Resource Allocation Algorithms
Lecture 03

Lecture by Dr. Tami Tamir
Typeset by Steven Karas

2018-03-20
Last edited 18:12:25 2018-04-10

Disclaimer These lecture notes are based on the lecture for the course Resource Allocation Algorithms, taught by Dr. Tami Tamir at IDC Herzliyah in the spring semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Tami Tamir.

Agenda

- Branch and bound

1 Review

1.1 Scheduling/Queueing Theory

In a system with n jobs that are processed over m machines:

- J_j - the j-th job.
- p_j - the length of J_j = how many processing units it requires.
- r_j - the release time of J_j = when J_j is available for execution.
- d_j - deadline for J_j = when execution of J_j needs to be completed by.
- C_j - the completion time of J_j in a given schedule.

A scheduling problem is represented as $\alpha|\beta|\gamma$:

- α represents the processing environment. E.g. number of machines, concurrency, etc.
- β represents additional constraints. E.g. preemption, precedence, etc.
- γ represents the desired objective function. E.g. $\sum_j C_j$.

We proved that some classes of objective functions are equivalent:

$$\underbrace{\min C_{\max}}_{\text{makespan}} = \underbrace{\max \text{avg}(N_p)}_{\text{avg. throughput}} = \underbrace{\min \sum_k I_k}_{\text{total idle time}}$$

$$\begin{aligned}\sum_j C_j &= \sum_j F_j + \sum_j r_j \\ &= \sum_j W_j + \sum_j r_j + \sum_j p_j \\ &= \sum_j L_j + \sum_j d_j\end{aligned}$$

We also showed some optimal solutions to simple problems such as SPT and variants.
We also showed that both $1|\text{setup}|C_{\max}$ and $1|r_j|T_{\max}$ are $\mathcal{NP}$-hard.

2 Branch and bound

This is a general technique in combinatorial optimization which yields an optimal solution. The running time is not predictable, and in the worst case, all possible configurations are examined. However, it may be very efficient.

Generally branches by dividing the problem into subproblems, and finds a lower bound for the optimal subproblem. This is predicated upon the subproblems being disjoint and providing a partial solution to the original problem.

Example We will work out a branch and bound solution to $1||\sum T_j$:

j	p_j	d_j
1	4	5
2	3	6
3	7	8
4	2	8
5	2	17
total	18	17

We can start from an EDD solution: 1,2,3,4,5 which gives us:

j	1	2	3	4	5
start	0	4	7	14	16
C_j	4	7	14	16	18
T_j	0	1	6	8	1

We branch things out based upon which job is scheduled last. [1]

The drawing was given on the board, but also appears on slide 43 of the previous lecture slides.

3 Scheduling theory in nontrivial environments

We will now consider n jobs being scheduled on m machines, where each machine processes at most one job at a time.

An identical machine environment where all machines work at the same rate such that the processing time of job J_j on machine m is p_j. This environment is denoted as P.

A uniform machine environment denoted as Q is where each machine has a processing rate s_i which is uniform for all jobs such that the processing time of J_j on M_i is $\frac{p_j}{s_i}$

An unrelated machine environment denoted as R is where each machine gives specific values for p_{ij} which is the processing time of J_j on M_i.

A good resource for finding currently known best scheduling algorithms can be found at http://schedulingzoo.lip6.fr

3.1 P - Service time

SPT is optimal for $P||\sum_j C_j$.

Optimality proof sketch Assume $n = zm$, given that we can introduce synthetic jobs with $p_j = 0$. Sort the jobs $p_1 \leq \ldots \leq p_n$. Each ordering of job on a specific machine is counted multiple times (because they affect following jobs multiple times).

3.2 P - Weighted service time

Provably $\mathcal{NP}$-hard.

Solvable per machine by WSPT, but partition between the machines is $\mathcal{NP}$-hard.

3.3 P - Makespan

Provably $\mathcal{NP}$-hard by reduction from Partition.

The proof sketch follows from a direct translation of $P2||C_{\max}$

[1] It's generally a good idea to draw most of the branches off to the left and bunched together, since they tend to be bound quickly

3.3.1 Approximation: List Scheduling

Greedily schedule the next job on the least loaded machine. Provides a $\left(2 - \frac{1}{m}\right)$-approximation for $P||C_{\max}$.

Proof Let H_i be the last completion time on the i-th machine. Let k be the job that finishes last and determines C_{LS}. All the machines are busy when k starts its processing. Therefore $\forall i : H_i \geq C_{\text{LS}} - p_k$. For at least one machine (the one that processes J_k), it holds that $H_i = C_{\text{LS}}$.

$$\sum_j p_j = \sum_i H_i \geq (m-1)(C_{\text{LS}} - p_k) + C_{\text{LS}}$$

$$\sum_j p_j + (m-1)p_k \geq mC_{\text{LS}}$$

$$C_{\text{LS}} \leq \frac{1}{m}\sum_j p_j + p_k\frac{m-1}{m}$$

Consider an optimal schedule:

$$C_{\text{opt}} \geq \max_j p_j \geq p_k \qquad \text{some machine must process the longest job}$$

$$C_{\text{opt}} \geq \frac{1}{m}\sum_j p_j \qquad \text{if the load is perfectly balanced}$$

$$C_{\text{LS}} \leq C_{\text{opt}} + C_{\text{opt}}\frac{m-1}{m} = \left(2 - \frac{1}{m}\right) C_{\text{opt}}$$

Notably, this analysis is tight.

$$\frac{C_{\text{LS}}}{C_{\text{opt}}} = \frac{2m-1}{m} = 2 - \frac{1}{m}$$

3.3.2 LPT algorithm

List Scheduling is for an arbitrary ordering of the jobs. If the jobs are known in advance, we can determine the processing order offline. We sort the processing times descending, and apply list scheduling.

This provides a $\left(\frac{4}{3} - \frac{1}{3m}\right)$-approximation to $P||C_{\max}$

Proof We claim that if the optimal schedule has at most 2 jobs scheduled on every machine, then LPT is optimal. We can break this into the case where $n \leq m$, where there are fewer jobs than machines. In such as case, $C_{\max} = p_{\max}$.

Now consider the case where $m \leq n \leq 2m$. In the optimal schedule for $n = m + z$ where $0 < z \leq m$, the job J_k where $k \leq m - 2$ is scheduled alone on a machine, and the pair of jobs J_{m+k}, J_{m-k+1} are together for all $1 \leq k \leq z$...[2]

Assume for contradiction that for some input I, our claim is wrong. Denote $C^*(I)$ as the objective value for the optimal solution for I and $C_A(I)$ as the objective value as given by LPT. Let J_k be the job that decides $C_A(I)$. J_k must be the last job scheduled, and therefore the smallest; otherwise, we could remove from I all jobs shorter than J_k such that $C_A(I') = C_A(I)$ and $C^*(I') \leq C^*(I)$. Therefore, I' also contradicts the our claim, and wherein J_k is the shortest job. Our assumption from above can be shown as:

$$\frac{C_A(I)}{C^*(I)} > \frac{4}{3} - \frac{1}{3m}$$

In our analysis of LS, we showed that (and what follows):

$$C_A(I) \leq \frac{\sum_j p_j}{m} + \frac{p_k(m-1)}{m}$$

$$\frac{4}{3} - \frac{1}{3m} < \frac{C_A(I)}{C^*(I)} \leq \frac{\sum_j p_j}{mC^*(I)} + \frac{p_k(m-1)}{mC^*(I)}$$

Because $C^*(I) \geq \frac{\sum_j p_j}{m}$, it follows that:

[2] This part was erased while I was transcribing it

$$\frac{4}{3} - \frac{1}{3m} < 1 + \frac{p_k(m-1)}{mC^*(I)}$$
$$\frac{C^*(I)}{3}\left(1 - \frac{1}{m}\right) < \left(1 - \frac{1}{m}\right)$$
$$C^*(I) < 3 \cdot p_k$$

Therefore there are at most two jobs on each machine. However, we proved that LPT is optimal for such inputs, which contradicts our assumption.

This analysis is tight as shown on slide 13. Consider $n = 2m + 1$ jobs, such that $p = [2m - 1, 2m - 1, \ldots, m + 1, m + 1, \underbrace{m, m, m}_{3 \text{ such jobs}}]$. In this case, LPT will be perfectly balanced right before the last job, and will be penalized by exactly the approximation bound.

3.4 P - preemption

We can solve $P|\text{pmtn}|C_{\max}$ in polytime. Let $w = \max\left(\max_j p_j,\ \frac{1}{m}\sum_j p_j\right)$.

Consider the jobs by some arbitrary order. Schedule the jobs on the current machine (starting from the first machine). When M_i has been allocated w processing units, move onto the next machine, possibly preempting the last job.

3.5 P - precedence

For the problem $P|\text{prec}, p_j = 1|C_{\max}$, precedence is given as a DAG. On a single machine, any topological sort solves this optimally as $C_{\max} = n$. The problem is not complicated by much if we have non-unit jobs, but on parallel machines, it is $\mathcal{NP}$-hard.

We consider two special cases of precedence graphs: in-trees and out-trees. In an in-tree, each job has at most one predecessor (in-degree is at most 1). In an out-tree, each job has at most one consecutive (out-degree is at most 1).

3.5.1 Hu's Algorithm for in-trees

Given on slides 18-23 (including an execution example).

3.5.2 out-tree algorithm

Given on slide 24.

3.5.3 Other variants

in-forests and out-forests can be solved by introducing dummy jobs.

4 Next week

Due to the holiday, the lecture next week will be held 1500-1715 in PE203 (building at the southeast corner of campus).

References

[1] Michael L Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.