Speaker: Olufikayo Adetunji
Scribe: Robert Woodward
Dates: April 8, 2013 & April 10, 2013

Topic: Constraint-Based Scheduling and Planning (CP Handbook, Chapter 22)

The discussion focused on scheduling rather than planning. First, we give preliminaries of scheduling and planning (scheduling is the focus here). Then give a formal definition of scheduling, and present extensions to it. After that, we go over three examples of scheduling problems. Following that, we discuss disjunctive then conjunctive constraint propagation, and finally methods for solving scheduling as an optimization problem.

Overall you should take away that modeling scheduling problems is very open! The variety of constraints, resources, priorities, and optimization criteria in scheduling is too wide to capture in a unique, generic constraint model. Unlike CSP where the model is abstract and generic enough to allow us to plug-and-play, solving a large number of problem instances with a simple backtrack search, in scheduling, specific constraints have to be carefully crafted to properly model the situation at hand. How do products depend on resources, on time? Can tasks be broken, resources shared? Are there set-up times for equipment? How to model buffers? Overall, your resources define constraints.

Preliminaries:

Scheduling task: Allocate activities to resources over time, respecting constraints.
Planning task: Construct a sequence of actions to transfer the initial state of the world into a goal state. In general,

- The tasks are not known in advanced. We need to find the tasks to be executed.
- We do not care about time (release date, due date) or the duration of activities. We only care about the determining the sequence of tasks, a partial order of precedence between tasks.
- Figure 1 shows the block world (recall from the AI class), where you want to get from the initial state to the goal state.

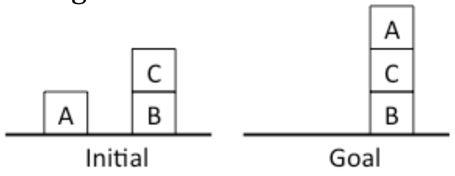


Figure 1: The initial and goal state of the block world.

The instructor interjected in the discussion inviting the class to further distinguish between scheduling and *resource allocate* where resources allocation is a simplification of scheduling.

- In resource allocation, we do not care about time, we only care about mapping resources.
- For example, Graduate TA assignment program, or allocating students to projects at CSE.

In *Constraint-Based* Scheduling & *Constraint-Based* Planning, the task is to solve Scheduling or Planning problems using CP techniques. However, the presentation focused on Constraint-Based Scheduling.

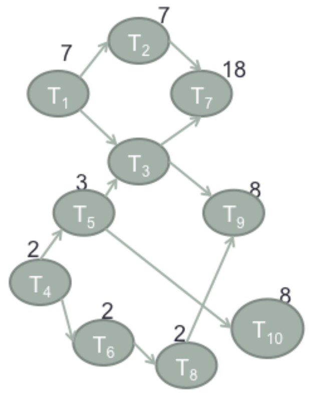


Figure 2: Precedence graph for bicycle assembly.

Simple Motivating Example:

Bicycle Assembly (Shown in Figure 2)

- 3 workers can perform tasks.
- 10 tasks with its own duration (T_1,...,T_{10}).
- Precedence constraint (e.g., T_1 must proceed T_2).
- No preemption (cannot interrupt tasks).

- Goal: Want to minimizes breaks in time the schedule.

Consider the random schedules in Figure 3 and optimal schedule in Figure 4. The optimal schedule (no time breaks) uses only two people (person 3 goes on vacation ☺).

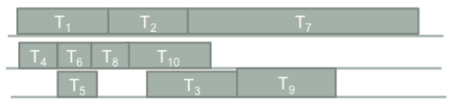


Figure 3: Random schedule for bicycle assembly.

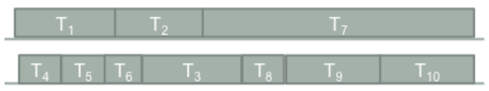


Figure 4: Optimal schedule for bicycle assembly.

Formal Definition of Scheduling:

Activity **A**: is an entity needing some resources & time

Has corresponding variables:

- start(**A**): Earliest start time of the activity
  - [est(**A**),lst(**A**)]
  - lst(**A**) = max(start(**A**)): latest start time
  - est(**A**) = min(start(**A**)): earliest start time
- end(**A**): Latest completion time of the activity
  - [eet(**A**),lct(**A**)]
  - eet(**A**) = min(end(**A**)), earliest end time
  - lct(**A**) = max(end(**A**)), latest competition time
- proc(**A**): processing time (duration) of the activity. Equivalent to est(**A**)-eet(**A**).

Note: start(**A**) and end(**A**) might also be represented as [r,d]: The *release* and *deadline* of A. r=est(**A**) and d=lct(**A**).

Types of Resources: Either *non-preemptive* or *preemptive*.

Non-preemptive: Activities cannot be interrupted: end(**A**)–start(**A**)=proc(**A**)

Preemptive: Activities can be interrupted: end(**A**)– start(**A**)≥proc(**A**).

Compute proc(**A**)=proc(**A**$_1$)+proc(**A**$_2$)+proc(**A**$_3$)

Figures 5 and 6 show examples of non-preemptive and preemptive schedules, respectively.

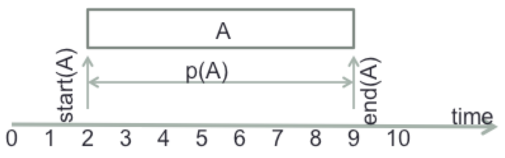


Figure 5: Non-preemptive schedule

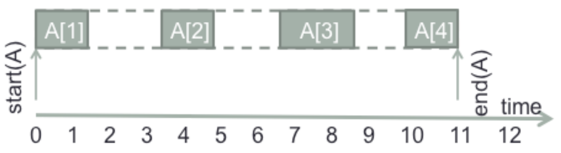


Figure 6: Preemptive schedule

Resource Constraints: Either *Disjunctive Scheduling* or *Cumulative Scheduling*.
Disjunctive Scheduling (also called Unary Constraints): All resources have a *unary* capacity (cap(**A**) = 1). Resources called machines, which can have at one activity executing at a time.
Cumulative Scheduling: Each activity uses some capacity of the resource, cap(**A**). Resources can execute in parallel if cap(**A**) is not exceeded.
Figures 7 and 8 show an example of disjunctive and cumulative scheduling, respectively.

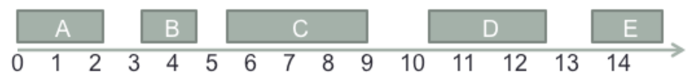


Figure 7: Disjunctive scheduling.

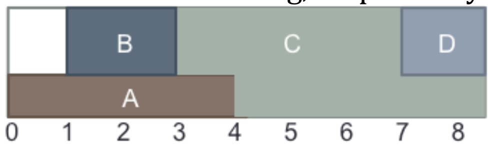


Figure 8: Cumulative scheduling.

Temporal Constraints: Either *precedence constraints* or *disjunctive constraints*.
We have already seen these when we studied Temporal CSPs! (See Temporal Reasoning/Temporal CSPs in the class handouts: http://cse.unl.edu/~choueiry/S13-921/handouts.html)

Precedence constraints: Sequencing of activities. **A**≪**B**, meaning end(**A**) ≤ start(**B**). Figure 9 shows an example where **A**≪**B**.

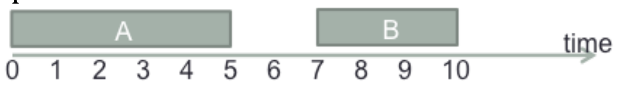


Figure 9: Activity A must come before activity B.

Disjunctive constraint: Activities cannot overlap. **A** ≪ **B** or **B** ≪ **A**, meaning end(**A**)≤start(**B**) or end(**B**)≤ start(**A**)

Optimization: We want to optimize an *objective function*.

- Objective function uses a variable criterion (equal to the value of the objective function)
- Here, we focus on the *makespan* criterion (completion time of last activity)
    - Introduce L, proc(L) = 0. Want to minimize L.
    - Add precedence constraint to each T with no successor (because L must comes last). Figure 10 shows an example of adding L.

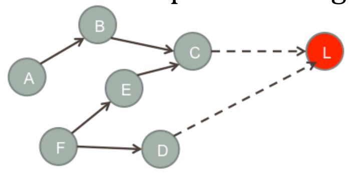


Figure 10: Adding the makespan L to the precedence graph.

Extensions: Below *alternative resources*, *reservoir resource, breakable activities,* and *state resources* are described, which are other types of resources that can be modeled (depending on the problem).

Alternative Resources: Activity **A** must be scheduled from its alternative resources.

- altern(**A**) = {R_1, R_2, R_3}; **A** can only be scheduled on resources R_1, R_2, or R_3.
- These different alternatives can have different costs.
- Can use disjunctive scheduling or cumulative scheduling (same as the regular disjunctive / cumulative scheduling, but must respect the alternative resources).

Question by Daniel D.: If workers are not equal, for example one can do a task faster, can this be represented with alternative resources?
Answer: Yes: the alternative resources can have different costs. However, might not be able to use cumulative scheduling (cumulative schedule can combine resources into one, which might not be acceptable if the workers are different).
As for modeling different time, when the duration depends on the resource the activity is being executed. You would have to enrich the model to not only focus on cost of resource, but also consider the time.

Reservoir Resource: (also called consumable / producible resource): Model the resources as consume and/or produce an activity

- When a resource consumes an activity, its resource decreases (taking up resources)
- When a resource produces an activity, the capacity increases (freeing up resources)
- Cumulative resource is a special case of reservoir
- Example of reservoir resource is a gas station. Cars come in, take gas, reservoir goes down. Gas truck eventually comes and re-fills the reservoir.

Calendars: (also called breakable activities): resources governed by calendar, and the calendar consists of breaks (e.g., Spring Break at UNL)

State Resources: resources have infinite capacity with varying state over time (need to know what state you are in to know what can be scheduled)

Examples

Below we cover multiple examples: timetabling, and machine scheduling (disjunctive and cumulative).

Timetabling

Create a schedule of N periods for classes, where classes have a given duration, lecturer, number of enrolled students, and prohibited time periods (e.g., Do not schedule grad students during colloquiums). There are M classrooms with specified seat capacities, and some classes creating a curriculum (cannot schedule these classes at the same time, for example, a course and its lab might make up a curriculum).

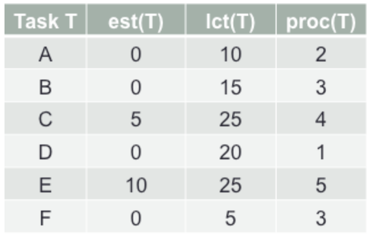
How the problem is modeled:
Class **A** is an activity with a given duration (e.g., our class is Constraint Processing Class). Initially a class can occur at any time: start(**A**) = {0,1,..., N-1}.

- Constraint specifying start(**A**) $\neq$ prohibited(A)
- Classrooms are resources. Order the classrooms by seat capacities.
- Alternative resource constraints for the classes: altern(**A**) = {k,..., M-1} (where k is the smallest classroom where the class size fits).
- Teaching represents a disjunctive resource: lecturer can only teach one class at a time (no overlap), all classes of each lecturer are constrained by unary resource constraints.
- Curriculum represents a unary resource, where classes of one curriculum define one unary resource constraint.
- At most one course can be taught at any classroom at each time slot (unary resource)

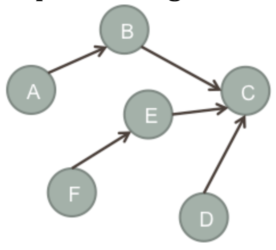
Machine scheduling with disjunctive scheduling

Given a set of tasks with [r,d] and proc, a set of precedence constraint from graph, and one machine of unary capacity, want to create a schedule while minimizing the makespan.

Critical Path: sequence of tasks that takes the longest time (makespan cannot be below this value). In Figure 11, the critical path is F→E→C, processing time = 19.



Task T	est(T)	lct(T)	proc(T)
A	0	10	2
B	0	15	3
C	5	25	4
D	0	20	1
E	10	25	5
F	0	5	3



$$\begin{tikzcd}
 & B \arrow[rd] & \\
A \arrow[ru] & E \arrow[r] & C \\
F \arrow[ru] & & D \arrow[u]
\end{tikzcd}$$

Figure 11: An example illustrating machine scheduling.

We can make a Gantt Chart, which shows the start/stop time of the schedule:

F	A	D	B		E	C

0 3 5 6 9 10 15 19

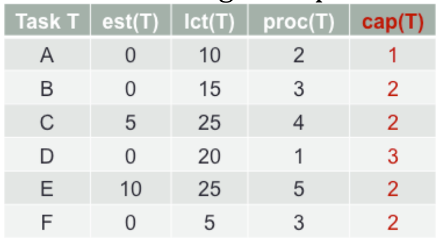
Variables: Start time start(**T**) for each task. start(**T**) = {r(**T**), ..., d(**T**)-proc(**T**)}
Domains: A: [0,8]; B: [0,12]; C: [5,21]; D: [0,19]; E: [10,20]; F: [0,2]
Constraints:

- Precedence constraints: For all **A** $\ll$ **B**, start(**A**) + proc(**A**) ≤ **B.**
- Unary constraints: For all tasks need to satisfy the start(**A**) and proc(**A**).
- Because we have one resource, we want to create a serial schedule, add a global serialize constraint.
- Minimize makespan, so add **L** after **C. C** + 4 ≤ **L**; minimize(makespan) = minimize(**L**).

Machine scheduling with cumulative scheduling

Problem is the same as the disjunctive scheduling, except we have a capacity 3 for our resource (run up to 3 activities at the same time).

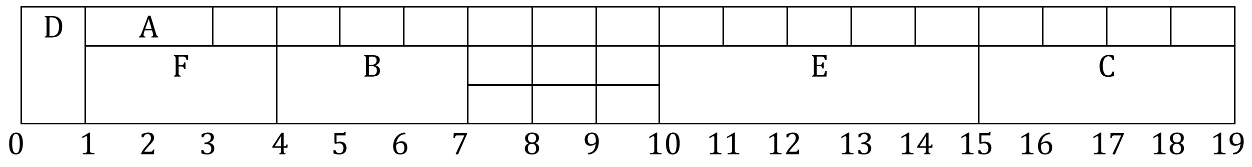
- Capacity for the resource is the capacity that the resource can take
- Capacity for an activity is really like the load on the resource that it uses. (For this example, need to have the whole capacity available.)
- Figure 12 updates the machine learning example to use capacities.



Task T	est(T)	lct(T)	proc(T)	cap(T)
A	0	10	2	1
B	0	15	3	2
C	5	25	4	2
D	0	20	1	3
E	10	25	5	2
F	0	5	3	2

Figure 12: Expanding the machine scheduling example with a capacity.

In addition to the unary constraints from before (enforces start time, etc.), add a new cumulative constraint: cumulative({Activities}, {proc()}, {cap()}, MachineCapacity) = cumulative([A,B,C,D,E,F], [2,3,4,1,5,3], [1,2,2,3,2,2],3), which enforces the load at a given time.



Disjunctive Propagation

First, we discuss disjunctive constraint propagation, edge finding, then “not-first” and “not-last” rules, focusing on disjunctive scheduling constraints (resources cannot overlap), and unary resources (only one resource).

Disjunctive constraint propagation

Two activities A_i and A_j requiring the same unary resource cannot overlap in time. Therefore either $A_i \ll A_j$ or $A_j \ll A_i$. Therefore, we want to maintain arc-B-Consistency on the formula: $[\mathrm{end}(A_i) \leq \mathrm{start}(A_j)] \vee [\mathrm{end}(A_j) \leq \mathrm{start}(A_i)]$.

Whenever $\mathrm{eet}(A_i) \geq \mathrm{lst}(A_j)$, A_i cannot precede A_j, therefore $A_j \ll A_i$.
Whenever $\mathrm{eet}(A_j) \geq \mathrm{lst}(A_i)$, A_j cannot precede A_i, therefore $A_i \ll A_j$.

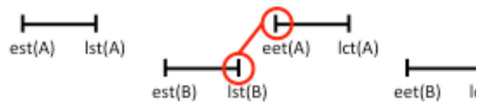


Figure 13: eet(A)≥ lst(B), therefore B≪

Edge Finding

In a given set Ω, decide that some activities must, can, or cannot execute first (or last) in Ω, which leads to new ordering relations (edges in the graph representing possible orderings). Different variants exist, but the one studied here is:

$$\forall_\Omega\, \forall_{A_i \notin \Omega}\, [d_{\Omega \cup \{A_i\}} - r_\Omega < p_\Omega + p_i] \Rightarrow [A_i \ll \Omega] \quad 1$$

$$\forall_\Omega\, \forall_{A_i \notin \Omega}\, [d_\Omega - r_{\Omega \cup \{A_i\}} < p_\Omega + p_i] \Rightarrow [A_i \gg \Omega] \quad 2$$

$$\forall_\Omega\, \forall_{A_i \notin \Omega}\, [A_i \ll \Omega] \Rightarrow [\mathit{end}(A_i) \leq \min_{\emptyset \neq \Omega' \subseteq \Omega} (d_{\Omega'} - p_{\Omega'})] \quad 3$$

$$\forall_\Omega\, \forall_{A_i \notin \Omega}\, [A_i \gg \Omega] \Rightarrow [\mathit{start}(A_i) \geq \max_{\emptyset \neq \Omega' \subseteq \Omega} (r_{\Omega'} + p_{\Omega'})] \quad 4$$

Where r_Ω, d_Ω, and p_Ω denote the smallest of the earliest start times, the largest of the latest ends times, and the sum of the minimal processing times of the activities in Ω, respectively.

Consider the example shown in Figure 14, where Ω = {B}. By equation 1, when A_i=A, $d_{\Omega \cup \{A\}}$=20, r_Ω=7, p_Ω=6, p_A=9, then 20-7 < 6+9 ⇒ 13 < 15 (*typo on slide 38*), which means that A must come first. Filtering end(A) with equation 3, yields that the ending time of A must be 20-6=14 (*typo on slide 38*). The bounds of B will stay the same, however, they will be updated after arc-B-consistency.

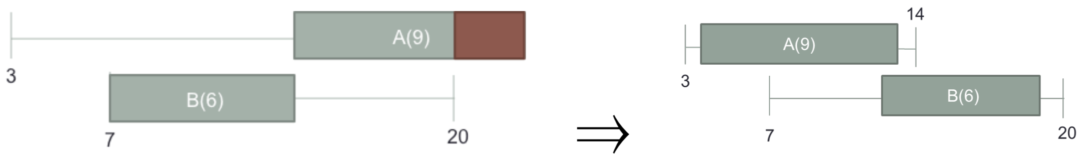


Figure 14: Example illustrating edge finding, where the bounds of A can be adjusted.

"Not-First" and "not-last" rules

Edge finding can only tell us if a must, can, or cannot execute first. However, we might run into a situation where we want to determine an activity cannot be first and cannot be last. The application is similar to edge-finding, but considering a different set of rules:

$$\forall_\Omega \ \forall_{A_i \notin \Omega} \ [d_{A_i} - r_\Omega < p_\Omega + p_i] \Rightarrow [end(A_i) \leq \max_{B \in \Omega} lst_B] 1$$

$$\forall_\Omega \ \forall_{A_i \notin \Omega} \ [d_\Omega - r_{A_i} < p_\Omega + p_i] \Rightarrow [start(A_i) \geq \min_{B \in \Omega} eet_B] 2$$

Consider the example shown in Figure 14, where Ω = {B,C}. By equation 1, when A_i=A, d_A=20, p_A=8, r_Ω=7, p_Ω=10, then 20-7=13<10+8=18, which means A must be before the latest start time of the any activity in Ω. Therefore, the new ending time of A is 15 (*typo on Slide 41*).

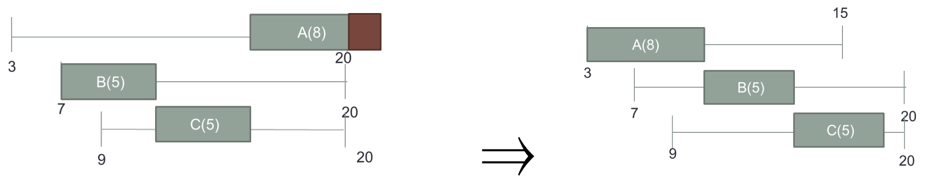


Figure 15: Example illustrating the "not-first" and "not-last" rule.

Conjunctive Propagation

First, we talk about timetable constraints, disjunctive constraints, then energy reasoning. *Arc-B-consistency* is enforcing arc-consistency on the bounds of an interval (will not break up the interval).

Timetable Constraints

Timetable is used to maintain information about resource utilization and resource availability over time. Apply arc-B-consistency on the formula:

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

Disjunctive Constraint

A_i and A_j are two activities such that $c_i + c_j > cap(R)$, and therefore they cannot overlap in time and either $A_i \ll A_j$, or $A_j \ll A_i$. Apply arc-B-consistency on the formula:

$$[cap(A_i) + cap(A_j) \leq cap(R)] \vee [end(A_i) \leq start(A_j)] \vee [end(A_j) \leq start(A_i)]$$

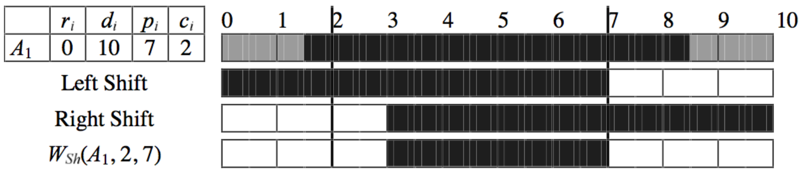
Energy Reasoning

The goal in *energy reasoning* is to determine how much capacity the resources will require at a minimum to process a given time interval, $[t_1,t_2)$.

The Left-Shift/Right-Shift, for an activity A_i and time interval $[t_1,t_2)$, denoted $W_{Sh}(A_i,t_1,t_2)$, is the c_i times minimum of the three durations:

1. $t_2 - t_1$
2. $p_i^+(t_1) = \max(0, p_i - \max(0,t_1-r_i))$: the number of time units during which A_i executes after time t_1 if A_i is left-shifted (i.e., scheduled as soon as possible).
3. $p_i^-(t_2) = \max(0, p_i - \max(0, d_i-t_2))$: the number of time units during which A_i executes before time t_2 if A_i is right shifted (i.e., scheduled as late as possible).

Consider the example shown in Figure 16, for a task A_1 find the energy consumption over $[2,7)$. Computing $W_{Sh}(A_1,2,7) = 2\cdot \min(5,5,4)=8$, a minimum of 8 energy units must be consumed by A_1 in the time interval $[2,7)$



	r_i	d_i	p_i	c_i
A_1	0	10	7	2

Figure 16: Computing the energy consumption of a task A_1 over $[2,7)$.

The Left-Shift/Right-Shift for a time interval $[t_1,t_2)$, denoted $W(t_1,t_2)$, is the sum over all the activities A_i, $W_{Sh}(A_i,t_1,t_2)$.

Conjunctive Reasoning between Temporal and Resource Constraints

Precedence graph: Temporal network representing the relations between the time points of all activities (start and end times) using Point Algebra is maintained during search.

For example, $end(A_i) \leq start(A_j)$ adds to the precedence graph the relation $e_i \leq s_j$.

During search, additional precedence relations can be added as decisions or as the result of constraint propagation.

Energy precedence constraint: For a given activity A_i, ensures that every subset of predecessor activities of A_i, ϕ, the resource provides enough energy to execute all the activities in ϕ between r_ϕ and s_i.

Solving using Optimization

Criterion (or multiple criteria)

Regular (e.g., makespan): If the criteria increases with the end time of the activities, then if solution S_1 is strictly better than S_2, replace value of criterion obtained by replacing each end time variable by its lower bound.

Sequence-dependent: Depends only on the relative order in which activities are executed. Want to solve resource constraints by ordering activities, once activities are sequenced, the earliest start and end times that result from constraint propagation can be used in a solution.
Other: If the optimization criteria are more difficult to optimize. It is often the case that once the resource constraints have been solved by sequencing activities, a linear program can be used to determine optimal solution. Therefore, use hybrid algorithms based on both CP and Mixed Integer Programming (MIP).

Local Search

Two types of local moves, repair and shuffle:

1. **repair**: swap two activities to shrink / reduce number of critical paths.
2. **shuffle**: move part of the solution and search through the rest of the solution space to complete it.

Mixed Integer Programming:

Hybrid combination of CP and MIP, use CP techniques to limit and select the explored branches, and MIP for solving the problem.