Scheduling
Lecture 1: Scheduling on One Machine

Loris Marchal

1 Generalities

1.1 Definition of scheduling

- allocation of limited resources to activities over time
- *activities*: tasks in computer environment, steps of a construction project, operations in a production process, lectures at the University, etc.
  tasks, jobs,
- *resources*: processors, workers, machines, lecturers, rooms, etc.
  processors, machines
- *objective*: minimize total time, energy consumption, average service time

Many variations on the model, on the resource/activity interaction and on the objective.

- Typical example: organize production of a workshop, by making a schedule for machines. Making different objects may need different series of machines, with different processing time. The number of objects of each type (or the ratio) may be fixed, and we want to minimize the overall processing time (or the throughput).
- Another example: in an airport, allocate gates to airplanes, arriving at different gates, not all gates can accomodate all planes, some gates are further than others: how to minimize the time spent by passengers (taxiing and walking in the terminal).
- Other examples: schedule the activity of workers in a construction process, allocate rooms for lecture at the University, etc.
- Our context: in a computing system, we have a number of machines at hand. These machines may be close to each other, or linked with a limited communication network. We want to use this platform to execute an application, composed of several tasks. Tasks may have precedence constraints, or other type of constraints. Several users may share the platform, which might be widely distributed, or hierarchical.

1.2 Graham notation

Classes of scheduling problems can be specified in terms of the three-field classification $\alpha|\beta|\gamma$ where

- α specifies the machine environment,
- β specifies the job characteristics,
- γ and describes the objective function(s).

We will illustrate this notation on all following scheduling problems.

2 Scheduling with a single machine

Motivations:

- understand the complexity of the problem for various objectives
- some of actual systems offers flexibility and may well be modeled by a single machine (e.g. virtual machines)
- practice usual scheduling techniques

2.1 First example, with new objective, $1||\sum w_i C_i$, polynomial (Smith-ratio)

Objectives using C_i:

- Makespan $C_{\max} = \max C_i$ **most common objective**
- Total *flow* time: $\sum_{j=1}^{n} C_j$
- Weighted (total) flow time: $\sum_{j=1}^{n} w_j C_j$

Let us consider the problem: $1||\sum w_i C_i$,

- 1 machine
- no constraints on tasks (length p_i)
- Objective: weighted sum of completion times
- Intuitions:
  - put high weight first
  - put longer tasks last
- $\Rightarrow$ Order task by non-increasing Smith ratio: $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$

Proof:

- Consider a different optimal schedule S
- Let i and j be two consecutive tasks in this schedule such that $w_i/p_i < w_j/p_j$
- contribution of these tasks in S:
  $S_i = (w_i + w_j)(t + p_i) + w_j p_j$
- contribution of these tasks if switched:
  $S_j = (w_i + w_j)(t + p_j) + w_i p_i$
- we have
  $\frac{S_i - S_j}{w_i w_j} = \frac{p_i}{w_i} - \frac{p_j}{w_j}$
  Thus we decrease the objective by switching these tasks.

2.2 Adding due-dates

Other objectives in the Graham notations, using *due dates* d_j ((sometimes) appears in the job characteristics):

- *lateness*: $L_j = C_j - d_j$
- *tardiness*: $T_j = \max\{0, C_j - d_j\}$
- *unit penalty*: $U_j = 0$ if $C_j \leq d_j$, 1 otherwise

wich gives the following objectives:

- (total lateness=sum flow)
- maximum lateness: $L_{\max} = \max L_j$
- total tardiness $\sum T_j$
- total weighted tardiness $\sum w_j T_j$
- number of late activities $\sum U_j$
- weighted number of late activities $\sum w_j U_j$

2.2.1 EDF for $1||L_{\max}$

Let's study the problem $1||L_{\max}$: minimize the maximum lateness on one machine. The strategy which places first the jobs with the earliest deadlines is optimal (Earliest Deadline First). See details in http://www.seas.upenn.edu/~deepc/Courses/CO454/Lectures/lecture5.pdf.

2.2.2 Moore-Hodgson algorithm for $1||\sum U_i$

Example:

job	1	2	3	4	5
d_j	6	7	8	9	11
p_j	4	3	2	5	6

Tasks are sorted by non-decreasing d_i : $d_1 \leq \cdots \leq d_n$

- $A := \emptyset$
- For $i = 1 \ldots n$
  - If $p(A) + p_i \leq d_i$, then $A := A \cup \{i\}$
  - Otherwise,
    - Let j be the longest task in $A \cup \{i\}$
    - $A := A \cup \{i\} - \{j\}$

Optimal solution : $A = \{2, 3, 5\}$

Proof.
- Feasibility:

  We first prove that the algorithm produces a feasible schedule, that is, all task of A can be scheduled in this order without missing their deadline
  - By induction: if no task is rejected, ok
  - Assume that A is feasible, prove that $A \cup \{i\} - \{j\}$ is feasible too
    - all tasks in A before j: no change
    - all tasks in A after j: shorter completion
    - task i: let k be the last task in A: $p(A) \leq d_k$
      since task j is the longest: $p_i \leq p_j$, thus $p(A \cup \{i\} - \{j\}) \leq p(A) \leq d_k \leq d_i$
      (because tasks are sorted)
      That is, the new task i terminates earlier than k before j was rejected. Since $d_i \geq d_k$, this is enough.
- Optimality:

  Assume that there exists an optimal set O different from the set A_f output by the Moore-Hodgson algorithm
  - Let j be the first task rejected by the algorithm
  - We prove that there exists an optimal solution without j
  - We consider the set $A = \{1, \ldots, i-1\}$ at the moment when task j is rejected from A, and i the task being added at this moment
  - $A + i$ is not feasible, thus O does not contain $\{1, \ldots, i\}$
  - Let k be a task of $\{1, \ldots, i\}$ which is not in O

- Since the algorithm rejects the longest task, $p(O \cup \{k\} - \{j\}) \leq p(O)$, and by the same arguments than before, $O \cup \{k\} - \{j\}$ is feasible
- We can suppress j from the problem instance, without modifying the behavior of the algorithm or the objective

We can repeat this process, until we get the set of tasks scheduled by the algorithm. □

2.3 Adding release dates, $1|r_i|\sum C_i$

All jobs are not released at the same time, but job j is available starting at time r_j.

$1|r_i|\sum C_i$ is NP-complete (admitted)

When adding preemption, we can derive an optimal algorithm for $1|r_i, pmtb|\sum C_i$, which serves as a basis for a 2-approximation algorithm for the initial problem.

See details in http://www.seas.upenn.edu/~deepc/Courses/CO454/Lectures/lecture4.pdf.