CSE 632: Analysis of Algorithms II **Fall 2017**

Lecture 4 (09/08/2017): Scheduling on identical machines

Lecturer: Shi Li *Scribe: Zhenggang Xue*

Let $[n]$ be the set $\{1, 2, 3, \cdots, n\}$.

Definition 4.1 (Scheduling on identical machines) *Given $P_1, P_2, P_3, \cdots, P_n \geq 0$ and $m \geq 1$, the goal is to divide $[n]$ into m sets $J_1, J_2, \cdots, J_m$, so as to minimize $max_{i\in[m]}\sum_{j\in J_i} P_j$.*

For example, we now have 3 machines and 7 jobs, whose processing time are 13, 10, 5, 6, 8, 20 and 11 respectively. In this case, optimal solution would be $\{10, 5\}, \{13, 10\}, \{6, 8, 11\}$ and optimal value would be 25.

Algorithm 1 Naive Greedy Algorithm for Scheduling on identical machines

```
1: $J_1, J_2, \cdots, J_m \leftarrow \emptyset$
2: for $j \leftarrow 1$ to $n$ do
3:     let $i \in [m]$ be the index with minimum $P(J_i) := \sum_{j'\in J_i} P_{j'}$
4:     $J_i \leftarrow J_i \bigcup \{j\}$
5: return $J_1, J_2, \cdots, J_m$
```

Since at the last step, we pick the machine with least heavily loaded. The other machines will be all busy at the time t, so it is trivial that $t \leq \frac{P([n])}{m}$ and the processing time of new added job $P(m)$ would be no great than the maximum processing time of all jobs, that is $P(m) \leq \max_{j\in[m]} P_j$.

Besides, for optimum solution, we have two inequilities: *opt* $\geq \frac{P([n])}{m}$ and *opt* $\geq \max_{j\in[n]} P_j$.

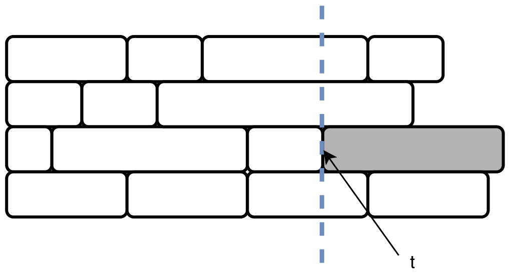

Figure 4.1: Analysis at last step. The gray job is inserted into the 3rd machine that is currently the least heavily loaded.

Thus, our naive greedy algorithm returns a solution of $cost = t + P(m) \leq \frac{f([n])}{m} + \max_{j\in[n]} P_j \leq 2 \cdot opt$.

Theorem 4.2 *Naive Greedy Algorithm for scheduling on identical machines is a 2-appoximation algorithm.*

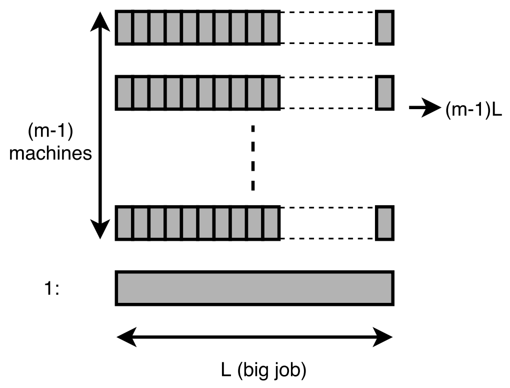

Figure 4.2: A bad instance for greedy algorithm 1

Here we have a intuitive bad example. Assuming m machines, $(m-1)L$ jobs with processing time 1, and 1 job which is the last one with time L. The value of optimum solution will absolutely be L. However, in our naive greedy algorithm, it will be $L + \frac{(m-1)L}{m} = (2 - \frac{1}{m})L$.

The running time of naive greedy algorithm is $O(nm)$. Can we improve the running time? The idea comes with sorting job according to sizes first, that is $P_{j_1} \geq P_{j_2} \geq P_{j_3} \cdots \geq P_{j_n}$ where $\{j_1, j_2, j_3, \cdots, \jmath_n\} = [n]$. So we have a improved greedy algorithm for scheduling on identical machines.

Algorithm 2 Improved Greedy Algorithm for Scheduling on identical machines

```
1: Build a max-heap for all jobs according to their processing time.
2: $J_1, J_2, \cdots, J_m \leftarrow \emptyset$
3: for $k \leftarrow 1$ to $n$ do
4:     let $i \in [m]$ be the index with minimum $P(J_i) := \sum_{j' \in J_i} P_{j'}$
5:     $J_i \leftarrow J_i \bigcup \{j_k\}$
6: return $J_1, J_2, \cdots, J_m$
```

So we get a better running time $O(n \lg m)$.

Lemma 4.3 *Assume we have* $\min_{j \in [n]} P_j > \frac{opt}{3}$*, the "improved algorithm" gives the optimum solution, in which every machine has at most 2 jobs.*

Proof: To prove that each machine has at most 2 jobs, we assume there exists a counterexample that a machine M has k jobs in optimum solution, $k \geq 3$. Then then solution will have at least $k \cdot \frac{opt}{3} > opt$. This is not a counterexample and every machine has at most 2 jobs.

Our improved algorithm will give us the solution with such combination that the largest value with smallest value, the 2nd largest with 2nd smallest, and so on so forth. This is true if we assuming that there are $2 \cdot m$ jobs, in which the rest $(2m - n)$ jobs can be considered as 0-processing-time job.

To prove this solution is optimum, we can arbitrarily change the combination on two machines. This will obviously increase one of the this two machines, and will lead to a solution that will no better than previous solution.

■

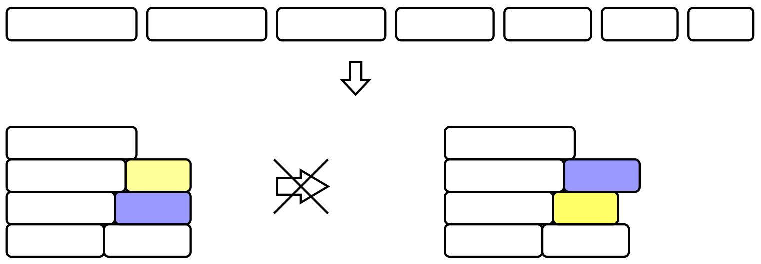

Figure 4.3: An example for Lemma 4.3