Scheduling
Lecture 1: Introduction

Loris Marchal

1 My first scheduling problem

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.
- *resources*: processors, workers, machines, lecturers, rooms, etc.
- *objective*: minimize total time, energy consumption, average service time

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

1.2 Small scheduling problem, to introduce the vocabulary

- n *jobs* (or tasks) $j = 1, \dots n$,
- r renewable *resources* $i = 1, \dots, r$
- R_k: amounts of resource k available at any time
- activity j processed for p_j time units, using an amount $r_{j,k}$ of resource k
- Integer numbers

Objective: find *starting time* S_j (or $\sigma(j)$) for each activity, such that

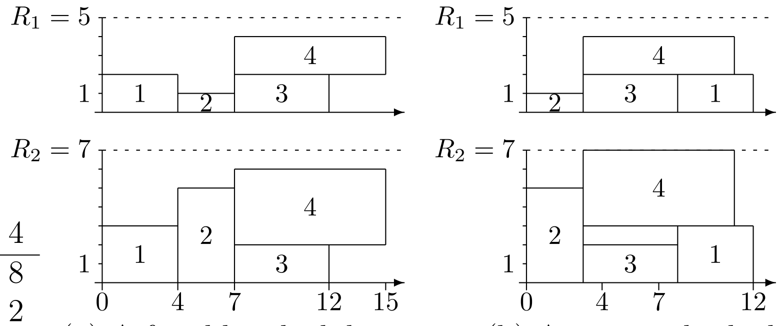
- At each time, the total resource demand is less than (or equal to) the resource availability for each resource
- Objective: *Makespan* $C_{\max} = \max C_j$ is minimized, with $C_j = S_j + p_j$
- S defines a *schedule*

- S is called *feasible* if all resource constraints are fulfilled

Exemples:

- Road construction, resource 1 is the number of available trucks, and resource 2 is the number of available caterpillars (or excavators). Jobs involve a given number of each machine.
- 2 resources with $R_1 = 5$ and $R_2 = 7$
- 4 jobs

j	1	2	3	4
p_j	4	3	5	8
$r_{j,1}$	2	1	2	2
$r_{j,2}$	3	5	2	4



(a) A feasible schedule (b) An optimal schedule

NB: these figures are called Gantt charts.

1.3 Common additionnal constraints

- precedence: j cannot start before i is completed
  precendence constraints are often modeled as a Directed Acyclic Graph (concept of predecessor and successor in this graph)
- communication delays $d_{i,j}$ is the delay between the completion of i and the starting time of j

2 Processor scheduling & 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.

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

- 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.

4 More machines, example of $P|prec|C_{\max}$, NP-completeness and Graham 2-approximation algorithm

More machines in the Graham notation:

- P parallel identical
- Q uniform machines
  each machine has a given speed $speed_i$, and all jobs have a size $size_j$, the processing time is given by $size_j/speed_i$
- R unrelated machines
  the processing time of job j on machine i is given by $t_{i,j}$, without any other constraints

- P: identical parallel machines
- prec: precedence constrants between tasks
- $C_{\max}$: minimizing the maximum makespan

Rersults:

- NP-complete
- reduction to 2-partition (or 3-partition, $\rightarrow$ unary NP-complete)

4.1 Recall on NP-completeness

Polynomial problems:

- a solution to a scheduling problem is a function h:
  - x is the input (parameters)
  - $h(x)$ is the solution (starting times, etc.)
- $|x|$ defined as the length of some encoding of x
  - usually, binary encoding: integer a encoded in $|a|_2 = \log_2 a$ bits
- Complexity of an algorithm computing $h(x)$ for all x: running time
- An algorithm is called polynomial, if it computes $h(x)$ for all x it at most $O(p(|x|_2))$ steps, where P is a polynomial
- A problem is called polynomial if it can be solved by a polynomial algorithm

Pseudo-polynomial problems

If we replace the binary encoding by an unary encoding (integer a encoded with size $|a|_1 = O(a)$): we can solve more difficult problems in time polynomial with $|x|$.

- An algorithm is pseudo-polynomial if it solves the problem for all x with a number of steps at most $O(p(x))$ steps, where P is a polynomial.

Example:
An algorithm for a scheduling problem, whose running time is $O(p_j)$ is pseudo-polynomial.

P and NP

- Decision problems
- To each optimization problem, we can define a decision problem
- P: class of polynomially solvable decision problems
- NP: class of polynomially *checkable* decision problems
  for each "yes"-answer, a certificate exists which can be used to check the answer in polynomial time

- Decisions problems of scheduling problems belongs to NP
- $P \subseteq NP$. $P \stackrel{?}{=} NP$ still open

NP-complete problems

- a decision problem Q is NP-complete if all problems in NP can be polynomially reduced to Q
- if any single NP-complete decision problem Q could be solved in polynomial time then we would have P = NP.
- To prove that a problem is NP-complete: reduction to a well-known NP-complete problem
- Weakly NP-complete (or binary NP-complete): strongly depends on the binary coding of the input. If unary coding is used, the problem might become polynomial (pseudo-polynomial).
  - 2-Partition vs 3-Partition

How to solve NP-complete problems ?
Exact methods:

- Mixed integer linear programming/Constraint Programming
- Dynamic programming
- Branch and bound methods (A^*)

(usually limited to small or simple instances)

Approximate methods:

- Heuristics (no guarantee)
- Approximation algorithms

Approximation algorithms
Consider a minimization problem. On a given instance x,
$f(x)$: value of the objective in the solution given by the algorithm
$f^*(x)$: optimal value of the objective

An algorithm is a *ρ-approximation* if for any instance x , $f(x) \leq \rho \times f^*(x)$

APX class: problems for which there exists a polynomial-time ρ-approximation algorithm, for some $\rho > 0$

An algorithm is a *PTAS* (Polynomial Time Approximation Scheme) if for any instance x and any $\epsilon > 0$, the algorithm computes a solution $f(x)$ with $f(x) \leq (1+\epsilon) \times f^*(x)$ in time polynomial in the problem size.

An algorithm is a *FPTAS* (Fully Polynomial Time Approximation Scheme) if for any instance and any $\epsilon > 0$, it produces a solution $f(x)$ such that $f(x) \leq (1+\epsilon) \times f^*(x)$, in time polynomial in the problem size and in 1ϵ.

4.2 Graham list scheduling approximation

A *list scheduling* algorithm is a heuristic which never leaves a processor *idle* when there is some *free* tasks to schedule.

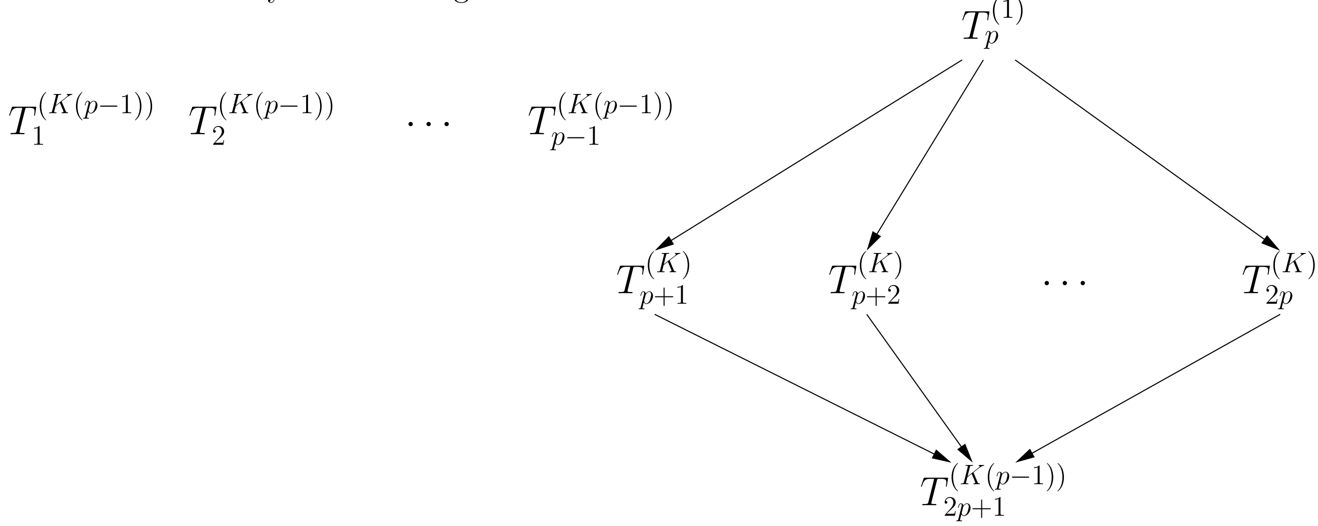
- Theorem: Any list scheduling heuristic gives a schedule, whose makespan is at most $2 - 1/p$ times the optimal.
- Lemma: there exists a precedence path Ψ such that
  $\text{Idle} \leq (p-1) \times w(\Psi)$
  - Consider the task with maximum termination time T_1
  - Let t_1 be the last moment before $\sigma(T_1)$ when a processor is not active
  - Since a processor is inactive at time t_1, there exists a task T_2, finishing at time t_1 (since at time t_1, a new task can be started on the idle processor, freed by the completion of T_2). T_2 is an ancestor of T_1, otherwise T_1 would T_1 would be free and scheduled at time t_1 (or before).
  - We iterate the processor and build up a path of dependent tasks Ψ.
  - All idle times occur during the processing of the tasks on this dependency path, and there are at most $p-1$ processors, which concludes the proof of the lemma.
- Notice that $pC_{\max} = \text{Idle} + \text{Seq}$, with $\text{Seq} = \sum w(T_i)$
- We also have $\text{Seq} \leq pC_{\max}^{\text{opt}}$, thus $C_{\max} \leq \big((p-1) \times w(\Psi)\big) + \big(pC_{\max}^{\text{opt}}\big)$
- We also have $w(\Psi) \leq C_{\max}^{\text{opt}}$, qed.

4.3 The approximation bound is tight

Let $K > 0$ be some large integer. Consider the following problem:

- $p-1$ tasks $T_1, \dots T_{p-1}$ of weight $K(p-1)$,
- a task T_p of weight 1,
- p tasks $T_{p+1}, \dots, T_{2p}$ of weight K
- one task T_{2p+1} of weight $K(p-1)$.

as discribed by the following DAG:



A list scheduling heuristic will schedule all tasks $T_1, \ldots, T_p$ at time 0. Then $p-1$ of the tasks of weight K will be scheduled on the processor holding T_p, and the last one on another processor. The last task then starts at time $K(p-1)+K$, and last for $K(p-1)$ time-units, reaching a makespan of:

$$C_{\max}^{list} = Kp + K(p-1) = K(2p-1)$$

In the optimal schedule, we delay the processing of tasks $T_1, \ldots, T_{p-1}$ to the end: a single processor processes T_p, then all processors process one task of weight K, then all tasks of weight $K(p-1)$ are processed in parallel. This gives a makespan of

$$C_{\max}^{opt} = 1 + K + K(p-1) = Kp + 1$$

$$\frac{C_{\max}^{list}}{C_{\max}^{opt}} \geqslant \frac{K(2p-1)}{Kp+1} = \frac{2p-1}{p} - \frac{2p-1}{p(Kp+1)} = (2 - \frac{1}{p}) - o(1/K).$$

5 More objectives, example of $1||\sum U_i$, Moore-Hodgson algorithm

Other objectives in the Graham notations:

- Using C_i
  - Total *flow* time: $\sum_{j=1}^{n} C_j$
  - Weighted (total) flow time: $\sum_{j=1}^{n} w_j C_j$
- With *due dates* d_j (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:

  - 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 $\sum w_j U_j$

- With *release dates* r_j: flow becomes $C_j - r_j$
  (online, stretch)

One machine, minimize the number of late jobs

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:
  - 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 exist 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 $A = \{1, \dots, i-1\}$ 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 $\{1, \dots, i\}$

– Let k be a task of $\{1, \dots, i\}$ 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. □

6 Shop and Job-Shop problems, and other variants

6.1 Shop scheduling

Jobs consist in several operations, to be processed on different resources.

General shop scheduling problem:

- jobs $J_1, \dots J_n$
- processors $P_1, \dots, P_m$
- J_j consists in n_j operation $O_{1,j}, \dots, O_{n_j,j}$
- two operations of the same job cannot be processed at the same time
- a processor can process one operation at a time
- operations $O_{i,j}$ has processing time $p_{i,j}$ and makes use of processor $\mu_{i,j}$
- arbitrary precedence pattern

6.2 Job-Shop scheduling problem

- chain of precedence constraints:

$$O_{1,j} \to O_{2,j} \to \dots \to O_{n_j,j}$$

<u>flow-shop scheduling problem:</u>

- special job-shop scheduling problem

- $n_j = m$ for all j, and $\mu_{i,j} = P_i$ for all i, j:
  operation $O_{i,j}$ must be processed by P_i

open-shop scheduling problem:

- like a flow-shop, but no precedence constraints

Graham notations:

- J job-shop
- F flow-shop
- O open-shop

6.3 Other variants

Other job characteristics in Graham notation:

- $p_j = 1$ or $p_j = p$ or $p_j \in 1, 2$: restricted processing times
- prec : arbitrary precedence constraints
- intree: (outtree) intree (or outtree) precedences
- chains: chain precedences
- series-parallel: a series- parallel precedence graph

Other types of scheduling problems, that we will discover in the next lectures:

- Online problems
  - contrarily to offline, information about future jobs is not known in advance
  - competitive ratio: ratio to the optimal offline algorithm

- Distributed scheduling
  - use only local information

- Multi-criteria scheduling
  - several objectives to optimize simultaneously
  - and/or several users, link with game theory

- Cyclic scheduling
  - infinite but regular pattern of tasks