JPOC'11
June 24th, 2019 Metz

On Scheduling Problems

Tutorial by:

Imed Kacem

imed.kacem@univ-lorraine.fr

Professor –LCOMS Lab –France

UNIVERSITÉ DE LORRAINE

Outline

- **Scheduling problems**
  - Structure, complexity and possible applications
  - Standard used approaches (exact and heuristic)
- **Polynomial approximation**
  - Constant approximation
  - Polynomial Schemes (PTAS, FPTAS)
- **Effective approximation algorithms for scheduling under non-availability constraints**
- **Conclusions**

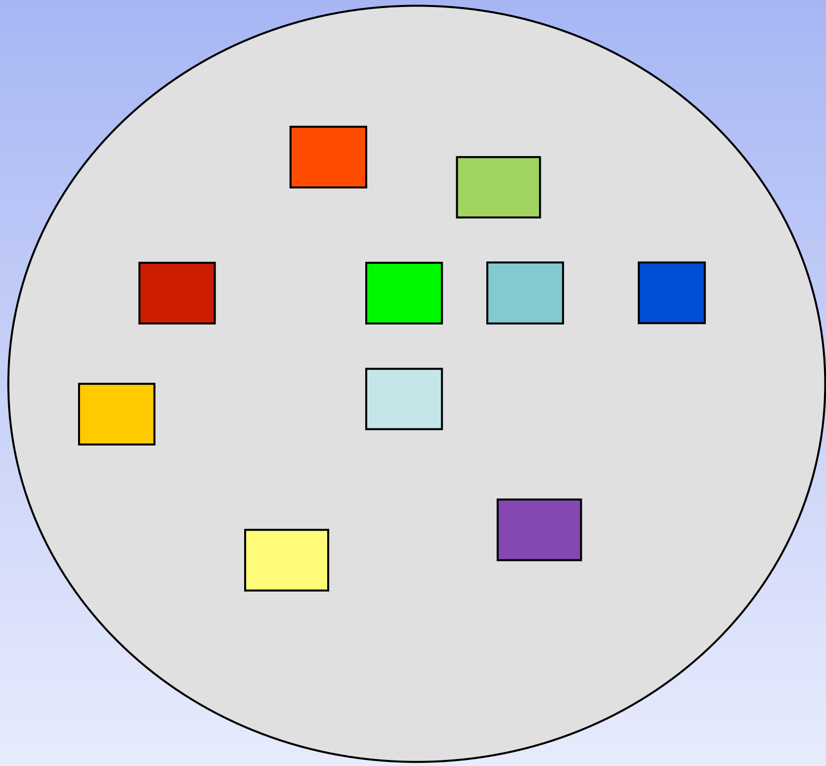
Structure of scheduling problems

Discrete optimization problems: definition

Solving a discrete optimization problem can be reduced to the selection of a solution among a set B of feasible solutions. The set B is finite and its cardinal depends on the problem size N.

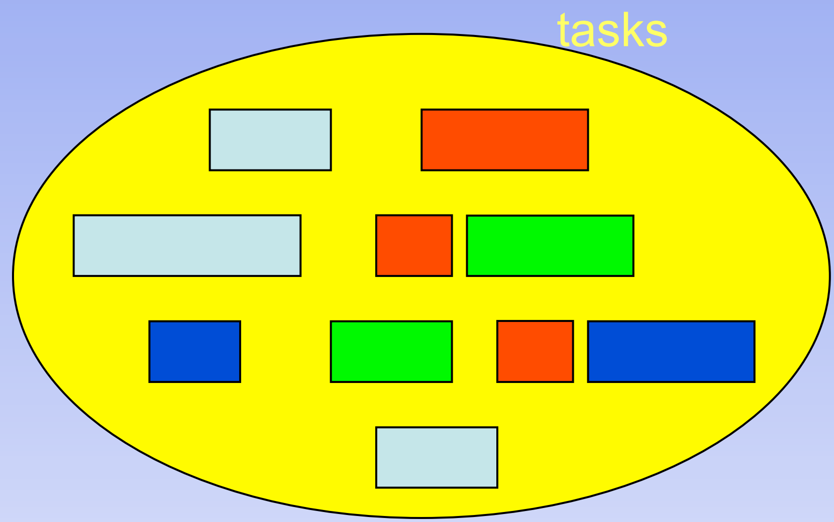
We have to find one optimal solution which can provide the maximum of effectiveness (according to a set of criteria or objectives).

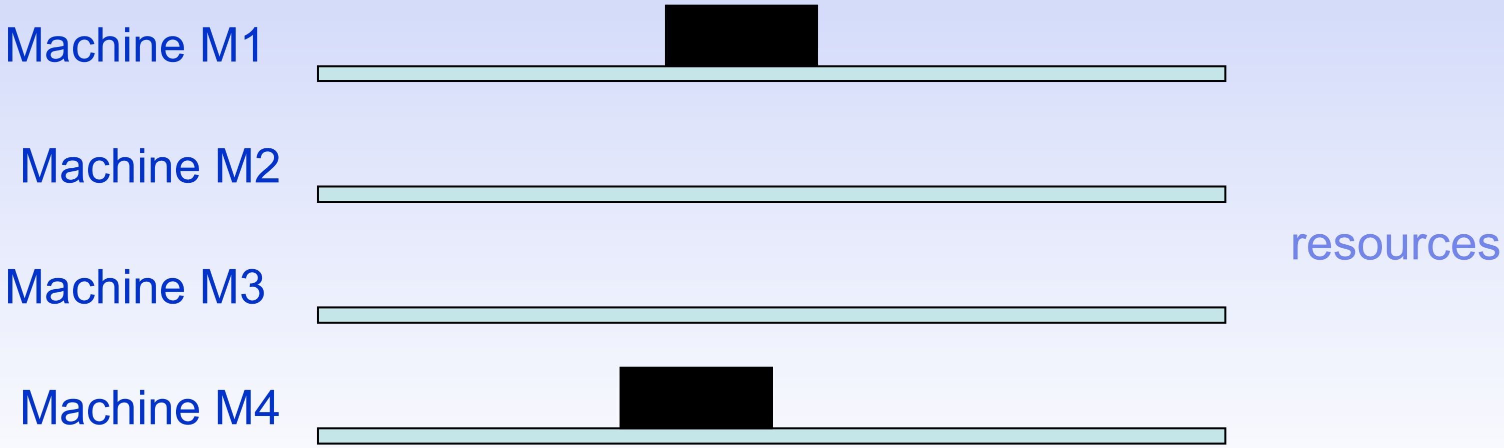
Set B of feasible solutions



Scheduling problems: a definition

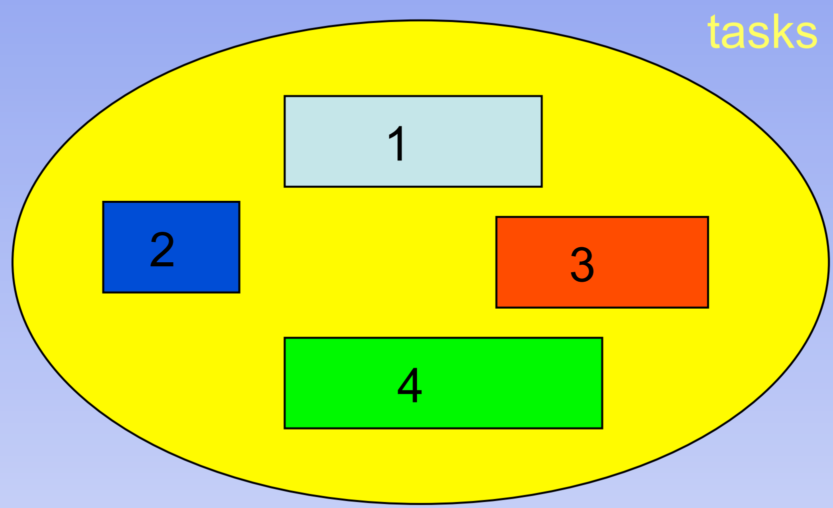
solving a scheduling problem can be reduced to the organization of a set of activities (jobs or tasks) by exploiting the available capacities (resources). This execution has to respect different technical rules (constraints) and to provide the maximum of effectiveness (according to a set of criteria or objectives).





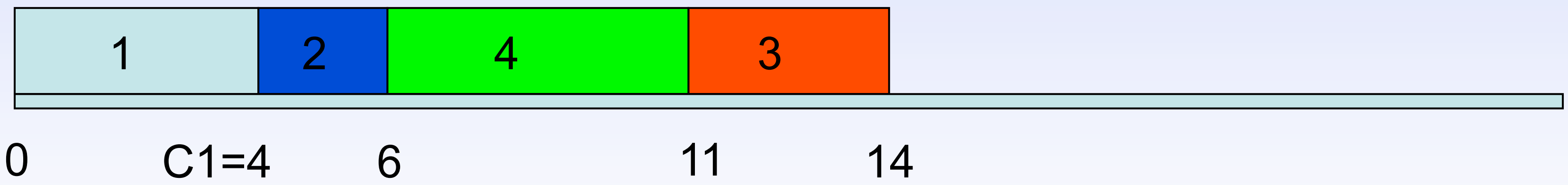
Scheduling problems: a first example 1//Sum(Tj)

- We have 4 jobs to be performed on a single machine,
- Every job j has a processing time pj and a due date dj,
- The objective is to schedule all the jobs by minimizing the total tardiness T,
- Tardiness of job j is equal to Tj=max(0, Cj-dj), where Cj is the completion time of job j.



Job j	pj	dj
1	4	5
2	2	4
3	3	7
4	5	10

Feasible schedule: T=0+2+7+1=10



Example: how to formulate 1//Sum(Tj)?

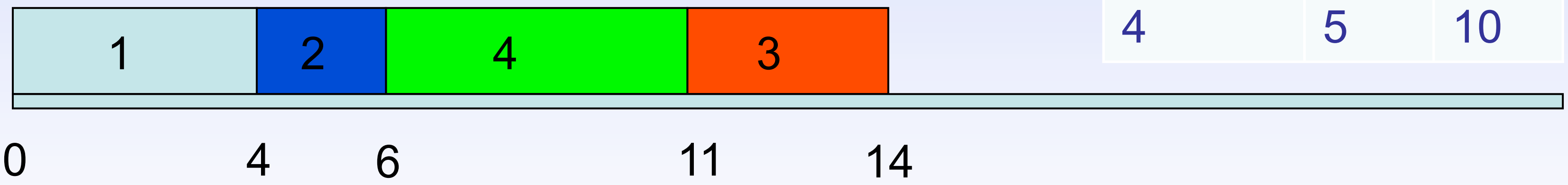
- Decision: for every job i and job j we must define the order;
- We could use a binary variable $X_{i,j}$ which is equal to 1 if job i is performed before job j (and equal to 0 otherwise);
- Tardiness of job j is equal to Tj=max(0, Cj-dj), where Cj is the completion time of job j, can be linearized;
- Question: find a constraint to compute Tj according to $X_{i,j}$;

Exercise: find an ILP model

Job j	pj	dj
1	4	5
2	2	4
3	3	7
4	5	10

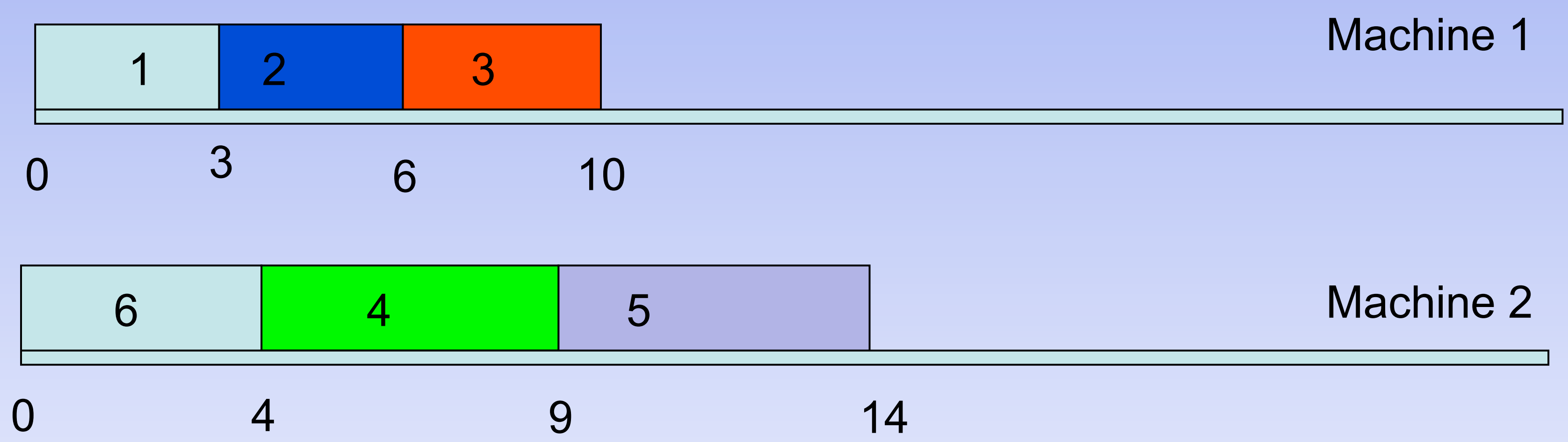
Feasible solution:

$X_{1,2}=1$; $X_{1,4}=1$; $X_{1,3}=1$; $X_{2,1}=0$; $X_{2,3}=0$…



How to present a schedule?

GANTT Diagram

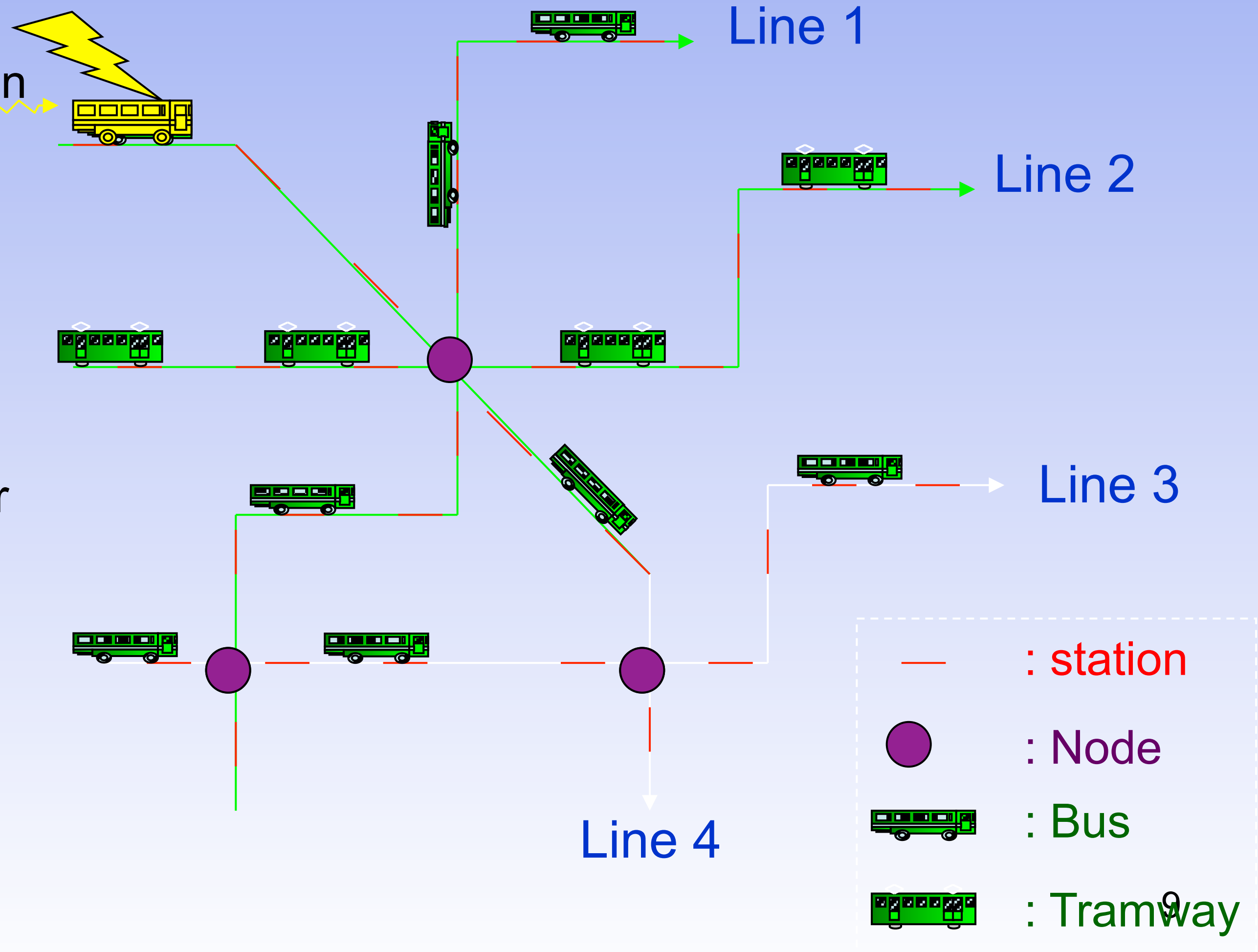


Application: the organization of a transportation network

- organize the transportation network

Objective

- schedule the trips in order to:
  - Maximize the service quality
  - Respect the temporal contraints

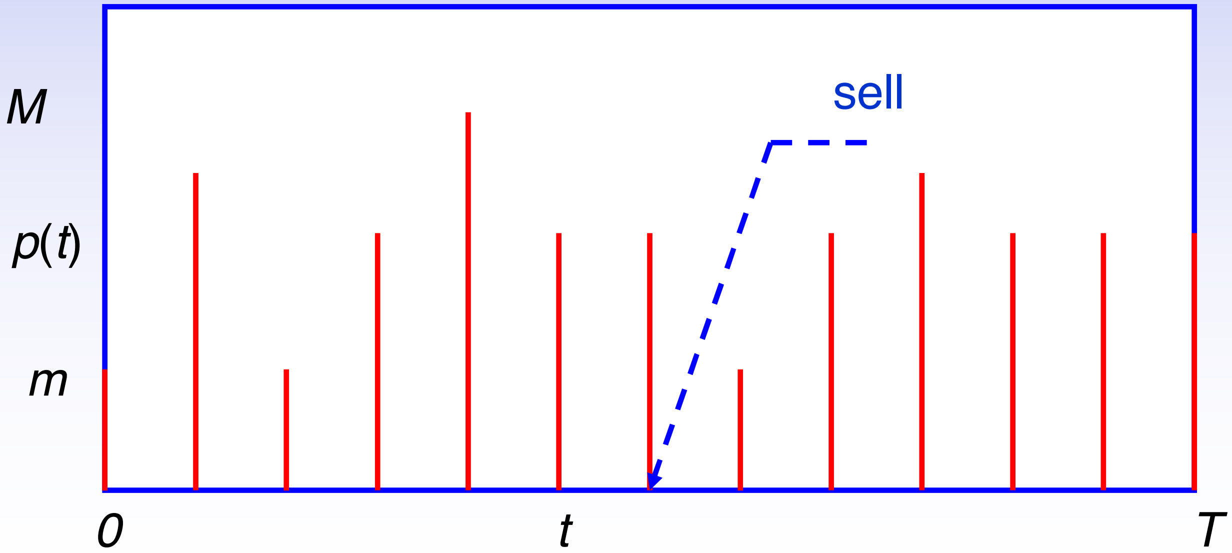


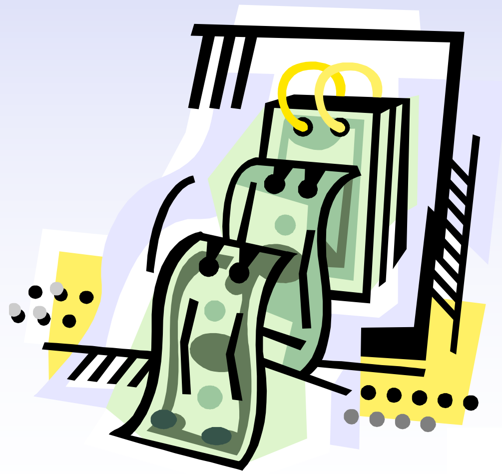
Online Trading Problems in Financial Markets

We aim to buy electronic market places at low and to sell them at high prices (see *Schmidt*, EJOR).

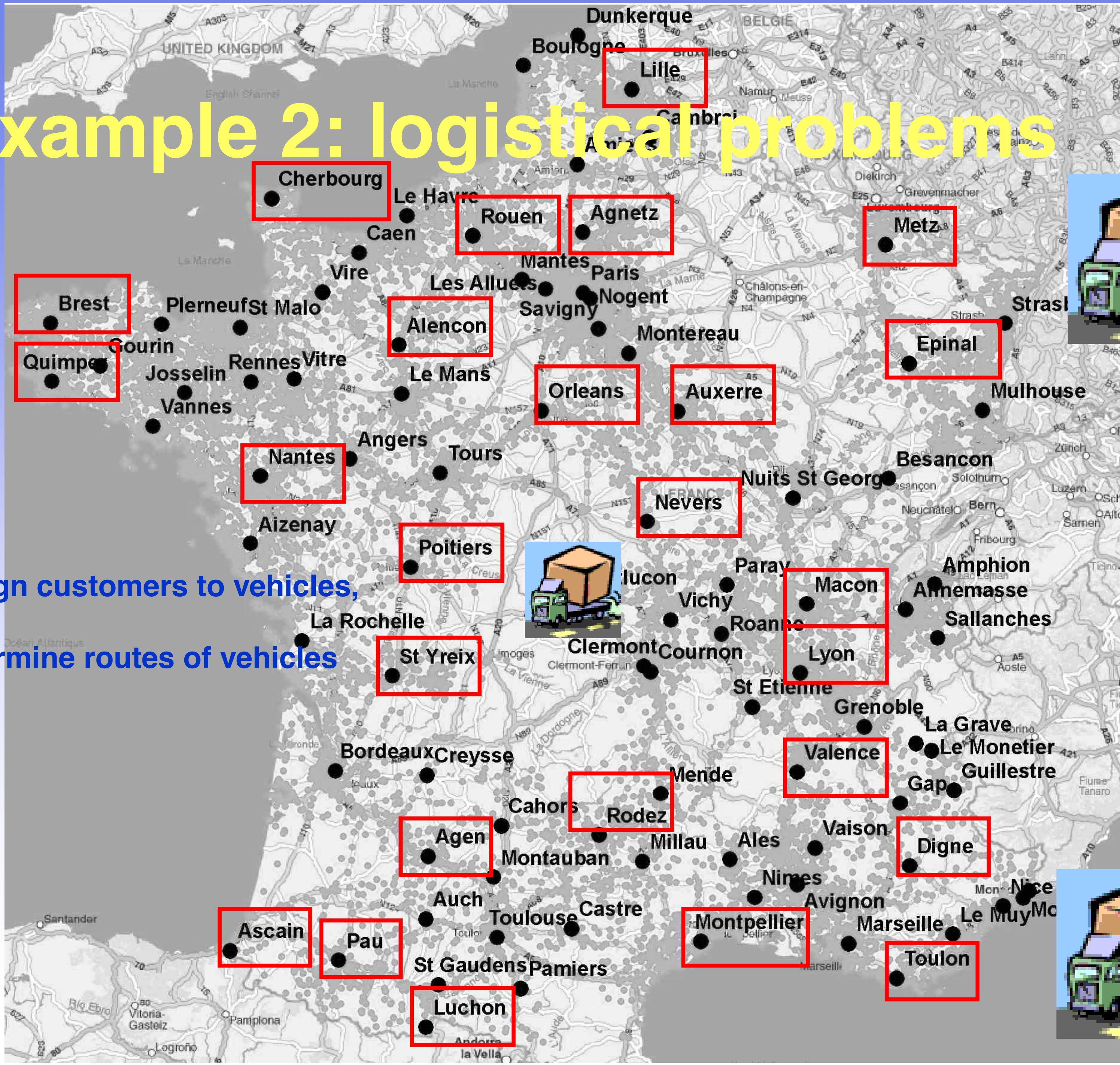
We (the trader) own an initial asset A at time $t = 0$. We can obtain dynamically a price quotation $m \leq p(t) \leq M$ at every time $t = 1, 2, \ldots, T$. Parameters m and M are known in advance. Hence, we have to decide at time t if accept this price for selling. Trading is closed once we accepted some $p(t)$. If we did not accept any price until time $T - 1$ we will be obliged to accept the last proposed price at time T .

General problem: buy and sell at given periods.





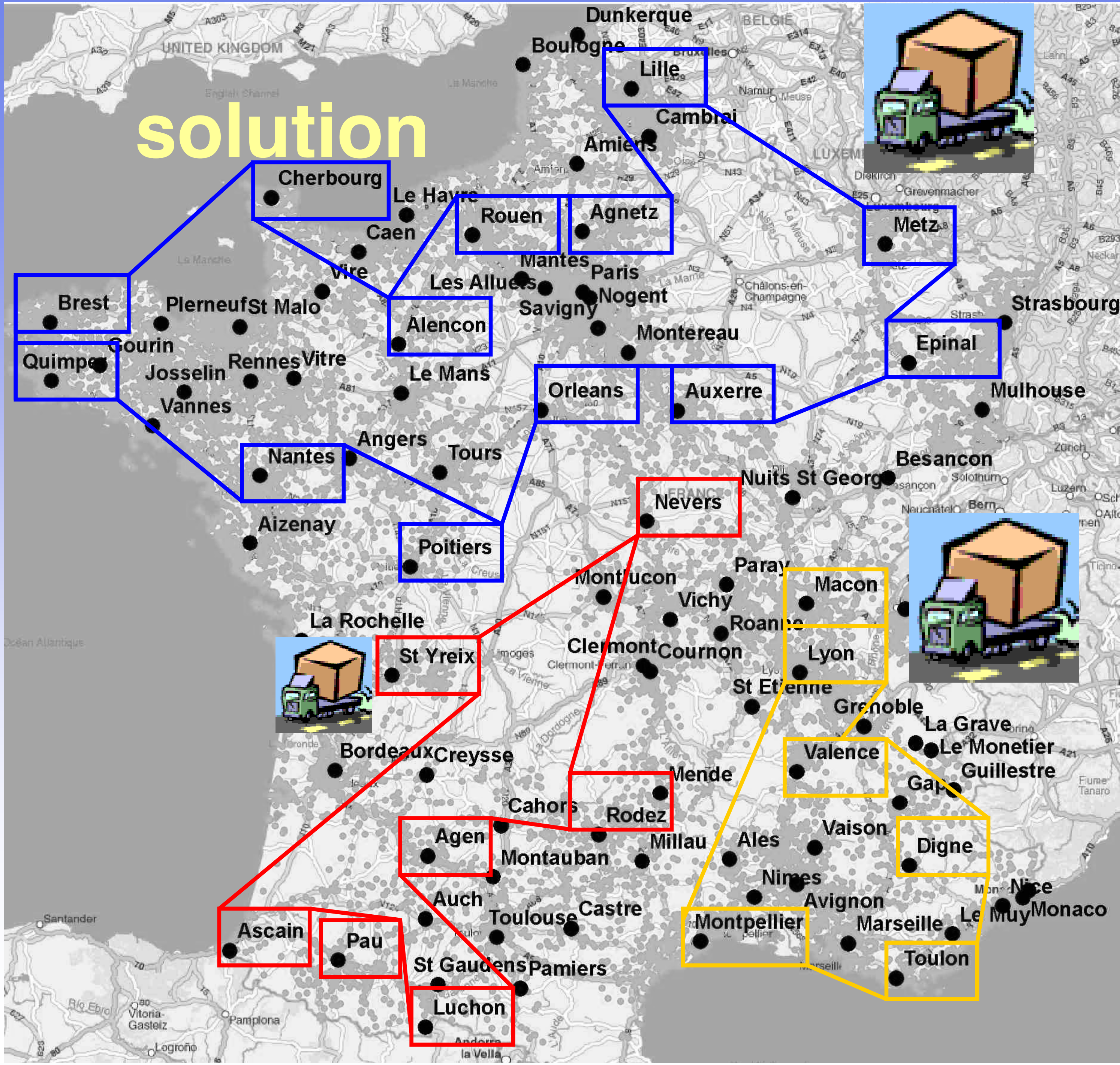
Example 2: logistical problems



Assign customers to vehicles,

Determine routes of vehicles

solution

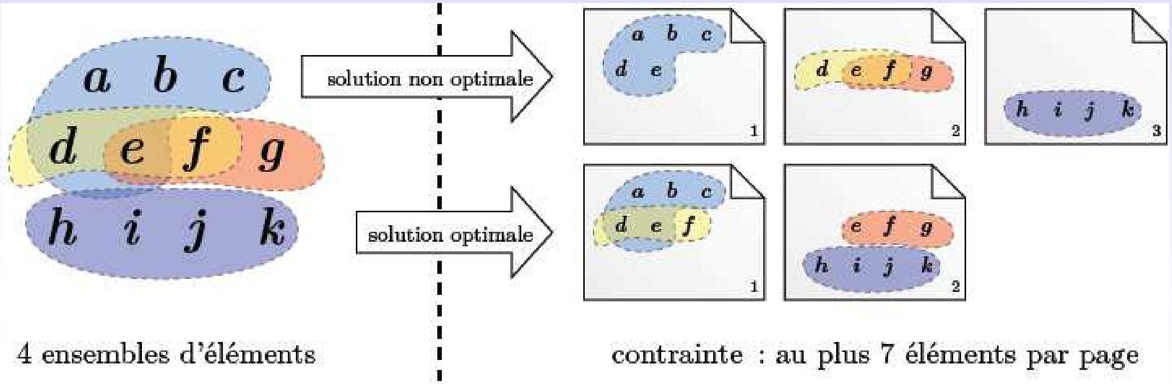

Dunkerque
Boulogne
Lille
Cambrai
Amiens
Cherbourg
Le Havre
Caen
Rouen
Agnetz
Metz
Vire
Mantes
Paris
Les Allues
Nogent
Savigny
Brest
Plerneuf
St Malo
Alencon
Strasbourg
Montereau
Epinal
Gourin
Quimper
Josselin
Rennes
Vitre
Le Mans
Vannes
Orleans
Auxerre
Mulhouse
Angers
Nantes
Tours
Besancon
Nuits St Georg
Nevers
Aizenay
Poitiers
Paray
Montlucon
Macon
Vichy
Roanne
La Rochelle
St Yreix
Clermont
Cournon
Lyon
St Etienne
Grenoble
La Grave
Le Monetier
Bordeaux
Creysse
Valence
Guillestre
Mende
Gap
Cahors
Rodez
Agen
Vaison
Millau
Ales
Digne
Montauban
Nimes
Nice
Auch
Avignon
Castre
Toulouse
Le Muy
Monaco
Ascain
Montpellier
Marseille
Pau
Toulon
St Gaudens
Pamiers
Luchon
UNITED KINGDOM
BELGIE
LUXEM
Océan Atlantique

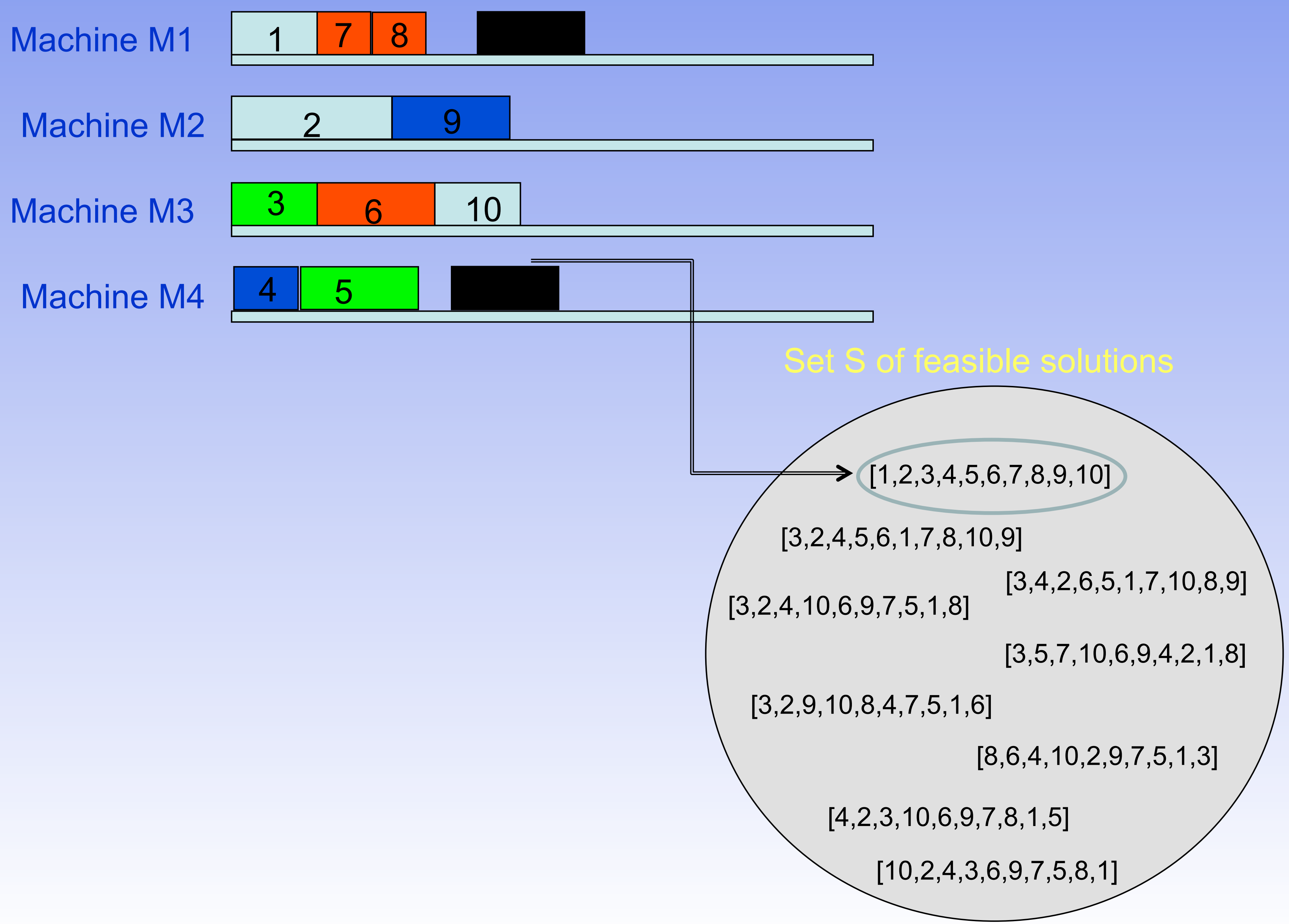
Scheduling for New Technologies: Touch Screen

Data visualisation in touch screens

Optimization in some specific keyboards

Grange, Kacem, Martin
Comp. & IE, 2018




Machine M1
1
7
8
Machine M2
2
9
Machine M3
3
6
10
Machine M4
4
5
Set S of feasible solutions
[1,2,3,4,5,6,7,8,9,10]
[3,2,4,5,6,1,7,8,10,9]
[3,4,2,6,5,1,7,10,8,9]
[3,2,4,10,6,9,7,5,1,8]
[3,5,7,10,6,9,4,2,1,8]
[3,2,9,10,8,4,7,5,1,6]
[8,6,4,10,2,9,7,5,1,3]
[4,2,3,10,6,9,7,8,1,5]
[10,2,4,3,6,9,7,5,8,1]

Scheduling problems: Main components

Resources

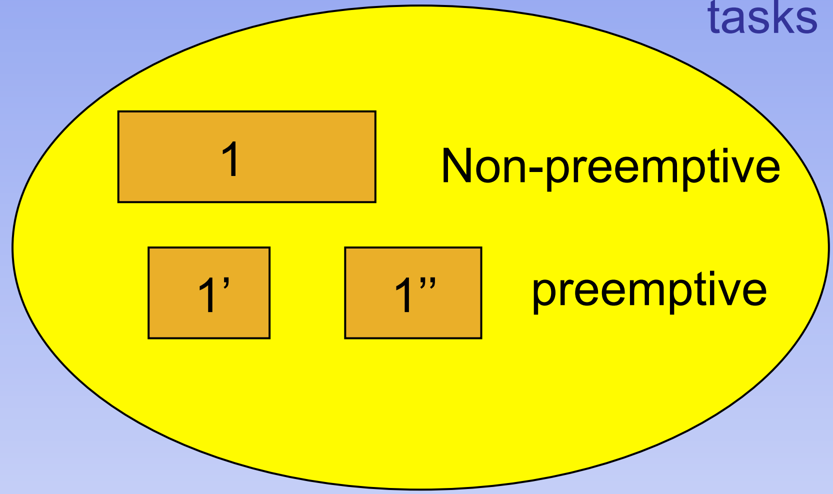
- Resources: can be used once,

Example: money, energy, …

- They can be renewed in other cases,

Example: machines, operators,..

tasks



Constraints

- They can be related to resources: non-availability constraints, capacity, placement in the shop, …
- They can be related to time: release times, due dates, precedence, delivery times …
- We can also distinguish internal and external constraints.

Objectives

- Mono-criterion: here, the objective can be formulated in a single function,
- Multiple-criteria: here, we have several objectives to be optimized at the same time…

Scheduling problems: Notations

Polynomial (exercise)

1|rj|Cmax ➔ use FIFO (jobs are sorted in increasing order of rj).
1|qj|Lmax ➔ use Jackson (jobs are sorted in decreasing order of qj).
1||Sum(wjCj) ➔ use WSPT (jobs are sorted in increasing order of pj/wj).
P||Sum(Cj) ➔ use SPT (jobs are sorted in increasing order of pj).

Notation α|β|γ

α: description of the resources
β: description of tasks (precedence, release times, deadlines, preemption, no-wait…
γ: objective function(s)

NP-hard

1|rj, qj|Lmax ➔ NP-hard in the strong sense.
2||Cmax ➔ NP-hard in the ordinary sense.
1|rj|Sum(Tj) ➔ NP-hard in the strong sense.

1||Sum(Tj) ➔ NP-hard in the ordinary sense.

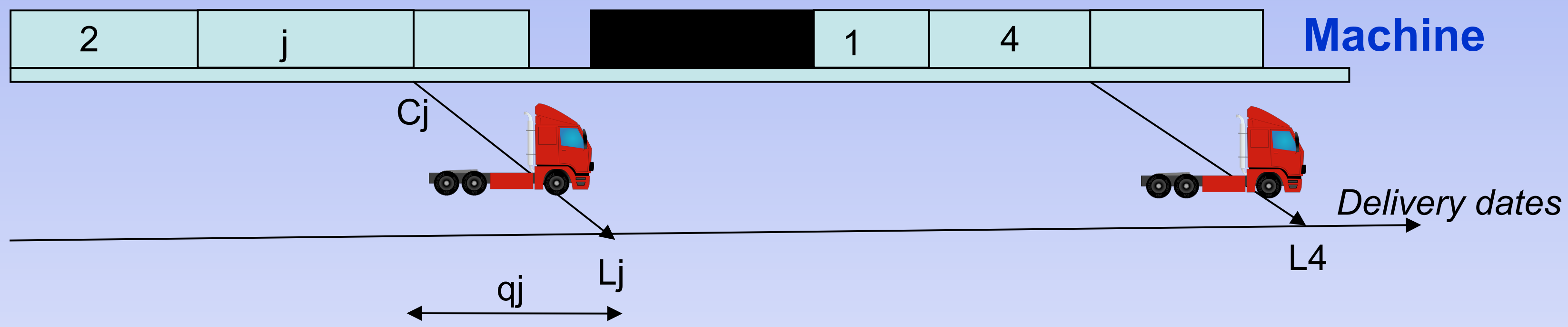
Open

1||Sum(wjTj) ➔ Strongly or weakly NP-hard?

1|rj, pre|Sum(wjCj) ➔ NP-hard in the stronge sense?

Delivery times: illustration

1, h1|qj|Lmax



Lj=Cj+qj

Sufficient transportation resources

Complexity

- Let us consider a problem which consists in finding a sequence of N jobs on a set of machines with the aim of minimizing the total tardiness where every job is characterized by a processing time and due date.
- Obviously, we have N possibilities for determining the job to be scheduled in the first position. Then, N-1 remaining jobs will be candidate to the second position. More generally, we will have (N-i+1) possible remaining jobs to put in the i-th position. Hence, we deduce that there are $N.(N-1)...(N-i+1)...2.1$ possible sequences (i.e., $N!$ possibilities).
-To select one of these solutions, we need to measure the performance of each solution.
- Let suppose for example that N = 10 and we have a computer able in 0.1 *s* to explore the search space (constituted of the $N!$ sequences), to evaluate these sequences and to select the best one. Moreover, let us assume that the same computer spends the same time for evaluating sequences of different values of N.

Complexity

N	*Computation time*
15	**4 days**
20	**220 centuries**
25	**136 billions of years!**

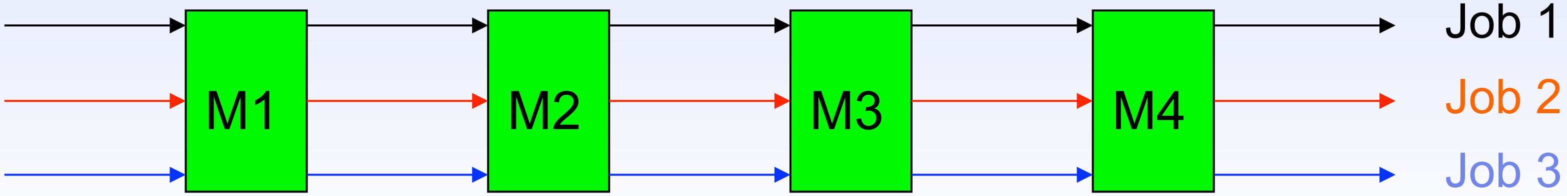
-This phenomenon is called the combinatorial explosion.
- Most of scheduling (and discrete optimization) problems are combinatorial and belong to a special class of hard problems: the *NP-Hard* class.
- Most of researchers think we cannot construct polynomial time algorithms for solving problems of the *NP-hard* class (see Garey & Johnson 1979).

Diversity

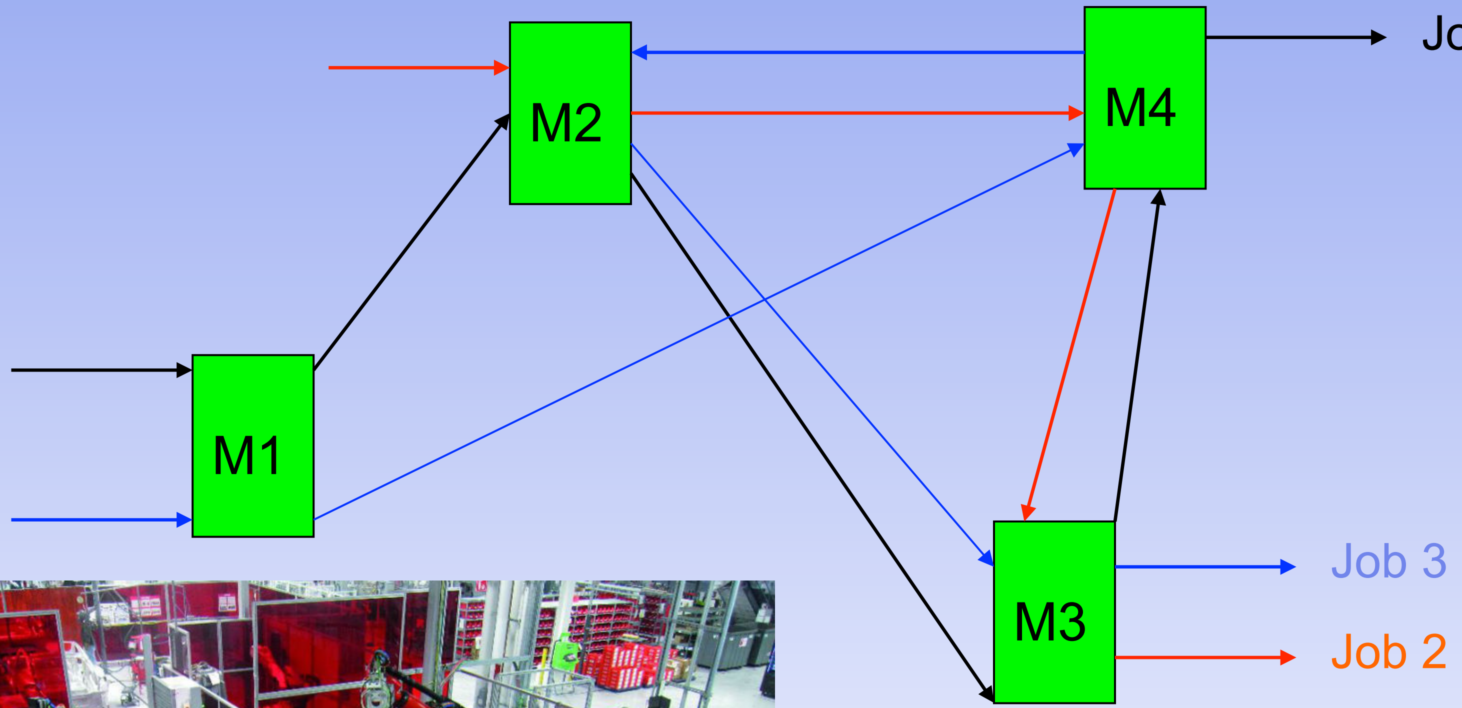
-The diversity leads to the specificity of any discrete optimization problem.
- Consequently, we cannot imagine a generic solution.
- Such a diversity is due to the existence of different and numerous industrial configurations and systems to optimize.
- Numerous classes of problems are related to different structures and types of systems. For example, in scheduling we can have several types of shops:
    - -Flow-shops
    - -Job-shops
    - -Open-shops
    - -Flexible job-shops

Flow-shop

- For every job, all the operations follow the same sequence of machines.
- This type is generally met in the production systems where the objective is to maximize the produced quantity.
- In a flow-shop problem, we have to schedule a set of jobs on a set of machines.
- Every job is carried out by passing by all the resources.
- The order must be identical for all the jobs.
- The literature shows other possible extensions and types of this configuration (hybrid flow-shop, permutation job-shop…).



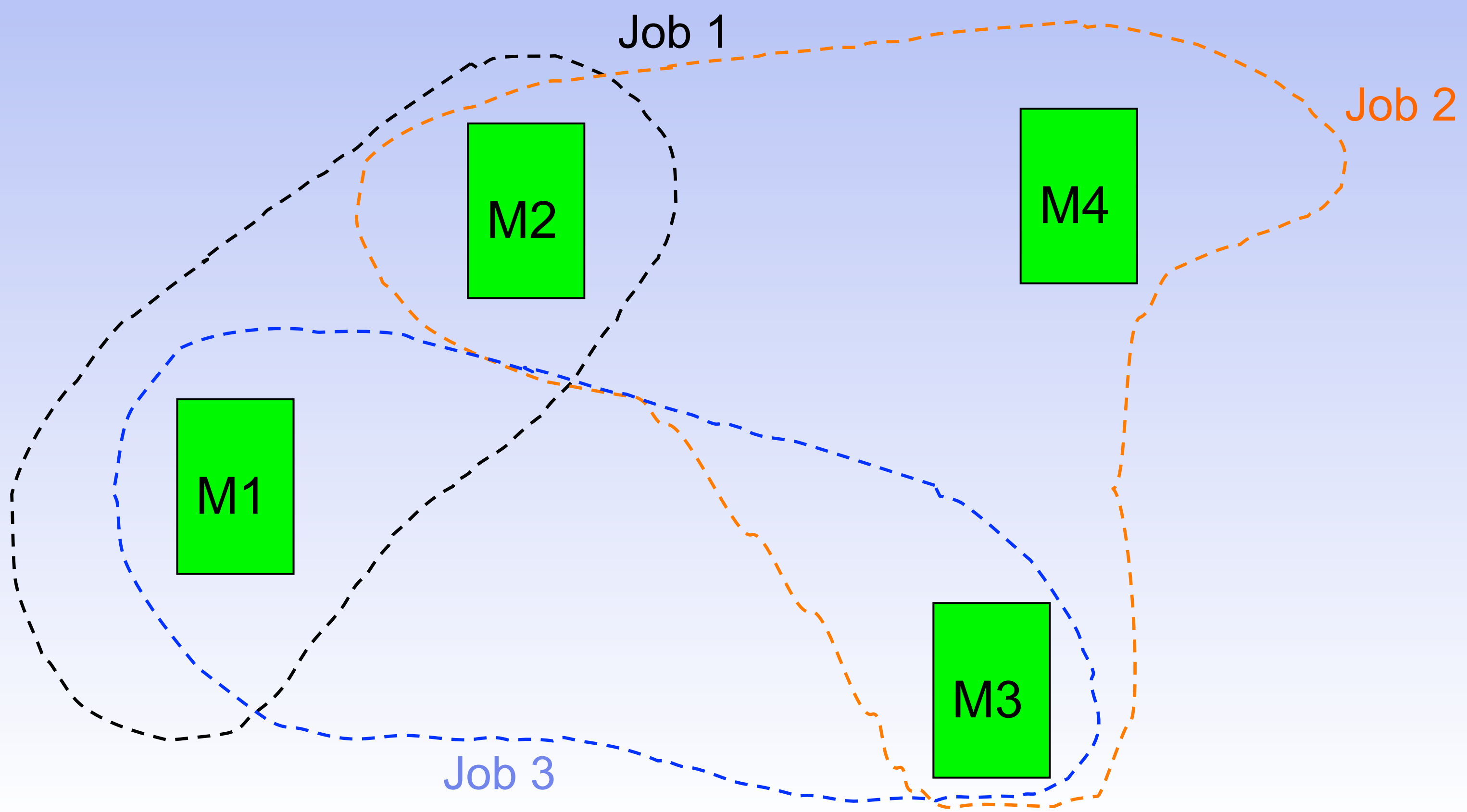
Job-shop



- The operations of every job follow a fixed sequence.
- The sequence depends on the job.
- This type of configuration is suitable for the production of different types of products.

Open-shop

- For all the operations of jobs, no fixed order is imposed.
- Every job has to be performed on a determined set of resources.
- No sequence of machines is imposed.



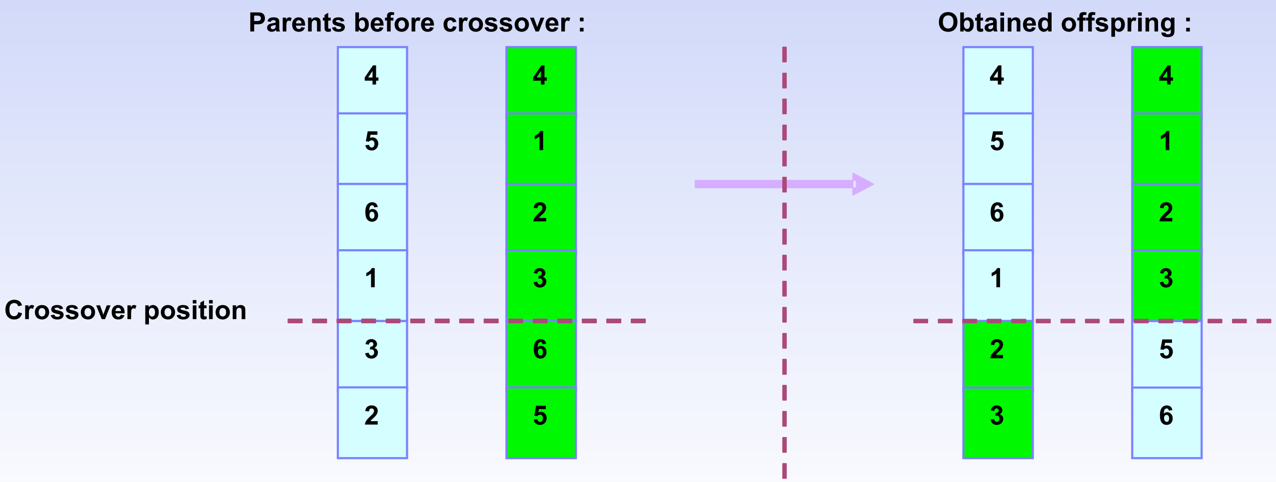
Standard used approaches

Heuristic Methods

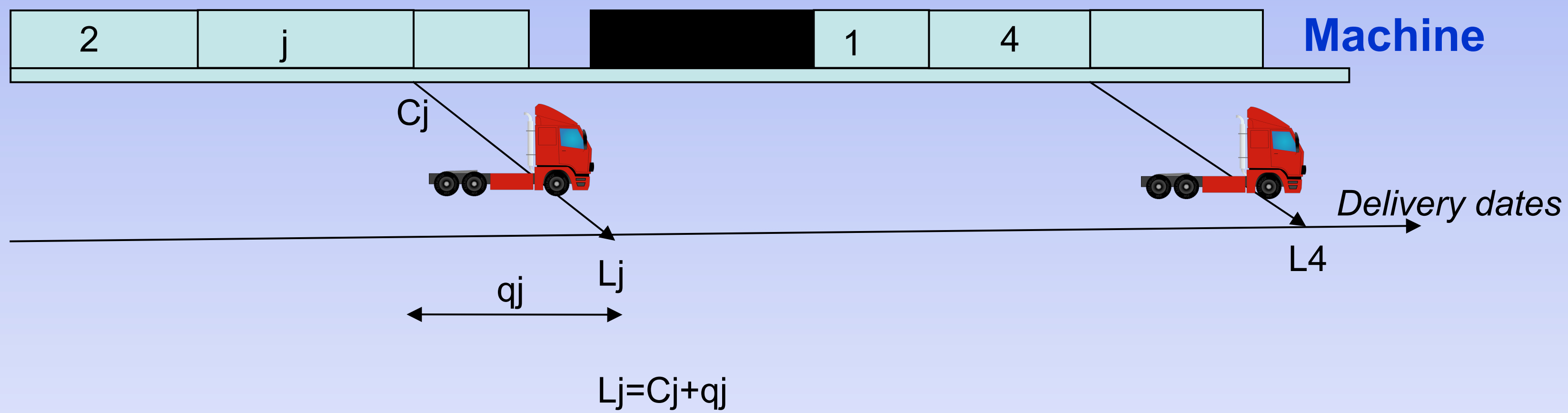
- These methods cannot usually yield an optimal solution.
- They are fast and they can ensure a compromise between quality and computation time if they are rigorously implemented.
- The use of these methods can be recommended when the studied problem has a high level of hardness (for example, the *NP-hard* problems in the strong sense).
- Any effective heuristic needs a minimum effort in its design (local-optimality properties).
- The results that we can obtain by a heuristic can be experimentally evaluated by comparing to the literature results.
- The performance of a heuristic can also be compared to some lower bounds and/or by establishing its worst-case performance analysis.
- Different types of heuristics exist and are widely-used.
    - constructive heuristic methods (based on some priority rules),
    - local-search methods based on the exploration of the neighbourhood of an existing solution (Tabu Search, Simulated Annealing...)
    - population-based methods consisting in iterative improvements of a set of solutions (Genetic Algorithms, Evolutionary Algorithms, Ant Colonies...).

Heuristic Methods

- The disadvantage of these methods consists in the empirical performance of solution (we cannot evaluate the difference with the optimal solution).
- Another type of heuristics can overcome this disadvantage by yielding a guarantied performance for any instance of the problem.
- These heuristics can be obtained by using the polynomial approximation techniques and the performance analysis in the worst-case.
- This type of methods is generally very hard to construct and to analyze (see Pinedo).



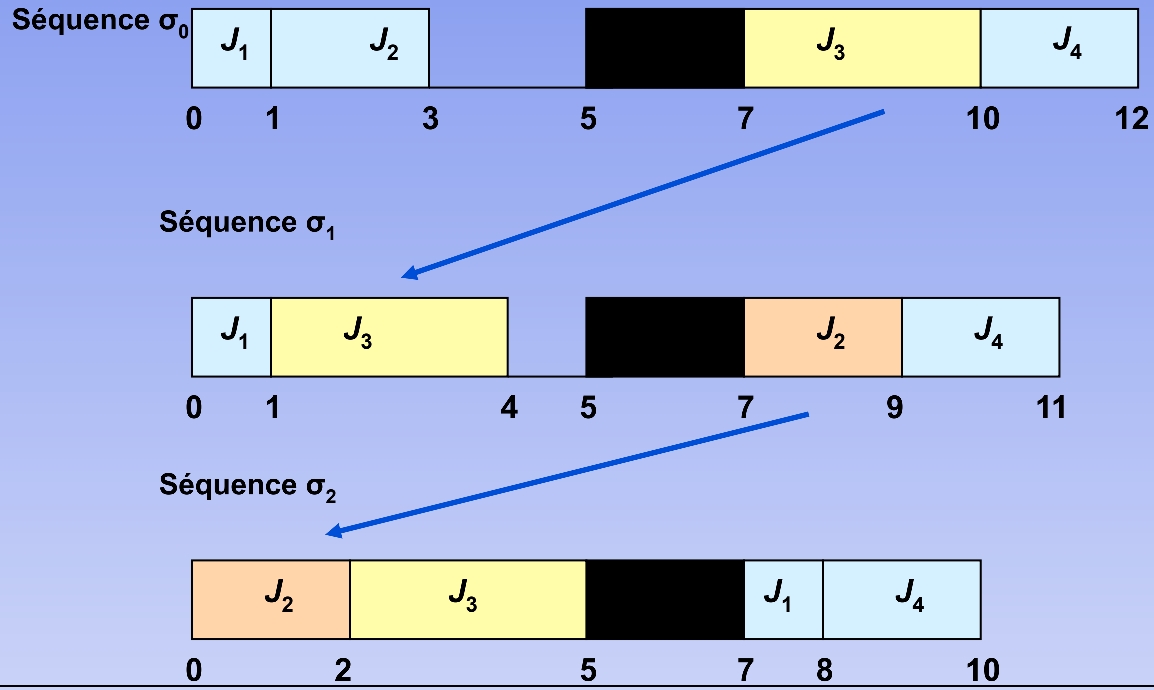
Heuristic Methods with guaranteed performance

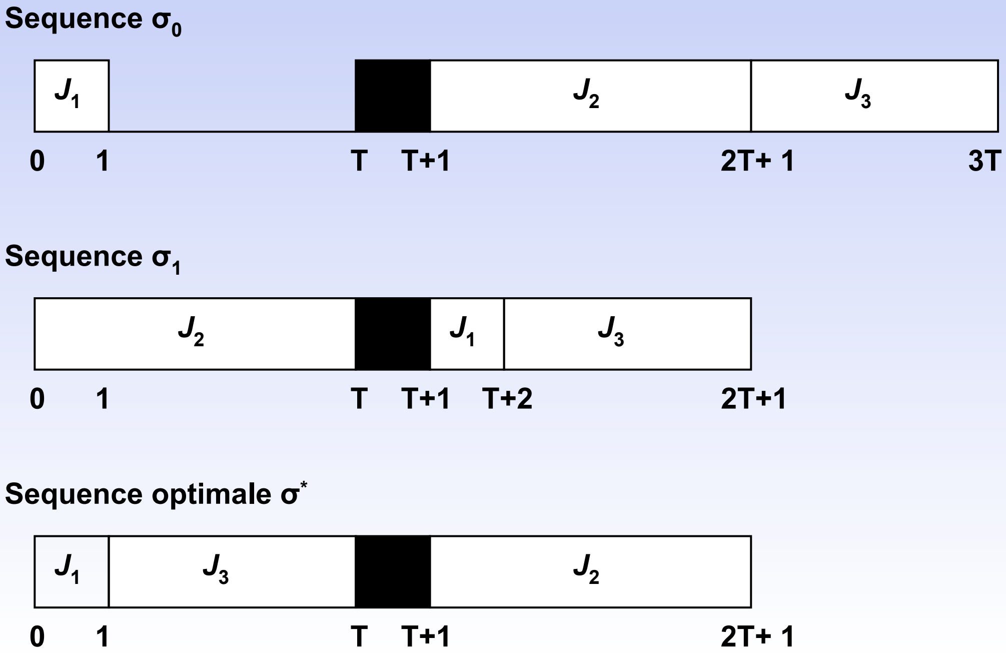


Sufficient transportation resources

Heuristic Methods with guaranteed performance

$p1 = 1 ; q1 = 7 ;$
$p2 = 2 ; q2 = 6 ;$
$p3 = 3 ; q3 = 5 ;$
$p4 = 2 ; q4 = 4 ;$
$T1 = 5 ; \Delta T = 2$
$g0 = \varnothing;$
$g1 = \{J3\} ;$
$g2 = \{J2, J3\} ;$
$\phi\sigma0 (P) = 16 ;$
$\phi\sigma1 (P) = 15 ;$
$\phi\sigma2 (P) = 15.$





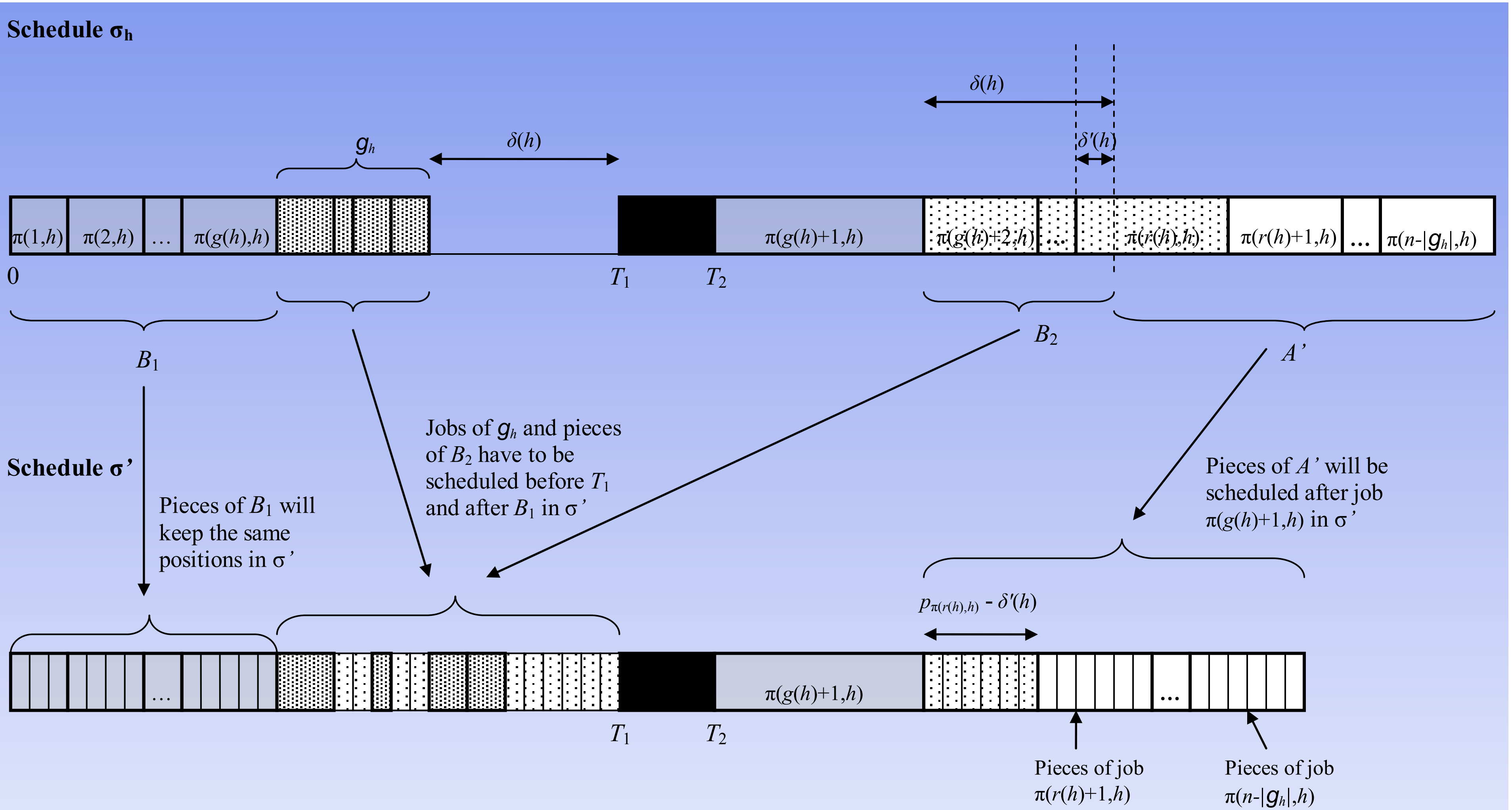
$p1 = 1 ; q1 = 2T ;$

$p2 = T ; q2 = \varepsilon ;$

$p3 = T - 1 ; q3 = 0 ;$

$T1 = T ; \Delta T = 1$ où $\varepsilon << T.$

$\Rightarrow$ $\rho(H) \geq 3/2.$



1) **Heuristic H generates at least one sequence σh ($0 \le h \le l$) such that in the optimal solution σ* :**
   - **All the fixed jobs of gh have to be scheduled before T1**
   - **The critical job π (g(h) + 1, h) has to be scheduled after T2.**

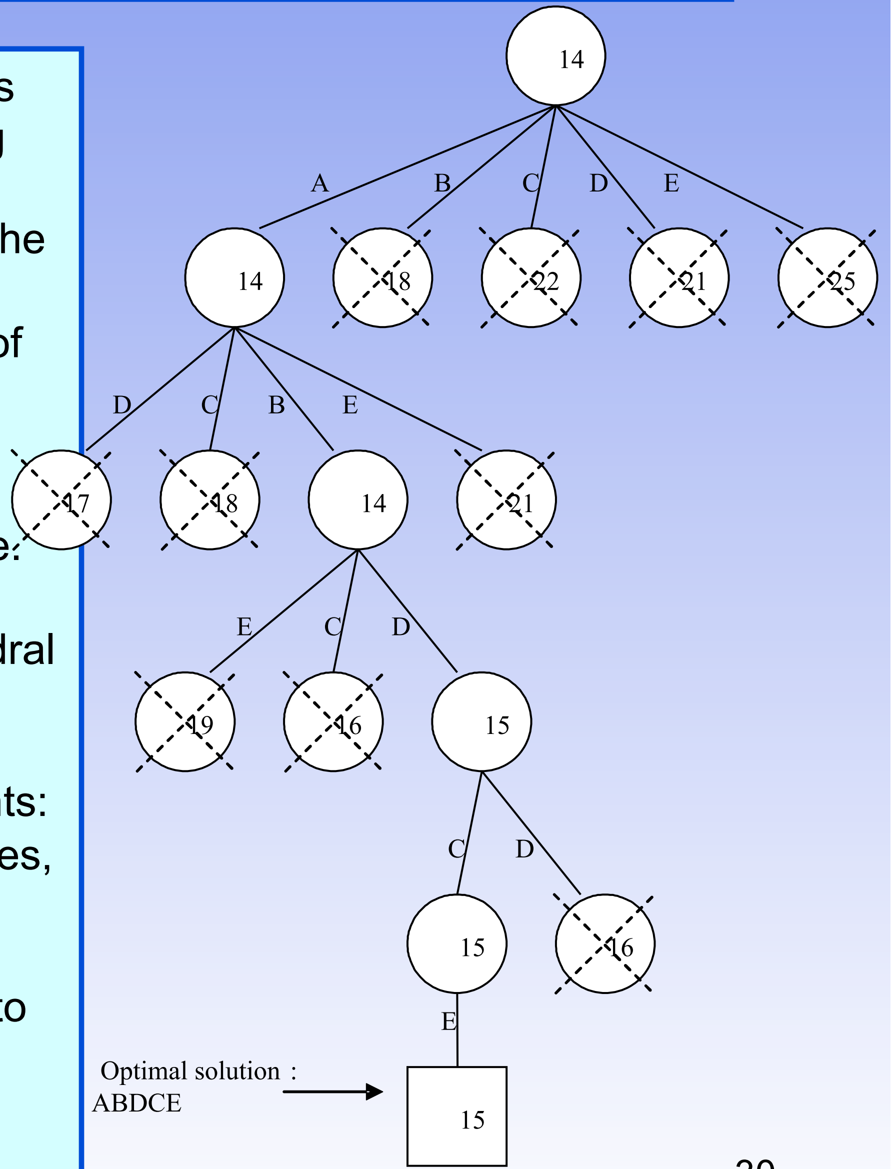
2) **For every job i of gh, we have the following relation : $q_i \le q_{\pi(g(h)+1,h)}$**

3) **By using the splitting principle, we can establish that:**

$\Rightarrow$ $\rho(H) \le 3/2.$

Exact Methods

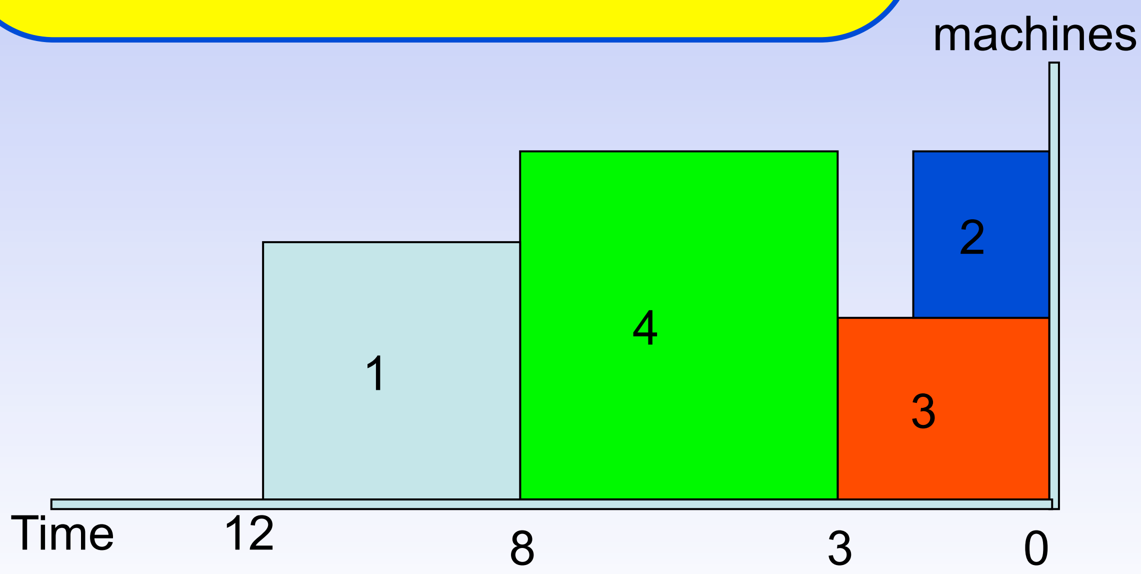
-when the structure of the problem is suitable it is possible to reach an optimal solution by applying an exact method.
- necessity of tools capable to explore implicitly the search space.
- this exploration allows us to reduce the space of visited solutions to a sub-space in which at least one optimal solution exists.
- we can discard the sub-spaces of dominated solutions in order to reduce the computation time.
- this technique can be applied for arborescent methods (branch-and-bound algorithms, polyhedral approaches, integer programming methods, dynamic programming…).
-It can be applied based on the following elements: heuristic solutions, lower bounds, dominance rules, valid inequalities, cuts, exploration strategies, relations based on induction…
-the elaborated exact methods allow us at least to improve the heuristic solutions when the optimal solution cannot be reached in a reasonable computation time.



Exercise 1: how to formulate parallel-machine problems?

- We have m machines for performing N jobs;
- Every job j has a processing time pj and requires a number of Neighbor machines mj;
- Overlopping of jobs is not allowed;
- Preemption is not allowed;
- The objective is to minimize the makespan;

Exercise: give an ILP model

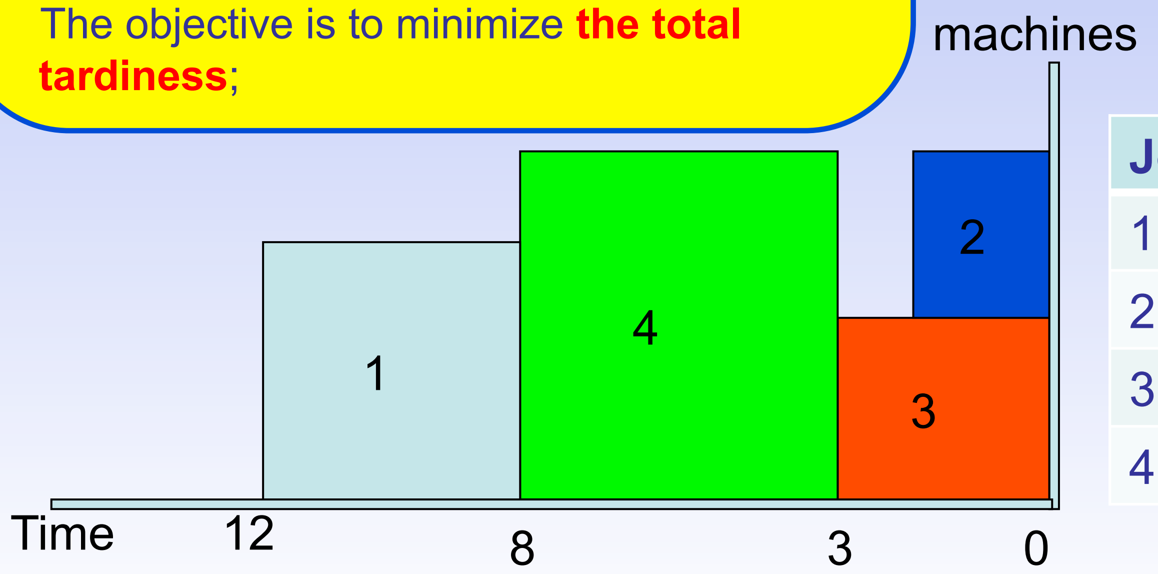


Job j	pj	mj
1	4	3
2	2	2
3	3	2
4	5	4

Exercise 2: how to formulate parallel-machine problems?

- We have m machines for performing N jobs, **with precedence constraints**;
- Every job j has a processing time pj, a **due date dj** and it requires a number of neighbor machines mj;
- Overlopping of jobs is not allowed;
- Preemption of jobs is not allowed;
- The objective is to minimize **the total tardiness**;

Exercise: give an ILP model



Job j	pj	mj	dj
1	4	3	10
2	2	2	3
3	3	2	4
4	5	4	5

Exercise 3: polynomial cases

We would like to prove the following problems are polynomial:

Problem 1:
1||Sum(Cj) ➔ use SPT (jobs are sorted in increasing order of pj).

Problem 2:
1||Sum(wjCj) ➔ use WSPT (jobs are sorted in increasing order of pj/wj).

Problem 3:
1|rj|Cmax ➔ use FIFO (jobs are sorted in increasing order of rj).

Problem 4:
1|qj|Lmax ➔ use Jackson (jobs are sorted in decreasing order of qj).

Problem 5:
1|dj|Tmax ➔ use EDD (jobs are sorted in increasing order of dj).

Polynomial Approximation

Polynomial approximation

« It is the ***art*** to achieve, in polynomial time, feasible solutions with objective value as close as possible (in some predefined sense) to the optimal » Vangelis Paschos

The motivations:

-Practical motivation:

- In several situations, we need to achieve feasible solutions in a reasonable time (polynomial complexity).
- The quality of a solution is generally very important.

-Theoretical motivation:

- Polynomial approximation and combinatorial optimization are strongly related and the knowledge in one field of them can significantly contribute to the other.
- Polynomial approximation can be used to evaluate and to study the complexity of discrete optimization problems.

Polynomial approximation: Notations

- *I* denotes an instance of a discrete optimization problem π (minimization problem).

- OPT(*I*) is the value of an optimal solution for *I*

- H is an algorithm for solving π
- H(*I*) is the value of the solution produced by H for *I*

- r(H) is the standard approximation ratio defined as the maximum of H(*I*)/OPT(*I*) over *I*

- The closer the ratio r(H) to 1, the better the performance of H

Polynomial approximation: Classes

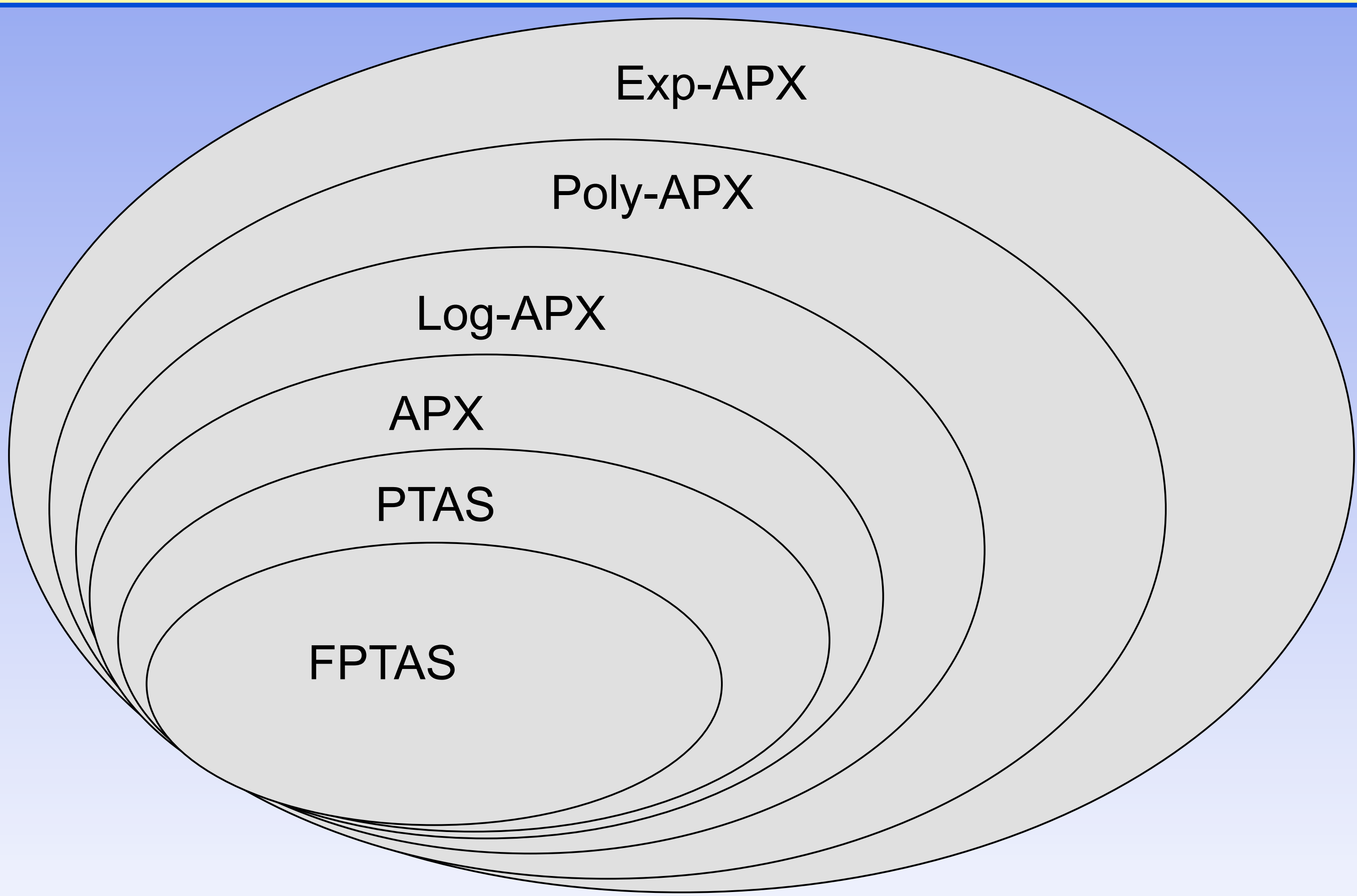
- ***Ratios depending on*** the size of *I* (| *I* |)
  - **Exp-APX** (Travelling Salesman Problem)
  - **Poly-APX** (Graph Coloring Problem)
  - **Log-APX** (Set Covering Problem)

- ***Constant ratios (independent of*** |*I*|***)***
  - **APX** (R||Cmax)

- ***Ratios*** $1 + \varepsilon$***, for any*** $\varepsilon > 0$:
  - *Polynomial time approximation scheme* (complexity polynomial in |*I*| but, eventually, exponential in $1/\varepsilon$)
    PTAS (Pm||Cmax)

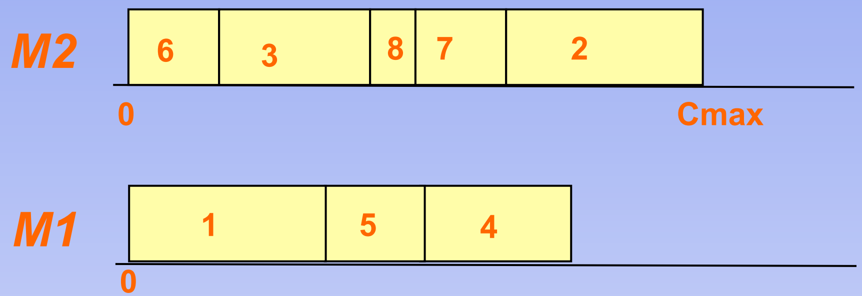
  - *Fully polynomial time approximation scheme* (polynomial in both |*I*| and $1/\varepsilon$)
    FPTAS (2||Cmax)

Polynomial approximation: Classes



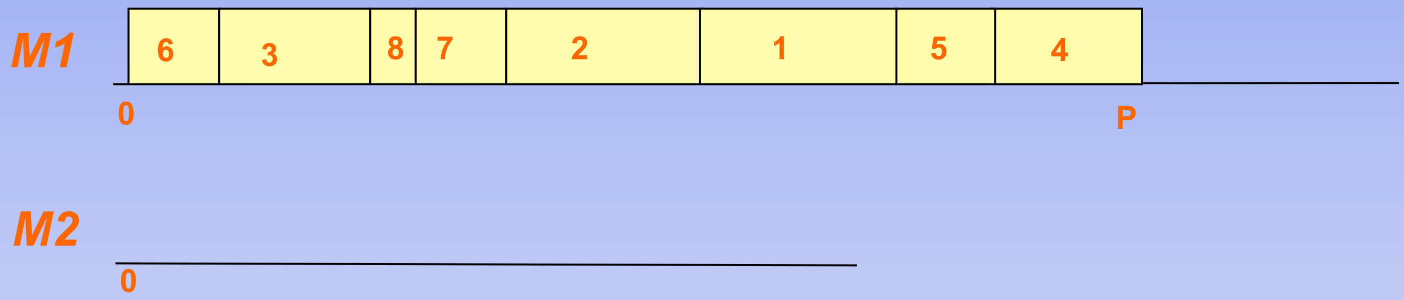
Approximability classes for NP-hard problems (*under the assumption* P ≠ NP)
From the book by: Vangelis PASCHOS

Polynomial Approximation: Illustrations on 2||Cmax



- The problem is to schedule n jobs on two parallel identical machines, with the aim of minimizing the makespan (Cmax).
- Every job i has a processing time pi.
- The machine is available at time 0 and can process at most one job at a time.
- Without loss of generality, we consider that all the data are integers and that jobs are sorted in the LPT order : $p_1 \geq p_2 \geq \ldots \geq p_n$.
- An optimal solution is composed of two sequences of jobs assigned to the machines. In the two sequences any order is optimal. P is the total processing time.
- The problem is NP-hard in the ordinary sense.

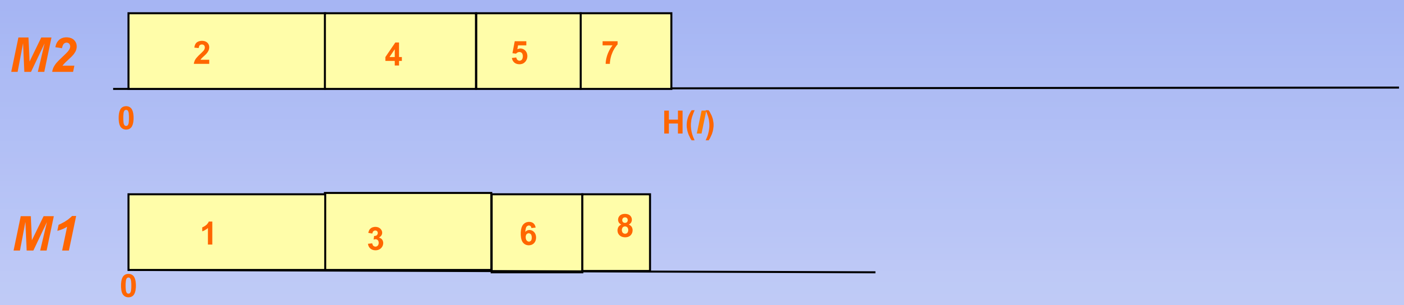
Constant Approximation: Illustrations on 2||Cmax



PROPERTY:
Any assignment of jobs to the two machines is a constant approximation with a standard ratio no more than 2.

Proof:
For any instance I, we have $OPT(I) \geq P/2$ and $H(I) \leq P$.
Then, $H(I) \leq P \leq 2.OPT(I)$.

Constant Approximation: A better heuristic H for 2||Cmax

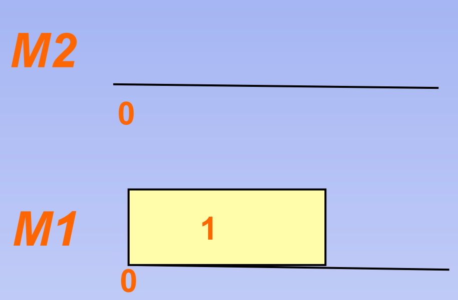


Heuristic description:
Assign the jobs in the LPT order as soon as possible to the available machine.

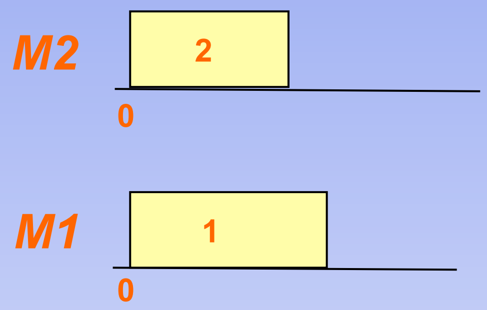
Standard approximation ratio:
For any instance I, we have H(I) ≤ 7/6.OPT(I).

Sketch of proof:
H is optimal for n=1, 2, 3 and 4.

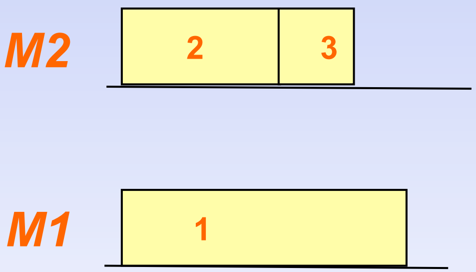
Constant Approximation: A better heuristic H for 2||Cmax



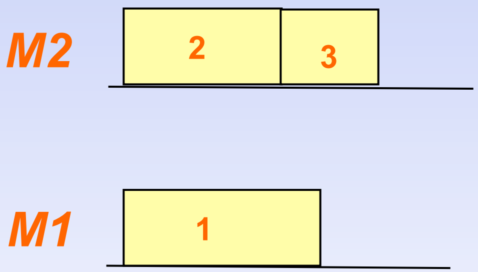
H is optimal for n=1.



H is optimal for n=2.

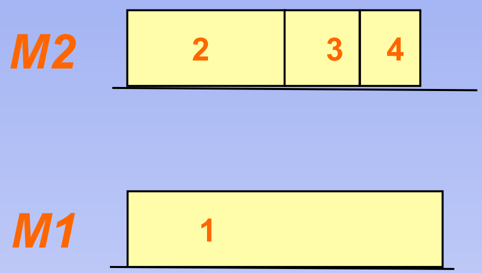


H is optimal for n=3 (case 1: p1>p2+p3).

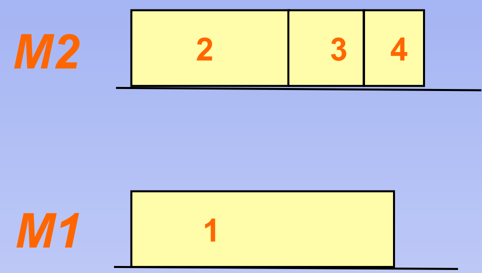


H is optimal for n=3 (case 2: p1 ≤ p2+p3).

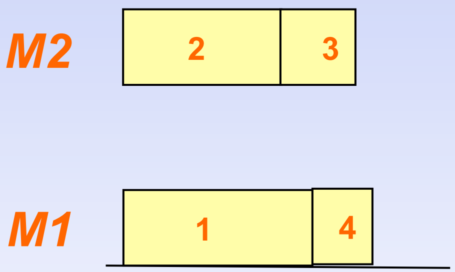
Constant Approximation: A better heuristic H for 2||Cmax



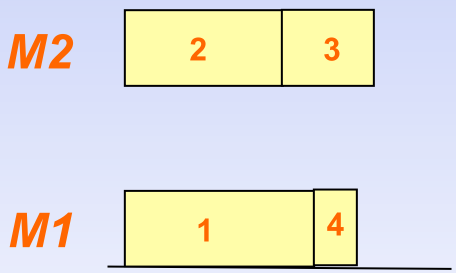
H is optimal for n=4
(case 1: p1>p2+p3+p4).



H is optimal for n=4
(case 2: p2+p3 ≤ p1 < p2+p3+p4).

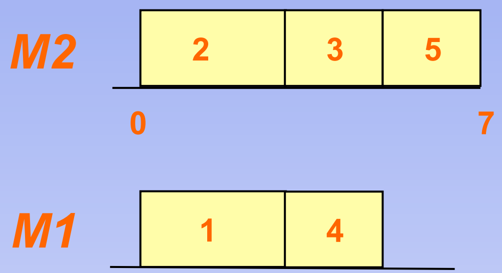


H is optimal for n=4
(case 3: p1 < p2+p3).

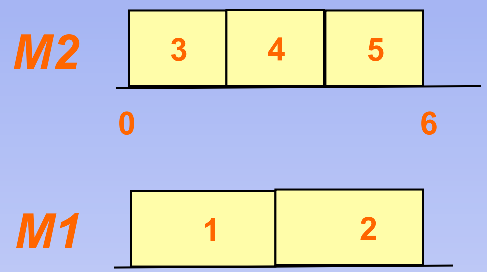


H is optimal for n=4
(case 4: p1 < p2+p3).

Constant Approximation: A better heuristic H for 2||Cmax



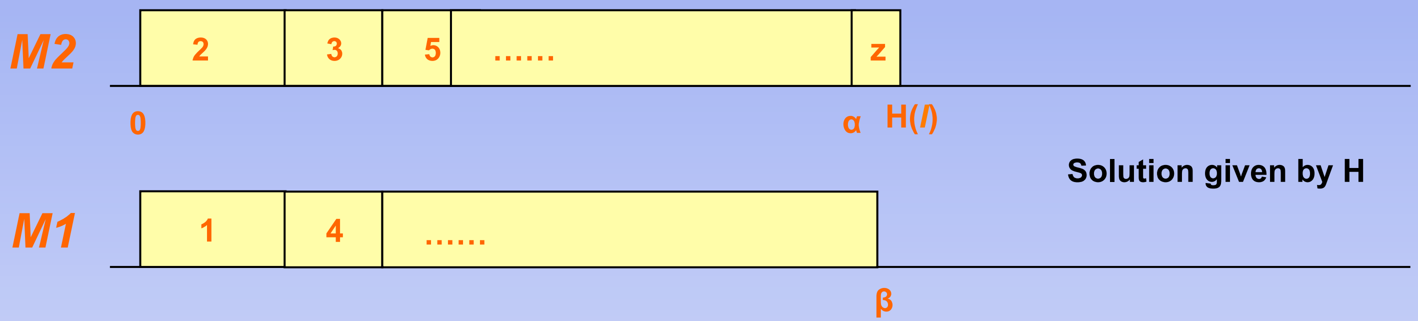
Solution given by H



Optimal solution

H is not optimal for n=5
p1 = 3
p2 = 3
p3 = 2
p4 = 2
p5 = 2

Constant Approximation: A better heuristic H for 2||Cmax



Let z denote the last job scheduled in this sequence. Let **α** be the starting time of *z* and let **β** be the completion time on the other machine. We have:

$$\mathbf{H(I) = \alpha + pz}$$

$$\mathbf{\alpha \leq \beta}$$

$$\mathbf{OPT(I) \geq P/2 = (\alpha + pz + \beta)/2}$$

Hence,

$$\mathbf{H(I) - OPT(I) \leq \alpha + pz - (\alpha + pz + \beta)/2 = (\alpha + pz - \beta)/2 \leq pz/2}$$

Moreover, **OPT(I) ≥ E(z/2).pz** and **z** can be considered **≥ 5.**

E(x) is the smallest integer greater or equal to x.

Then, **(H(I) – OPT(I))/OPT(I) ≤ 1/(2.E(z/2)**) **≤ 1/6 = 0.16667** which implies that:

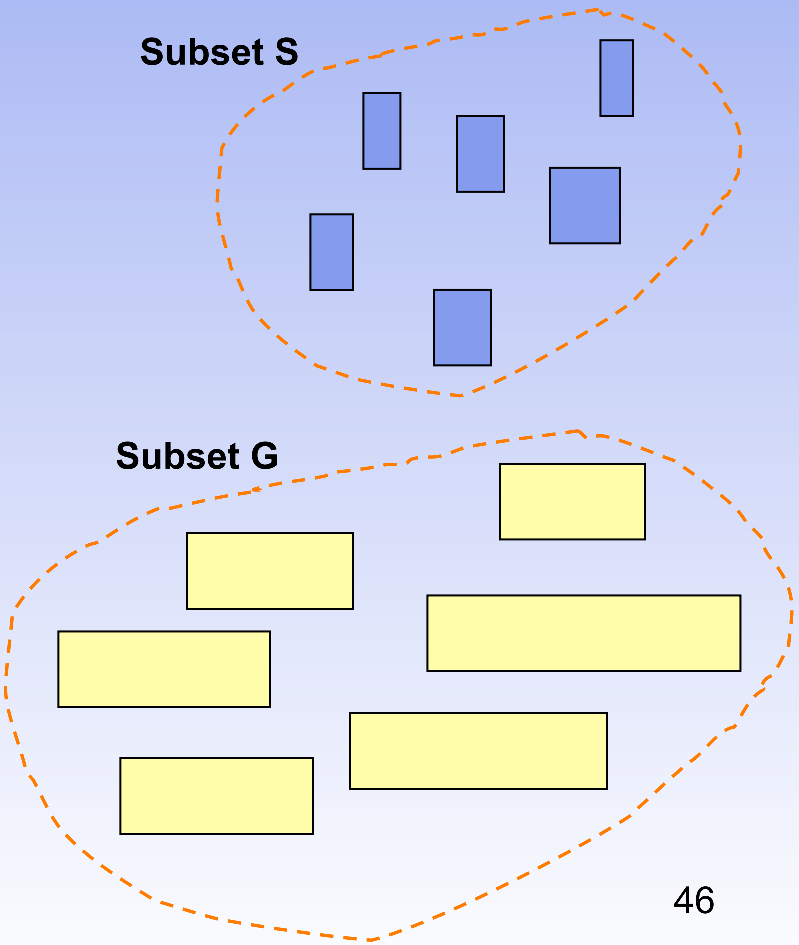
$$\mathbf{H(I) \leq 1.1667\ OPT(I)}.$$

PTAS exists:

A better algorithm than H for 2||Cmax

The idea is very simple!
Let $\varepsilon > 0$.

1. **We divide the set of jobs in two subsets G and S where G = { i | pi >** εP/2**} and S = { i | pi ≤** εP/2**}.**
2. **Enumerate all the assignments of G on the two machines (their cardinal is limited to** $2^{2/\varepsilon}$**).**
3. **For every assignment A generated in Step (2) complete by the jobs of S by scheduling them iteratively on the less loaded machine (in any order).**
4. **Select the best schedule from the solutions obtained in Step (3).**



PTAS exists:

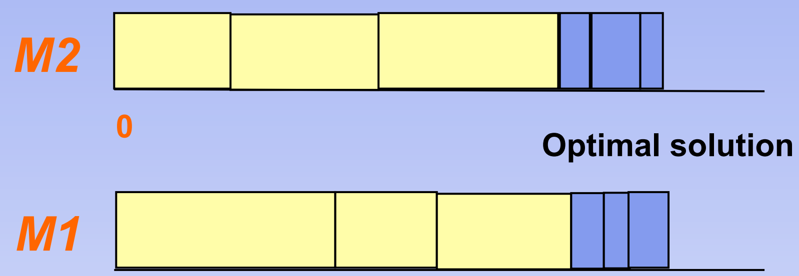
A better heuristic algorithm than H for 2||Cmax

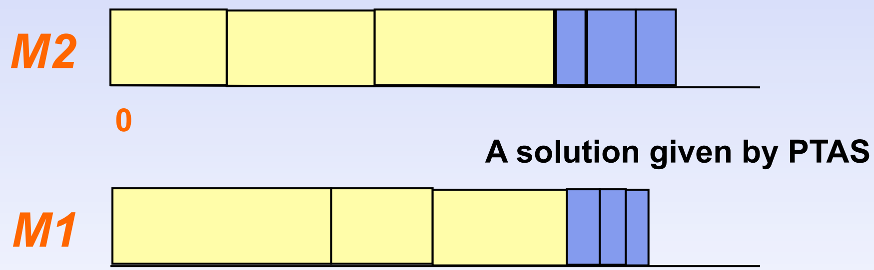
For any optimal solution OPT there exists a solution generated by PTAS where we have the same assignment of great jobs (from G).

Only the assignments of S may be different.

The difference cannot be more than the size of a small job $\varepsilon P/2 \leq \varepsilon OPT$.

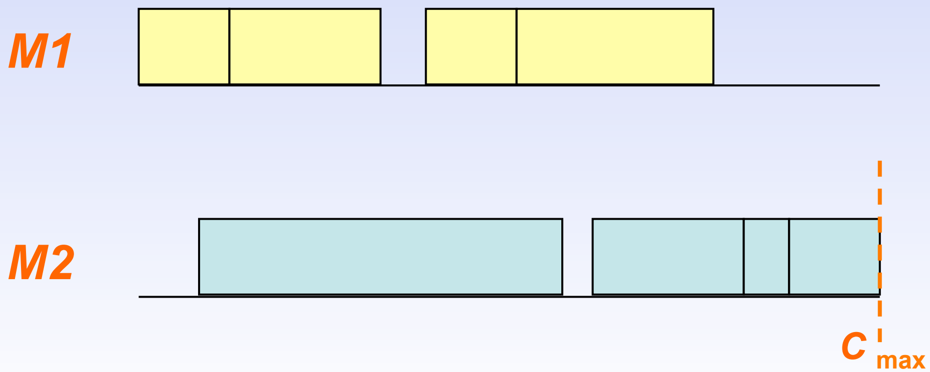
The complexity is bounded by $O(n.2^{1/\varepsilon})$.





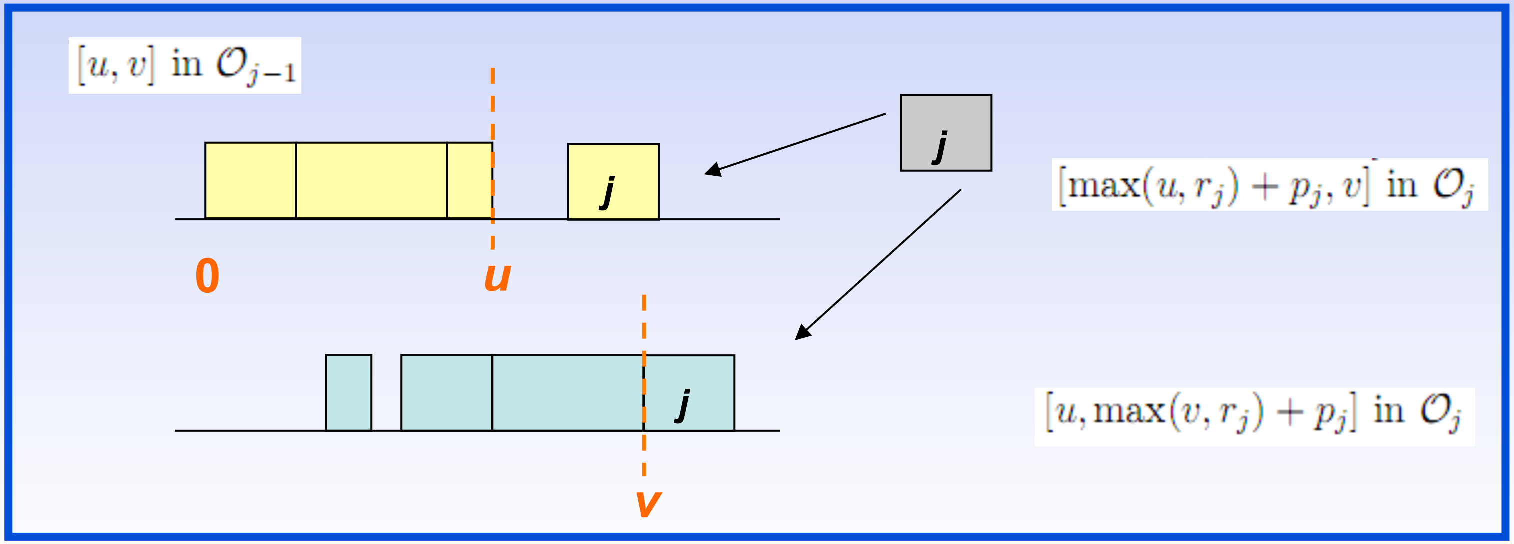
FPTAS: An illustration for 2|rj|Cmax

- The problem is to schedule n jobs on two parallel identical machines, with the aim of minimizing the makespan (Cmax).
- Every job i has a processing time pi and a release date ri.
- The machine is available at time 0 and can process at most one job at a time.
-Without loss of generality, we consider that all the data are integers and that jobs are sorted in the FIFO order : $r1 \leq r2 \leq \ldots \leq rn$.
- An optimal solution is composed of two FIFO sequences of jobs assigned to the machines. In the two sequences only the FIFO order is optimal. The problem is NP-hard in the ordinary sense.

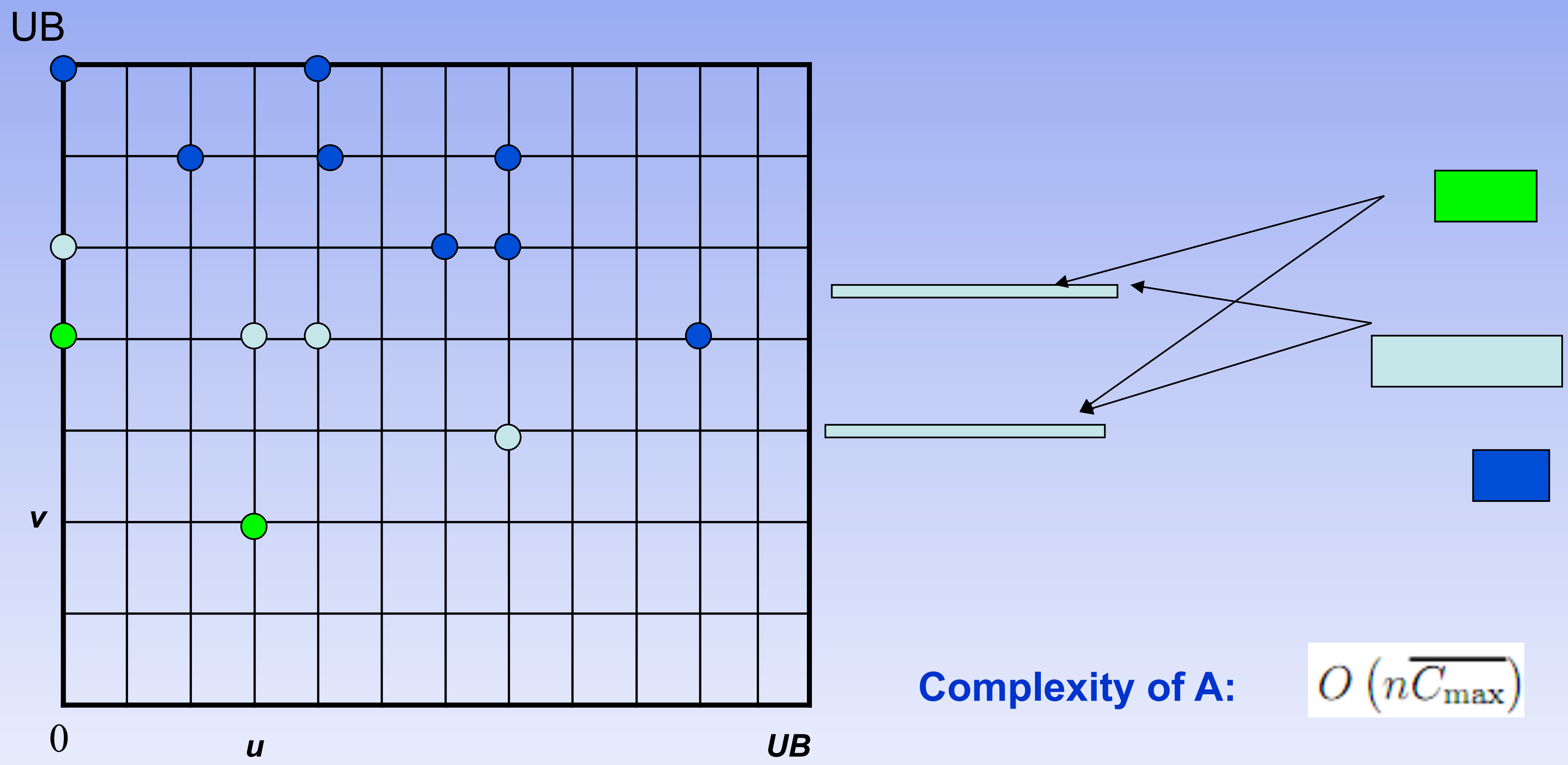


Dynamic Programming

The problem can be optimally solved by applying the following standard dynamic programming algorithm A. This algorithm generates iteratively some sets of states. At every iteration j, a set $\mathcal{O}_j$ composed of states is generated $(1 \leq j \leq n)$. Each state $[u, v]$ in $\mathcal{O}_j$ can be associated with a feasible schedule for the first j jobs. Variable u denotes the completion time of the last job scheduled on the first machine and v is the completion time of the last job scheduled on the second machine. This algorithm can be described as follows:



New Dynamic Programming: Illustration



Complexity of A: $O\left(n\overline{C_{\max}}\right)$

Fully Polynomial Time Approximation Scheme (FPTAS)

$$LB = \frac{2C_{\max}^{H'}(\mathcal{P})}{3},$$

$$\beta = \left\lceil \frac{3n}{2\varepsilon} \right\rceil,$$

$$\gamma = \frac{C_{\max}^{H'}(\mathcal{P})}{\beta}.$$

DEFINITION: Given ε>0, an FPTAS finds (1+ ε)-approximation with a time-complexity polynomial in (1/ ε) and in the input size.

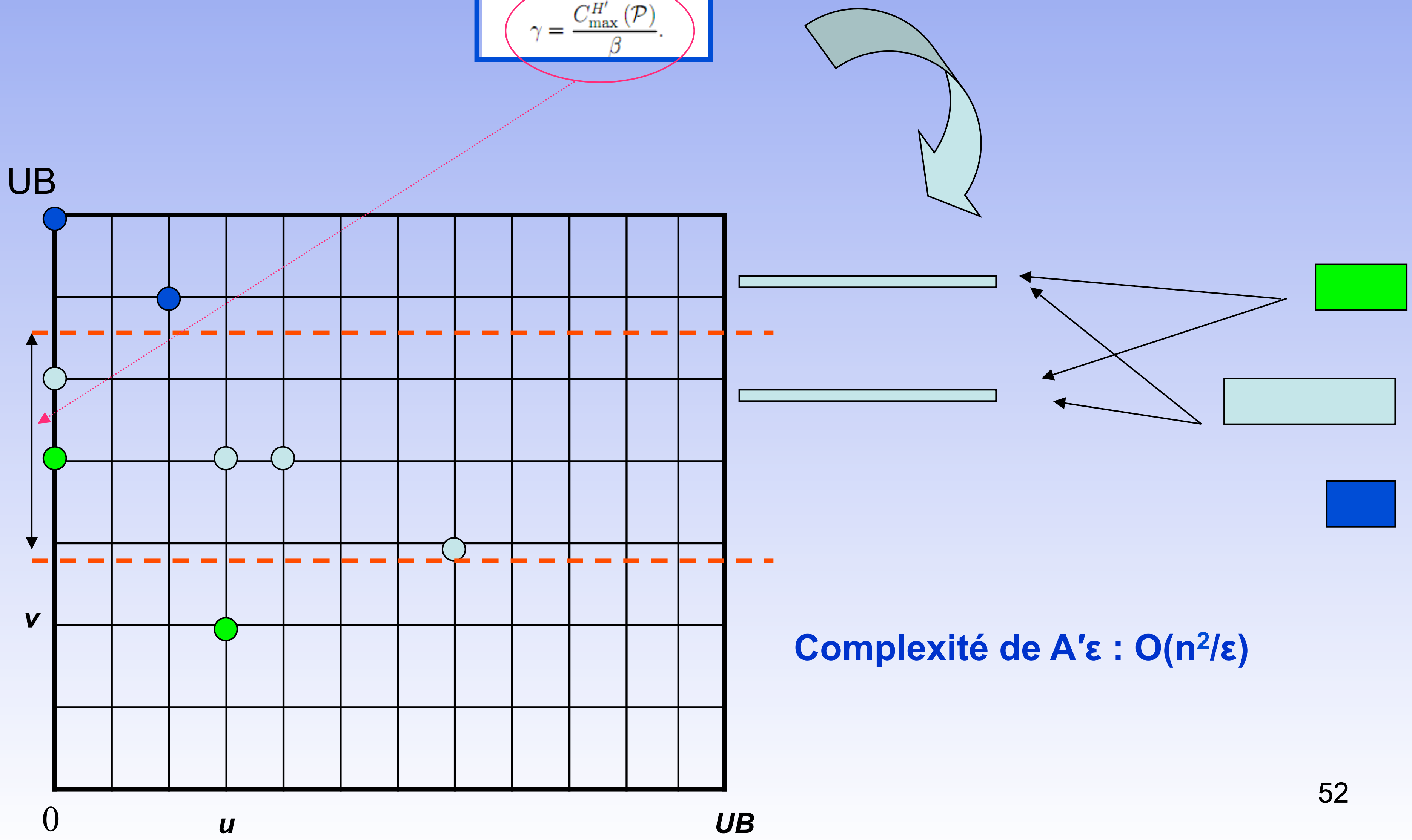
PRINCIPLE: modification of the execution of an exact algorithm

We split the interval $[0, \overline{C_{\max}}]$ into β equal subintervals $I_h = [(h-1)\gamma, h\gamma]_{1 \le h \le \beta}$ of length γ. Our algorithm $\mathcal{A}_\varepsilon$ generates reduced sets $\mathcal{O}_j^\#$ instead of sets $\mathcal{O}_j$. It can be described as follows:

$$LB = \frac{2C_{\max}^{H'}(\mathcal{P})}{3},$$

$$\beta = \left\lceil \frac{3n}{2\varepsilon} \right\rceil,$$

$$\gamma = \frac{C_{\max}^{H'}(\mathcal{P})}{\beta}.$$



Complexité de A′ε : O(n²/ε)

Fully Polynomial Time Approximation Scheme (FPTAS): the main results [Kacem 2009]

Lemma 3 *For every state* $[u, v]$ *in* $\mathcal{O}_j$ *there exists a state* $[u^{\#}, v^{\#}]$ *in* $\mathcal{O}_j^{\#}$ *such that:*

$$u^{\#} - u \leq 0 \tag{7}$$

and

$$v^{\#} - v \leq j\gamma \tag{8}$$

Theorem 1 *Given an arbitrary* $\varepsilon > 0$*, algorithm* $\mathcal{A}_\varepsilon$ *yields an output* $C_{\max}^{\mathcal{A}_\varepsilon}(\mathcal{P})$ *such that:*

$$\frac{C_{\max}^{\mathcal{A}_\varepsilon}(\mathcal{P})}{C_{\max}^{*}(\mathcal{P})} \leq (1 + \varepsilon). \tag{9}$$

Lemma 4 *Given an arbitrary* $\varepsilon > 0$*, algorithm* $A(\varepsilon)$ *can be implemented in* $O(n^2/\varepsilon)$ *time.*

Some results in approximation theory

Topics	Surveys
Bin Packing	L. HALL
Covering and Packing	D. HOCHBAUM
Scheduling	D. SCHMOYS
Knapsack	H. KELLERER
Symmetric Quadratic Knapsack	I. KACEM, H. KELLERER, V. STRUSEVICH
Graph Coloring	V. PASCHOS