TD Optimisation – session 5 – Task graph scheduling

3 novembre 2025

Learning objective : study classical heuristics and anomalies for the scheduling problem with task precedence $P|pred|C_{\max}$.

The first 5 exercises are essential.

Before starting, we recall some concepts from the lecture :

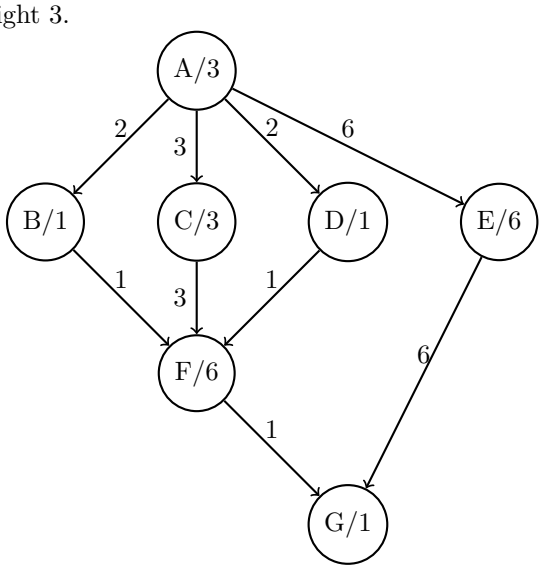
— The *bottom level* of a task is the longest length from this task to the sink, including its own cost.

— The *critical path scheduling* algorithm first sorts the tasks by their bottom-level in non-increasing order. Each task becomes available as soon as their predecessors are completed. At each time step, the algorithm schedules any available task with highest bottom-level on the first available processor if any (i.e., neither computing, nor transmitting data). If data are transferred, the selected processor is marked as busy until the task is completed.

— Contention is ignored in the communication model : the cost to transfer data is independent of what occurs on the network.

1 Task graph scheduling heuristics

Consider the following DAG where a pair X/w means that task X has weight w. For instance, A has weight 3.



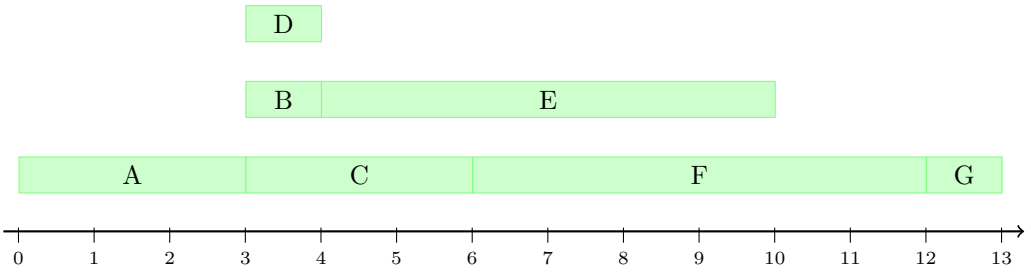
Exercice 1 : Scheduling without communications

We first disregard the labelling of the edges and assume communications come for free.

Compute the bottom level for each node.

A : 13, B : 8, C : 10, D : 8, E : 7, F : 7, G : 1.

Schedule the task graph on 3 processors using a list heuristic. What is the makespan of our schedule ? Is it optimal ?



It schedules A, C, F and G on processor 1, B and E on processor 2 and D on processor 3, leading to a makespan of 13. This is optimal because the bottom level is a lower bound on the makespan. It is therefore not possible to obtain a better makespan.

Exercice 2 : Critical path scheduling

From now on, we will consider the communication costs which have to be accounted for when two adjacent tasks are scheduled on different processors.

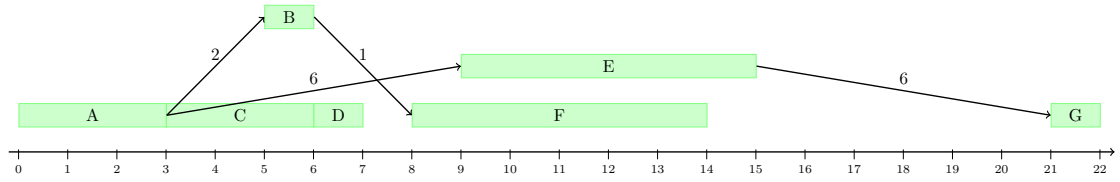
How communications should be taken into consideration when computing the bottom level ?

As we do not known the allocation yet, we must assume the worst-case : communication always occur between each dependent tasks. This amounts to assume one distinct processor per task.

Compute the bottom level for each node.

A : 22, B : 10, C : 15, D : 10, E : 14, F : 8, G : 1.

Schedule the task graph on 3 processors using the critical path heuristic. What is the makespan of our schedule ?



It schedules A, C, D, F and G on processor 1, E on processor 2 and B on processor 3, leading to a makespan of 22. It would have been much better to put the final G on processor 2 but the algorithm selects the first free processor.

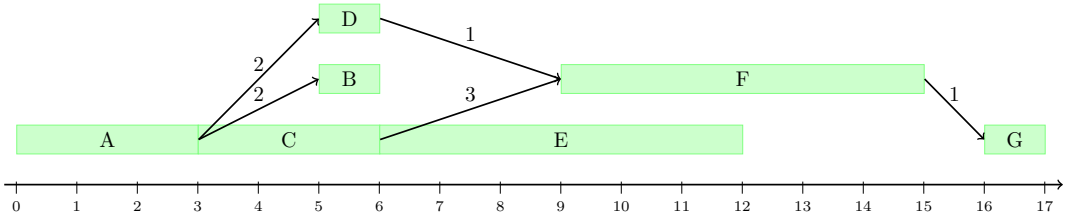
Exercice 3 : Modified critical path scheduling

Sometimes, it is worth waiting to schedule a task on a busy processor rather then using the first processor available. This means that we would not schedule tasks on the first available processor, but we would have waiting queue for each processor and we would add the tasks to theses queues.

Which wrong decision, made in the previous section, would be avoided by using this new heuristic ?

Putting task E on another processor incurs a significant communication overhead. Instead of that, we could wait for task C to finish to save the communication time.

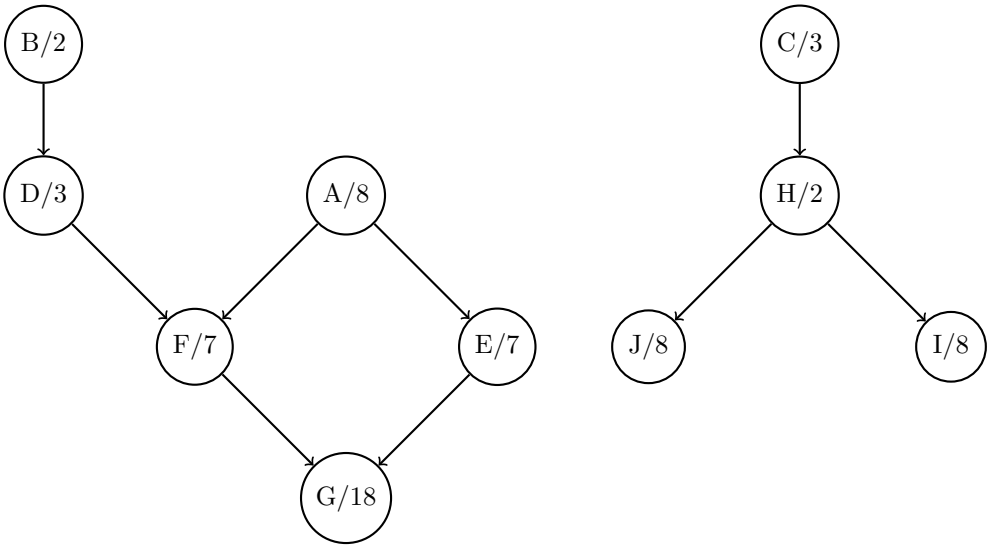
Using this approach, propose a new schedule for our task graph with 3 processors. What is its makespan ?



The idea is to avoid choosing the first free processor and instead select the processor that will complete the task first (Earliest Finish Time). Therefore, instead of making a scheduling decision at each time step, we consider a queue of tasks on each processor and add each newly available task to the queue of the processor that minimizes its completion time. It schedules A, C, E and G on processor 1, B and F on processor 2 and D on processor 3, leading to a makespan of 17.

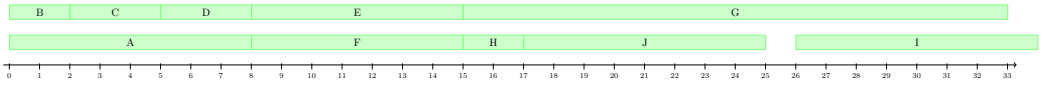
2 List scheduling anomalies

Consider the following DAG with two components.



Exercice 4 : No anomalies

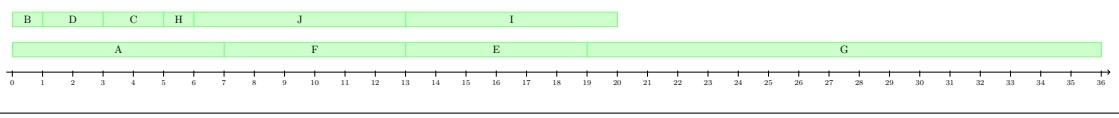
What is the makespan achieved by the critical path list scheduling with 2 processors ? Is it optimal ?



It is optimal because the makespan is the maximum bottom-level.

Exercice 5 : Anomalies on weights

Assume that each task weight is decreased by one unit (now A has weight 7, B has weight 1, and so on). Show that the makespan achieved by the critical path list scheduling increases. Show that, somewhat shockingly, it is impossible to get a lower makespan than before with a list scheduling algorithm.



It is never possible to start E and F in parallel because the other processor starts processing either I or J juste before.

Exercice 6 : Anomalies on processors

Going back to original task weights, assume that we have 3 processors. Show that the makespan achieved by the critical path list scheduling increases. Show that the makespan achieved by any list scheduling algorithm, shockingly again, increases.