Resource Allocation Algorithms

Lecture 02

Lecture by Dr. Tami Tamir
Typeset by Steven Karas

2018-03-20
Last edited 18:14:38 2018-03-20

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

- Scheduling algorithms

1 Scheduling Algorithms

Some of the reference material will be uploaded to the course website later this week.

In scheduling theory, a set of jobs needs to be processed by a set of machines. Jobs need to be scheduled on machines to satisfy some objective function.

Problems can vary, from a single machine, to multiple machines, to asymmetric machines. Sometimes there is a transfer cost. Sometimes there will be a preemption cost (although typically infinite). Typically, the objective is to minimize the number of jobs that finish after their deadline (or some variant thereof).

Examples A single machine executing tasks.

Two identical machines executing tasks.

Two machines, one fast, one slow, executing tasks.

Exams with four questions, where each question is graded by a different grader. Questions can be graded in any order.

Now where questions need to be graded in a specific order.

Now where there's a cost to transfer exams between graders (but can be done in batches).

1.1 Notation

- 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 visual version of these definitions can be found on slide 6.

We denote K_j as the cost associated with job j. There are two main types of objective functions:

1. Minimize the maximal cost: $\min K_{\max} = \min \max_i K_i$
2. Minimize the sum of costs: $\min \sum_j K_j$

Example objective functions

- $\min C_{\max}$ - the makespan
- $\min \sum_j C_j$ - the sum of completion times
- $\min \sum_j w_j C_j$ - weighted sum of completion times

When jobs have release times, the service time is denoted as $F_j = C_j - r_j$.

- $\min F_{\max}, \quad \min \sum_j F_j, \quad \min \sum_j w_j F_j$

Out of the service time, the waiting time is denoted as $W_j = F_j - p_j$.

- $\min W_{\max}, \quad \min \sum_j W_j, \quad \min \sum_j w_j W_j$

We denote the lateness of a job as $L_j = C_j - d_j$, but more practical is the tardiness: $T_j = \max(0, L_j)$. We also denote the lateness indicator as $U_j = 0$ if $C_j \leq d_j$ else 1.

- $\min T_{\max}, \quad \min \sum_j T_j, \quad \min \sum_j w_j T_j$
- $\min L_{\max}, \quad \min \sum_j L_j, \quad \min \sum_j w_j L_j$
- $\min \sum_j U_j, \quad \min \sum_j w_j U_j$

There are also functions based upon the system performance. We denote the number of jobs processed at time t as $N_p(t)$. We denote the number of jobs waiting at time t as $N_W(t)$. We denote as I_k the idle time of machine k during the interval $[0, C_{\max}]$

In class examples Can be found on slides 11-13.

1.2 Equivalence of makespan, average process, and idle time

We will prove that some objective functions are equivalent. That is, that a schedule that is optimal for one is optimal for the others.

1. $\min C_{\max}$ - makespan
2. $\max \text{avg}(N_p)$ - the average number of processed jobs
3. $\min \sum_k I_k$ - the total idle time

Equivalence 1-2

$$\text{avg}(N_p) = \frac{\sum_j p_j}{C_{\max}}$$

Job j is processed for p_j units of time in the interval $[0, C_{\max}]$. The numerator is constant and independent on the schedule. Therefore, the maximum $\text{avg}\, N_p$ is achieved by minimizing the makespan.

Equivalence 1-3 The idle time of machine k is for those jobs scheduled on that machine:

$$I_k = C_{\max} - \sum_j p_j$$

$$\sum_k I_k = m \cdot C_{\max} - \sum_j p_j$$

Because $\sum_j p_j$ is constant and independent of the schedule, and m is similarly constant and independent. Therefore, we want to minimize $C_{\max}$ is set by the schedule.

1.3 Equivalence of completion, service, waiting, and lateness

Theorem 2 is from slide 15.

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

Note the constant terms in all of these.

These are also equivalent for the respective weighted objective functions, but the proof is left as an exercise to the reader.

1.4 More notation

A scheduling problem is defined by the 3-tuple $\alpha|\beta|\gamma$.

α represents the processing environment. Examples:

1 a single machine

P identical parallel machines

Q parallel machines with different rates

R unrelated parallel machines - p_{kj} defines the processing time of job j on machine k

O open shop scheduling - order is undefined between subtasks

F flow shop scheduling - order is well defined between subtasks

J job shop scheduling - the order is job-dependent, and will not be covered in this course

β represents additional constraints. Examples:

pmtn preemptions are allowed

prec precedence constraints. A DAG defines the allowed

r_j jobs cannot be processed prior to their release time

$t_j = 1$ all jobs have the same processing time

set-up comes with a pairwise matrix S_{j_1,j_2} that defines the set up time when switching between different jobs

γ represents the desired objective function. Examples of some problems:

$1||\sum_j C_j$ minimize average completion time on a single machine

$1|prec|L_{\max}$

1.5 Single machine problems

1.5.1 Shortest processing time

For the problem $1||\sum_j C_j$, we have an $O(n \log n)$ algorithm.

The SPT rule is to sort the jobs such that $p_1 \leq \ldots \leq p_n$. Process the jobs according to this order.

We will prove this is optimal for $1||\sum_j C_j$. The proof sketch follows from the fact that the completion time for each job is dependent on the completion time of all the jobs that completed before it.

$$C_1 = P_i; \quad C_2 = p_1 + p_2; \quad C_j = \sum_{i \leq j} p_i$$

$$\sum_j C_j = np_1 + (n-1)p_2 + \ldots + p_n = \sum_j (n-j+1)p_j$$

This is the product of two vectors, one of which is decreasing $(n, n-1, \ldots, 1)$. To minimize the product, the other vector should be increasing.

An alternative proof using an exchanging argument (as a matter of practicing different proof styles): Assume S is an optimal schedule not according to SPT. For some pair J_i, J_k of adjacent job, J_k is scheduled after J_i and $p_i > p_K$. Build a new schedule S' in which we swap the scheduled of J_i, J_k. We will show that S' is a better schedule (formally: $\sum_j C_j(S') < \sum_j C_j(S)$).

Let A be the set of jobs starting before J_i and J_k. Let B be the set of jobs starting after J_i and J_k.

$$\begin{aligned}
\sum_j C_j(S) &= \sum_{j\in A} C_j + \sum_{j\in B} C_j + C_i + C_k \\
&= \sum_{j\in A} C_j + \sum_{j\in B} C_j + (p_A + p_i) + (p_A + p_i + p_k) \\
\sum_j C_j(S) &= \sum_{j\in A} C_j + \sum_{j\in B} C_j + C_k + C_i \\
&\ldots
\end{aligned}$$

The remainder of the proof is algebraic and located on slide 24.

Preemptive variant on SPT For $1|r_j, \text{pmtn}|\sum_j C_j$, the Shortest Remaining Processing Time (SRPT) rule is optimal. This rule has us process the job with the shortest remaining time.

The complexity of this is $O(n \log n)$. There are at most n decision points, and we do $O(\log n)$ work for each released or preempted job.

This is optimal, proof is on slide 25.

Weights variant on SPT Details on slide 26.

Variant with setup time For the problem $1|\text{set-up}|C_max$, we define a $n \times n$ matrix where s_{ij} is the set up time required between processing i and j.

Without setup times, or identical setup times $\forall i, j \; s_i j = s$, any order is optimal ($C_{\max} = \sum_j p_j + (n-1)s$). For arbitrary setup times, the problem is $\mathcal{NP}$-hard.

The proof follows from a reduction from the traveling salesman problem, where the setup times are the distances between cities. A formal proof was presented on the board.

1.5.2 Earliest Due Date

For an instance with due dates and a given schedule, define the EDD rule as sorting the jobs by increasing d_j.

Note that $\sum_j T_j$ and $\sum_j T_j$ are $\mathcal{NP}$-hard. We will prove that EDD is optimal for $1||T_{\max}$ and $1||L_{\max}$. The proof sketch follows an exchange argument.

Assume S to be an optimal schedule that does not follow EDD. For some pair of adjacent jobs J_i, J_k, J_k is scheduled after J_i even though $d_i > d_k$. Construct a new schedule S' by swapping the schedule for this pair. We will show that $L_{\max}(S') \leq L_{\max}(S)$.

$$\begin{aligned}
L_k(S') = C_k(S') - d_k < C_k(S) - d_k = L_k(S) \\
L_i(S') = C_i(S') - d_i = C_k(S) - d_i < C_k(S) - d_k = L_k(S)
\end{aligned}$$

From this we see that $\max(L_k(S'), L_i(S')) < \max(L_k(S), L_i(S))$.

Let $L = \max\{L_j \mid j \text{ is not } i \text{ or } k\}$. L is identical in S and S'.

$$\begin{aligned}
L_{\max} = \max\{L, L_i, L_k\} \\
L_{\max}(S') \leq L_{\max}(S)
\end{aligned}$$

Optimality for $1||T_{\max}$ As above.

Variant with preemption for $1|r_j, \text{pmnt}|T_{\max}$, EDD is optimal. Proof is on slide 32.

Variant with release times This is $\mathcal{NP}$-hard, because intentionally leaving a single machine idle may be useful. An example of this can be found on slide 33.

We will prove that $1|r_j|T_{\max}$ is $\mathcal{NP}$-hard by showing a reduction from PARTITION.

Recall that the PARTITION problem is given a set A of n numbers such that $\sum_{j\in A} a_j = 2B$, we need to decide if there is a subset A' of A such that $\sum_{j\in A'} a_j = B$.

Given an instance for PARTITION, we will construct an instance for $1|r_j|T_{\max}$ such that $T_{\max} = 0$ if and only if A has a partition. The proof sketch is to schedule a job in the middle that arrives at B and is ready at $B+1$, and all the other elements arrive at 0 and need to be ready at $2B+1$.

For each item $a_j \in A$, let J_j be a job with $p_j = a_j$, $r_j = 0$, and $d_j = 2B+1$. Also let J_{n+1} be a job with $p_{n+1} = 1$, $r_{n+1} = B$, and $d_{n+1} = B+1$.

To achieve $T_{n+1} = 0$, J_{n+1} must be scheduled in $[B, B+1]$. Therefore, the schedule of J_{n+1} induces a partition.

1.5.3 Moore's Algorithm for number of late jobs

For $1||\sum_j U_j$, we define the following[1] algorithm[3]:

1. Order the jobs by EDD, call this A^*, and let R^* be empty.
2. If no job in A^* is late, A^*R^* is an optimal order.
3. Else, let k be the first job to be late in A^*.
4. Move to R^* the longest job among the first k jobs in A^*.
5. Update the completion times in A^* and go to step 2.

An example execution of this algorithm can be found on slide 37. The intuition as to why this algorithm is optimal is because one of the jobs in k has to be late, so we choose to make the worst one as late as possible.

2 Branch and bound

We will cover this now and finish it next lecture.

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.

3 Next lecture

Tomorrow the first homework will be published.

There will not be a lecture over the holiday. The next lecture will take place on April 10th. We will continue covering branch and bound.

References

[1] Peter Brucker and P Brucker. *Scheduling algorithms*, volume 3. Springer, 2007.

[2] David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms. 1997.

[3] J Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management science*, 15(1):102–109, 1968.

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

[1] We will have the full proof as a homework problem