Constraint Based Scheduling

Philippe Baptiste

CNRS LIX, École Polytechnique

June 22, 2006

Scheduling

Given a set of resources with fixed capacities, a set of activities with given durations and resource requirements, a set of temporal constraints between activities, and a cost function, a "pure" scheduling problem consists of deciding when to execute each activity to minimize the overall cost, under both temporal and resource constraints.

Scheduling Situations

- In **disjunctive scheduling**, each resource (= machine) can execute one activity at a time.
- In **cumulative scheduling**, a resource can run several activities in parallel.
- In **non-preemptive scheduling**, activities cannot be interrupted.
- In **preemptive scheduling**, activities can be interrupted.
- On an **elastic schedule** the amount of resource assigned to A_i can, at any time t, pick any value between 0 and C_R resource capacity, provided that the sum the assigned capacity equals a given energy.

Objective Functions

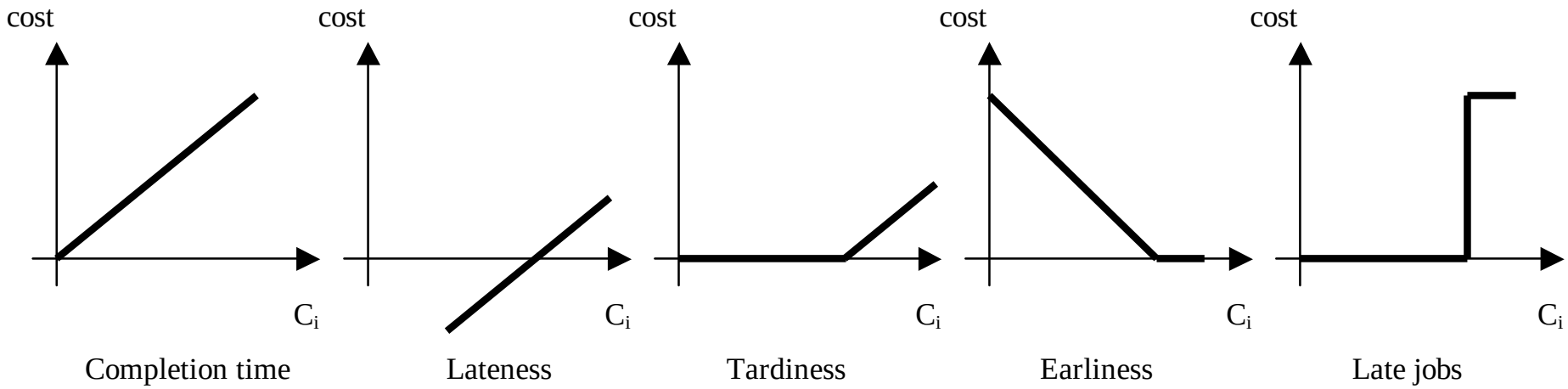
- In **decision** problems, one has to determine if there is a schedule that meets all constraints.
- In **optimization** problems, an objective function has to be minimized (minimization of the makespan, *i.e.*, the finishing time of the schedule, the number of activities performed with given delays, the peak resource utilization, the sum of setup times or costs).

Objective Functions

Scheduling criteria F are either formulated as a sum or as a maximum. A weight per job w_i may be used to give more importance to some jobs. C_i denotes the completion time of A_i.

- **Makespan:** $F = C_{\max} = \max_i C_i$
- **Maximum Tardiness:** $F = T_{\max} = \max_i T_i$
- **Total weighted flow time:** $F = \sum w_i C_i$
- **Total weighted number of late jobs:** $F = \sum w_i U_i$
- **Total weighted tardiness:** $F = w_i T_i$

Objective Functions



Aujourd'hui

- A CP Model
- Disjunctive Scheduling
  - Constraint Propagation
  - Job-Shop
- Cumulative Scheduling
  - Constraint Propagation
  - RCPSP
- Complex Objective Function
  - Constraint Propagation
  - Number of Late Jobs
- ATC
  - Problem Definition
  - Branch & Cut
  - Constraint Programing

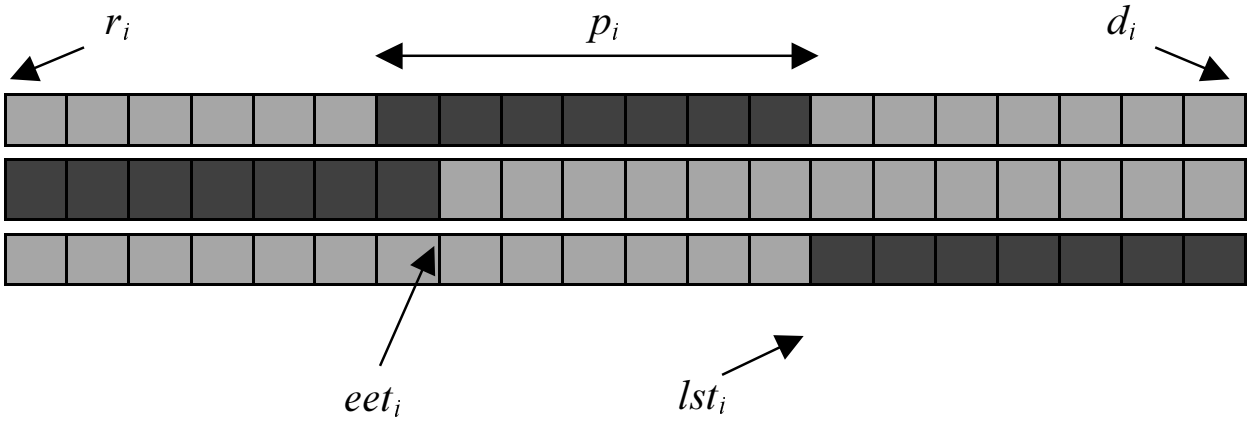
A Constraint Based Scheduling Model

- Activities
- Resources
- Temporal Relations
- Resource Constraints

Activities (non-preemptive case)

- Two variables, $start(A_i)$ and $end(A_i)$, are associated with A_i.
- We use $[r_i, d_i]$ to denote the time window in which A_i has to execute.
- r_i is the smallest value in $start(A_i)$ and d_i is the largest value in $end(A_i)$ (also called release date and the deadline)
- The processing time of the activity is an additional variable $proc(A_i)$ with $proc(A_i) = end(A_i) - start(A_i)$. We note $p_i = lb(proc(A_i))$.

Activities (Non-Preemptive Case)



Most often : Domains = Intervals

Temporal Relations

- Temporal relations between activities can be expressed by linear constraints between the start and end variables of activities.
- $A_i \rightarrow A_j$ is modeled by the linear ct $end(A_i) \leq start(A_j)$.
- Such constraints can be easily propagated using a standard arc-B-consistency algorithm.

Temporal Relations

- If we only have precedence constraints
- and if the resulting CSP is Arc-B-Consistant
- then there is a solution to the CSP (*i.e.*, there is a feasible schedule)
- Moreover, for each activity A_i and each starting time $t \in start(A_i)$, there is a feasible schedule where A_i starts at t.

The resulting algorithm is strongly related to Ford & Belllman's algorithm.

Resource Constraints

Given A_i and a resource R whose capacity is $cap(R)$, let $cap(A_i, R)$ be the amount of R required by A_i.
Generally speaking, the resource constraint is

$$\forall t \sum_{start(A_i) \leq t < end(A_i)} cap(A_i, R) \leq cap(R) \quad (1)$$

In the following, we note $c_{i,R}$ the minimal amount of the capacity (resp. of the energy) of the resource R required by the activity A_i.
Finally we note C_R the maximum value in the domain of the resource capacity.

Objective Functions

A variable *criterion* represents the value of the objective function. We have

$$criterion = F(end(A_1), \cdots, end(A_n))$$

F is often a simple arithmetic expression ($\Rightarrow$ arc-B-consistency).

Objective Functions : Min sum vs. Min Max

- Considering the objective constraint and the resource constraints independantly is ok when F is a "maximum" ($C_{\max}$ or $T_{\max}$).
- Indeed, the upper-bound on *criterion* is directly propagated on the completion time of each activity. In the search for a solution to the decision problem, the objective function can be forgotten.
- The situation is much more complex for sum functions such as $\sum w_i C_i, \sum w_i T_i$ or $\sum w_i U_i$.

Efficient constraint propagation techniques must handle resource cts and objective constraint simultaneously.

Computing an Optimal Solution

Once all constraints of the problem are posted, solve successive decision variants to get the optimum, *e.g.*, a dichotomizing algorithm can be used:

1. Compute an *ub* and a *lb* for *criterion*.
2. Set $D = (lb + ub)/2$
3. Constrain $criterion \leq D$. Solve the resulting CSP (branching procedure with constraint propagation). If a solution is found, set *ub* the solution value; otherwise, set *lb* to $D + 1$.
4. Iterate steps 2 and 3 until $ub = lb$.

Refinements of the Model

- Alternative Resources
- Transition Time
- Transition Cost

Aujourd'hui

- A CP Model
- Disjunctive Scheduling
  - Constraint Propagation
  - Job-Shop
- Cumulative Scheduling
  - Constraint Propagation
  - RCPSP
- Complex Objective Function
  - Constraint Propagation
  - Number of Late Jobs
- ATC
  - Problem Definition
  - Branch & Cut
  - Constraint Programing

Propagation of the One Machine Constraint

How to propagate that a set of n activities $\{A_1, \cdots, A_n\}$ require the same resource of capacity 1 ?
Several techniques:

- Time-Table Constraint
- Disjunctive Constraint
- Edge-Finding

Time-Tables

Time-Tables use an explicit data structure called "timetable" to maintain information about resource utilization and resource availability over time.
We define a 0-1 variable $X(A_i, t)$ for each activity A_i and time t.
The propagation mainly consists of maintaining arc-B-consistency on the formula:

$$\forall t, \sum_{start(A_i) \leq t < end(A_i)} X(A_i, t) \leq 1 \tag{2}$$

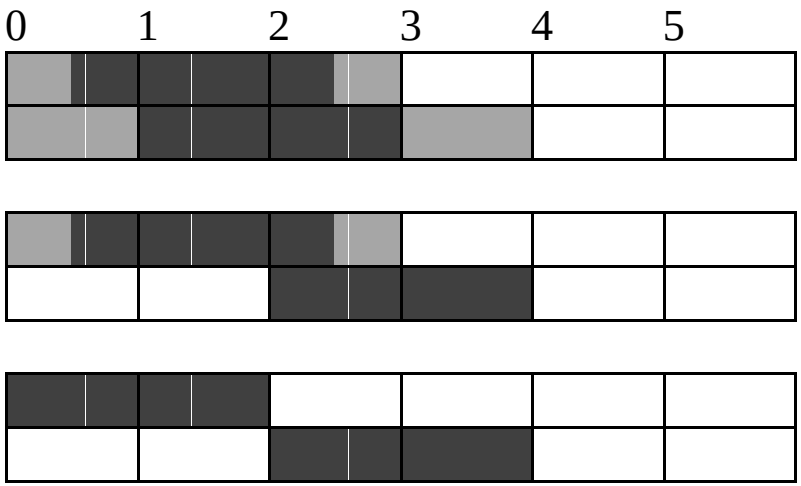
Time-Tables

The previous mechanism leads in turn to some adjustments of r_i, eet_i, lst_i and d_i:

$$start \leq \min\{t : X(A_i, t) = 1\}$$
$$start > \max\{t : \forall u < t, X(A_i, u) = 0\}$$
$$end > \max\{t : X(A_i, t) = 1\}$$
$$end \leq \min\{t : \forall u \geq t, X(A_i, u) = 0\}$$

Time-Tables

Before Propagation	r_i	d_i	p_i
A_1	0	3	2
A_2	0	4	2
Propagation 1	r_i	d_i	p_i
A_1	0	3	2
A_2	2	4	2
Propagation 2	r_i	d_i	p_i
A_1	0	2	2
A_2	2	4	2



Disjunctive Constraint Propagation

In non-preemptive scheduling, two activities A_i and A_j requiring the same machine cannot overlap in time.

- either A_i precedes A_j or A_j precedes A_i.
- $n(n-1)/2$ (explicit or implicit) disjunctive constraints

$$[end(A_i) \leq start(A_j)] \vee [end(A_j) \leq start(A_i)] \qquad (3)$$

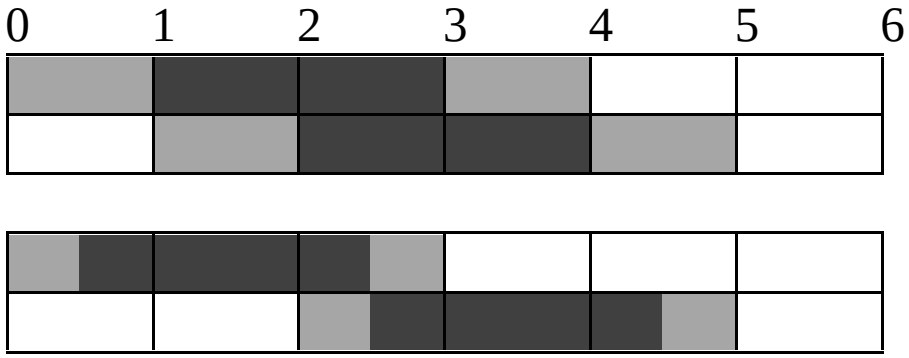
Disjunctive Constraint Propagation

Enforcing arc-B-Consistency on
$[end(A_i) \leq start(A_j)] \vee [end(A_j) \leq start(A_i)]$ done as follows:

- Whenever the smallest possible value of $end(A_i)$ (earliest end time of A_i) exceeds the greatest possible value of $start(A_j)$ (latest start time of A_j), A_i cannot precede A_j;
- hence A_j must precede A_i; the time-bounds of A_i and A_j are consequently updated
- When neither of the two activities can precede the other, a contradiction is detected.

Disjunctive Constraint

Before Propagation	r_i	d_i	p_i
A_1	0	4	2
A_2	1	5	2
Propagation	r_i	d_i	p_i
A_1	0	3	2
A_2	2	5	2



Disjunctive Constraint *vs.* time-tables

- Disjunctive constraints provide more precise time bounds than timetables. Indeed, if A_j is known to execute at some time t between r_i and eet_i of A_i, then the first disjunct of $[end(A_i) \leq start(A_j)] \vee [end(A_j) \leq start(A_i)]$ is false and thus, A_j must precede A_i and the propagation of the disjunctive constraint implies $start(A_i) \geq r_j + p_j > t$.
- disjunctive constraints may propagate more than time-table constraints (see prev. ex)

Necessary Conditions of Existence

- The global constraint is feasible iff there is a feasible schedule (NP-Hard in the strong sense)
- Relax non-preemption constraint. Polynomially solvable
  - Flow
  - Duality (Max Flow Min Cut) $\rightsquigarrow$ Check over intervals
  - Jacskon rule (EDD order)

Necessary Conditions of Existence (Flow)

Have a look to the white board

Necessary Conditions of Existence (Dual)

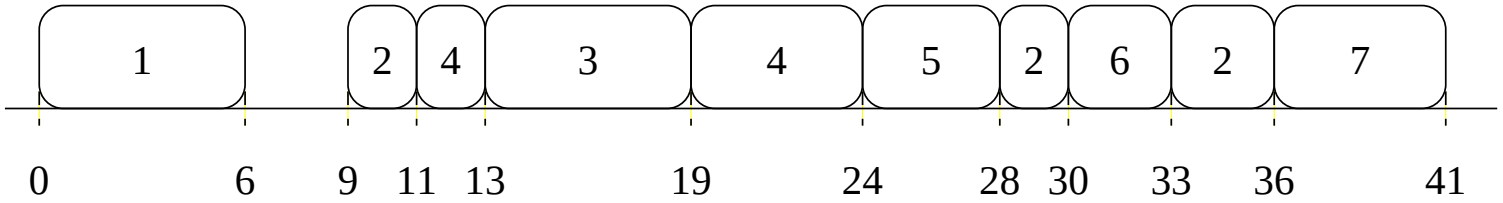
There is a feasible preemptive schedule if and only if over any time interval $[t_1, t_2]$, the sum of the processing times of the jobs i such that $t_1 \leq r_i$ and $d_i \leq t_2$ is not greater that $t_2 - t_1$.
$\rightsquigarrow$ simple algorithms

Necessary Conditions of Existence (Jackson)

- based on EDD dispatching rule
- Build the schedule from left to right
- At any time point t schedule the available piece of job with minimal deadline
- Where available at t means with release date greater than t

Necessary Conditions of Existence (Jackson)

i	1	2	3	4	5	6	7
r_i	0	9	13	11	20	30	30
p_i	6	7	6	7	4	3	5
d_i	31	41	22	24	27	40	48



Edge-Finding

- It consists of deducing that some activities from a given set Ω **must, can, or cannot**, execute first (or last) in Ω.
- Such deductions lead to new time-bounds.

In the following,

$$r_\Omega = \min_{A_i \in \Omega} r_i, \quad d_\Omega = \max_{A_i \in \Omega} d_i, \quad p_\Omega = \sum_{A_i \in \Omega} p_i$$

Let $A_i \ll A_j$ ($A_i \gg A_j$) mean that A_i executes before (after) A_j and $A_i \ll \Omega$ ($A_i \gg \Omega$) mean that A_i executes before (after) all the activities in Ω.

Edge-Finding Rules

$$\forall \Omega, \forall A_i \notin \Omega, [d_{\Omega \cup \{A_i\}} - r_\Omega < p_\Omega + p_i] \Rightarrow A_i \ll \Omega$$

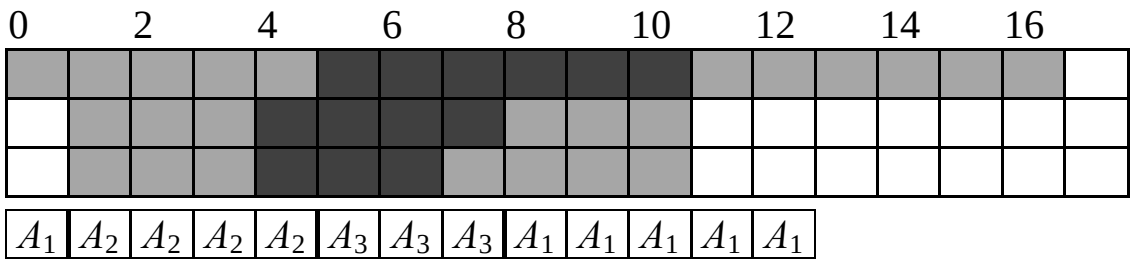
$$\forall \Omega, \forall A_i \notin \Omega, [d_\Omega - r_{\Omega \cup \{A_i\}} < p_\Omega + p_i] \Rightarrow A_i \gg \Omega$$

$$\forall \Omega, \forall A_i \notin \Omega, [A_i \ll \Omega] \Rightarrow [end(A_i) \leq \min_{\Omega' \subseteq \Omega} (d_{\Omega'} - p_{\Omega'})]$$

$$\forall \Omega, \forall A_i \notin \Omega, [A_i \gg \Omega] \Rightarrow [start(A_i) \geq \max_{\Omega' \subseteq \Omega} (r_{\Omega'} + p_{\Omega'})]$$

Edge-Finding Example

	r_i	d_i	p_i
A_1	0	17	6
A_2	1	11	4
A_3	1	11	3



On this example, the edge-finding propagation algorithm deduces $start(A_1) \geq 8$ (A_1 must execute after $\{A_2, A_3\}$), when the timetable and the disjunctive constraint propagation algorithms deduce nothing.

Edge-Finding: Algorithmitic Aspect

- If n activities require the resource, there are a priori $O(n * 2^n)$ pairs (A_i, Ω) to consider!
- Note that one can only consider the sets $\Omega = \Omega_{u,v}$

$$\Omega_{u,v} = \{J_i : r_u \leq r_i \wedge d_i \leq d_v\}$$

- So one can easiely build a polynomial algorithm: there are at most $O(n^3)$ pairs $(A_i, \Omega_{u,v})$ to consider and each time the rules run in linear time so we have an $O(n^4)$ algorithm.
- The best algorithm runs in $O(n \log n)$!

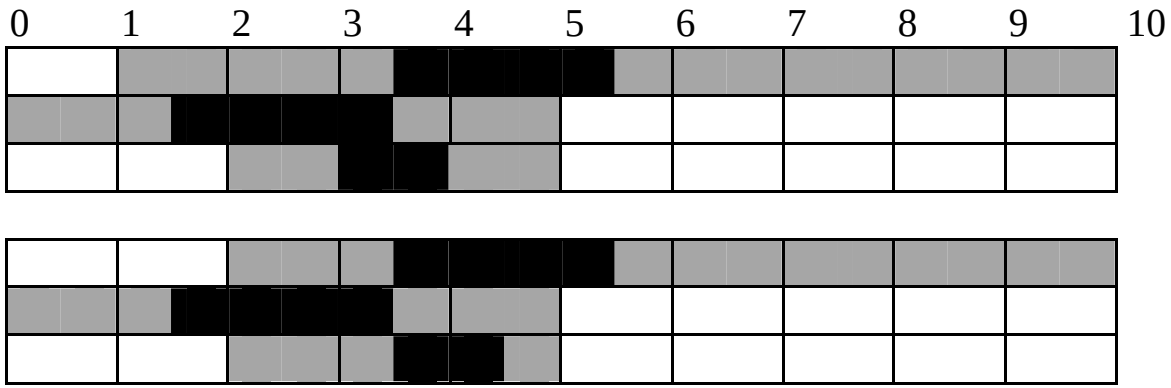
Not-First Not-Last

- The algorithms presented mostly focus on determining if A_i must execute before (or after) Ω.
- A natural complement consists of determining whether A_i can execute before (or after) Ω.
- In the non-preemptive case, this leads to the rules

$$\forall \Omega, \forall A_i \notin \Omega, [d_\Omega - r_i < p_\Omega + p_i] \Rightarrow \neg(A_i \ll \Omega)$$
$$\forall \Omega, \forall A_i \notin \Omega, [d_i - r_\Omega < p_\Omega + p_i] \Rightarrow \neg(A_i \gg \Omega)$$
$$\forall \Omega, \forall A_i \notin \Omega, \neg(A_i \ll \Omega) \Rightarrow start(A_i) \geq \min_{A_j \in \Omega}(r_j + p_j)$$
$$\forall \Omega, \forall A_i \notin \Omega, \neg(A_i \gg \Omega) \Rightarrow end(A_i) \leq \max_{A_j \in \Omega}(d_j - p_j)$$

Not-First Not-Last

Before Prop.	r_i	d_i	p_i
A_1	1	10	2
A_2	0	5	2
A_3	2	5	1
After Prop.	r_i	d_i	p_i
A_1	2	10	2
A_2	0	5	2
A_3	2	5	1



Not-First Not-Last deduces that A_i cannot start before 2 while the other deductive rules seen up to now deduce nothing.

Not-First Not-Last

- The "not-first" and "not-last" rules subsume the disjunctive constraint propagation.
- Hence, no disjunctive constraint propagation algorithm is needed when the "not-first" rule and its dual "not-last" are applied.

Job-Shop Scheduling (JSSP)

- n jobs to be performed using m machines.
- Each job consists of m activities (of given processing times) to be executed in a specified order.
- Each activity requires a specified machine and each machine is required by a unique activity of each job.

The goal is to determine a solution with minimal makespan and prove the optimality of the solution. Job-shop scheduling is an NP-complete problem, known to be among the worst in this class.

Branching Scheme for JSSP

We build a B&B with constraint propagation (disjunctive + edge-finding). It is used to determine whether the problem with makespan at most D.

1. Select a resource among those required by unordered activities.
2. Select the activity to execute first among unordered ones (keep others as alternatives upon backtrack).
3. Iterate step 2 until all the activities on the resource are ordered.
4. Iterate steps 1 to 3 until all the activities that require a common resource are ordered.

Heuristics for JSSP: Resource Selection

- Some "critical" resources are, over some periods of time, more relied upon than others.
- It is, in general, very important to schedule critical resources first
- the "criticality" of a resource over $[t_1, t_2]$ is the "demand" (sum of the durations of activities) minus the "supply" $(t_2 - t_1)$.
- When all activities of the critical reosurce are ordered, there is very little work left to do!

Heuristics for JSSP: Activity Selection

We choose the activity

- with the soonest earliest start time;
- in addition, the activity with the soonest latest start time is chosen when two or several activities share the same earliest start time.

Computational Results

On a VERY old Pentium 90...

PROB	ALGO	BT	CPU	BT-PR	CPU-PR
BDG	ARC	176	.3	149	.1
	EF	14	.5	12	.1
MT10	EF	51980	2194	8137	355
ABZ5	EF	45187	1369	19574	590
ABZ6	EF	1068	52	302	14

The best techniques available so far allow to solve 10*10 instances in a few seconds.

Open-Shop Scheduling (OSSP)

- n jobs $\{J_1, \ldots, J_n\}$ that have to be scheduled on m parallel identical machines $\{M_1, \ldots, M_m\}$.
- Each job J_i consists of m operations $\{O_{i1}, \ldots, O_{im}\}$, each of which being described by its processing time p_{ij}.
- Operations of the same job cannot overlap in time and are assigned to a machine μ_{ij}
- The goal is to find a schedule with minimal makespan.

OSSP is NP-Hard and within its class it has been shown to be extremly hard to solve in practice.

Dominance property for OSSP

- The symetric counterpart of any solution of the OSSP is also a solution of the OSSP
- So we can arbitrary pick any operation O_{ij} and impose, *a priori*, that it starts in the right part of the schedule:

$$start(O_{ij}) \leq \left\lceil \frac{makespan - p_{ij}}{2} \right\rceil$$

Dominance property for OSSP

- How to determine the operation on which this domain reduction is to be achieved ?
- To estimate the impact of this dominance property, it is "tried" on each operation and each time, the sum of the domain size of each variables of the problem is computed
- The operation that minimizes this sum is picked.

Shaving

Shaving is a powerful propagation method it relies on other propagation mechanism:

- for all job, try to set $start(J_i)$ to r_i and propagate constraints.
- If a contradiction occurs, set $r_i := r_i + 1$.
- otherwise fail.

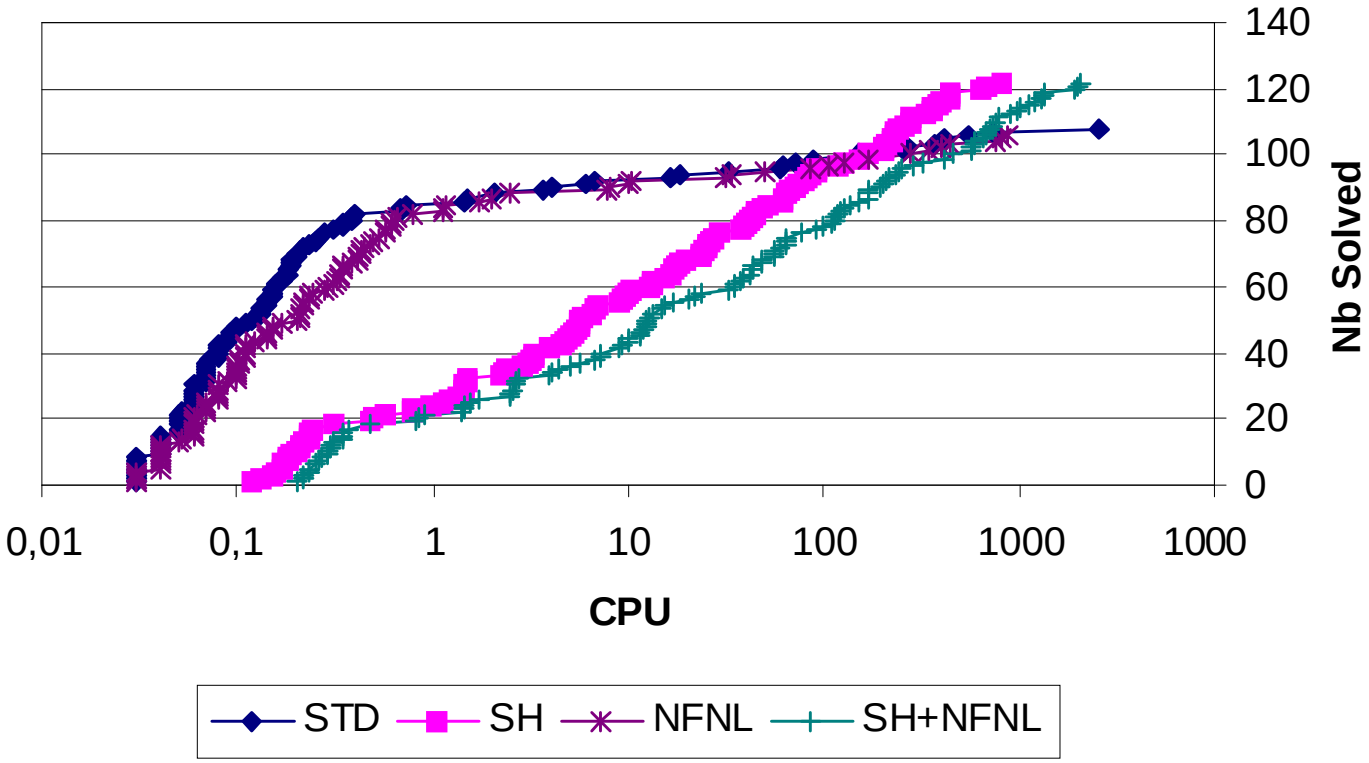
It may prove to be extremly costly but sometimes very efficient!

Solving the OSSP

To evaluate the efficiency of these propagation techniques, four variants of the algorithm have been tested:

- **STD.** No additional propagation is performed.
- **SH.** Shaving is performed.
- **NFNL.** Not-First Not-Last propagation is used.
- **SH+NFNL.** Both Shaving and Not-First Not-Last propagation are used.

Solving the OSSP



Aujourd'hui

A CP Model
Disjunctive Scheduling
- Constraint Propagation
- Job-Shop

Cumulative Scheduling
- Constraint Propagation
- RCPSP

Complex Objective Function
- Constraint Propagation
- Number of Late Jobs

ATC
- Problem Definition
- Branch & Cut
- Constraint Programing

Cumulative Scheduling

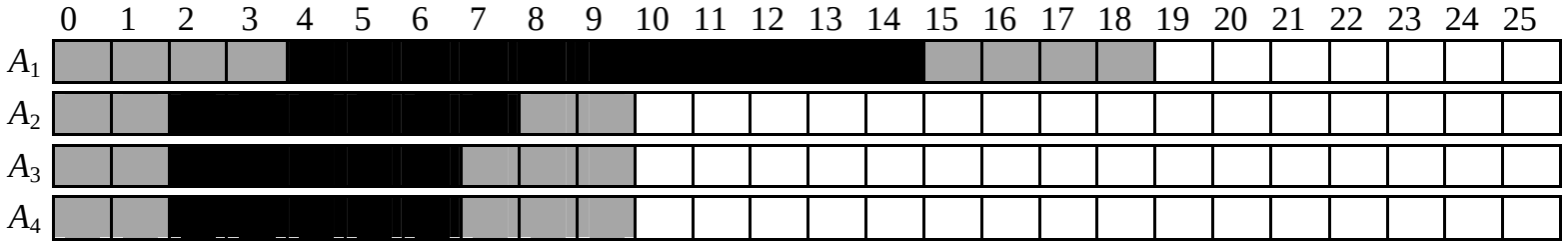
Given A_i and a resource R whose capacity is C, let c_i be the amount of R required by A_i.

$$\forall t \sum_{start(A_i) \leq t < end(A_i)} c_i \leq C$$

Non-Preemptive Problems

- Time-Table Constraint and disjunctive constraints are easiely adapted.
- Edge-Finding Not-First Not-Last deductions rules and energetic reasoning can also be adapted.

Edge Finding Example



Four activities on the same resource with capacity 2

The sum of the energy of the activities is 22, whereas the available energy between times 0 and 10 is 20. This implies that A_1 cannot be started at time $0, 1, 2, 3$.

Edge Finding

Proposition

Let Ω be a subset of jobs and let $A_i \notin \Omega$. Then, (i) if

$$p_{\Omega \cup \{A_i\}} > (d_\Omega - r_{\Omega \cup \{A_i\}})C, \tag{4}$$

then all activities in Ω end before the end of A_i, and (ii) if

$$p_{\Omega \cup \{A_i\}} > (d_{\Omega \cup \{A_i\}} - r_\Omega)C,$$

then all activities in Ω start after the start of A_i.

Edge Finding: Adjustment

Let Ω be a set of activities that end before the end of $A_i \notin \Omega$. With $rest(\Omega, c_i) = p_\Omega - (d_\Omega - r_\Omega)(C - c_i)$, it is derived that if $rest(\Omega, c_i) > 0$, a lower bound on the earliest end time of all activities in Ω, and thus a lower bound on r_i, equals $r_\Omega + \lceil rest(\Omega, c_i)/c_i \rceil$.
This lower bound has to be computed for all subsets Ω' of Ω. All this can be done in $O(n^2)$.

Energetic Reasoning

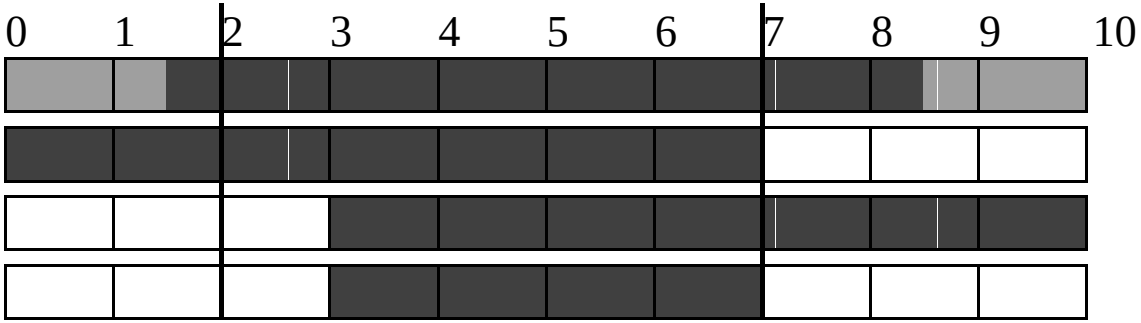
Given A_i and $[t_1, t_2]$, let $W_{Sh}(A_i, t_1, t_2)$, be the "left-shift / right-shift" required energy consumption of A_i over $[t_1, t_2]$. It is c_i times the minimum of

- $t_2 - t_1$, the length of the interval;
- $p_i^+(t_1) = \max(0, p_i - \max(0, t_1 - r_i))$, the number of time units during which A_i executes after time t_1 if A_i is left-shifted
- $p_i^-(t_2) = \max(0, p_i - \max(0, d_i - t_2))$, the number of time units during which A_i executes before time t_2 if A_i is right-shifted

Energetic Reasoning: Example

The required energy consumption of A_1 over $[2, 7]$ is 8. Indeed, at least 4 time units of A_1 have to be executed in [2, 7]; *i.e.*, $W_{Sh}(A, 2, 7) = 2 \min(5, 5, 4) = 8$.

	r_i	d_i	p_i	c_i
A_1	0	10	7	2



Energetic Reasoning: Necessary Condition

The left-shift / right-shift overall required energy consumption $W_{Sh}(t_1, t_2)$ over $[t_1, t_2]$ is the sum over all activities A_i of $W_{Sh}(A_i, t_1, t_2)$.

Proposition

If there is a feasible schedule then

$$\forall t_1, \forall t_2 W_{Sh}(t_1, t_2) \leq C(t_2 - t_1)$$

Energetic Reasoning: Necessary Condition

On what intervals do we have to compute $W_{Sh}(t_1, t_2)$? $O(n^2)$ only but they have a very complex structure.
In practice we limit ourselves to $t_1 \in O_1$ and $t_2 \in O_2$ with

$$\begin{aligned} O_1 &= \{r_i, 1 \leq i \leq n\} \cup \{d_i - p_i, 1 \leq i \leq n\} \cup \\ &\quad \{r_i + p_i, 1 \leq i \leq n\} \\ O_2 &= \{d_i, 1 \leq i \leq n\} \cup \{r_i + p_i, 1 \leq i \leq n\} \cup \\ &\quad \{d_i - p_i, 1 \leq i \leq n\} \end{aligned}$$

Energetic Reasoning: Adjustments

Proposition

If

$$W_{Sh}(t_1, t_2) - W_{Sh}(A_i, t_1, t_2) + c_i \min(t_2 - t_1, p_i^+(t_1)) > C(t_2 - t_1)$$

then a valid lower bound of the start time of A_i is

$$t_2 - \frac{1}{c_i}(C(t_2 - t_1) - W_{Sh}(t_1, t_2) + W_{Sh}(A_i, t_1, t_2)) \qquad (5)$$

Energetic Reasoning: Adjustments

for All relevant time-intervals $[t_1, t_2]$ **do**
  $W := 0$
  for $i \in \{1, \cdots, n\}$ **do**
    $W := W + c_i \min(t_2 - t_1, p_i^+(t_1), p_i^-(t_2))$
  if $W > C(t_2 - t_1)$ **then**
    Contradiction detected, backtrack
  else
    for $i \in \{1, \cdots, n\}$ **do**
      $SL := C(t_2 - t_1) - W + c_i \min(t_2 - t_1, p_i^+(t_1), p_i^-(t_2))$
      if $SL < c_i \min(t_2 - t_1, p_i^+(t_1))$ **then**
        Adjust the earliest start time $r_i := \max(r_i, t_2 - \lceil SL/c_i \rceil)$
      if $SL < c_i \min(t_2 - t_1, p_i^-(t_2))$ **then**
        Adjust the latest end time $d_i := \min(d_i, t_1 + \lceil SL/c_i \rceil)$

Resource-Constrained Project Scheduling

Given

- A set of resources of given capacities,
- a set of non-interruptible activities of given processing times
- for each activity and each resource the amount of the resource required by the activity over its execution,

the goal of the RCPSP is to find a schedule meeting all the constraints whose makespan is minimal. The RCPSP is NP-hard in the strong sense.

Branching scheme for the RCPSP

1. Initialize the set of **selectable** acts to the complete set.
2. If all acts have fixed start times, a solution is found, exit. Otherwise, remove from the set of selectable acts those with fixed start times.
3. If the set of selectable acts is not empty, select an activity from the set, create a choice point and schedule the selected activity from its earliest start time to its earliest end time. Then goto step 2.
4. If the set of selectable acts is empty, backtrack.
5. Upon backtracking, mark the activity as not selectable as long as its earliest start has not changed. Then goto step 2.

Branching scheme for the RCPSP

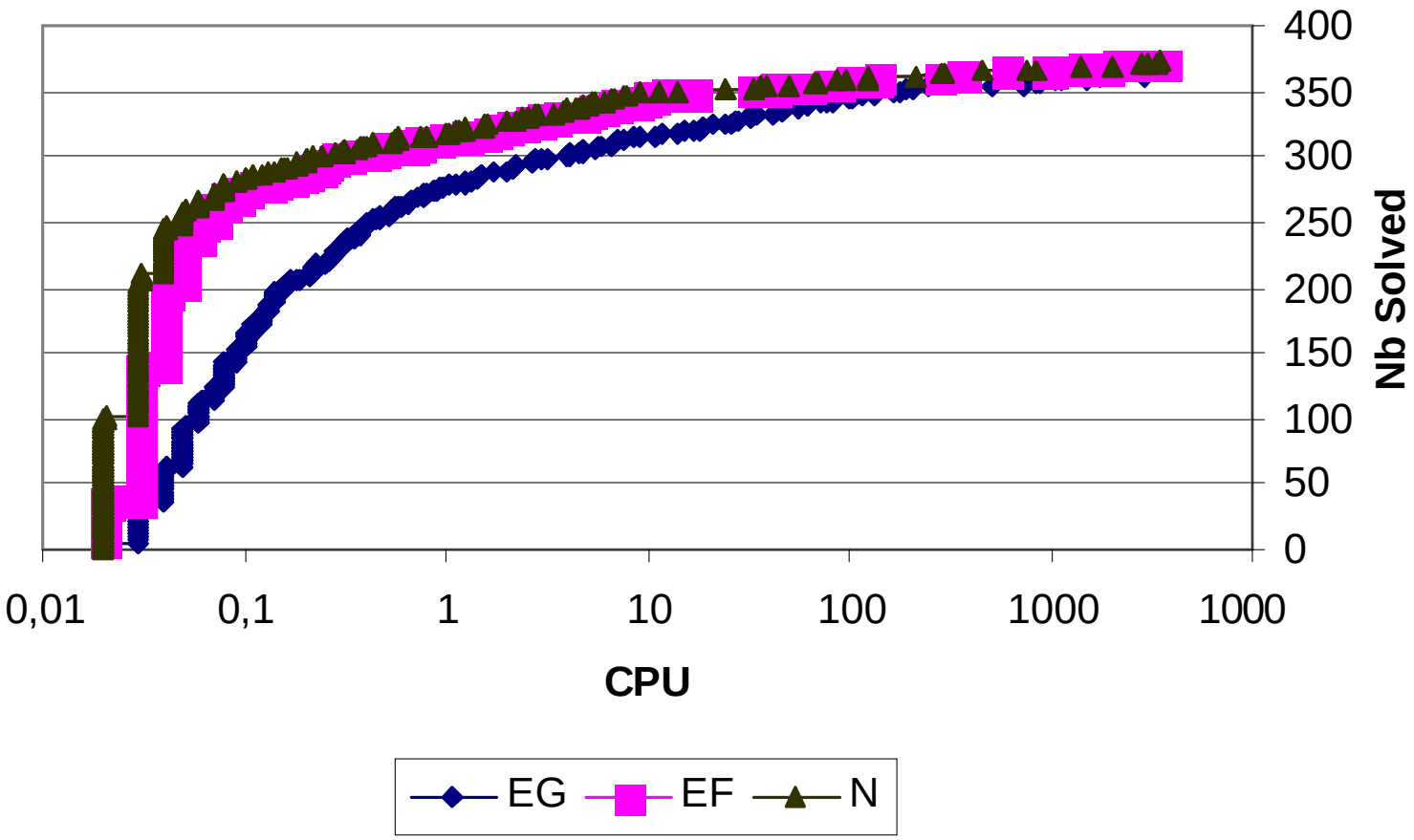
Rationale: The activity A_i chosen at step 3 must either start at its earliest start time or must be postponed to start later. But starting A_i later makes sense only if other activities prevent A_i from starting at its earliest start time, in which case the scheduling of these other activities must eventually result (thanks to constraint propagation!) in the update of the earliest start and end times of A_i.

Experimental Results

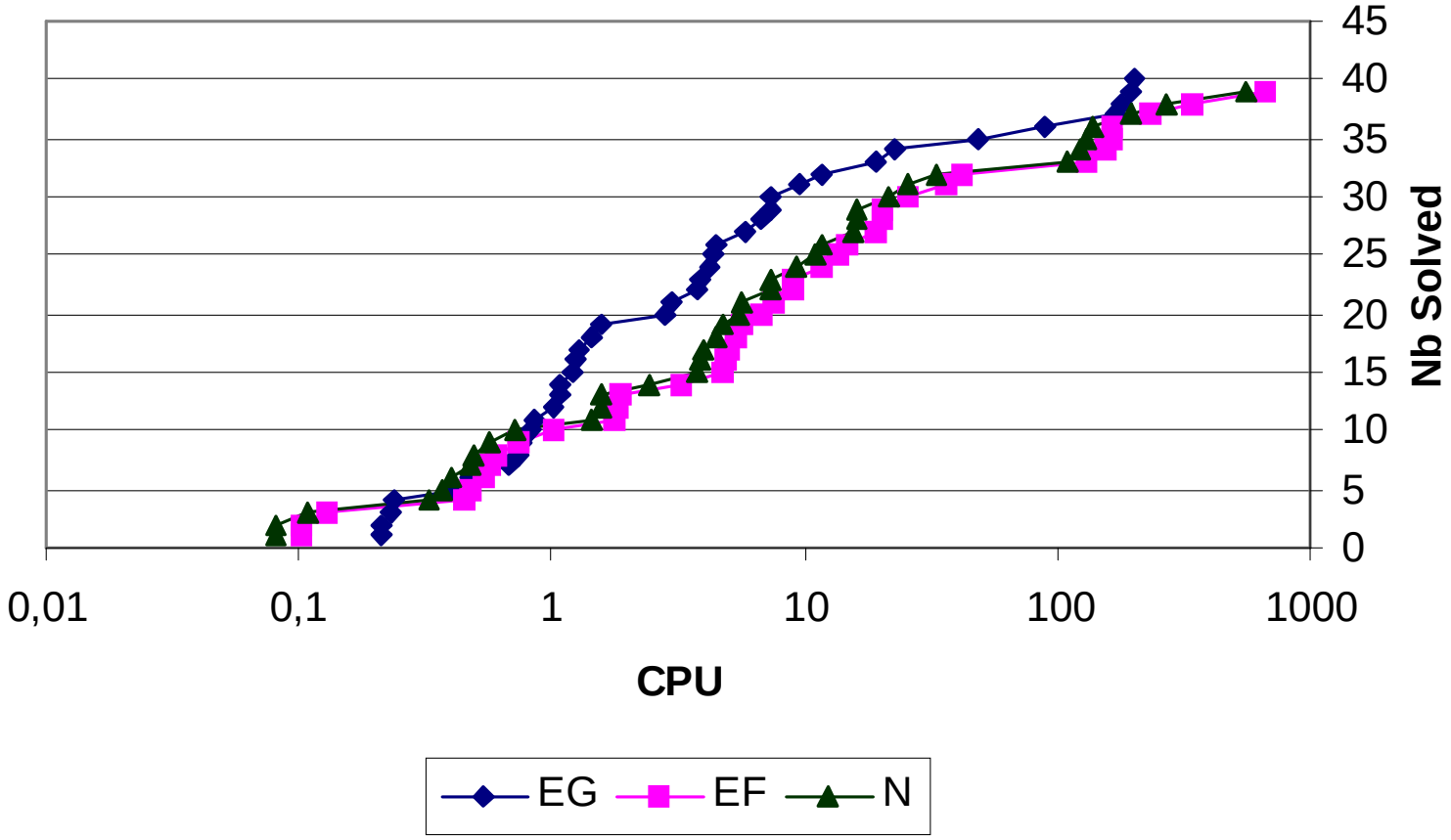
The three versions of the algorithm were tested on two sets of data.

- The "KSD" set, known to contain highly disjunctive instances (few activities can execute in parallel)
- The "BL" set where many activities can execute in parallel.

Experimental Results on the KSD set



Experimental Results on the BL set



Aujourd'hui

A CP Model
Disjunctive Scheduling
- Constraint Propagation
- Job-Shop

Cumulative Scheduling
- Constraint Propagation
- RCPSP

Complex Objective Function
- Constraint Propagation
- Number of Late Jobs

ATC
- Problem Definition
- Branch & Cut
- Constraint Programing

On the Propagation of a Min Sum Objective Functions

A variable *criterion* represents the value of the objective function. We have

$$criterion = F(end(A_1), \cdots, end(A_n)) = \sum_1^n F_i(end(A_i))$$

To be efficient constraint propagation has to consider resource constraints and obective function at the same time.
Here, $F = \sum_1^n w_i U_i$ with $U_i = 1$ if and only if $end(A_i) > \delta_i$.

Goal

- Compute good lower-bounds of the number of late jobs
- Given an *ub* on the number of late jobs, prove that some jobs are late or on-time and adjust time windows
- Look for dominance properties

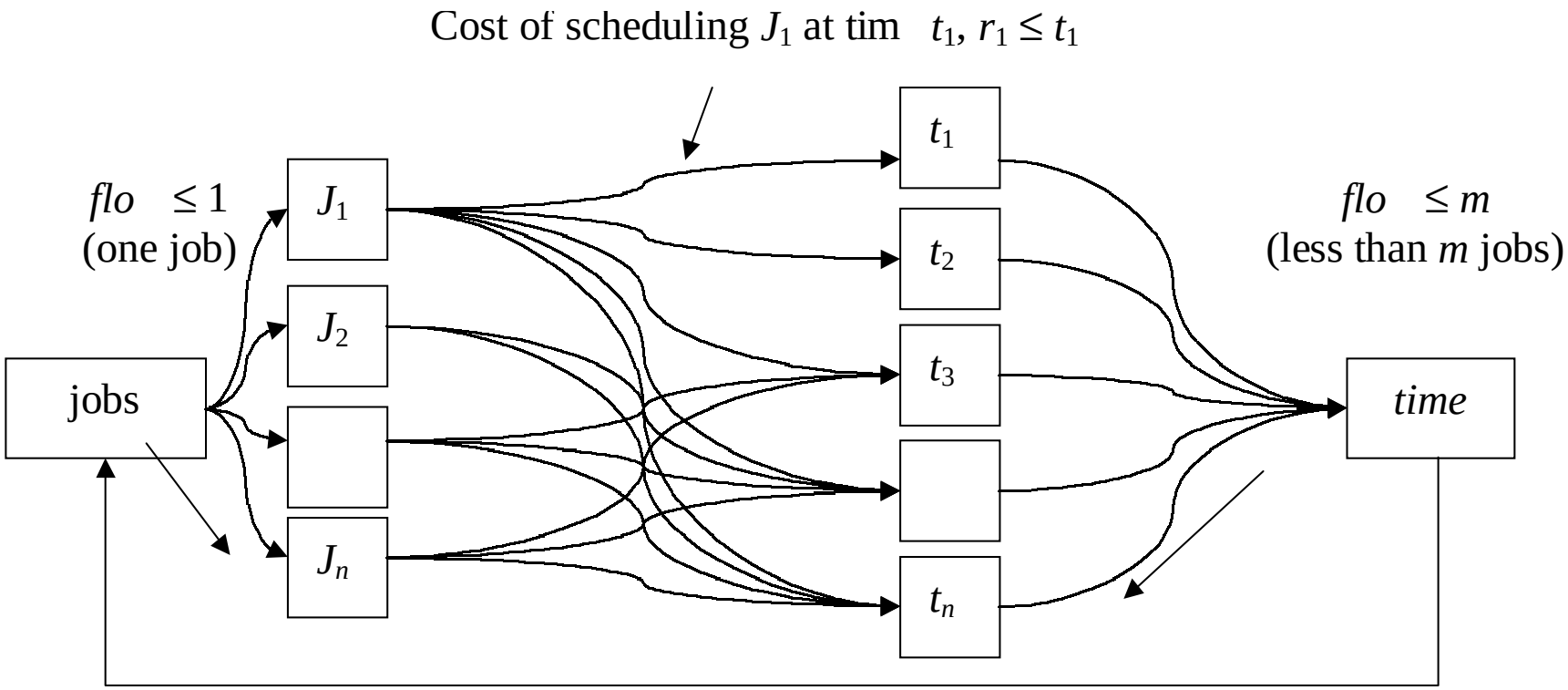
First of all, we have to theoretically study the problem.

Unit or Equal Processing Times

When processing times are unitary, the problem is polynomial.

- There is a polynomial number of relevant time points on left-shifted schedules.
- The cost of scheduling a job J_i at a time point t ($r_i \leq t$) is either 0 (if $t < \delta_i$) or w_i otherwise.
- At each time point t_i, less than m jobs can execute.

Unit or Equal Processing Times



Unweighted Case $w_i = 1$

- NP-hard in the strong sense
- But, when there is no release dates, polynomially solvable!
- When release and due dates of jobs are ordered similarly ($r_i < r_j \Rightarrow \delta_i \leq \delta_j$), the problem is solvable in $O(n^2)$

1 machine *vs.* m machines

- Recall that the *fully elastic* relaxation transforms a cumulative resource constraint into a preemptive one machine one
- If the earliest start time of the activity A_i is adjusted to x on the preemptive problem, the corresponding adjustment will be $\lceil \frac{x}{C} \rceil$.
- Lower bounds can be obtained by relaxing data to reach a polynomial case.

Lower Bounds for the 1 machine case

Proposition

There is a feasible preemptive schedule iff over any interval $[t_1, t_2]$, *the sum of the processing times of* $\{A_i : [t_1 \leq r_i] \wedge [d_i \leq t_2]\}$ *is lower than or equal to* $t_2 - t_1$.

It is well known that relevant values for t_1 and t_2 are respectively the release dates and the deadlines.

Lower Bounds for the 1 machine case

We introduce a binary decision variable x_i per activity (on-time *vs.* late) and let us define a MIP

$$\begin{aligned} S(t_1, t_2) &= \{A_i : r_i \geq t_1 \wedge d_i \leq t_2\} \\ P(t_1, t_2) &= \{A_i : r_i \geq t_1 \wedge d_i > t_2 \ \wedge \delta_i \leq t_2\} \end{aligned}$$

$$\min \sum_{1}^{n} w_i(1 - x_i)$$

$$\text{u.c.} \begin{cases} \forall t_1, \forall t_2 > t_1, \sum_{S(t_1,t_2)} p_i + \sum_{P(t_1,t_2)} p_i x_i \leq t_2 - t_1 \\ \forall i \in \{1, \cdots, n\}, x_i \in \{0, 1\} \end{cases}$$

Lower Bounds for the 1 machine case

The optimum of the MIP is the preemptive optimum of the problem and it is easy to see that the relevant values of t_1 and t_2 are the release dates, the due dates and the deadlines. Now focus on the continuous relaxation

$$\min \sum_{1}^{n} w_i(1 - x_i)$$

$$\text{u.c.} \begin{cases} \forall t_1 \in \{r_i\}, \forall t_2 \in \{d_i\} \cup \{\delta_i\} \ (t_2 > t_1), \\ \quad \sum_{S(t_1,t_2)} p_i + \sum_{P(t_1,t_2)} p_i x_i \leq t_2 - t_1 \\ \forall i, r_i + p_i > \delta_i \Rightarrow x_i = 0 \\ \forall i \in \{1, \cdots, n\}, x_i \in [0, 1] \end{cases} \tag{6}$$

Lower Bounds for the 1 machine case

The LP can be solved by a linear package but it can also be solved in $O(n^2 \log n)$ by a greedy algorithm.
Sort activities in non-increasing order of w_i/p_i (cost per unit).

Proposition

The largest vector (according to the lexicographical order) satisfying all the constraints of 6 realizes the optimum of 6.

Lower Bounds for the 1 machine case

An $O(n^4)$ algorithm to comute the violations

for $i := 1$ to n **do**
  $X_i := 0.0$
for $i := 1$ to n **do**
  if $r_i + p_i \leq \delta_i$ (otherwise A_i is late, *i.e.*, $X_i := 0$) **then**
    $X_i := 1.0$, $Violation := 0$
    for all constraint (t_1, t_2) s.t. $t_1 \in \{r_x\}, t_2 \in \{d_x\} \cup \{\delta_x\}$ **do**
      $total := 0.0$
      for $u := 1$ to n **do**
        if $t_1 \leq r_u$ and $d_u \leq t_2$ (*i.e.*, $A_i \in S(t_1, t_2)$) **then**
          $total := total + p_u$
        if $t_1 \leq r_u$ and $\delta_u \leq t_2 < d_u$ (*i.e.*, $A_i \in P(t_1, t_2)$) **then**
          $total := total + p_u * X_u$
      $Violation := \max(Violation, total - (t_2 - t_1))$
    $X_i := (p_i - Violation)/p_i$

Lower Bounds for the 1 machine case

- Each time, we compute the maximum resource constraint violation if the activity is fully on-time (lines 6–18).
- Given this violation, the maximum value X_i that the variable x_i can take is computed.
- This algorithm runs in $O(n^4)$ since there are n activities A_i and since for each of them $O(n^2)$ violations are computed, each of them in linear time.

Lower Bounds for the 1 machine case

Act	*est*	*p*	*let*	*d*	*w*	*w/p*
A1	7	2	12	10	10	5,0
A2	3	7	12	10	30	4,3
A3	0	5	14	6	20	4,0
A4	4	3	18	9	5	1,7

Initial instance. In the following, we focus on the fictive activities.

Act	est	p	let
A'1	7	0	10
A'2	3	0	10
A'3	0	0	6
A'4	4	0	9
A'5	7	2	12
A'6	3	7	12
A'7	0	5	14
A'8	4	3	18

X1	0,0
X2	0,0
X3	0,0
X4	0,0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A'7	A'7	A'7	A'6	A'6	A'6	A'6	A'6	A'6	A'6	A'5	A'5	A'7	A'7	A'8	A'8	A'8	

violation 0

Step 1: Is there a schedule where A1, A, A3, A4 end before their latest end time ? Yes.

Act	est	p	let
A'1	7	2	10
A'2	3	0	10
A'3	0	0	6
A'4	4	0	9
A'5	7	0	12
A'6	3	7	12
A'7	0	5	14
A'8	4	3	18

X1	**1,0**
X2	0,0
X3	0,0
X4	0,0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A'7	A'7	A'7	A'6	A'6	A'6	A'6	A'6	A'1	A'1	A'6	A'6	A'6	A'7	A'7	A'8	A'8	A'8

violation 0

Step 2: Try to put A1 on time (i.e., the processing time of the fictive activities A'1 and A'5 are set to 2 and 0) and compute JPS. The maximal violation is 0 hence, X1 = 1.0.

Act	est	p	let
A'1	**7**	**2**	**10**
A'2	3	7	10
A'3	0	0	6
A'4	4	0	9
A'5	**7**	**0**	**12**
A'6	3	0	12
A'7	0	5	14
A'8	4	3	18

X1	**1,0**
X2	**0,7**
X3	0,0
X4	0,0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A'7	A'7	A'7	A'2	A'2	A'2	A'2	A'2	A'2	A'2	**A'1**	**A'1**	A'7	A'7	A'8	A'8	A'8	

violation 2

Step 3: Try to put A2 on time (i.e., the processing time of the fictive activities A'2 and A'6 are set to 7 and 0) and compute JPS. The maximal violation is 2 hence, at most 5 units of A2 can be on time, i.e., X2 = 5/7.

Act	est	p	let
A'1	**7**	**2**	**10**
A'2	**3**	**5**	**10**
A'3	0	5	6
A'4	4	0	9
A'5	**7**	**0**	**12**
A'6	**3**	**2**	**12**
A'7	0	0	14
A'8	4	3	18

X1	**1,0**
X2	**0,7**
X3	**0,6**
X4	0,0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A'3	A'3	A'3	A'3	A'3	A'2	A'2	A'2	A'2	A'2	**A'1**	**A'1**	**A'6**	A'6	A'8	A'8	A'8	

violation 2

Step 4: Try to put A3 on time (i.e., the processing time of the fictive activities A'3 and A'7 are set to 5 and 0) and compute JPS. The maximal violation is 2 hence, at most 3 units of A3 can be on time, i.e., X3 = 3/5.

Act	est	p	let
A'1	**7**	**2**	**10**
A'2	**3**	**5**	**10**
A'3	**0**	**3**	**6**
A'4	4	3	9
A'5	**7**	**0**	**12**
A'6	**3**	**2**	**12**
A'7	**0**	**2**	**14**
A'8	4	0	18

X1	**1,0**
X2	**0,7**
X3	**0,6**
X4	**0,0**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A'3	A'3	A'3	A'2	A'4	A'4	A'4	A'2	A'2	A'2	**A'2**	**A'1**	**A'1**	**A'6**	**A'6**	**A'7**	**A'7**	

violation 3

Step 5: Try to put A4 on time (i.e., the processing time of the fictive activities A'4 and A'8 are set to 3 and 0) and compute JPS. The maximal violation is 3 hence, A4 is late, i.e., X4 = 0.

Constraint Propagation for $\sum w_i U_i$

- Let A_u denote an activity that can be either late or on-time ($eet_u \leq \delta_u < d_u$)
- Our objective is to compute efficiently a lower bound of the weighted number of late activities if A_u is on-time (conversely if A_u is late).
- If this lower bound is greater than *criterion* then, A_u must be late (conversely on-time).
- Straightforward but expensive agorithm

Constraint Propagation for $\sum w_i U_i$

First we relax deadlines to a large value and we compute the optimum X of the new LP.

$$\min \sum_{1}^{n} w_i(1 - x_i)$$
$$\text{u.c.} \begin{cases} \forall r_j, \forall \delta_k > r_j, \sum_{P(r_j,\delta_k)} p_i x_i \leq \delta_k - r_j \\ \forall i, r_i + p_i > \delta_i \Rightarrow x_i = 0 \\ \forall i, d_i \leq \delta_i \Rightarrow x_i = 1 \\ \forall i \in \{1, \cdots, n\}, x_i \in [0, 1] \end{cases} \quad (7)$$

Constraint Propagation for $\sum w_i U_i$

Let (8) be the linear program (7) to which the constraint $x_u = 1$ has been added and let Xo be the optimal vector of (8).

$$\min \sum_{1}^{n} w_i(1 - x_i)$$

$$\text{u.c.} \begin{cases} \forall r_j, \forall \delta_k > r_j, \sum_{P(r_j,\delta_k)} p_i x_i \leq \delta_k - r_j \\ \forall i, r_i + p_i > \delta_i \Rightarrow x_i = 0 \\ \forall i, d_i \leq \delta_i \Rightarrow x_i = 1 \\ \forall i \in \{1, \cdots, n\}, x_i \in [0, 1] \\ x_u = 1 \end{cases} \tag{8}$$

Constraint Propagation for $\sum w_i U_i$

Propositions 6 and 7 exhibit two relations that X and Xo satisfy.

Proposition

$\sum p_i Xo_i \leq \sum p_i X_i$

Proposition

$\forall i \neq u, Xo_i \leq X_i$

Constraint Propagation for $\sum w_i U_i$

Thanks to Propositions 6 and 7, we can add the constraints $\sum p_i Xo_i \leq \sum p_i X_i$ and $\forall i \neq u, x_i \leq X_i$ to the LP (8). We can also relax the resource constraints.

$$\min \sum_{1}^{n} w_i(1 - x_i)$$
$$\text{u.c.} \begin{cases} \forall r_j, \forall \delta_k > r_j, \sum_{P(r_j,\delta_k)} p_i x_i \leq \delta_k - r_j \\ \sum p_i x_i \leq \sum p_i X_i \\ \forall i \neq u, x_i \leq X_i \\ x_u = 1 \\ \forall i \in \{1, \cdots, n\}, x_i \in [0, 1] \end{cases} \quad (9)$$

Constraint Propagation for $\sum w_i U_i$

A linear time greedy algorithm for late job detection

1: **for** $i := 1$ to n **do**
2: $\quad Xo_i := 0.0$
3: $Xo_u := 1$
4: $MaxVal := \sum p_i X_i - p_u$
5: **for** $i := 1$ to n and $i \neq u$ **do**
6: $\quad Xo_i := \min(X_i, MaxVal/p_i)$
7: $\quad MaxVal = MaxVal - p_i * Xo_i$

Constraint Propagation for $\sum w_i U_i$

	r_i	d_i	δ_i	p_i	w_i	w_i/p_i
A_1	7	12	10	2	10	5.0
A_2	3	12	10	7	30	4.3
A_3	0	14	6	5	20	4.0
A_4	4	18	9	3	5	1.7

Deadlines are relaxed. The opt vector is $X_1 = 1$, $X_2 = 5/7$, $X_3 = 3/5$, $X_4 = 0$. The *lb* is then
$(0 * 10)/2 + (2 * 30)/7 + (2 * 20)/5 + (3 * 5)/3 = 21.57$.
Assume that *criterion* is $[22, 26]$ and put A_4 on-time. The relaxed opt is $Xo_1 = 1$, $Xo_2 = 5/7$, $Xo_3 = 0$, $Xo_4 = 1$ and
$lb = (0 * 10)/2 + (2 * 30)/7 + (5 * 20)/5 + (0 * 5)/3 = 28.57$

Minimizing the number of late jobs

An instance of the decision-variant of this problem consists of $\{J_1, ..., J_n\}$, a set of m identical parallel machines , and an integer W. Each of the jobs J_i is described by a release date r_i, a due-date δ_i, a processing time p_i, and a weight w_i.
The problem is to find an assignment of start times such that

- less than m jobs are scheduled at each time point,
- each job starts after its release date,
- the weighted number of jobs that end after their due date is lower than or equal to W.

Dominance Properties

We rely on the observation that on any solution, if a large job J_j is on-time and is scheduled inside the time window $[r_i, \delta_i]$ of a smaller job J_i that is late and if J_i is heavier than J_j, the jobs J_i and J_j can be "exchanged", *i.e.*, J_i becomes on-time and J_j becomes late.

Dominance Properties

Definition: For any pair of jobs J_i, J_j, $J_i \prec J_j$ if and only if

$$\begin{cases} (p_i < p_j) \vee (p_i = p_j \wedge i < j) \\ w_i \geq w_j \\ r_i + p_i \leq r_j + p_j \\ \delta_j - p_j \leq \delta_i - p_i \end{cases} \tag{10}$$

Dominance Properties

Proposition

There is an optimal schedule such that for any pair of jobs $J_i \prec J_j$, if J_j is on-time then J_i is also on-time, i.e.,

$$\forall J_i, \forall J_j, \neg[(J_i \prec J_j) \wedge (end(J_j) \leq \delta_j) \wedge (end(J_i) > \delta_i)] \qquad (11)$$

Branching Scheme

While there are some jobs that can be late or on-time (*i.e.*, $eet_i \leq \delta_i < d_i$),

1. select a job J_i such that $eet_i \leq \delta_i < d_i$
2. make J_i on-time, *i.e.*, $end(A_i) \leq \delta_i$ (if a backtrack occurs, J_i is late),
3. apply dominance properties and propagate constraints
4. check that there exists a feasible schedule of the jobs, *i.e.*, a schedule where all jobs are scheduled in their time-window, (if not, the problem is inconsistent and a backtrack occurs).

Job Selection Heuristic

- Let U be the set of jobs that can be late or on-time.
- Let $\max(w_i/p_i)$ be the maximum value of w_i/p_i among activities in U and finally, let S be the subset of the jobs J_j in U such that $w_j/p_j > 0.9 * \max(w_i/p_i)$.
- Among jobs in S, we select a job whose time window, if it becomes on-time,*i.e.*, $[r_i, \delta_i]$, is the largest.

This heuristic "bets" that it is better to schedule small jobs with large time windows rather than large jobs with tight time windows.

Feasibility Check

Even on a single machine, this Feasibility Check is NP-hard in the strong sense! But it's easy to solve in practice.
We use a B&B similar to the one described for the RCPSP. The Left-Shift Right-Shift propagation technique is used at this point.

Experimental Results

n	% solved	Avg. CPU	Max CPU	Avg. BCK
10	100.0	0.2	0.7	2.8
20	100.0	5.3	35.2	17.7
30	99.6	84.4	567.0	48.4
40	94.8	217.6	589.9	73.4
50	80.0	419.9	585.2	39.5

Table: Experimental results for $m = 1$

Experimental Results

n	% solved	Avg. CPU	Max CPU	Avg. BCK
10	100.0	0.2	0.7	4.9
20	100.0	6.8	44.8	60.4
30	96.7	94.9	554.5	601.7
40	87.0	203.0	570.8	3252.5
50	74.4	475.8	597.0	947.0

Table: Experimental results for $m = 3$

Experimental Results

n	% solved	Avg. CPU	Max CPU	Avg. BCK
10	100.0	0.1	0.2	0.2
20	100.0	9.9	67.0	238.3
30	98.2	70.5	551.6	1361.6
40	91.1	128.6	530.5	6676.6
50	84.8	336.4	539.4	6937.7

Table: Experimental results for $m = 6$

Aujourd'hui

A CP Model
Disjunctive Scheduling
- Constraint Propagation
- Job-Shop

Cumulative Scheduling
- Constraint Propagation
- RCPSP

Complex Objective Function
- Constraint Propagation
- Number of Late Jobs

ATC
- Problem Definition
- Branch & Cut
- Constraint Programing

Airport arrival and departure management

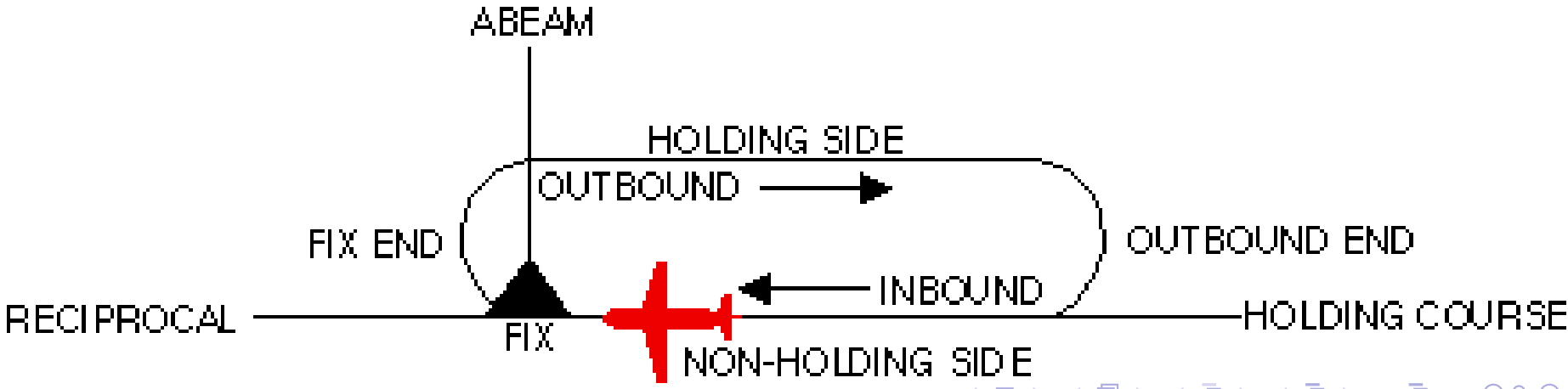
"Time slot Allocation" Problem

- Both the number of runways and the number of air traffic controllers are limited
- Traffic has to be carefully planned to limit peaks of activity
- while matching as much as possible airlines landing times requirements.

Unpredictable delays → hardly impossible to precisely schedule aircraft in advance → initial planning has to be refined on line in the TRACON

Terminal Radar Approach CONontrol

- TRACON controls aircraft approaching and departing between 5 and 50 miles of the airport.
- Air traffic controllers make some aircraft wait before landing
  - speed can be slightly decreased to increase the arrival time and/or the flight plan can be lengthened by a "Vector For Spacing"
  - "Holding Patterns" generate a constant prescribed delay for an aircraft.



Terminal Radar Approach CONtrol

a set of *time windows* in which the landing is possible can be associated to each aircraft entering the TRACON area

- Each time window = combination of holding patterns + vectors for spacing

Wake Vortex

- At landing time, air-traffic controllers space aircraft to reduce the WAKE VORTEX effects.
- wake turbulence is one of the most limiting factors for the take-off and landing frequency in the airports.
- Wake vortex effects are proportional to aircraft weight
  - the lighter the following aircraft, the more it suffers from wake vortex effects, demanding greater separation from the leading aircraft

Our problem

- A single runway + several aircraft with identical weight waiting around
- Objective: assign landing times to aircraft (within their time windows). Two objective functions are considered:
  - Given a fixed minimum time p between any two consecutive landings, minimize the maximal number of times a plane enters a Holding Pattern.
  - Maximize the minimum time elapsed between two consecutive landings.

Problem definition

- n aircraft in the TRACON
- A set of s_i time intervals is assigned to each aircraft i
- two objective functions are considered:
  1. Given a maximal number of times a plane can enter a holding pattern, maximize the minimum time elapsed between two consecutive landings.
  2. Given a minimum time which can elapse between two consecutive landings, minimize the maximal number of times a plane enters a holding pattern.

The decision variants of these two problems are identical :

Single machine scheduling problem in which n "landing" jobs with identical processing time p have to be scheduled within some time windows on a single "runway" machine

Problem definition

THE RUNWAY SCHEDULING PROBLEM (DECISION VARIANT)

input integers $n, p, (s_1, \ldots, s_n)$, $(r_{1,1}, d_{1,1}, \ldots, r_{1,s_1}, d_{1,s_1})$, $\ldots$, $(r_{n,1}, d_{n,1}, \ldots, r_{n,s_n}, d_{n,s_n})$.

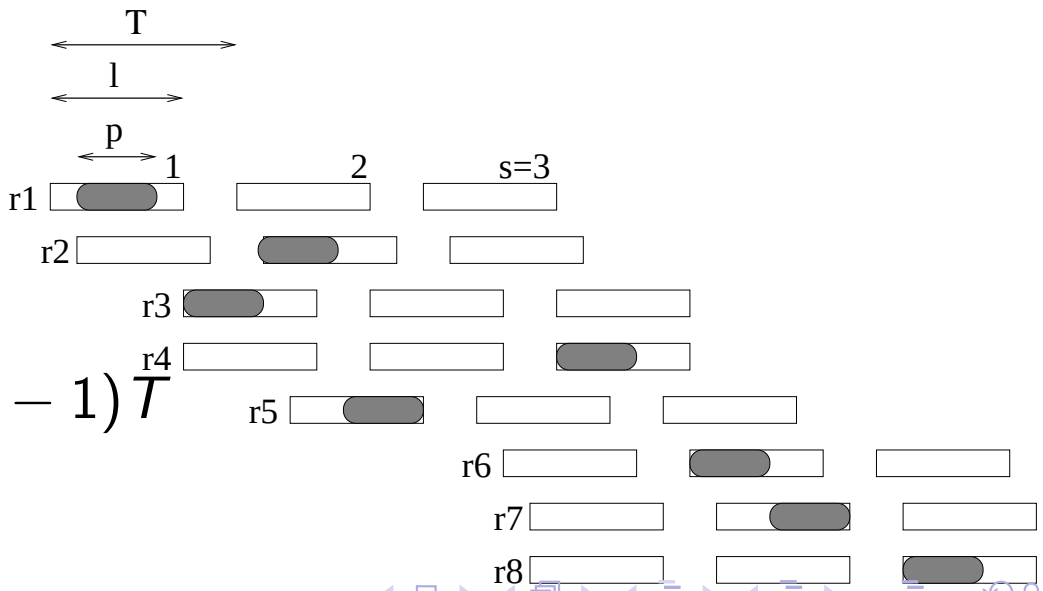
meaning each job i has processing time p and has to be fully scheduled (*i.e.*, started and completed) in one of the intervals $[r_{iu}, d_{iu}]$. We wish to find a schedule such that every job is scheduled non-preemptively, and no two jobs overlap.

output a set of starting times $S_1, \ldots, S_n \in \mathbb{N}$ such that (1) $\forall i \in \{1, \ldots n\}$, $\exists j \in \{1, \ldots, s_i\}$ such that $S_i \in [r_{ij}, d_{ij} - p]$ and (2) $\forall i, k \in \{1, \ldots n\}$ with $k \neq i$, $|S_i - S_k| \geq p$.

Problem definition : the mono-pattern case

- Special case: single pattern for the time windows (same number s of windows per aircraft, same window size l and identical distances T between windows)
- All aircraft are identical and where a single holding pattern is considered

$$\begin{cases} \forall i \leq n, s_i = s \\ \forall i \leq n, \forall j \leq s, d_{ij} - r_{ij} = l \\ \forall i \leq n, \forall j \leq s, r_{ij} = r_i + (j-1)T \end{cases}$$



Complexity

- NP-Complete as soon as processing times are larger than 2 and when we have more than three time windows per aircraft.
- When $p = 2$, $\forall i, s_i = 3$ and $\forall i \leq n, \forall j \leq s, d_{ij} - r_{ij} = 2$, this problem is a direct generalization of the problem of scheduling short tasks with few starting times [**?**].

Complexity (Unit processing times)

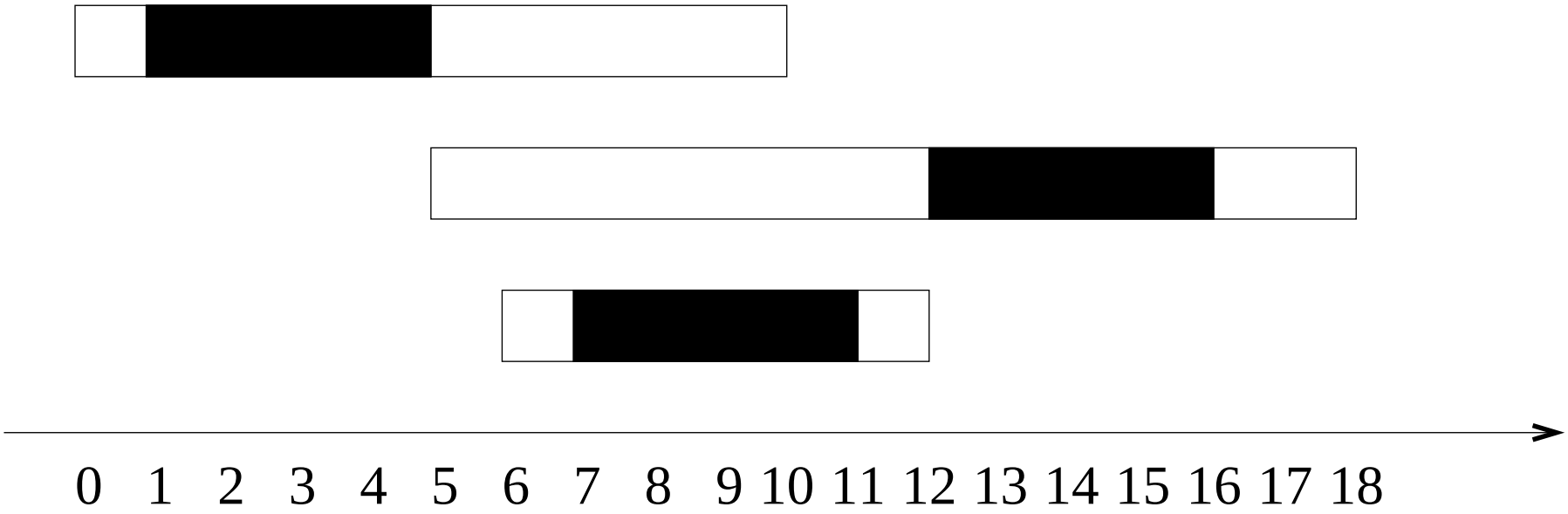
- When $p = 1$, the runway scheduling problem can be solved in polynomial time.
- Flow problem in a bipartite graph $G = (\mathcal{J} \cup \mathcal{T}, E)$
  - $\mathcal{J}$ is the set of job vertices $1, .., n$
  - $\mathcal{T}$ is the set of time points t corresponding to the beginning or to the end of a time window of a job
  - There is an edge between job i and t iff job i can start at t.
  - A source vertex σ is connected to all jobs and time vertices are connected to a sink ε.
  - All capacities are 1 except on the edges connecting a time vertex t to ε (capacity = distance between t and the next time point in $\mathcal{T}$)
  - There is a feasible schedule iff there is a flow of capacity n.

Complexity (Single Time Window)

- Arbitrary p, but one window per job (no holding pattern)
- Single machine scheduling problem with time windows and equal processing times
  - Care! Equal processing times does not reduce to unit processing times
- Garey, Johnson, Simons and Tarjan's Algorithm based on EDD
  - EDD (Earliest DeaDline) builds a schedule chronologically: Whenever the machine is idle, select among jobs that are released before or at the current time point, the job with minimal deadline . Schedule the job and iterate.
  - When preemption is allowed, EDD rule computes a feasible schedule iff one such exists
  - but...

Complexity (Single Time Window)

EDD fails in the non preemptive case.



Complexity (Single Time Window)

- Introduce of a set of forbidden regions F in which no job can start on any feasible schedule.
- The modified EDD rule keeps the schedule idle when the current time point t belongs to F

$U := \{1, ..., n\}$
$t := \min_{i \in U} r_i$
while $U \neq \emptyset$ **do**
  $t := \min(t, \min_{i \in U} r_i)$
  if $t \in F$ **then**
    $t := \min\{t' \geq t : t' \notin F\}$
  Find k be the job with smallest deadline s.t. $r_k \leq t$
  Start job k at time t, $U := U - \{k\}$, $t := t + p$

Complexity (Single Time Window)

- Crucial point: How to compute the set F of "forbidden regions" ?
- F is built step by step, starting from $F = \emptyset$.
- Given a set of jobs X, compute an upper bound lst of the largest time point such that there is a feasible schedule of the jobs in X that is idle before lst and in which no job starts in F
- If lst is smaller than $\min_{i \in X} r_i$ then there is no feasible schedule.
- If $lst - p < \min_{i \in X} r_i$ then no job can start after $lst - p$ and before $\min_{i \in X} r_i$
- The interval $[lst - p + 1, \min_{i \in X} r_i - 1]$ is added to the set of forbidden regions F.

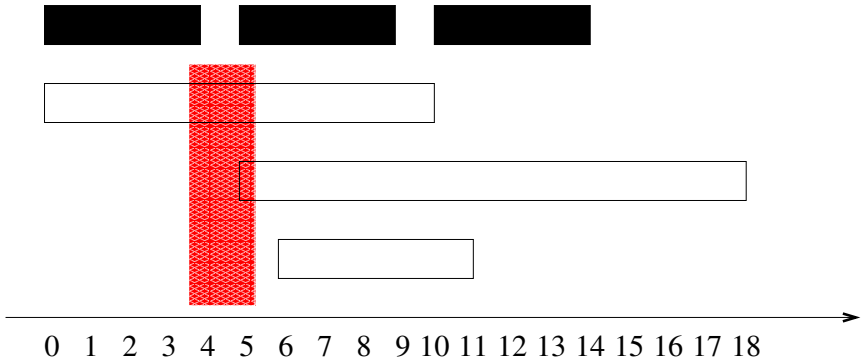
Forbidden regions

- EDD fails in this case because no task should start at 4 or 5

- A forbidden region $[4, 5]$ is created
- EDD is modified in order to avoid forbidden regions

Forbidden regions

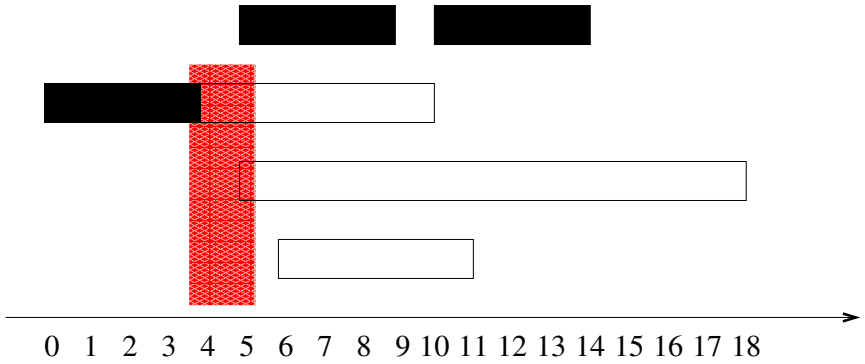
- EDD fails in this case because no task should start at 4 or 5



- A forbidden region $[4, 5]$ is created
- EDD is modified in order to avoid forbidden regions

Forbidden regions

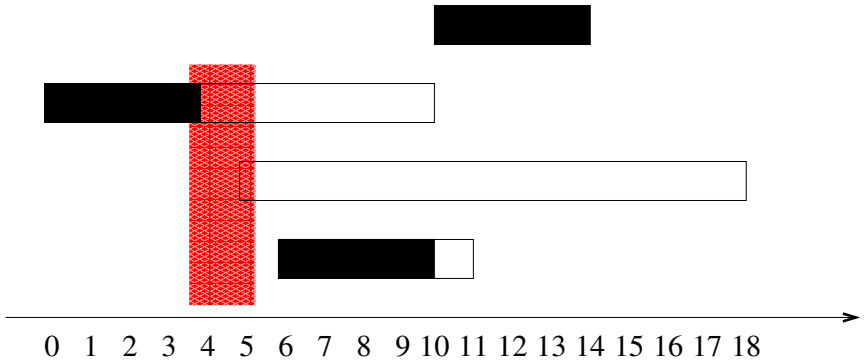
- EDD fails in this case because no task should start at 4 or 5



- A forbidden region $[4, 5]$ is created
- EDD is modified in order to avoid forbidden regions

Forbidden regions

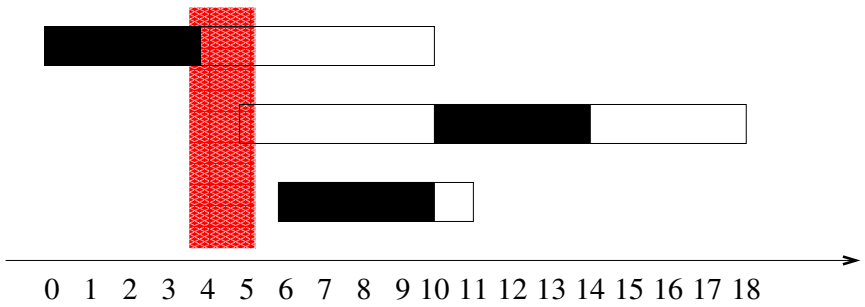
- EDD fails in this case because no task should start at 4 or 5



- A forbidden region $[4, 5]$ is created
- EDD is modified in order to avoid forbidden regions

Forbidden regions

- EDD fails in this case because no task should start at 4 or 5



- A forbidden region $[4, 5]$ is created
- EDD is modified in order to avoid forbidden regions

Complexity (tight windows)

- Special case: Time windows are 'tight", i.e., their size is exactly p
- *interval selection* problem (extensively studied by Arkin, Silverberg, Crama, Spieksma,Erlebach, Fischetti, Martello, Toth, Kroon, Salomon, etc..)
- When there are two time windows per job and when they are "tight" : 2-SAT.
  - We basically have to decide which one to pick.
  - To every job i associate a boolean variable x_i which is true if i is scheduled in its first time window and false if it is scheduled in its second time window.
  - For every pair of intersecting time windows with associated literals a, b there will be a clause $\bar{a} \vee \bar{b}$

Complexity

- Several other polynomial case
- Open problem : 2 windows that are not tight
- Now, come back to the general problem

MIP (Mixed Integer Programming)

- The general problem studied by Bayen, Tomlin, Ye and Zhang
  - $P.$ = elapsed time between consecutive landings
  - Each job i is associated with a starting time variable t_i.
  - x_{iu} variables are binary variables that indicate whether aircraft i is schedule in one of its u-th first time windows or not.
  - y_{ij} variables are binary sequencing variables that indicate whether aircraft i precedes j or not.

$$\min P$$

$$\begin{cases} t_i \geq r_{i1} & 1 \leq i \leq n \\ t_i \leq d_{is_i} & 1 \leq i \leq n \\ t_i \geq r_{iu} - Mx_{i\ u-1} & 1 \leq i \leq n, 2 \leq u \leq s_i \\ t_i \leq d_{iu} + M(1 - x_{i\ u}) & 1 \leq i \leq n, 1 \leq u \leq s_i - 1 \\ t_i - t_j \geq P - M'y_{ij} & 1 \leq j < i \leq n \\ t_i - t_j \leq M'(1 - y_{ij}) - P & 1 \leq j < i \leq n \\ x_{iu} \in \{0,1\} & 1 \leq i \leq n, 1 \leq u \leq s_i \\ y_{ij} \in \{0,1\} & 1 \leq i \leq n, 1 \leq j \leq n \end{cases}$$

Small MIP for the mono pattern

- Special case: single pattern for the time windows (same number s of windows per aircraft, same window size l and identical distances T between windows)
- To every j-th time window of every job i we associate a binary variable $x_{ij} \in \{0,1\}$, (job i is proc. in this window or not)
- In every interval $[a,b]$ at most $\lfloor (b-a)/p \rfloor$ jobs

$$\begin{cases} \sum_{u=1}^{s_i} x_{iu} = 1 & 1 \leq i \leq n \\ \sum_{[r_{iu},d_{iu}] \subseteq [a,b]} x_{iu} \leq \lfloor \frac{b-a}{p} \rfloor & 1 \leq i \leq n, a \in \{r_{jv}\}, b \in \{d_{jv}\}, a < b \\ x_{iu} \in \{0,1\} & 1 \leq i \leq n, 1 \leq u \leq s_{iu} \end{cases}$$

There is a solution to this MIP if and only if there is a feasible schedule

Small MIP (general case)

There is a solution to the previous MIP if and only if there is a feasible *preemptive* schedule.

What's next ?

- If there is non-preemptive schedule then we have found a solution to the runway problem.
- Otherwise, the assignment found by MIP does not lead to a feasible non-preemptive schedule.
- Hence, we can add an *iterative cut* to the MIP stating that the sum of the x variables that were equal to 1 in the previous solution cannot be greater than $n - 1$.
- We then re-solve the new MIP and iterate the same process.

This is a "Branch and Cut" Algorithm

Cuts to tighten the formulation

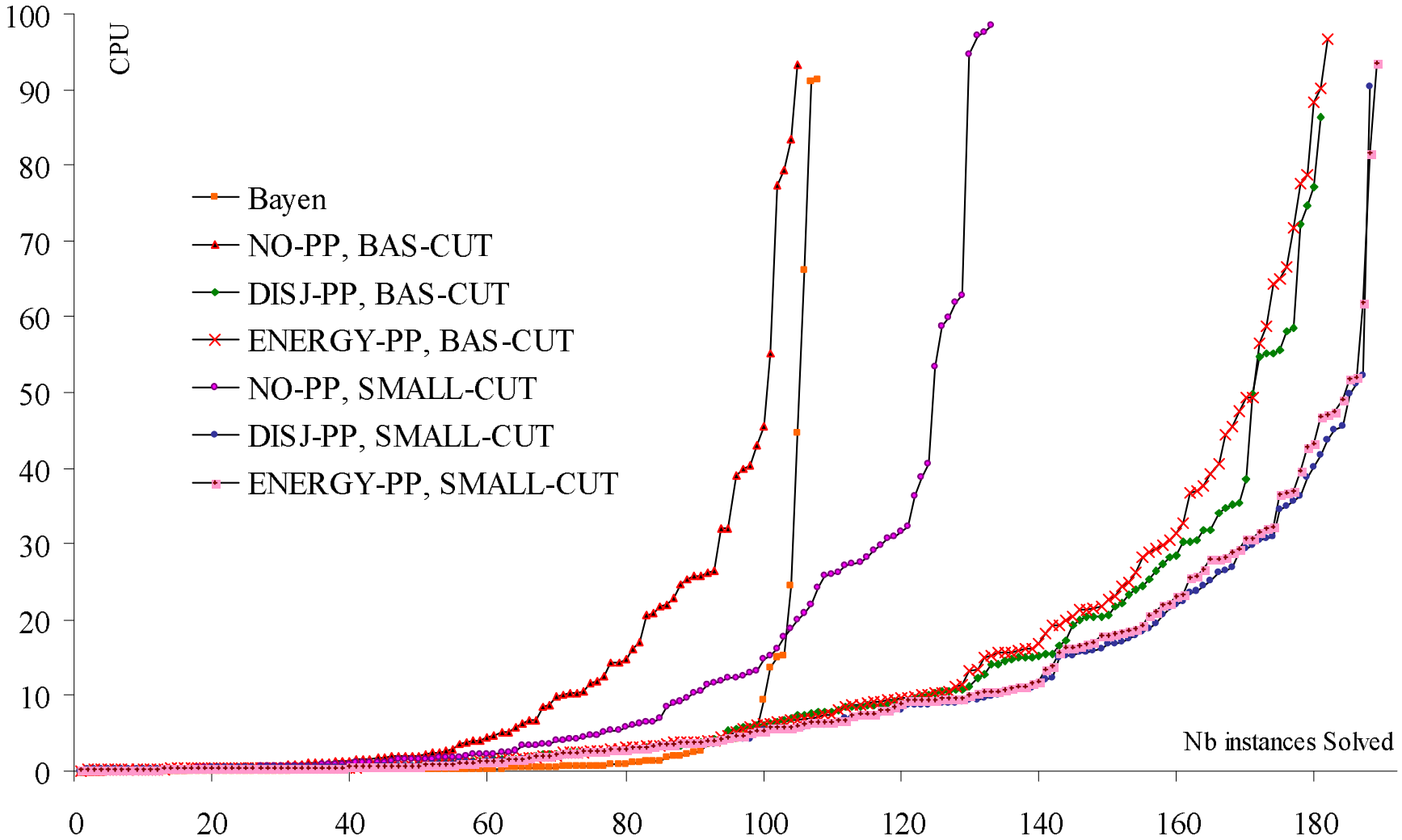
Objective : get rid of assignments that lead to preemptive schedules only

Disjunctive Cuts

- These cuts "$x_{iu} + x_{jv} \leq 1$" prevent the simultaneous assignment of a pair of jobs i, j to some time windows $[r_{iu}, d_{iu})$, $[r_{jv}, d_{jv})$ when this leads to an infeasible situation.
  - Relax all time windows of jobs $k \notin \{i, j\}$ to one single time window $[r_{k1}, d_{ks_k})$ that contain all initial ones.
  - Tighten the time windows of i, j to $[r_{iu}, d_{iu})$ and $[r_{jv}, d_{jv})$ respectively.
  - feasibility test can be achieved thanks to Garey et al. algorithm.

2 other cuts: Energy cuts + Small iterative cuts

Two other cuts



CP

- Solve the same problem in a CP framework
  - Start time variables + disjunctive constraints
- Main contribution : A global inter-distance constraint.
  - N finite domain variables $\{S_1, ..., S_N\}$
  - Integer p
  - Inter-distance constraints $\forall i, j, |S_i - S_j| \geq p$
  - This constraint represents a generalization of All-Different constraint

CP

- The consistency of the constraint is easy to compute (Garey et al)
- How to propagate the constraint ?
  - Propagation = adjust time bounds of jobs to remove the starting times that do not lead to non-preemptive schedules
  - Arc-B-Consistency of the constraint = for each job i there is a schedule in which it starts at the first (resp. last) point in its window
  - Why is it useful ? Reduce the search space

Arc-B-Consistency

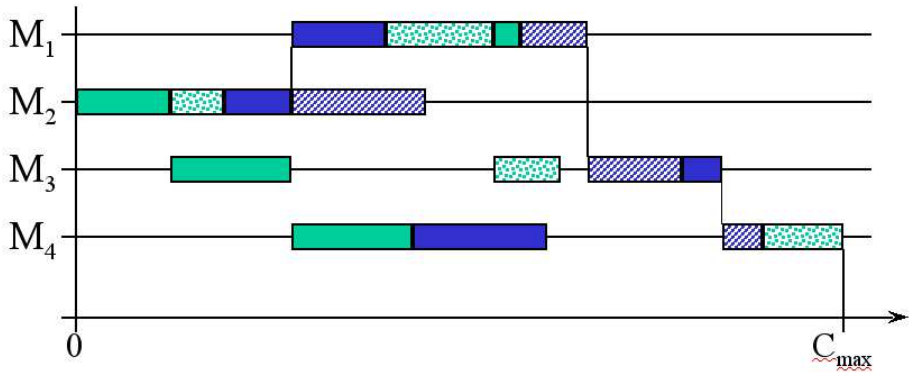
Simple algorithm

- Try all possible starting times of job i
- Apply the consistency check (and remove starting times that make the constraint inconsistent)
- Pseudopolynomial complexity.
  - Can be improved as there are few (polynomial number) possible starting times

Another polynomial propagation scheme has been setup. Leads to a 95% reduction of the search space.

Interdistance Constraint for job-shop

- A set of jobs consisting of tasks to be executed on several machines in given order
- Decision variant of this problem is considered
- All tasks on same machine have equal length in our instances



- More instances solved using our constraint than with any other constreaint propagation scheme
- 40% gain in number of backtracks on the instances solved by both variants