Scheduling with Constraint Programming

- Job Shop
- Cumulative Job Shop

CP vs MIP: Task Sequencing

- We need to sequence a set of tasks on a machine
  - Each task i has a specific fixed processing time p_i
  - Each task can be started after its release date r_i, and must be completed before its deadline d_i
  - Tasks cannot overlap in time

Time is represented as a discrete set of time points, say 1, 2,…, H (H stands for horizon)

MIP model

- Variables
  - Binary variable x_{ij} represents whether task i starts at time period j
- Constraints
  - Each task starts on exactly one time point

    $\sum_j x_{ij} = 1$ for all tasks i
  - Respect release date and deadline

    $j^*x_{ij} = 0$ for all tasks i and $(j < r_i)$ or $(j > d_i - p_i)$

MIP model (cont' d)

- Tasks cannot overlap
  - variant 1

    $\sum_i x_{ij} <= 1$ for all time points j

    - we also need to take processing times into account; try as an exercise
  - variant 2
    - introduce binary variable b_{ik} representing whether task i comes before task k
    - must be linked to x_{ij}; we need to add constraints to make them consistent with one another (i.e., triplets of tasks); (try as an exercise)

CP model

- Variables
  - Let $start_i$ represent the starting time of task i

    takes a value from domain {1,2,…, H}
  - This immediately ensures that each task starts at exactly one time point

- Constraints
  - Respect release date and deadline

    $r_i \leq start_i \leq d_i - p_i$

CP model

- Tasks cannot overlap:

  for all tasks i and j

  $(start_i + p_i < start_j)$ OR $(start_j + p_j < start_i)$

  That's it!

- an even more compact model is possible for this problem, using a 'global' scheduling constraint

Benefits of CP model

- The number of CP variables is equal to the number of tasks, while the number of MIP variables depends also on the time granularity (for a horizon H, and n tasks, we have H*n binary variables x_{ij})

- The sequencing constraints are quite messy in MIP, but straightforward and intuitive in CP

Advanced Scheduling

- Basic building blocks
  - Activities $a_1, a_2, \ldots, a_n$
  - Resources $r_1, r_2, \ldots, r_m$
- Variables corresponding to activity a_i
- Each activity requires one or more units of 'energy' from one or more resources
- Each resource has a capacity (usually fixed)

Activities

- Activities are defined with:
  - $start(a_i)$ start time
  - $end(a_i)$ end time
  - $proc(a_i)$ processing time (with: $proc(a_i) = end(a_i) - start(a_i)$)
  - $size(a_i)$ or intensity or energy

- Activities can further be:
  - Optional
  - Have priority

- All of these are transformed into CP (finite domain) variable which can be used to model constraints directly

Resources

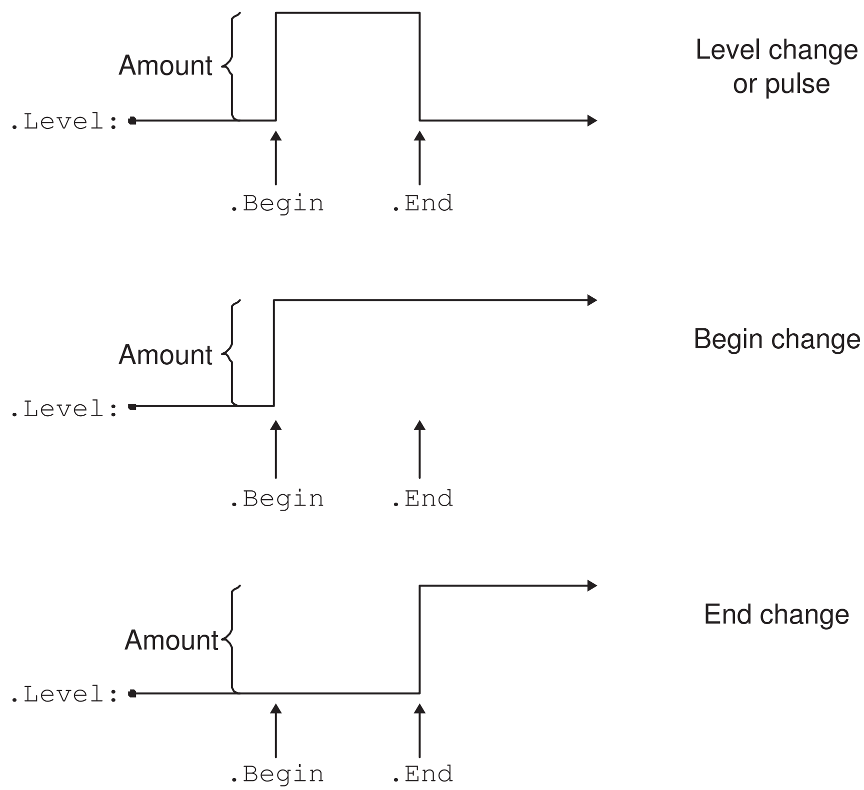
- There is essentially two types of resources:
  - Sequential
  - Parallel

- Sequential resources:
  - Execute only one activity at a time.
  - Allow for two form of precedencies
    - a “comes before” b, but c can be between a and b
    - or a “precedes” b, and nothing can be scheduled in between
  - Allow to define the first and last activities

- Transition
  - Occurs when switching from one activity to another one
  - Can be defined for each pair of activity (transitions between cities)
  - Can be defined for each pair of activity group (transitions between colors)

Resources

- Parallel resources
  - May execute many activities concurrently
  - Total size of activities run in parallel must not exceed resource capacity
  - Allow to track change in resource consumption level (a.k.a profile)



Scheduling constraints

- We have a set of possible constraints which can be stated

Scheduling Constraints	Interpretation
`cp::Span`(g,i,a_i)	The activity g spans the activities a_i g.Begin $= \min_i a_i$.Begin$\wedge$ g.End $= \max_i a_i$.End
`cp::Alternative`(g,i,a_i)	Activity g is the single selected activity a_i $\exists j : g = a_j \wedge \forall k, j \neq k : a_k$.present $= 0$
`cp::Synchronize`(g,i,a_i)	If g is present, all present activities a_i are scheduled at the same time. g.present $\Rightarrow (\forall i : a_i$.present $\Rightarrow g = a_i)$
Precedence Relations	
	When activities a and b are present and for a non-negative integer delay d
`cp::BeginBeforeBegin`(a,b,d)	a.Begin $+ d \leq b$.Begin
`cp::BeginBeforeEnd`(a,b,d)	a.Begin $+ d \leq b$.End
`cp::EndBeforeBegin`(a,b,d)	a.End $+ d \leq b$.Begin
`cp::EndBeforeEnd`(a,b,d)	a.End $+ d \leq b$.End
`cp::BeginAtBegin`(a,b,d)	a.Begin $+ d = b$.Begin
`cp::BeginAtEnd`(a,b,d)	a.Begin $+ d = b$.End
`cp::EndAtBegin`(a,b,d)	a.End $+ d = b$.Begin
`cp::EndAtEnd`(a,b,d)	a.End $+ d = b$.End
Adjacent Activity	
	r is the resource s is the scheduled activity e is extreme value (when s is first or last) a is absent value (s is not scheduled)
`cp::BeginOfNext`(r,s,e,a)	Beginning of next activity
`cp::BeginOfPrevious`(r,s,e,a)	Beginning of previous activity
`cp::EndOfNext`(r,s,e,a)	End of next activity
`cp::EndOfPrevious`(r,s,e,a)	End of previous activity
`cp::GroupOfNext`(r,s,e,a)	Group of next activity, see also page 316
`cp::GroupOfPrevious`(r,s,e,a)	Group of previous activity
`cp::LengthOfNext`(r,s,e,a)	Length of next activity
`cp::LengthOfPrevious`(r,s,e,a)	Length of previous activity
`cp::SizeOfNext`(r,s,e,a)	Size of next activity
`cp::SizeOfPrevious`(r,s,e,a)	Size of previous activity

Searching for solution

- During the solving process, constraint programming employs search heuristics that define the shape of the search tree, and the order in which the search tree nodes are visited.

- The shape of the search tree is typically defined by the order of the variables to branch on, and the corresponding value assignment.

- For example, to decide the next variable to branch on, a commonly used search heuristic is to choose a non-fixed variable with the minimum domain size, and assign it its minimum domain value.

- One can use the Priorities to force the system to branch first on activities with the highest priority.

Searching for solution

- Other heuristics are available…

Heuristic	Interpretation
Variable selection:	choose the non-fixed variable with:
`Automatic`	use the solver's default heuristic
`MinSize`	the smallest domain size
`MaxSize`	the largest domain size
`MinValue`	the smallest domain value
`MaxValue`	the largest domain value
Value selection:	assign:
`Automatic`	use the solver's default heuristic
`Min`	the smallest domain value
`Max`	the largest domain value
`Random`	a uniform-random domain value

Typical Objectives

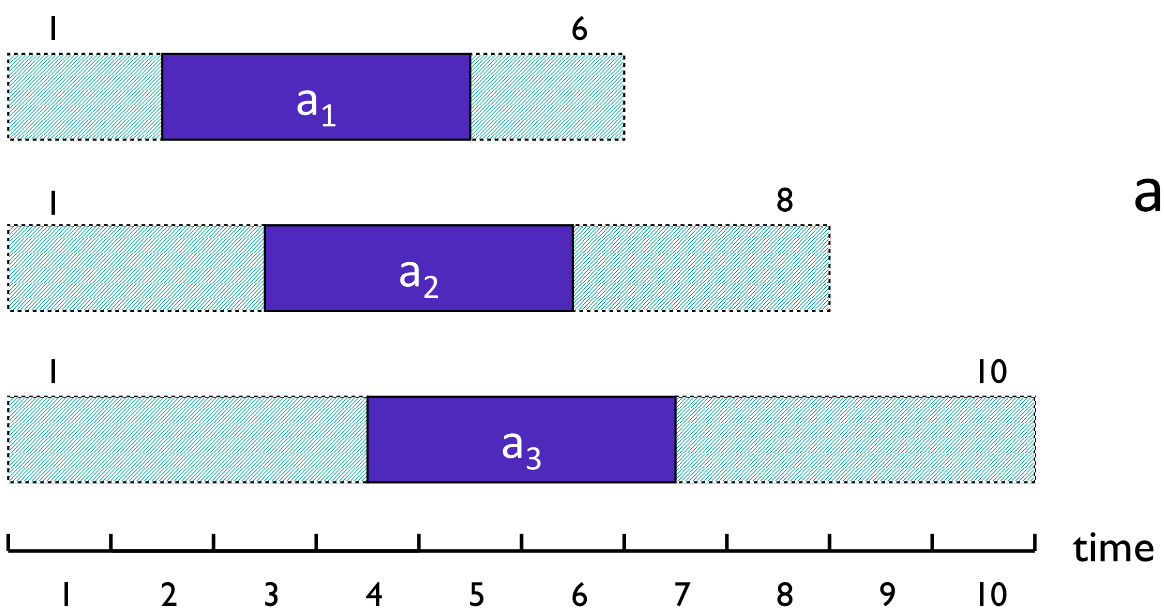
- Find feasible schedule
- Minimize makespan (latest end time)
- Minimize maximum tardiness (delay)
- Minimize total (weighted) number of late jobs

- Can you write these objectives using the variable derived from the activities ?

- Lets now look at filtering for scheduling constraints…

Sequential Scheduling (filtering)

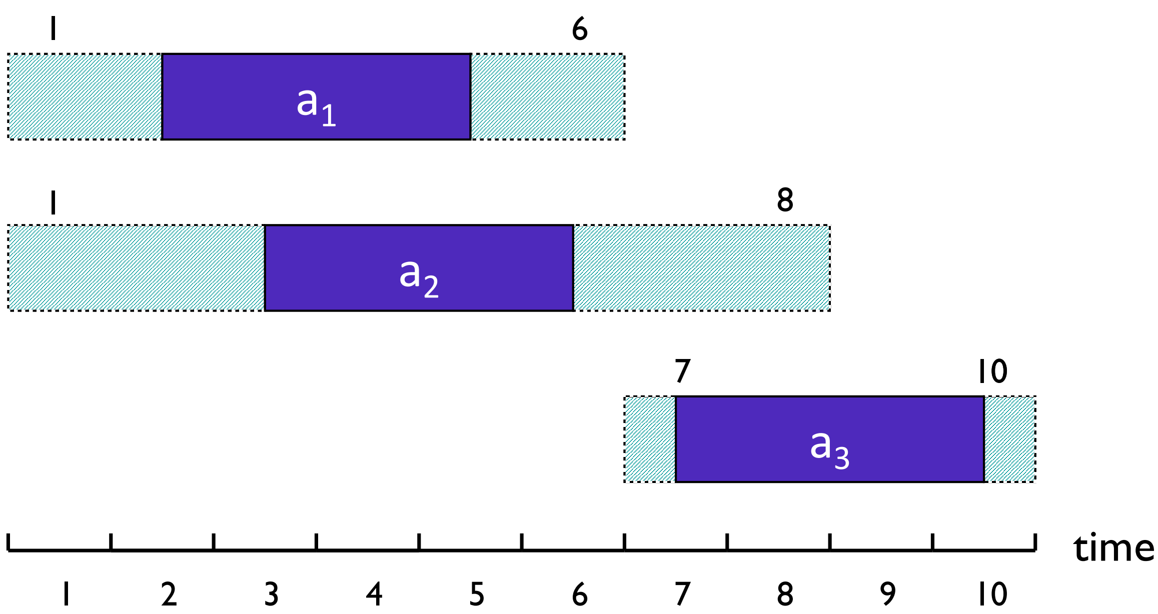
- Machine must execute three activities a_1, a_2, a_3 each with duration of 3 time units, time windows are indicated in figure. Activities cannot overlap in time.
- *Filtering task:* find earliest start time and latest end time for activities



filtering:
a_3 must start after time 6

Sequential Scheduling (filtering)

- Machine must execute three activities a_1, a_2, a_3 each with duration of 3 time units, time windows are indicated in figure. Activities cannot overlap in time.
- *Filtering task:* find earliest start time and latest end time for activities

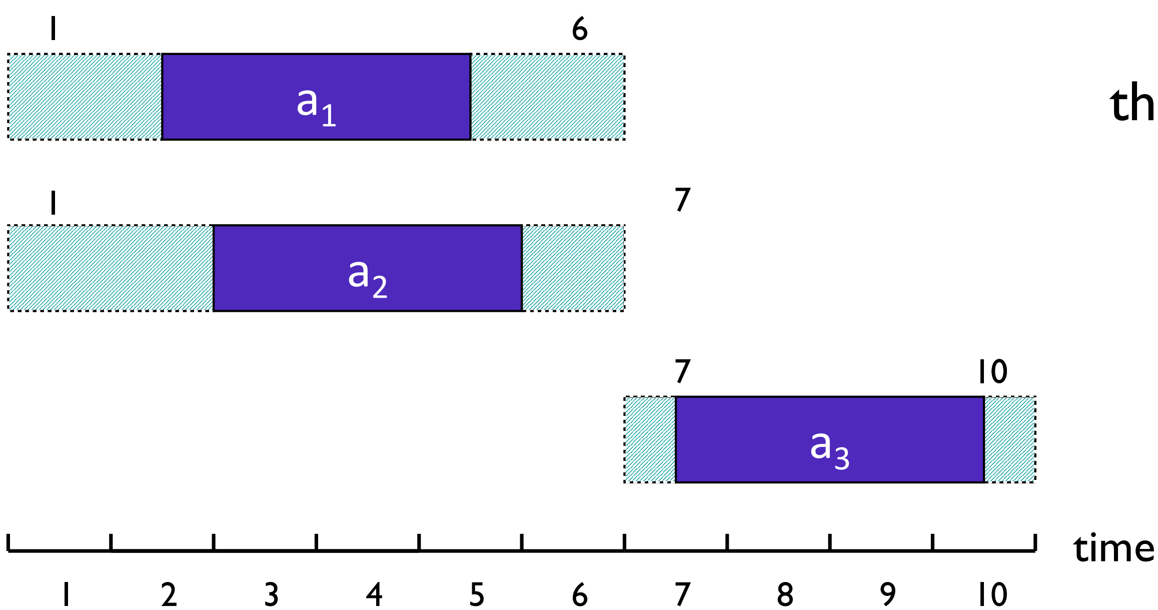


filtering:

a_2 must end before time 8

Sequential Scheduling (filtering)

- Machine must execute three activities a_1, a_2, a_3 each with duration of 3 time units, time windows are indicated in figure. Activities cannot overlap in time.
- *Filtering task:* find earliest start time and latest end time for activities



this domain filtering is applied in 'edge finding'

Task Sequencing Revisited

- Tasks cannot overlap: for all tasks i and j

  ($start_i$ + p_i < $start_j$) OR ($start_j$ + p_j < $start_i$)

- Can also be modeled with a single global constraint:

  SequentialSchedule([$start_1$,…, $start_n$], [p_1,…,p_n])

Advanced scheduling

- Several different filtering algorithms can be associated with the scheduling constraints
  - time-table, disjunctive, edge-finding, not-first not-last, network-flow based, precedence graph, ...
- These algorithms are called in sequence
- Dynamic search strategies can be defined using the information from the filtering algorithms
- In order to leverage the power of these algorithms, the model must explicitly use the dedicated scheduling syntax
  - i.e., activities, resources and global scheduling constraints

Integrated Methods

- Many applications contain an optimization component as well as a highly combinatorial (scheduling) component
- Use MIP or CP?

- Over the last decade, *integrated* methods combining CP, AI, and OR techniques have been developed (see conference CPAIOR)
  - Embed OR methods inside global constraints (e.g., network flows)
  - Double modeling (run CP and MIP separately)
  - Decomposition where MIP and CP solve different levels

Summarizing Strengths of CP

- Very expressive and intuitive modeling language
- Powerful domain filtering algorithms for global constraints
- Advanced search strategies
- Very effective on complex scheduling problems

Recognizing when to use CP

Definitely try **CP** if:

- ❑Finding feasible (or any) solution is more important than finding an optimal solution
- ❑The problem heavily relies on a ‘scheduling’ component: timetabling, employee rostering, production line sequencing, …

Definitely try **MIP** if:

- ❑Optimality is critical, and moreover the objective is naturally modeled as a linear expression
- ❑The problem structure corresponds to a `continuous’ allocation of resources, e.g., network models

Try an integrated **MIP+CP** approach

- ❑When both linear optimization and scheduling are present