1}, \ldots, S_l\}$$

Note that every group other than the last one has exactly k elements (the last one may have fewer). Now for all but the first group G_1, we round up to the largest element of that group. More formally, let $i \in I_{large} - G_1$ and suppose that $i \in G_j$. Then we set $S'_i = \max_{i' \in G_j} S_{i'} = S_{(j-1)k+1}$

Lemma 8.2.5 $S_{i-k} \geq S'_i \geq S_i, \; \forall i \in I_{large} - G_1$

Proof: The second inequality is obvious, as our rounding never decreased any value. For the first inequality, note that $i-k$ is an item which is in a different group from item i: if i is in G_j, then $i-k$ is in G_{j-1}. Since the rounding worked by setting all items in a group to the largest size in the group, we know that $S'_{i-k} = S_{(j-2)k+1} \geq S_{(j-1)k+1} = S'_i$. ■

We essentially ran out of time at this point. The following is an informal description of the rest of the proof. Let I' denote the new (rounded) instance.

Lemma 8.2.6 $OPT(I') \leq OPT(I_{large})$

Proof: Consider the optimal partition for I_{large}. Suppose each item $i \in I_{large}$ gets put in bin $f(i)$ (so f is the assignment function for this solution). To get a solution for I' with the same number of bins, we just place item i in bin $f(i-k)$. The first inequality of Lemma 8.2.5 implies that this is a valid solution to I' (note that this is why why could not include G_1 in I'. ■

Lemma 8.2.7 *Suppose there is a solution for I' using ℓ bins. Then there is a solution for I_{large} using at most $\ell + k$ bins.*

Proof: Consider a partition for I' into ℓ bins. By the second inequality in Lemma 8.2.5 this partition is also valid when using the original sizes. We just need to add the items in G_1. But we can trivially do this using k more bins by putting each item of G_1 into its own bin. ■

So suppose that we could solve I' optimally. Then we can use Lemma 8.2.7 to turn this into a solution for I_{large} using at most $OPT(I') + k$ bins. Now Lemma 8.2.6 implies that this is at most $OPT(I_{large}) + k$ bins. If we set k to be $\lfloor \epsilon \sum_{i \in I_{large}} S_i \rfloor$ we know that $k \leq \epsilon OPT(I_{large})$, and hence we have a solution using at most $(1+\epsilon)OPT(I_{large})$ bins. Adding small items using Theorem 8.2.4 now gives the overall approximation guarantee.

So we just need to solve I' optimally. Fortunately, this is essentially the exact same dynamic program from makespan scheduling! We may assume that $l \geq 2/\epsilon^2$ since otherwise it is trivial to solve I_{large} optimally through brute force. Then the number of *distinct* sizes in I' is at most

$$l/k \leq \frac{l}{\lfloor \epsilon \sum_{i \in I_{large}} S_i \rfloor} \leq \frac{l}{\lfloor \epsilon \cdot l \cdot \frac{\epsilon}{2} \rfloor} = \frac{l}{\lfloor l \cdot \epsilon^2/2 \rfloor} \leq \frac{l}{l \cdot \epsilon^2/4} = \frac{4}{\epsilon^2}$$

So there are only a constant number of distinct sizes, so as with makespan scheduling there are at most a constant number of configurations and there is a dynamic programming algorithm to find the minimum number of bins necessary that runs in time $n^{O(1/\epsilon^2)}$.