601.435/635 Approximation Algorithms **Lecturer:** Michael Dinitz
Topic: Rounding and DP: Min-Makespan Scheduling **Date:** 2/15/24
Scribe: Michael Dinitz

8.1 Makespan Scheduling

Consider the Makespan Scheduling Problem

Input:	Jobs $J = \{1, 2, \ldots, n\}$
	Processing times $p : J \to \mathbb{N}$
	Machines M
Feasable:	$\Phi : J \to M$
Objective:	Minimize makespan (max load): $\min \max_{i \in M} \sum_{j \in J : \Phi(j) = i} p(j)$

8.1.1 Greedy

Like with knapsack, there's a pretty easy greedy 2-approximation.

Algorithm 1 Greedy Min-Makespan Algorithm

Input: J, M, p
Output: $\Phi : J \to M$

```
Φ ← ∅
for j ∈ J do
    m ← least loaded machine in Φ
    Set Φ to assign job j to machine m.
end for
return Φ
```

Theorem 8.1.1 *This algorithm is a 2-approximation.*

Proof: Consider the following two simple observations:

1. $\forall j \in J, p(j) \leq \text{OPT}$ (since every job needs to be scheduled on some machine)
2. $\frac{\sum_{j \in J} p(j)}{|M|} \leq \text{OPT}$

Let m be the machine whose load equals the makespan of the greedy algorithm (i.e. the most heavily loaded machine after the algorithm completes). Let j be the last job assigned to m by the algorithm. Then just before j was assigned to m, by the second observation we know that the load of m was at most OPT. By the first observation the processing time for j is at most OPT, and since j was the last job assigned to m we get that the total load on m is at most $2 \times \text{OPT}$. ■

8.1.2 Dynamic Programming Algorithm

Now we're going to try to find a $(1+\epsilon)$-approximation.

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 8.1.2 *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 8.1.3 *Let $J_{small} = \{j \in J : p(j) \leq \epsilon T\}$ and $J_{large} = \{j \in J : p(j) > \epsilon T\}$*

Theorem 8.1.4 *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 earlier, 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 8.1.5 *Let $b = \lceil \frac{1}{\epsilon} \rceil$, so $\frac{1}{b} \leq \epsilon$.*

Definition 8.1.6 *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 $\epsilon T \leq p(j) \leq T$, so $T/b \leq p'(j) \leq T$ 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 8.1.7 *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 8.1.8 *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

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

proving the theorem. ■

Thus in order to find a solution with makespan at most $(1+\epsilon)T$ to the original instance (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 reasonably straightforward using dynamic programming.

Definition 8.1.9 *Let a* configuration *be a tuple $(a_b, a_{b+1}, \ldots, a_{b^2})$ with each $a_i \in \{0, 1, 2, \ldots, b\}$, 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 8.1.10 *Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines needed 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 8.1.7 no schedule of makespan at most T in the original instance. If yes, then by Theorem 8.1.8 this schedule has makespan at most $(1 + \epsilon)$ on the original instance.