PART IV: Constraint-Based Scheduling

Scheduling

- Ordering resource-requiring tasks over time.
- A very important (and tough) problem class.
- Many practical applications.

Some Scheduling Problems

- Project planning and scheduling.
  - Software project planning.
- Machine scheduling.
  - Allocation of jobs to computational resources.
- Scheduling of flexible assembly systems.
  - Car production.
- Employee scheduling.
  - Nurse rostering.
- Transport scheduling.
  - Gate assignment for flights.
- Sports scheduling.
  - Schedule for NHL, world cup, olympics.
- Educational timetabling.
  - Timetables at schools.

Resource Constrained Project Scheduling Problem (RCPSP)

- Given:
  - a set of resources with fixed capacities,
  - a set of tasks with given durations and resource requirements,
  - a set of temporal constraints between tasks,
  - and a performance metric,

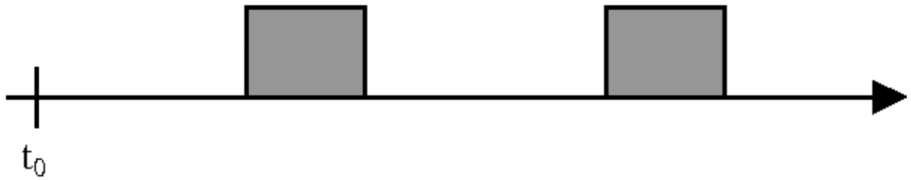
  RCPSP consists of deciding:
  - when to execute each task so as to optimize the performance metric, subject to temporal and resource constraints.

A Constraint-Based Model

- Tasks → (activity) variables.
- Resource constraints →
  - unary/disjunctive/sequential resource;
  - cumulative/parallel resource.
- Temporal constraints.
- Performance metric → schedule dependent cost function.

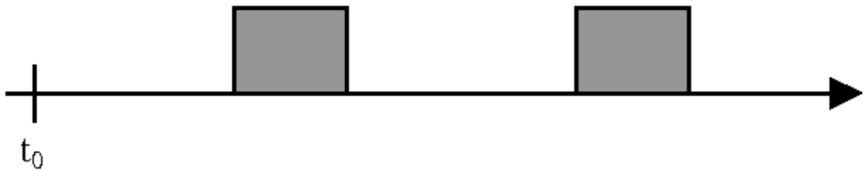
(Activity) Variables

- Correspond to the operations to be performed. E.g.,
  - processing an order,
  - executing a job,
  - working in a shift,
  - performing a loading operation.
- Need to decide the operation positions in the timeline of the schedule.



- Main variables of the scheduling problem.

(Activity) Variables



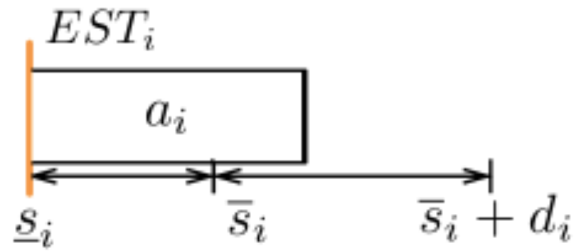
- Activity a_i
  - Start Time S_i
    - Starting time variable of an activity a_i, with domain $D(S_i)$
      - min(S_i) is the earliest start time (release date), also referred to as $EST_{i.}$
      - max(S_i) is the latest start time, also referred to as $LST_{i.}$
  - Duration d_i
    - usually assumed to be known.
  - End Time E_i
    - Ending time variable of an activity $a_{i,}$ with domain $D(E_i)$
      - max(E_i) is the latest end time (deadline), also referred to as $LET_{i.}$
      - min(E_i) is the earliest end time, also referred to as $EET_{i.}$

(Activity) Variables

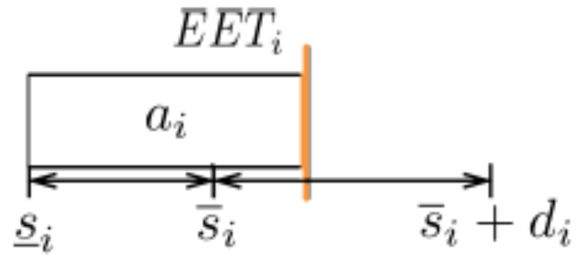
- Preemptive activity a_i
  - Can be interrupted at any time.
  - $S_i + d_i \leq E_i$
- Non-preemptive activity a_i
  - Cannot be interrupted at any time.
  - $S_i + d_i = E_i$
- We focus on non-preemptive activities.

Non-Preemptive Activity

Earliest Start Time: $\boldsymbol{EST_i = \underline{s}_i}$

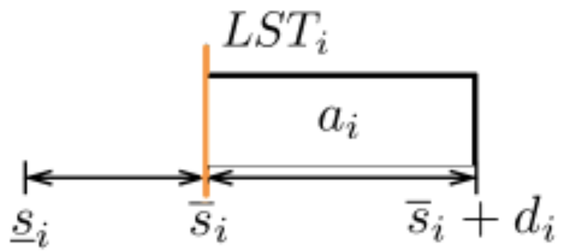


Earliest End Time: $\boldsymbol{EET_i = \underline{s}_i + d_i}$

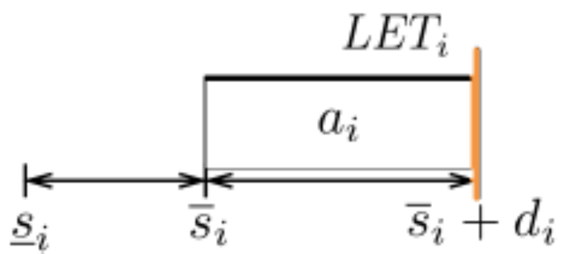


Non-Preemptive Activity

Latest Start Time: $LST_i = \bar{s}_i$



Latest End Time: $LET_i = \bar{s}_i + d_i$



Resources

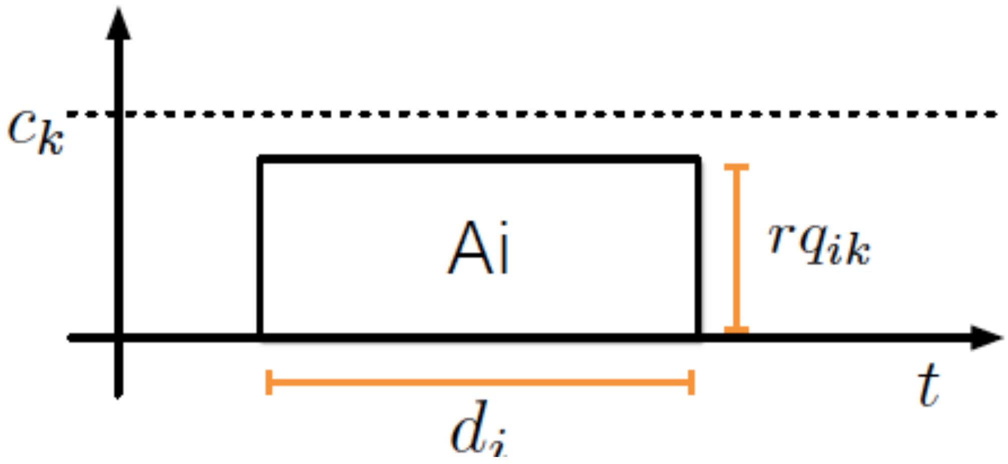
- A resource corresponds to an asset available to execute the operations. E.g.,
  - the capacity of a machine,
  - the volume of a truck,
  - the number of seats in a classroom,
  - number of available workers.

Cumulative/Parallel Resource

- Can execute multiple activities in parallel.
  - Activities can overlap in time!
  - E.g., a group of identically skilled workers, a delivery truck, a multi-core CPU.

Cumulative/Parallel Resource

- A resource r_k is associated to a capacity c_k.
- Each a_i requires some amount $rq_{ik} \geq 0$ of each resource r_k during its execution.
- During the execution of the schedule, the total usage of r_k by the activities a_i should not exceed c_k.



Cumulative/Parallel Resource

- Any two activity requiring the same resource is related by a cumulative constraint.
  - for all $r_k \in R$ with the capacity c_k:

cumulative([S_1, S_2, ..., S_n], [d_1, d_2, ..., d_n], [rq_{1k}, rq_{2k}, ..., rq_{nk}], c_k)

iff $\sum_{i | S_i \leq u < S_i + d_i} rq_{ik} \leq c_k$ for all u in D

- RCPSP resources are cumulative.

Another Cumulative Example

- You are moving house. You have 4 people to do the move and you must move in 1 hour. Piano must be moved before bed.

Item	Time	No. of people
piano	30 min	3
chair	10 min	1
bed	20 min	2
table	15 min	2

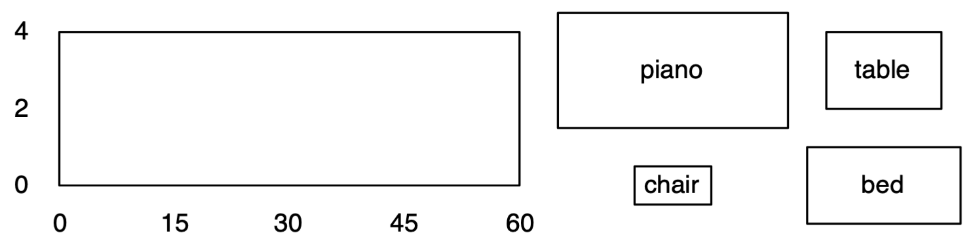
Another Cumulative Example

cumulative resource with capacity

- You are moving house. You have 4 people to do the move and you must move in 1 hour. Piano must be moved before bed.

timeline/horizon

Item	Time	No. of people
piano	30 min	3
chair	10 min	1
bed	20 min	2
table	15 min	2



activities durations resource requirements

Another Cumulative Example

cumulative resource with capacity

- You are moving house. You have 4 people to do the move and you must move in 1 hour. Piano must be moved before bed.

timeline/horizon

Item	Time	No. of people
piano	30 min	3
chair	10 min	1
bed	20 min	2
table	15 min	2

activities durations resource requirements

D(P) = D(C) = D(B) = D(T) = [0..60]
P + 30 ≤ 60, C + 10 ≤ 60,
B + 15 ≤ 60, T+ 15 ≤ 60,
P + 30 ≤ B,
cumulative([P,C,B,T], [30,10,20,15], [3,1,2,2], 4)

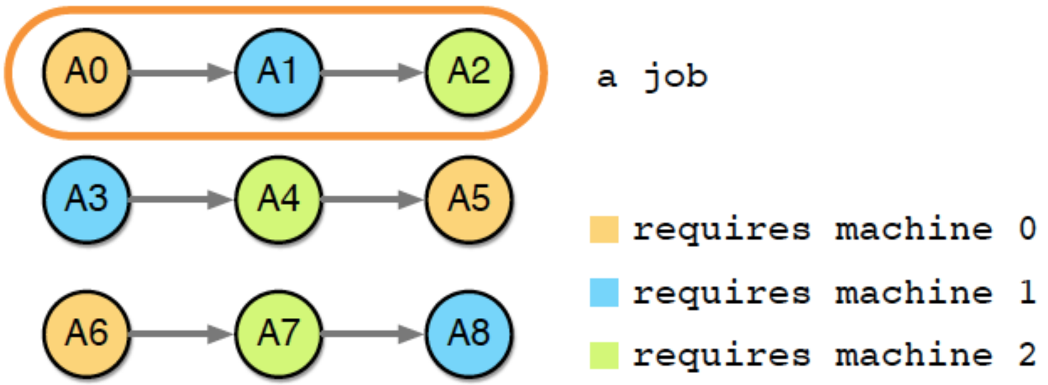
Unary/Disjunctive/Sequential Resource

- Can execute one activity at a time.
  - Activities cannot overlap in time independently of the resource capacity!
  - E.g., a classroom, a train track segment, a crane on a construction site.
- Any two activity is related by a disjunctive (noOverlap) constraint.
  - disjunctive($[S_1, S_2, ..., S_n], [d_1, d_2, ..., d_n]$) iff

    $S_i + d_i \leq S_j \lor S_j + d_j \leq S_i$ for all $1 \leq i < j \leq n$

A Disjunctive Example

- Job shop scheduling problem
  - A job is a sequence of activities (e.g., manufacture of an automobile).
  - Only disjunctive resources (machines).
  - Activities in a job require distinct machines.
  - There are as many activities as machines.

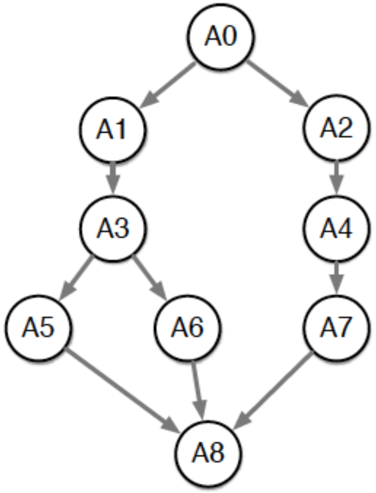


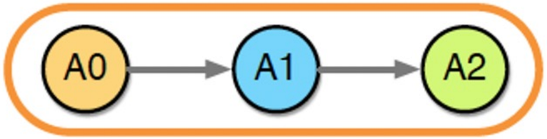
Temporal Constraints

- Precedence constraints
  - Forces one activity to end before another starts.
  - $a_i \rightarrow a_j$
    - $E_i \leq S_j$

Temporal Constraints

- House moving
  - Piano must be moved before bed
- RCPSP
  - Activities and precedence constraints form DAG, called Project Graph.
- Job shop scheduling problem
  - Tasks of a job are processed in a sequential order.





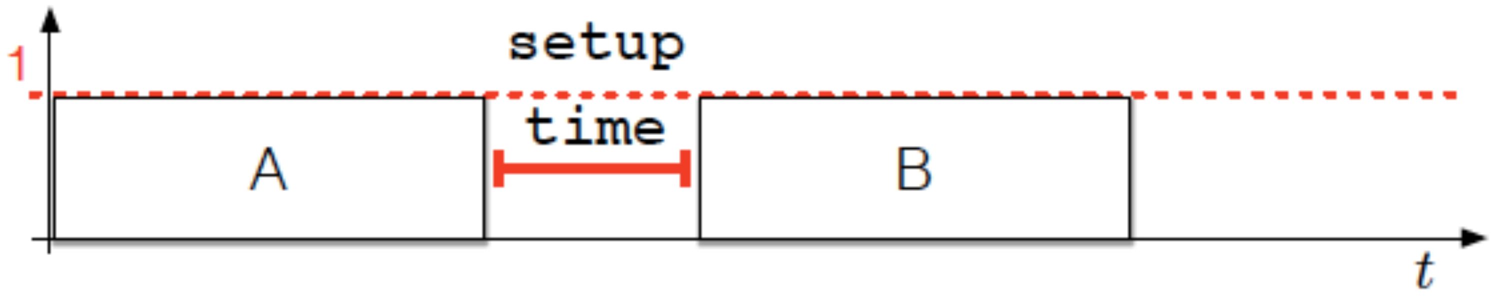
a job

Temporal Constraints

- Time-legs & Time windows
  - Bounds the difference between the end time and the start time of two activities.
  - $a_i \xrightarrow{[l_{ij},\, u_{ij}]} a_j$
    - $l_{ij} \leq S_j - E_i \leq u_{ij}$
  - Time windows are time legs from a dummy activity a_0 with $S_0 = 0$ and $d_0 = 0$.
    - $l_j \leq S_j \leq u_j$

Temporal Constraints

- Sequence-dependent set up times
  - Defined for unary resources.
  - If a_i and a_j are scheduled in a sequence, then they must obey a separation constraint.
    - $E_i \leq S_j \rightarrow E_i + d_{ij} \leq S_j$



Cost Function

- A common cost function: makespan
  - Completion time of the last activity.
  - Optimum makespan is the minimum makespan.
  - RCPSP and job shop scheduling cost functions are makespan.
- Minimum makespan can be modeled in different ways.
  - minimize max($[S_1+d_1,\dots, S_n+d_n]$)
  - Alternatively:
    - Introduce a dummy activity a_{n+1}, with $d_{n+1} = 0$ and constrain it to have the lowest precedence in the schedule:
      - $a_i \rightarrow a_{n+1}$ for all i
    - minimize S_{n+1}

Other Cost Functions

- (Weighted) Tardiness costs:

$$\sum_{a_i \in A} w_i \cdot \max(0, E_i - LET_i)$$

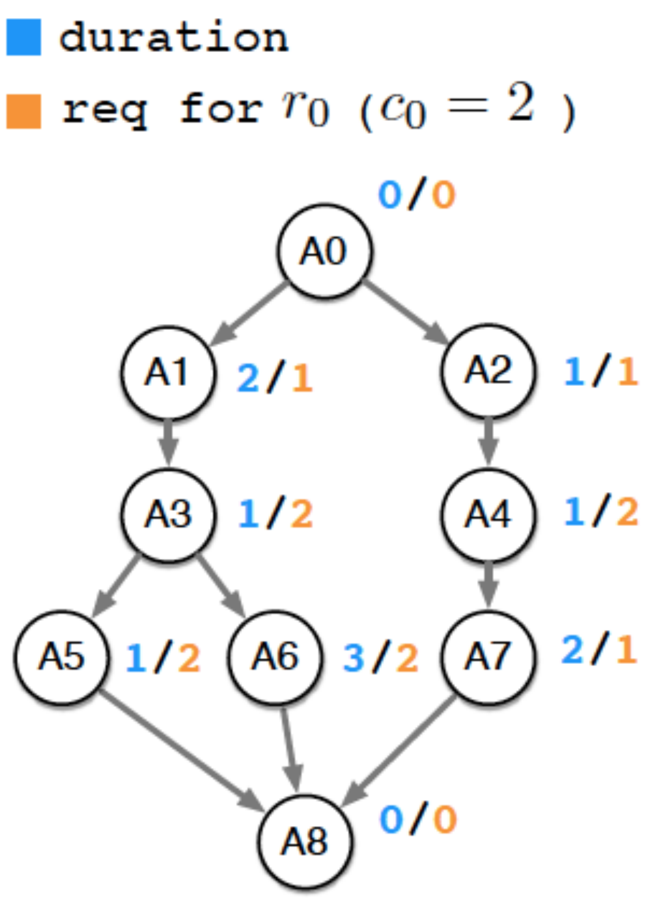
- (Weighted) Earliness costs:

$$\sum_{a_i \in A} w_i \cdot \max(0, LET_i - E_i)$$

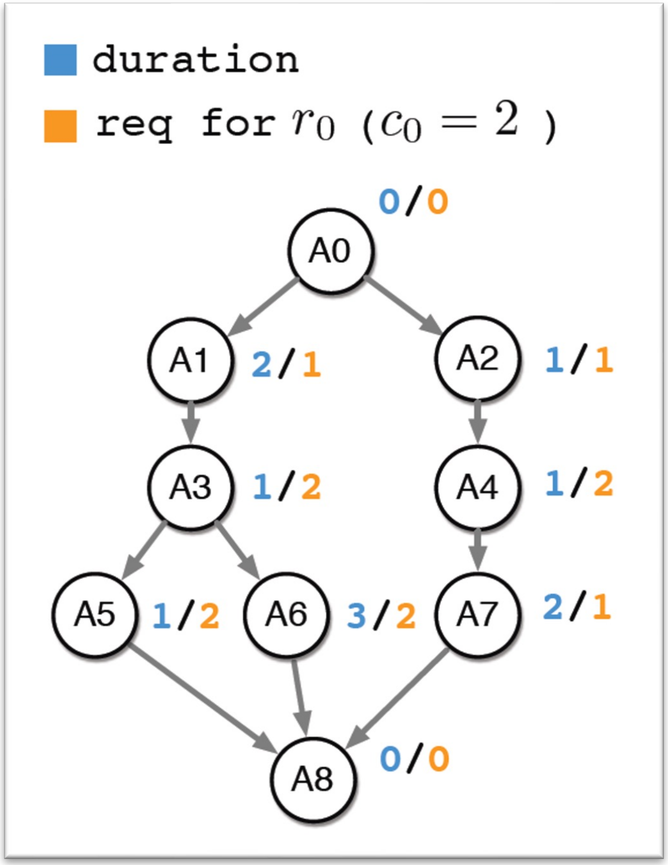
- The peak resource utilization.
- The sum of set up times and costs.

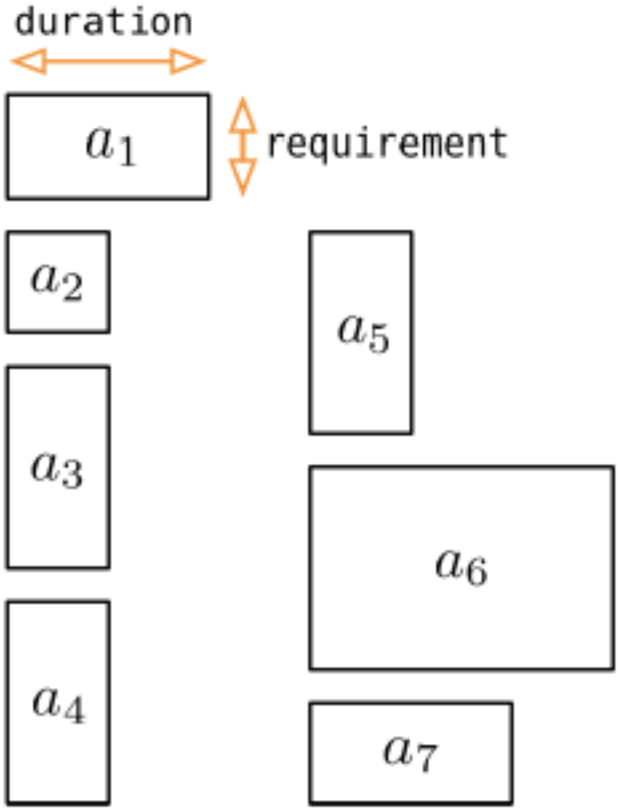
A Sample RCPSP

- A Project Graph $\langle A, E \rangle$
- A is the set of activities a_i
- E is a set of activity pairs (a_i, a_j), representing the precedence constraints
- Activity durations d_i
- Set R of resources r_k , with capacity c_k
- All resource requirements r_{ik}



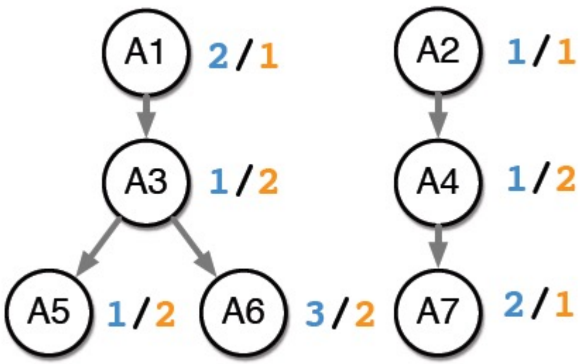
A Sample RCPSP



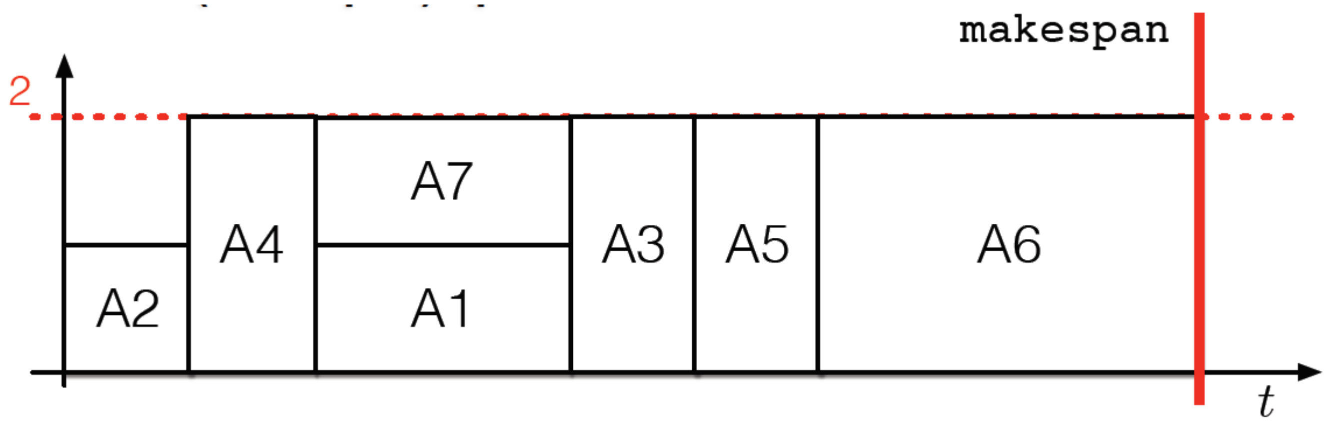


A Sample RCPSP

- The source and sink activities are fake and can be disregarded.



- Makespan optimal schedule:



Search Heuristics

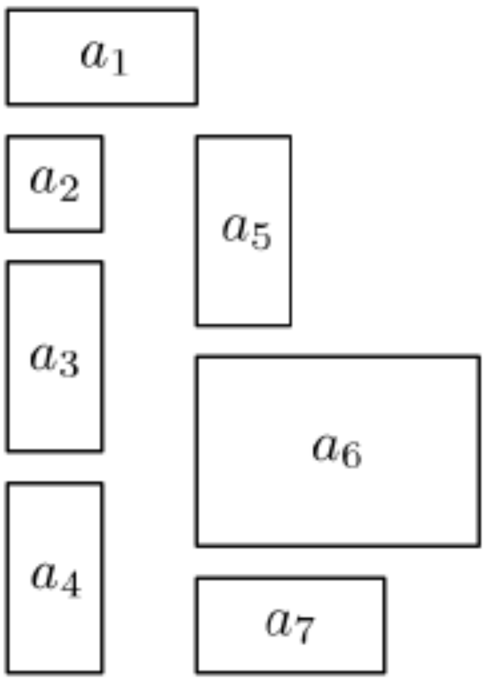
- Typical instances have large domains.
  - S_i and E_i domains are as large as the horizon.
- Which variable to pick next?
- Which value to assign?

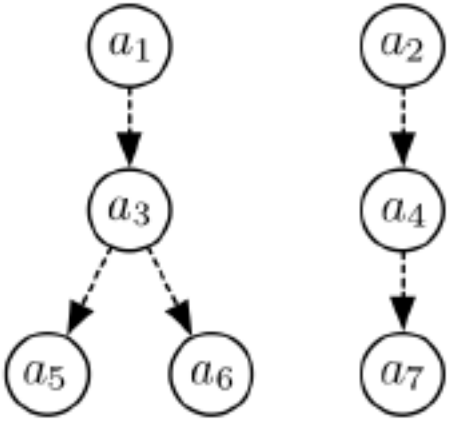
Value Selection for the RCPSP

- The objective is to minimize the makespan.
- Increasing an S_i value (with other S_j untouched) cannot improve the makespan.
  - → Select EST_i.
- This is true not only for the RCPSP.
  - Many scheduling problems have so-called regular cost metrics.
  - Regular = increasing a single S_i cannot improve the cost.

Variable Selection for the RCPSP

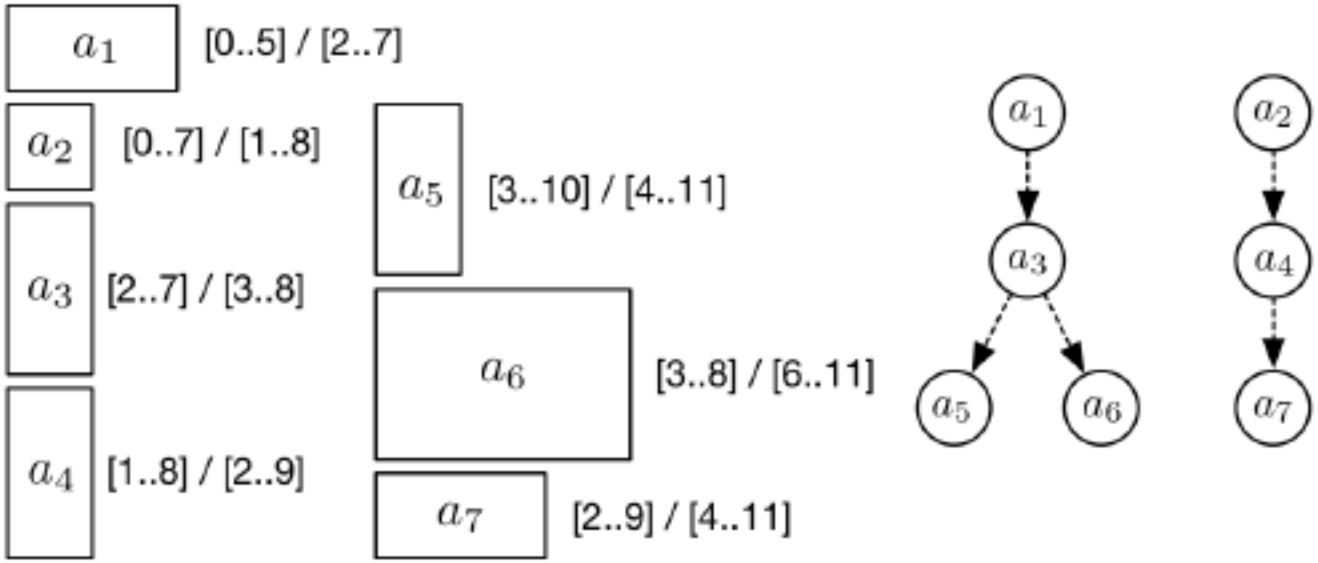
- Example





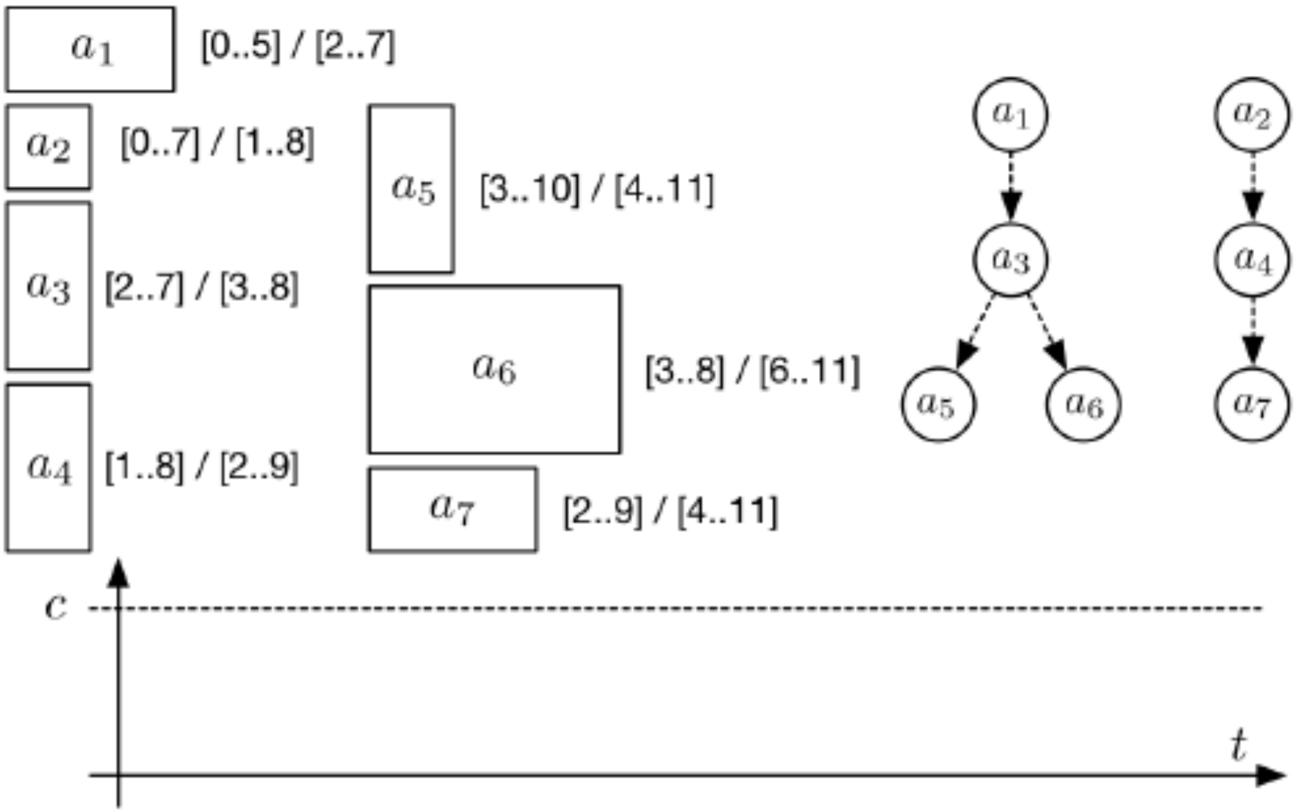
Variable Selection for the RCPSP

- Assume we have the following domains after propagating the precedence constraints:



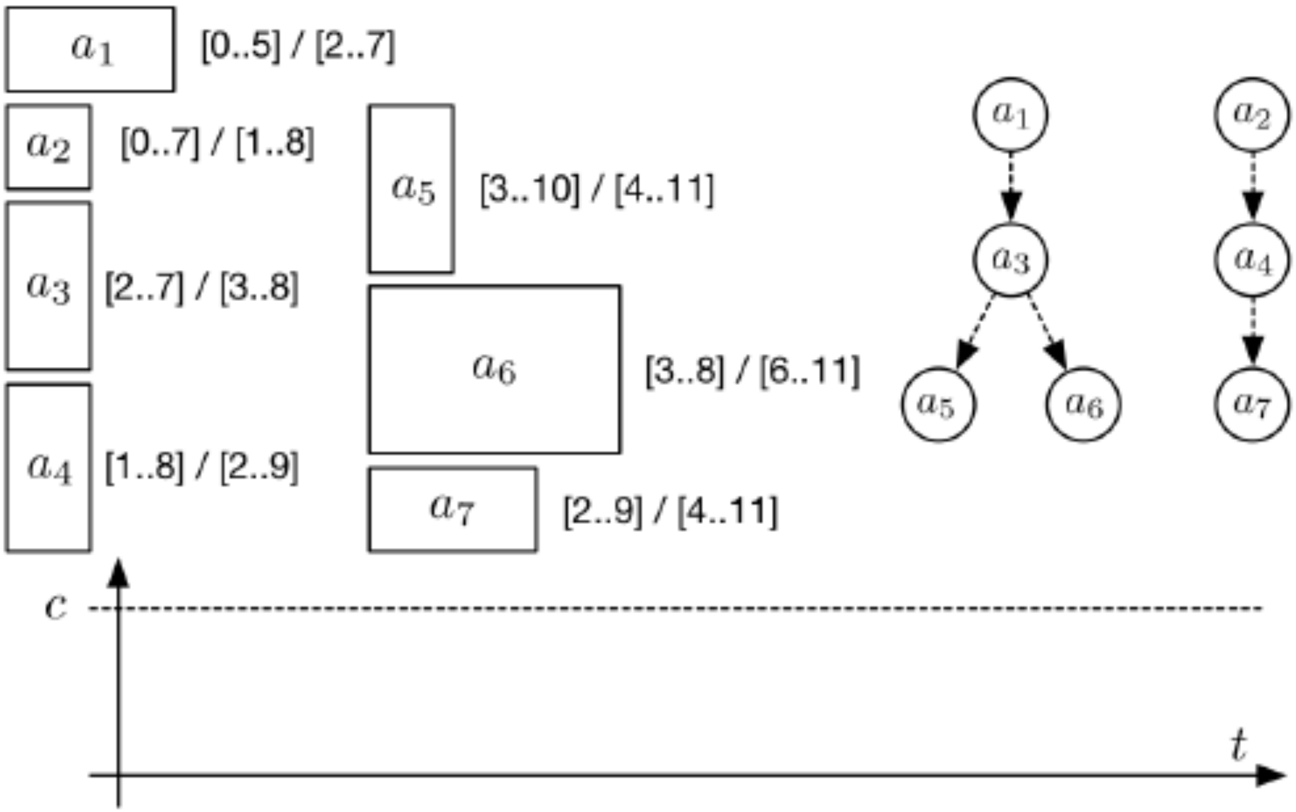
- Notation: $[EST_i..LST_i]/[EET_i..LET_i]$

Variable Selection for the RCPSP



- Which variable first?

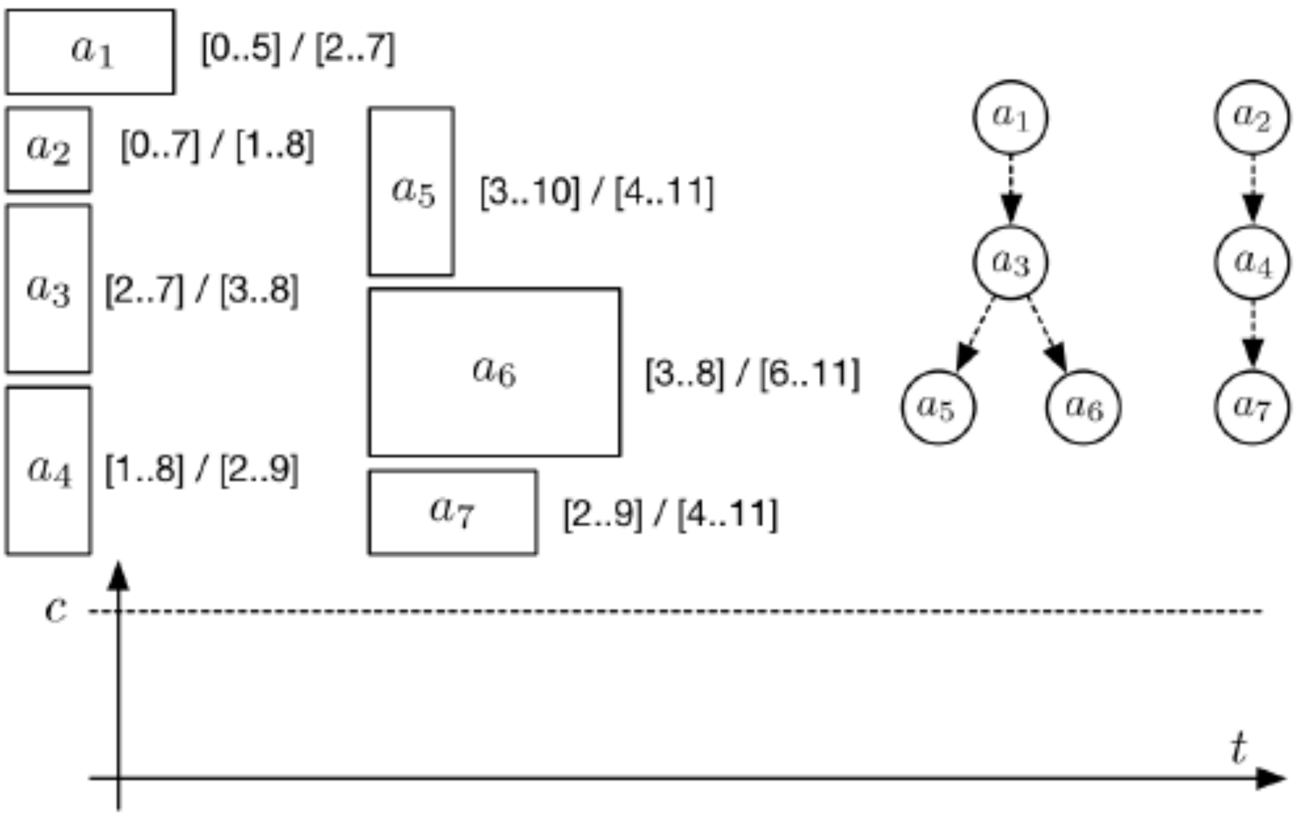
Variable Selection for the RCPSP



- A sensible criterion: minimum EST_i.

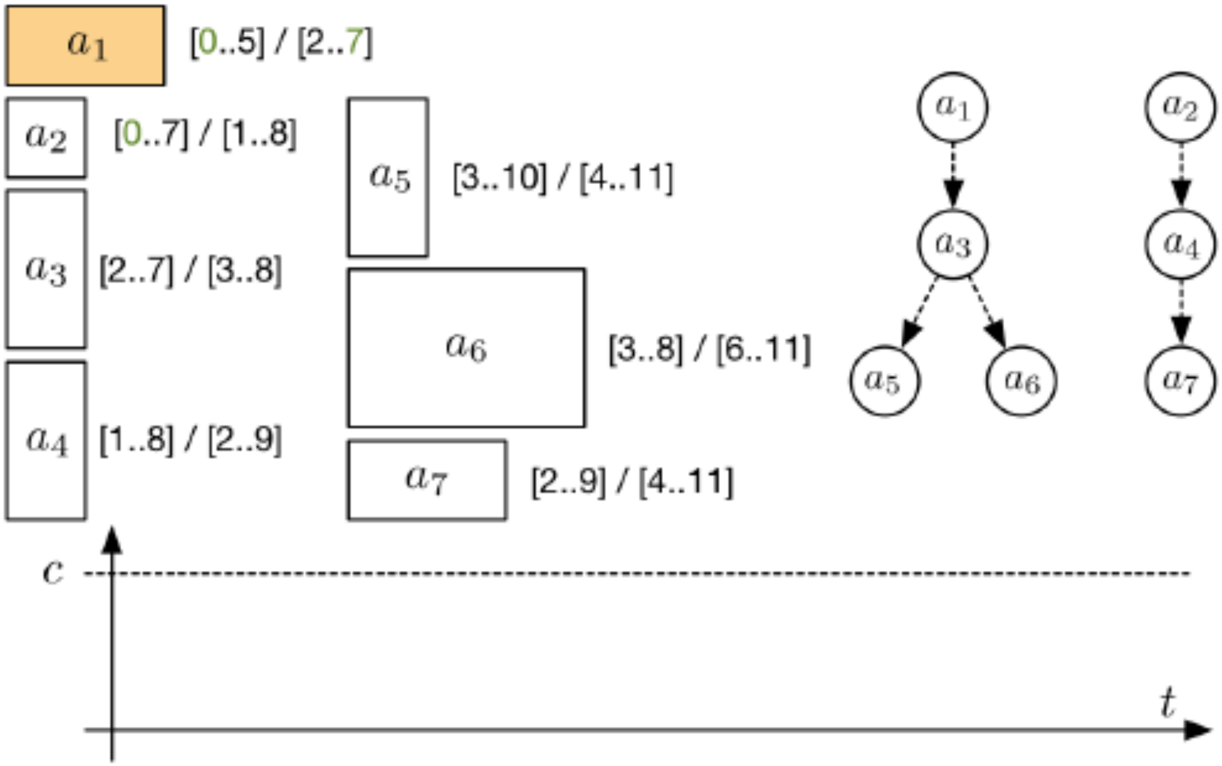
Variable Selection for the RCPSP

- How to break ties?

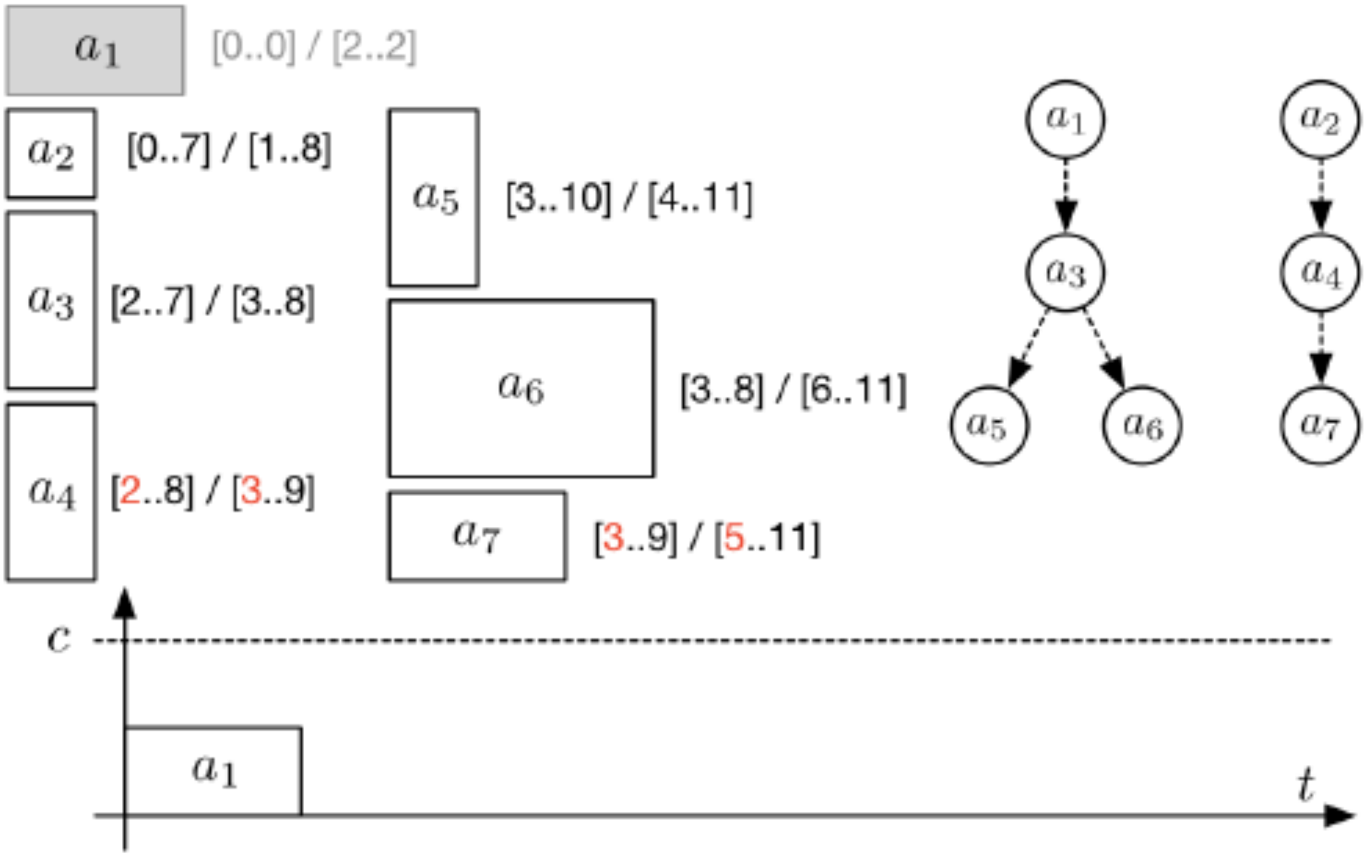


Variable Selection for the RCPSP

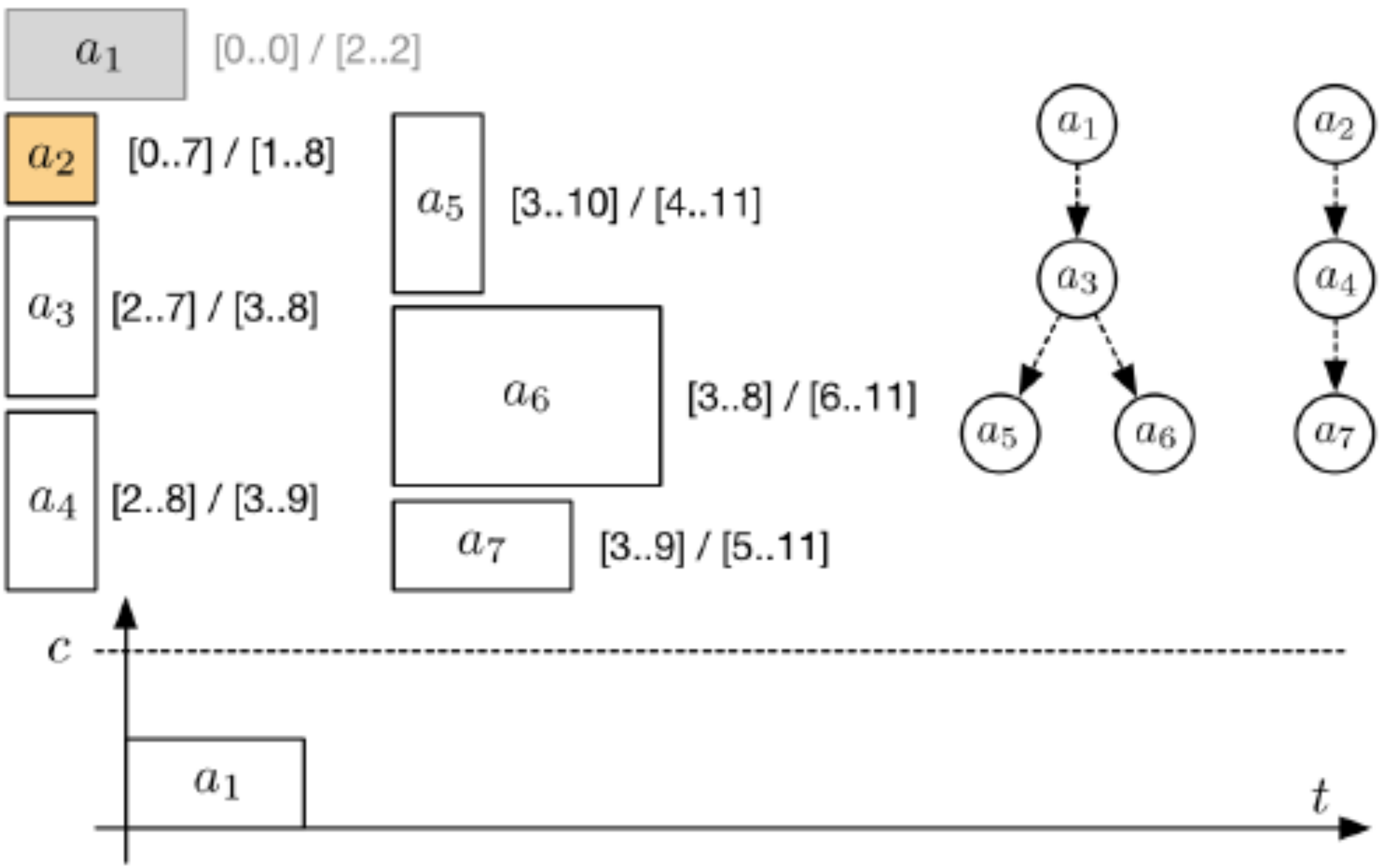
- How to break ties?
  - Tightest deadline, i.e. minimum LET_i.



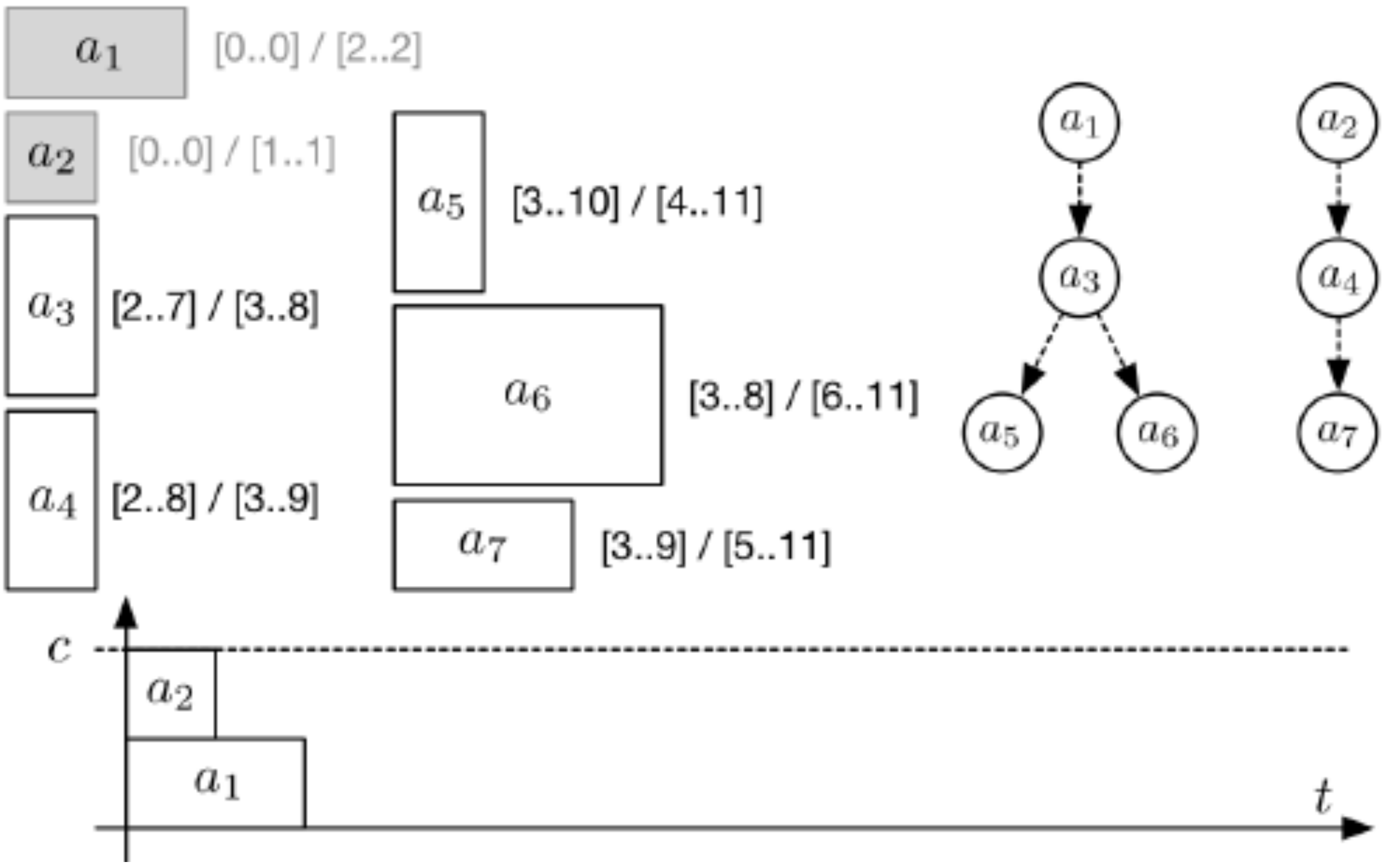
Variable Selection for the RCPSP



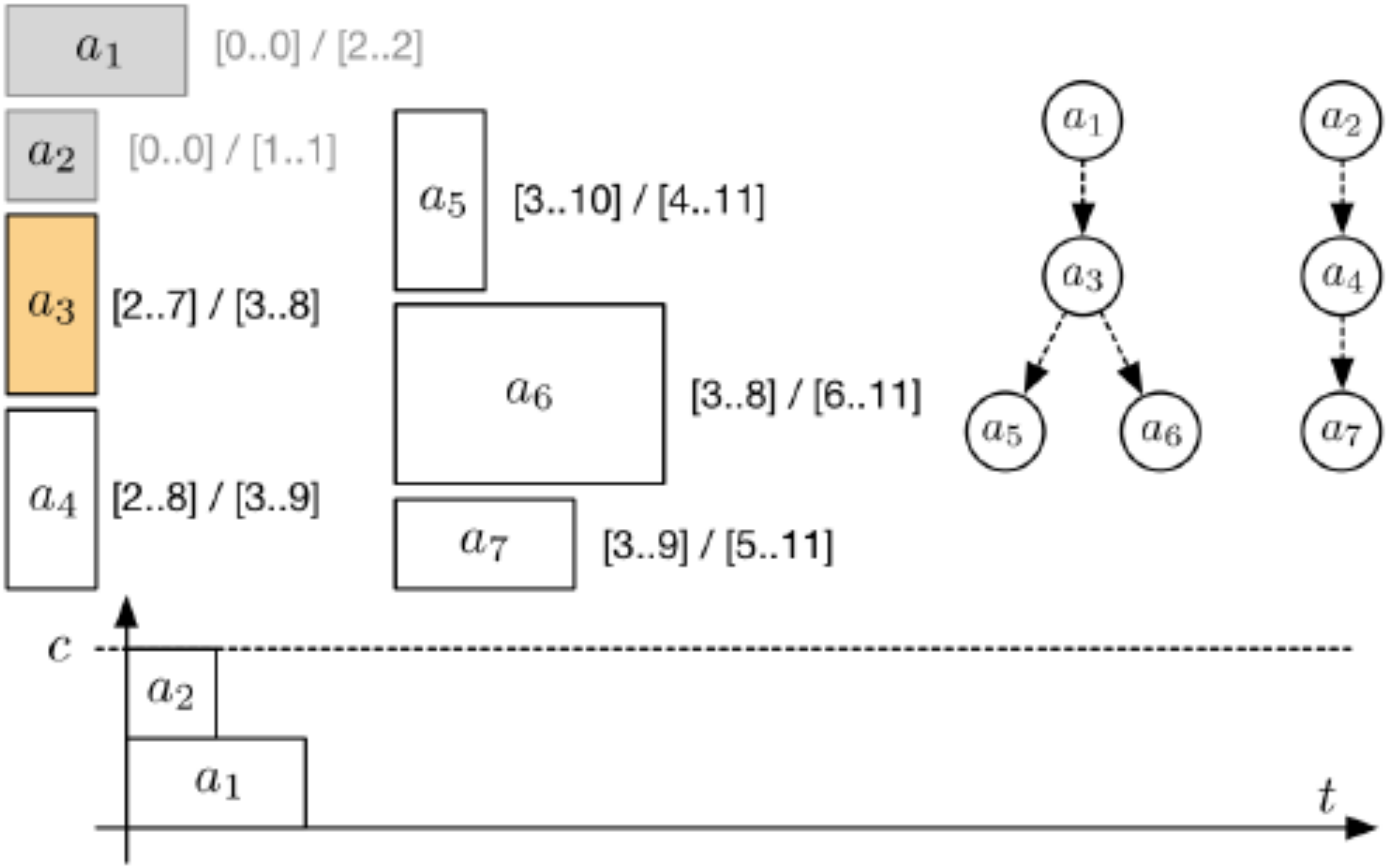
Variable Selection for the RCPSP



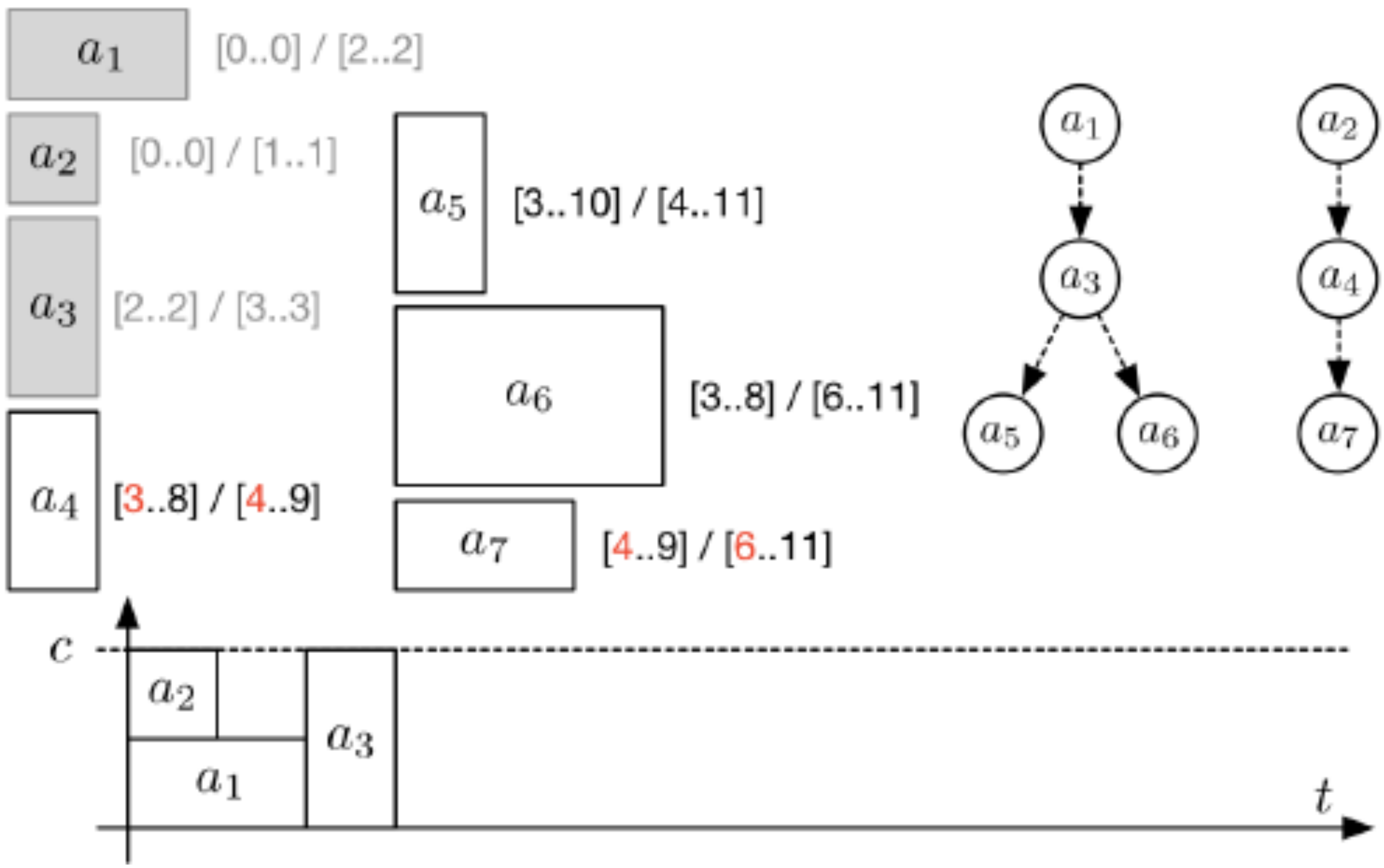
Variable Selection for the RCPSP



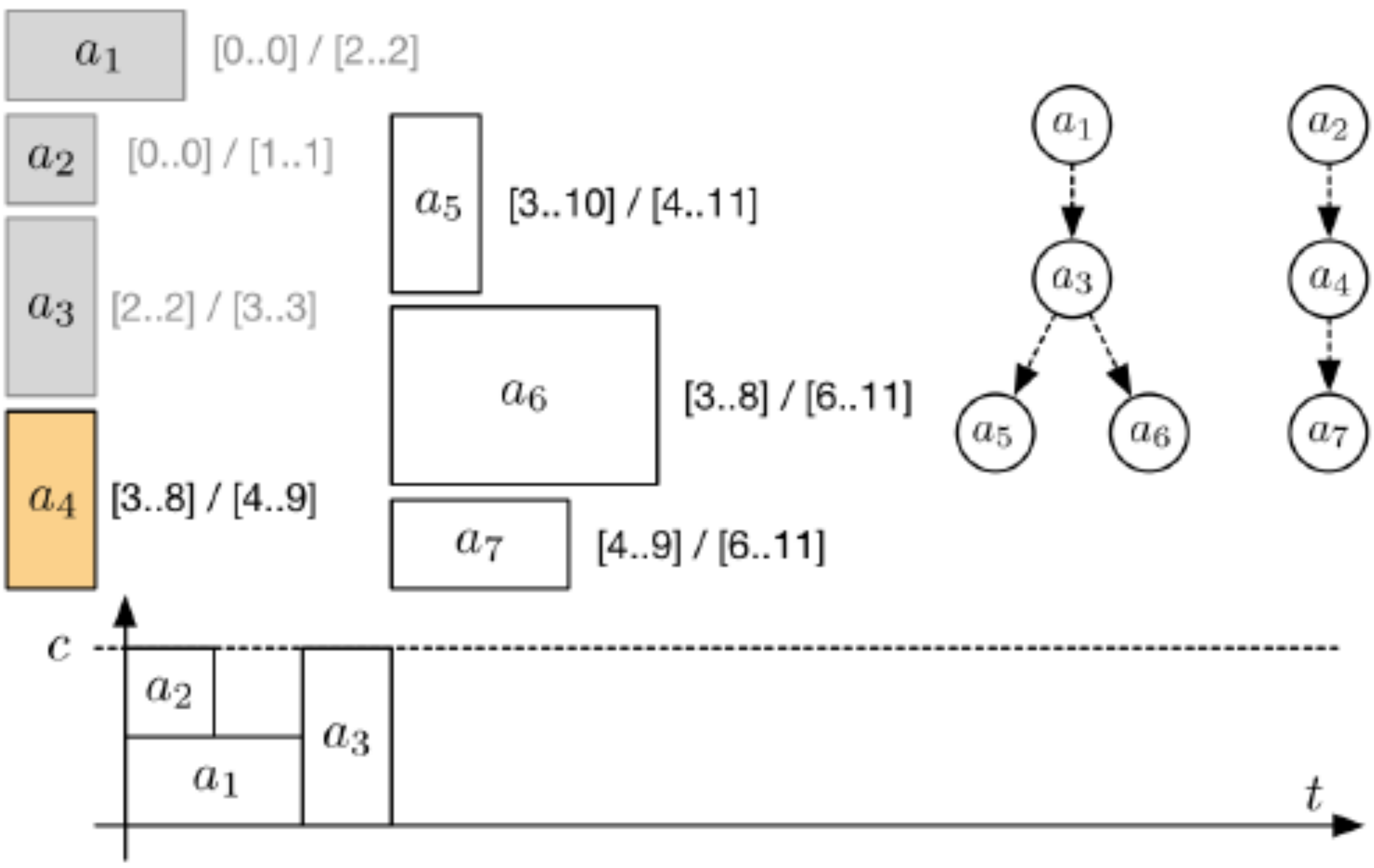
Variable Selection for the RCPSP



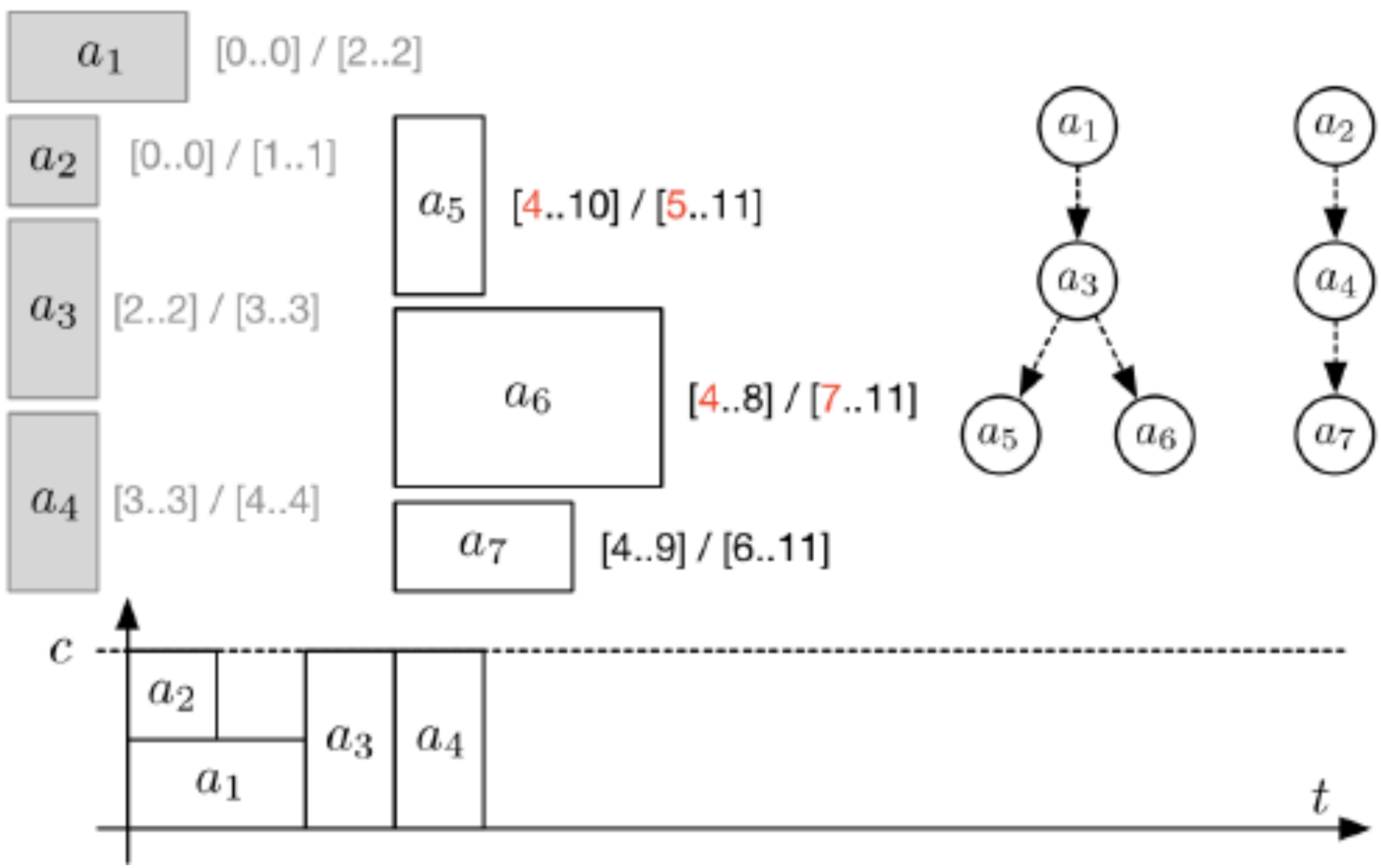
Variable Selection for the RCPSP



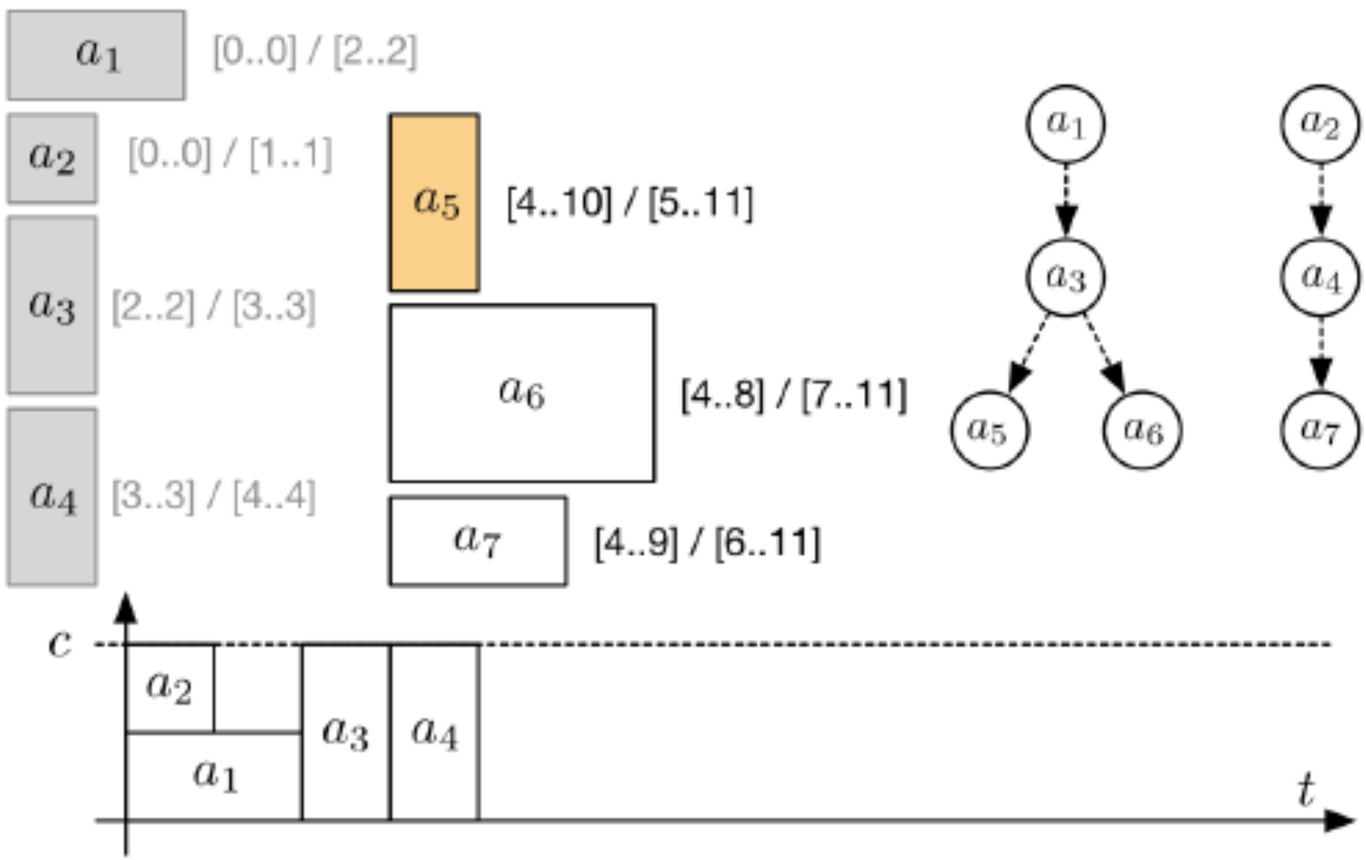
Variable Selection for the RCPSP



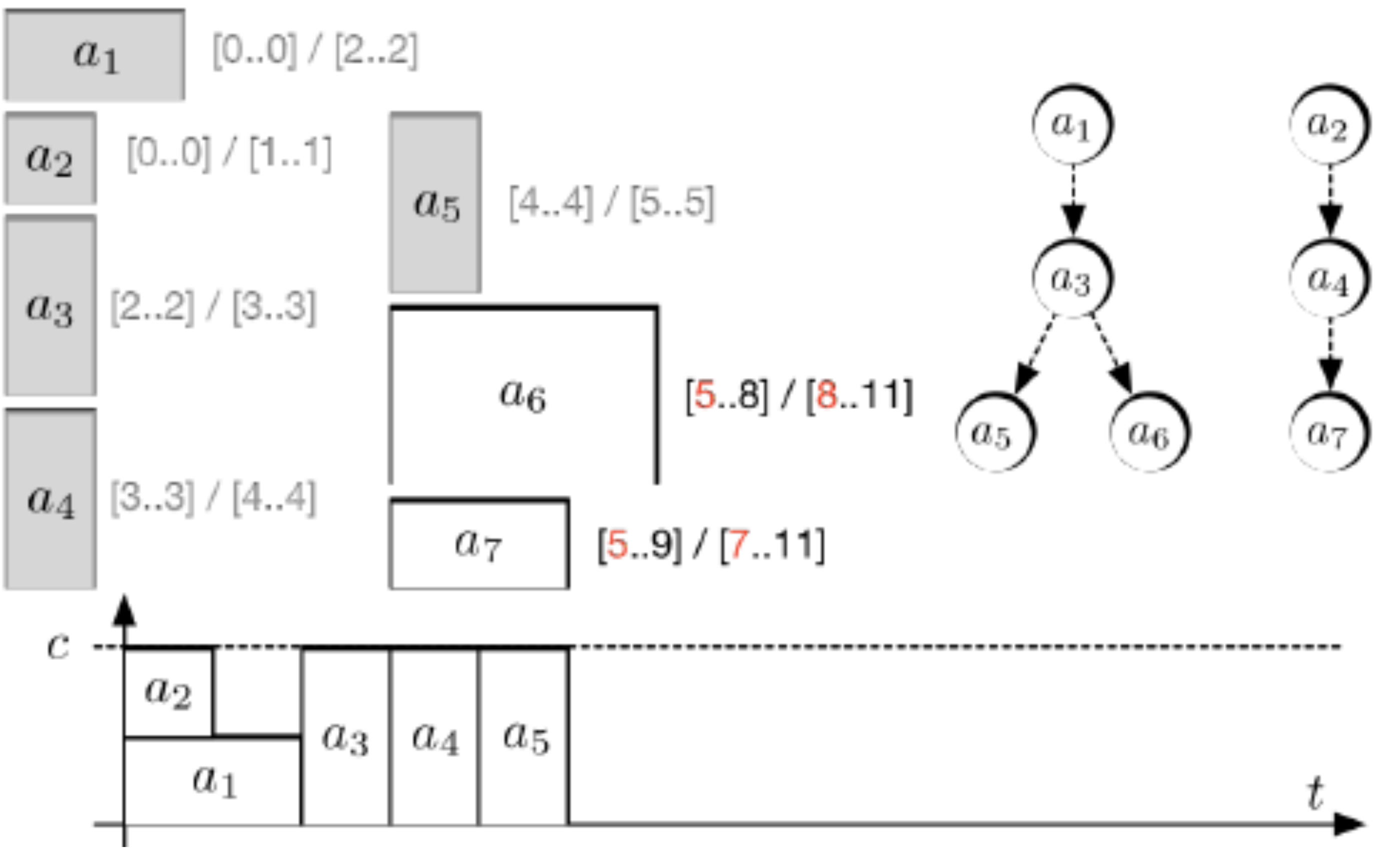
Variable Selection for the RCPSP



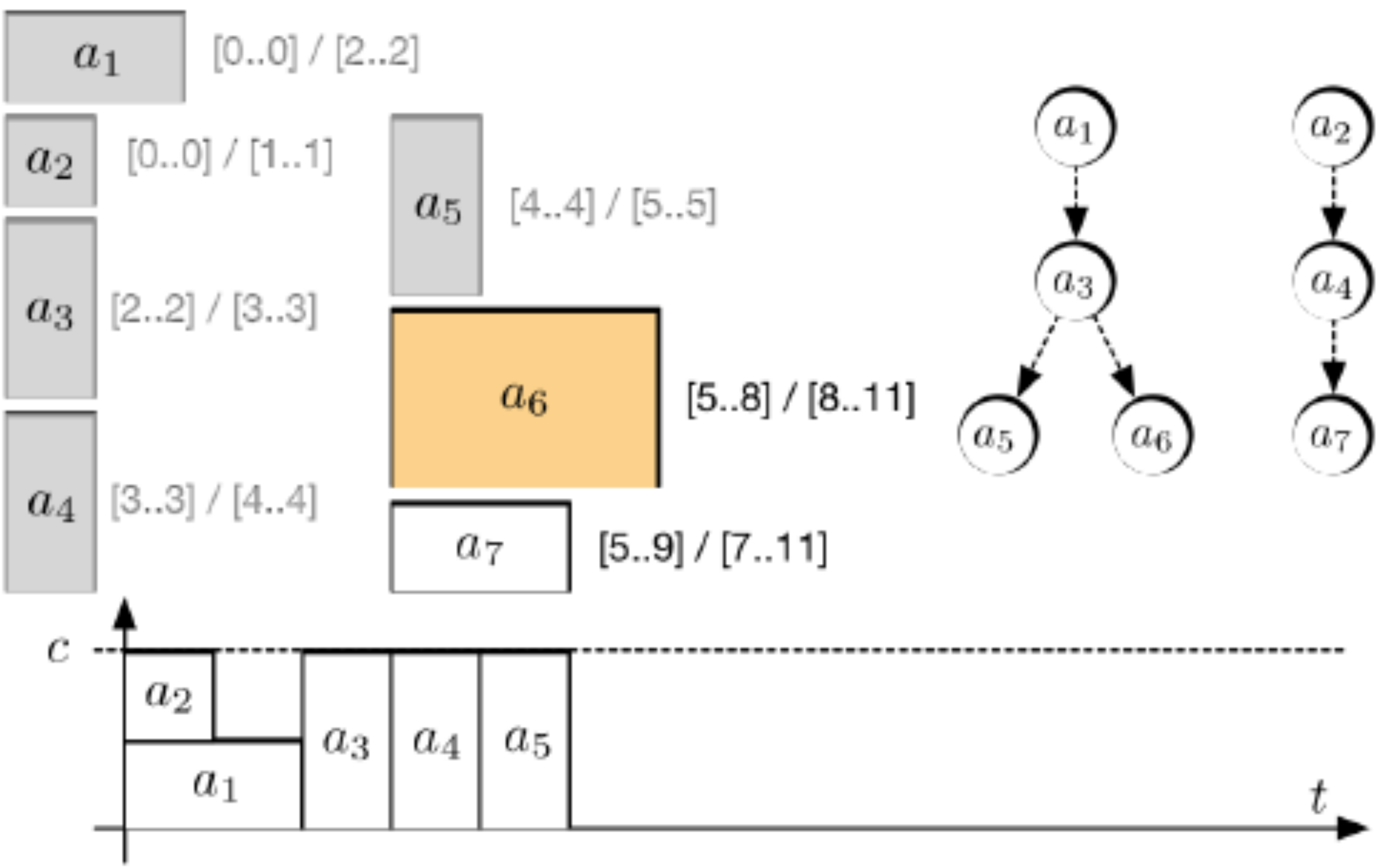
Variable Selection for the RCPSP



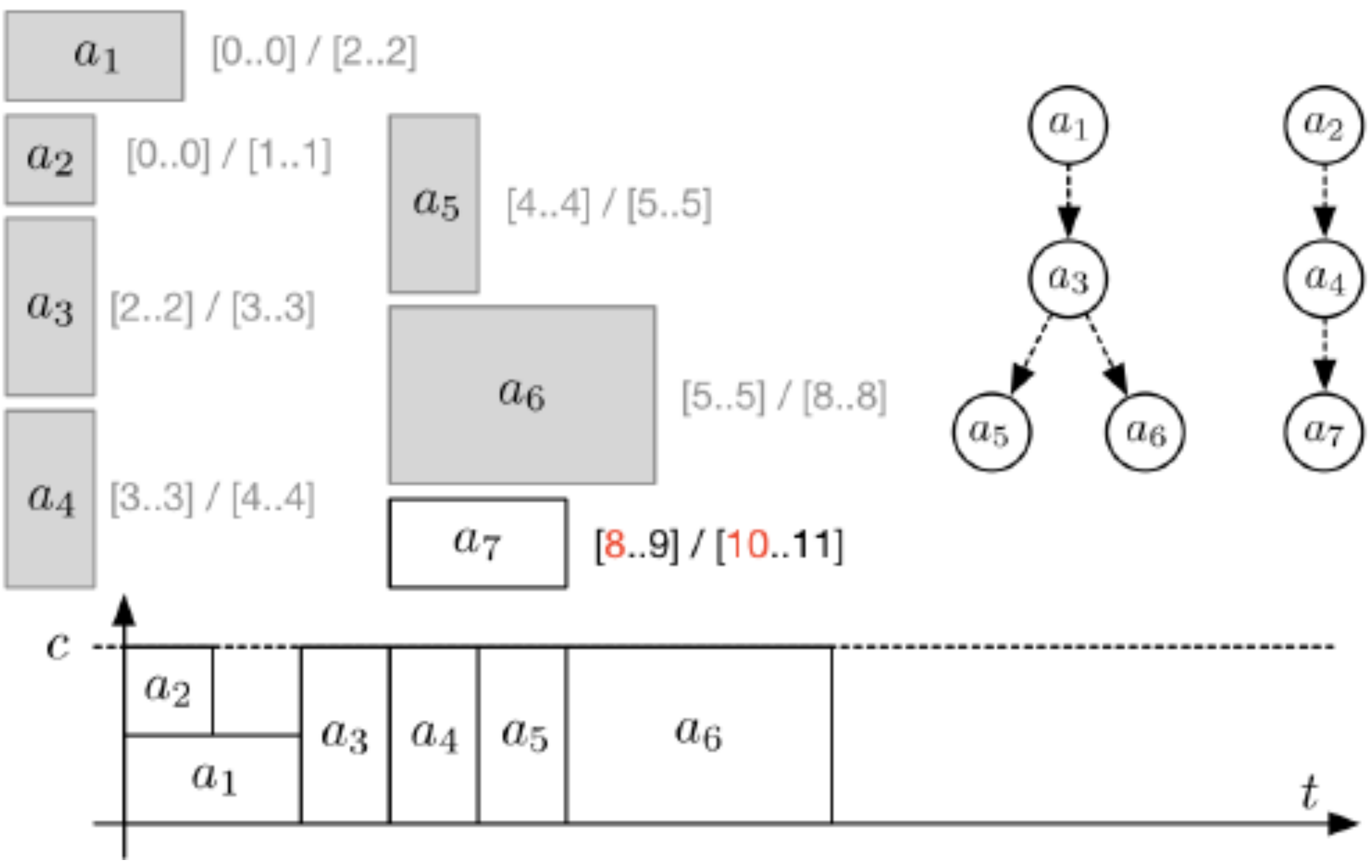
Variable Selection for the RCPSP



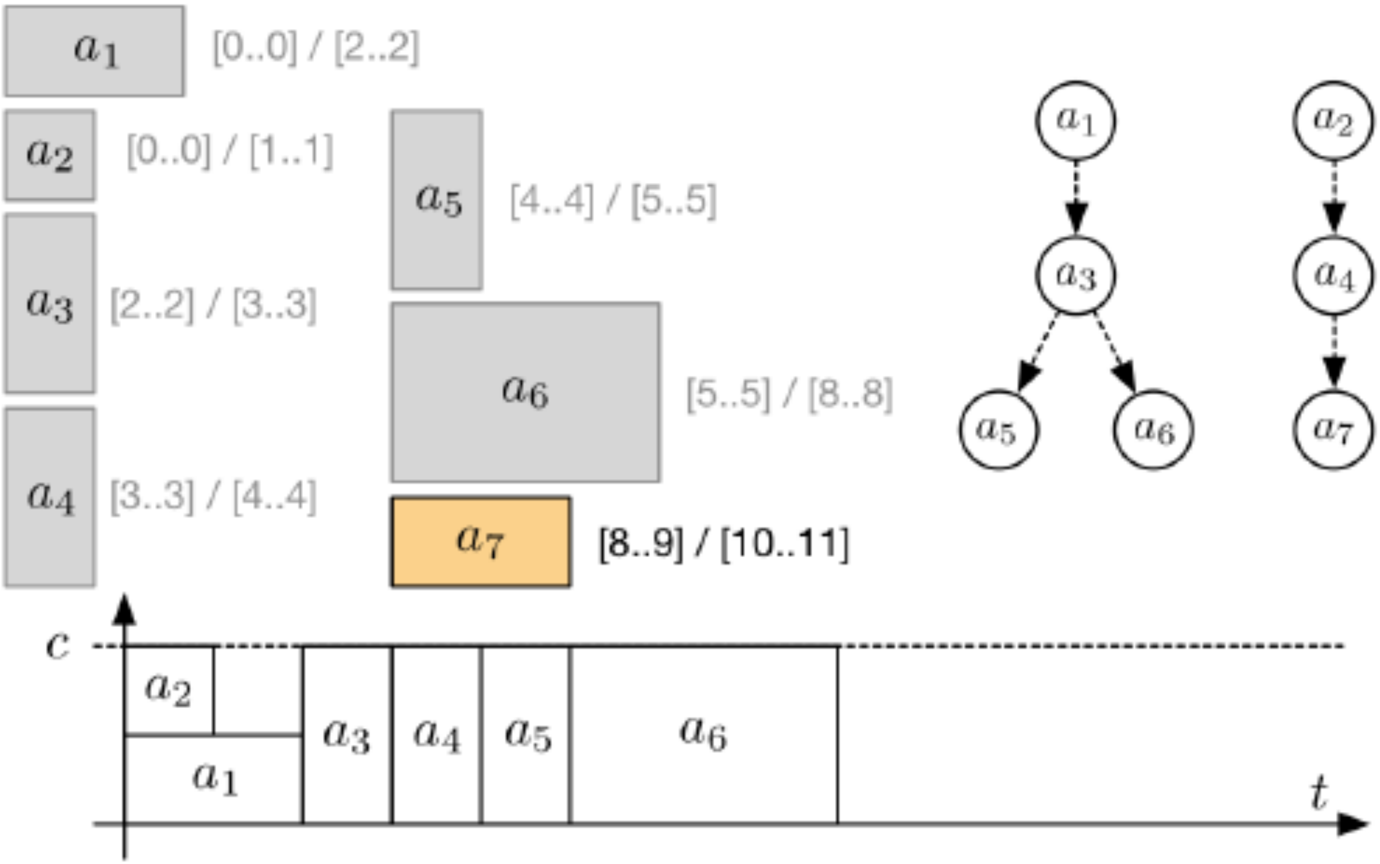
Variable Selection for the RCPSP



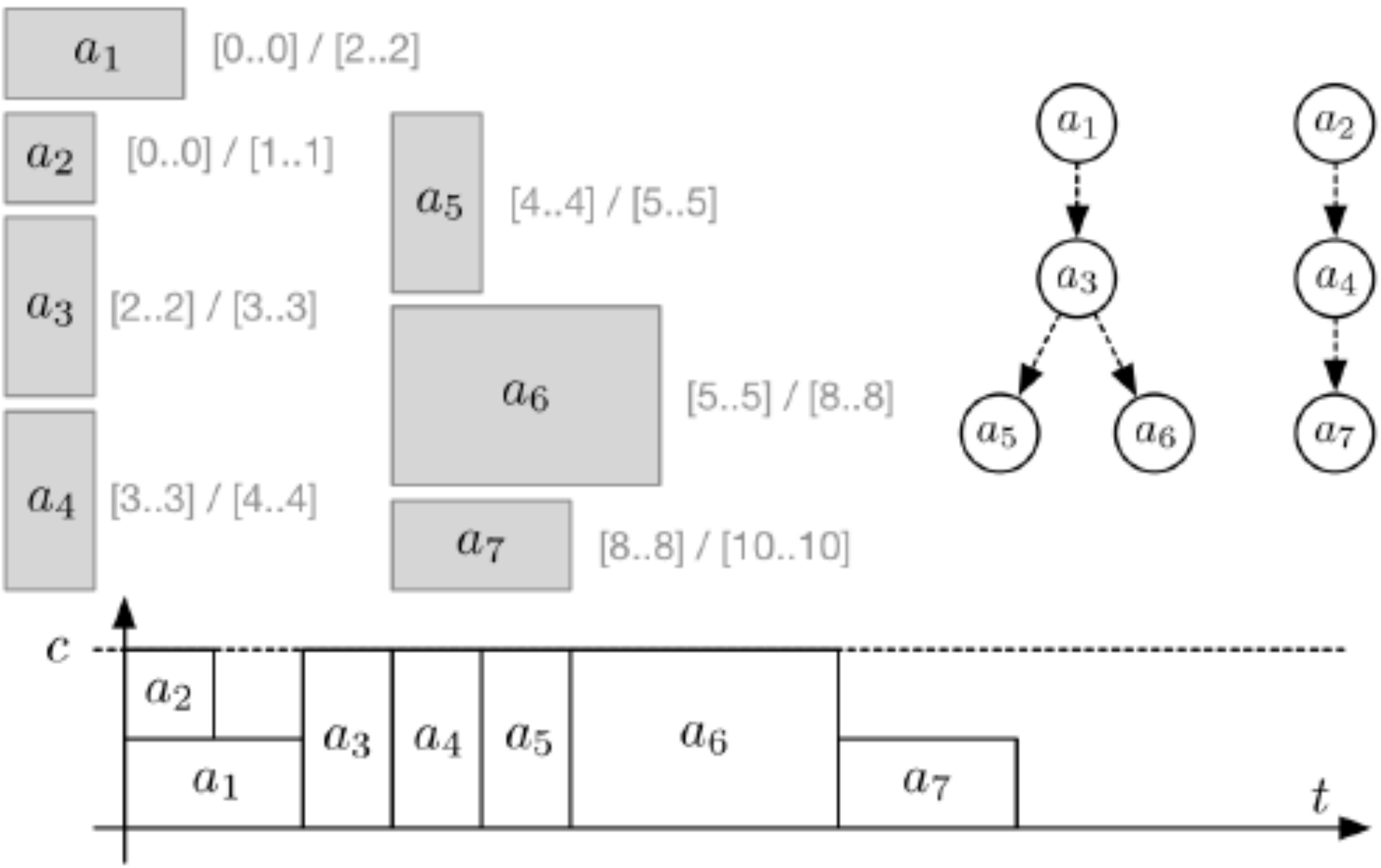
Variable Selection for the RCPSP



Variable Selection for the RCPSP

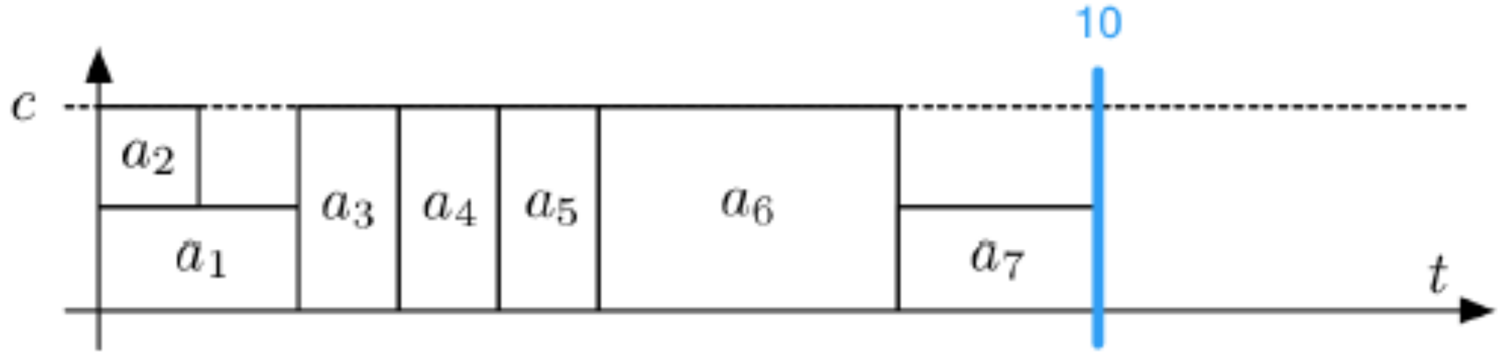


Variable Selection for the RCPSP



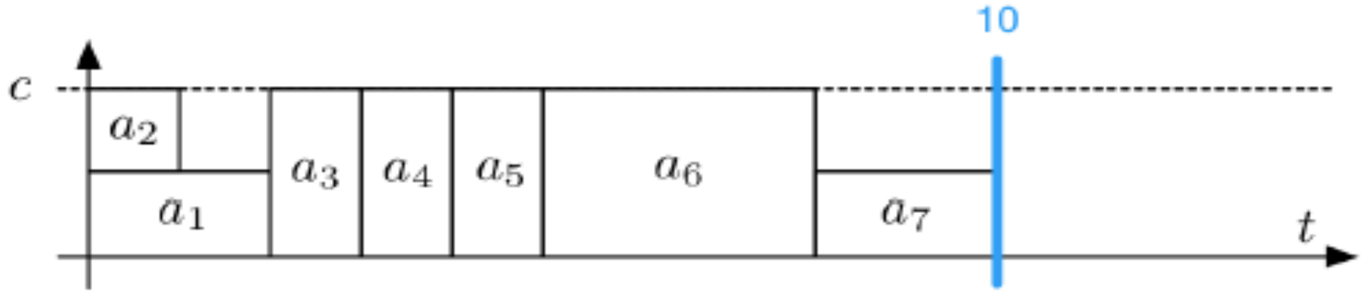
Priority Rule-Based Scheduling

- A simple greedy solution approach.
- Works well in many cases.

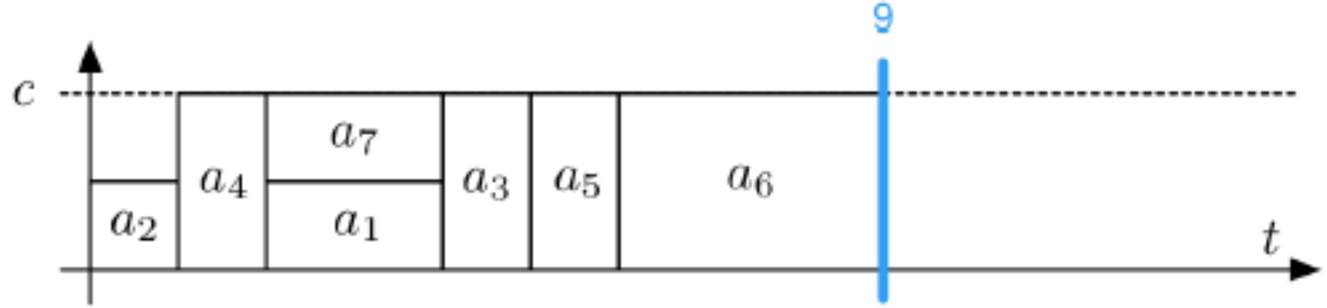


Priority Rule-Based Scheduling

- May not give the optimal solution.
- A PRB solution.

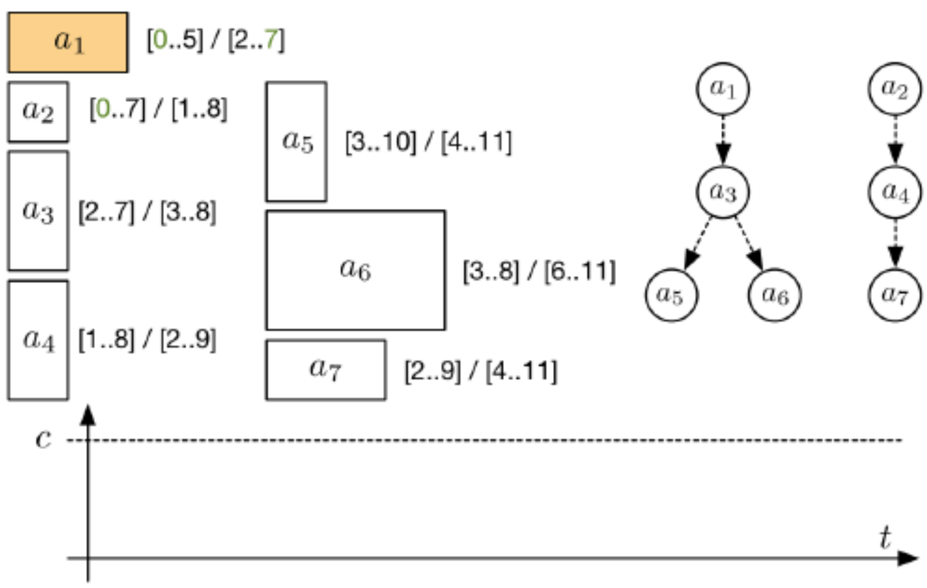


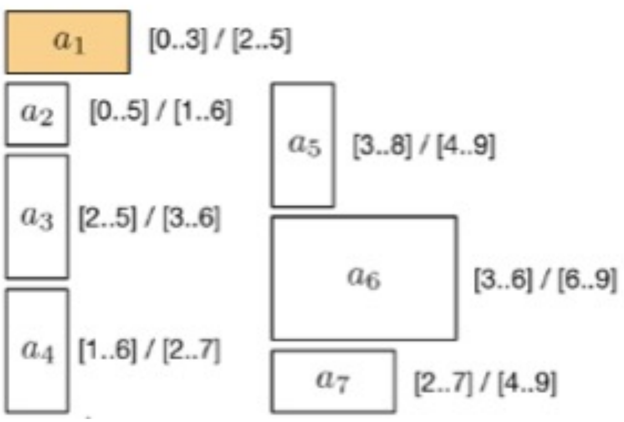
- An optimal solution.



Backtracking for Proving Optimality

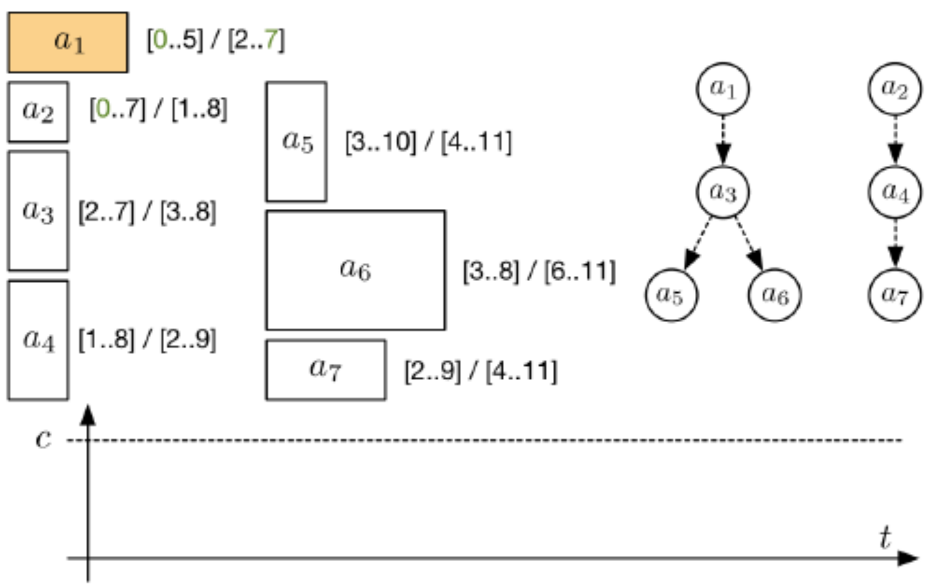
- Need to go back to the root node after posting $S_{n+1}<10$.

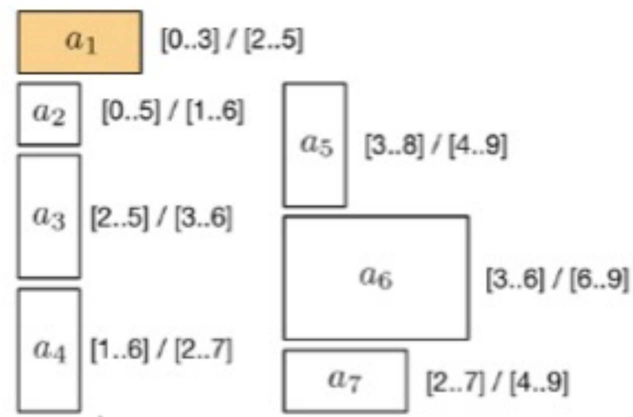




Backtracking for Proving Optimality

- Need to go back to the root node after posting $S_{n+1}<10$.
  - $S_1 \neq 0$?

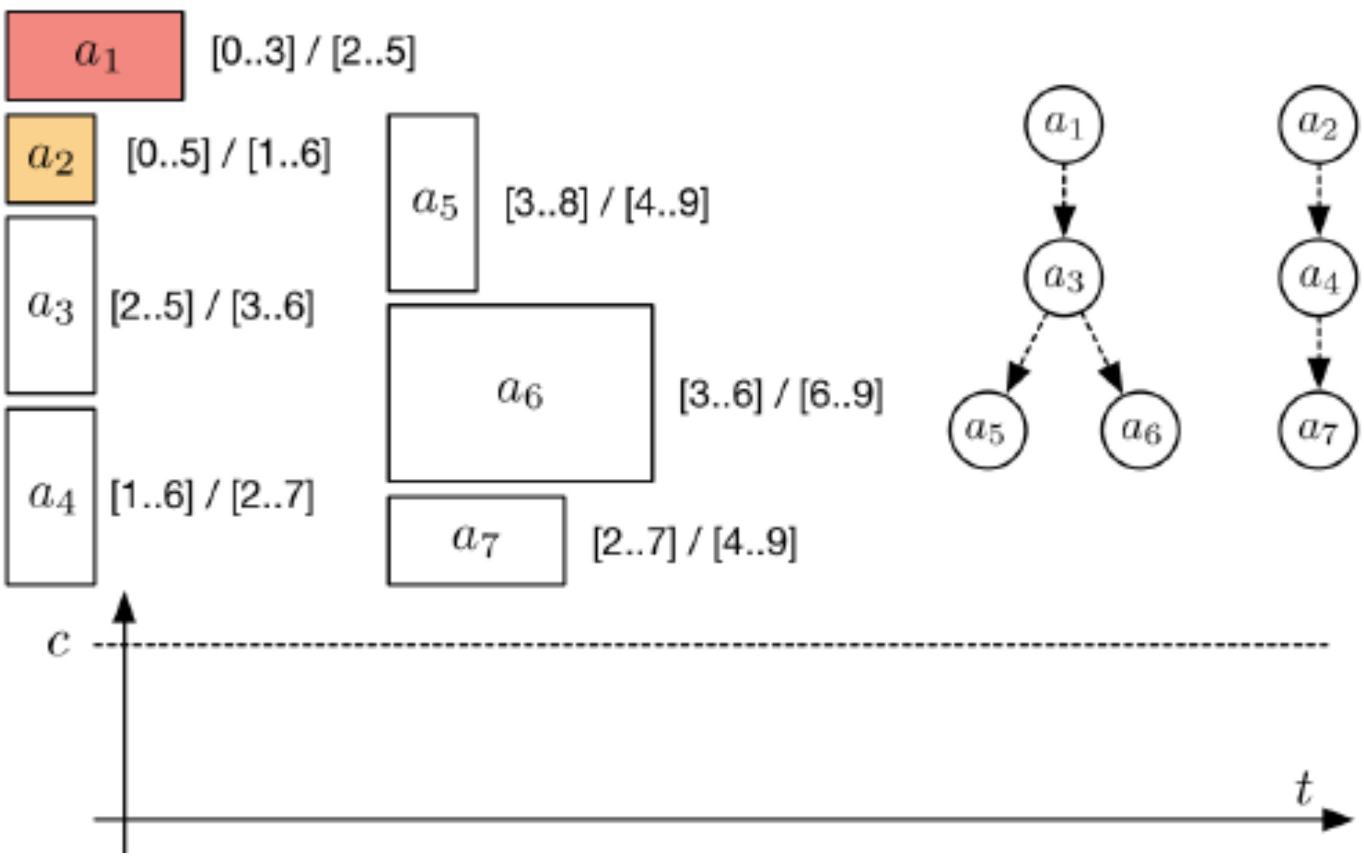




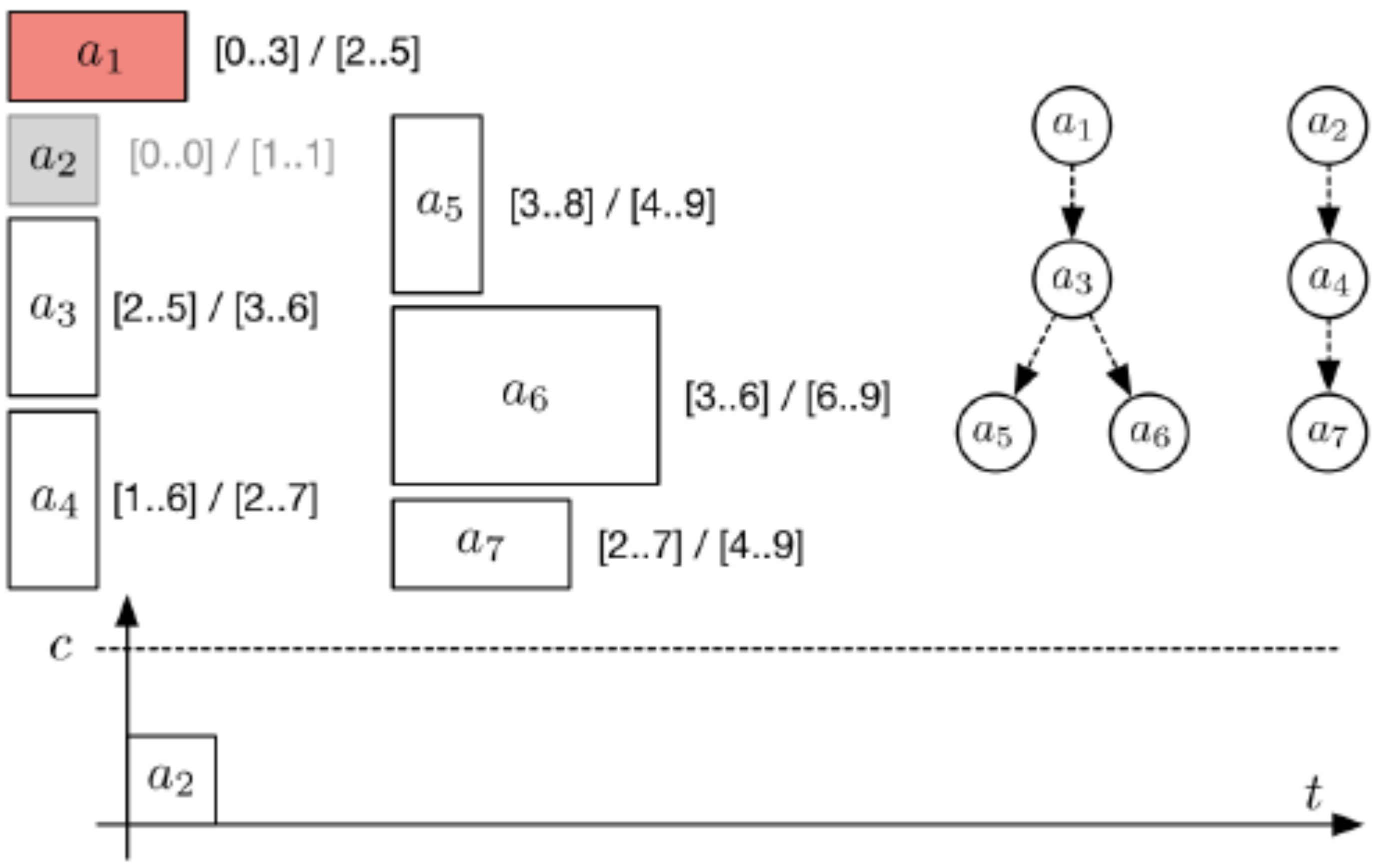
Backtracking for Proving Optimality

- $S_1 \neq 0$
  - It is weak, since S_i domains tend to be very large.
- **Alternative**: mark activity **i** as postponed.
  - A postponed activity cannot be selected for branching until its EST_i changes.
- **Rationale**: we want to explore a different branching decision.
  - We always schedule activities at their EST_i.
  - The scheduling decision changes when EST_i changes.

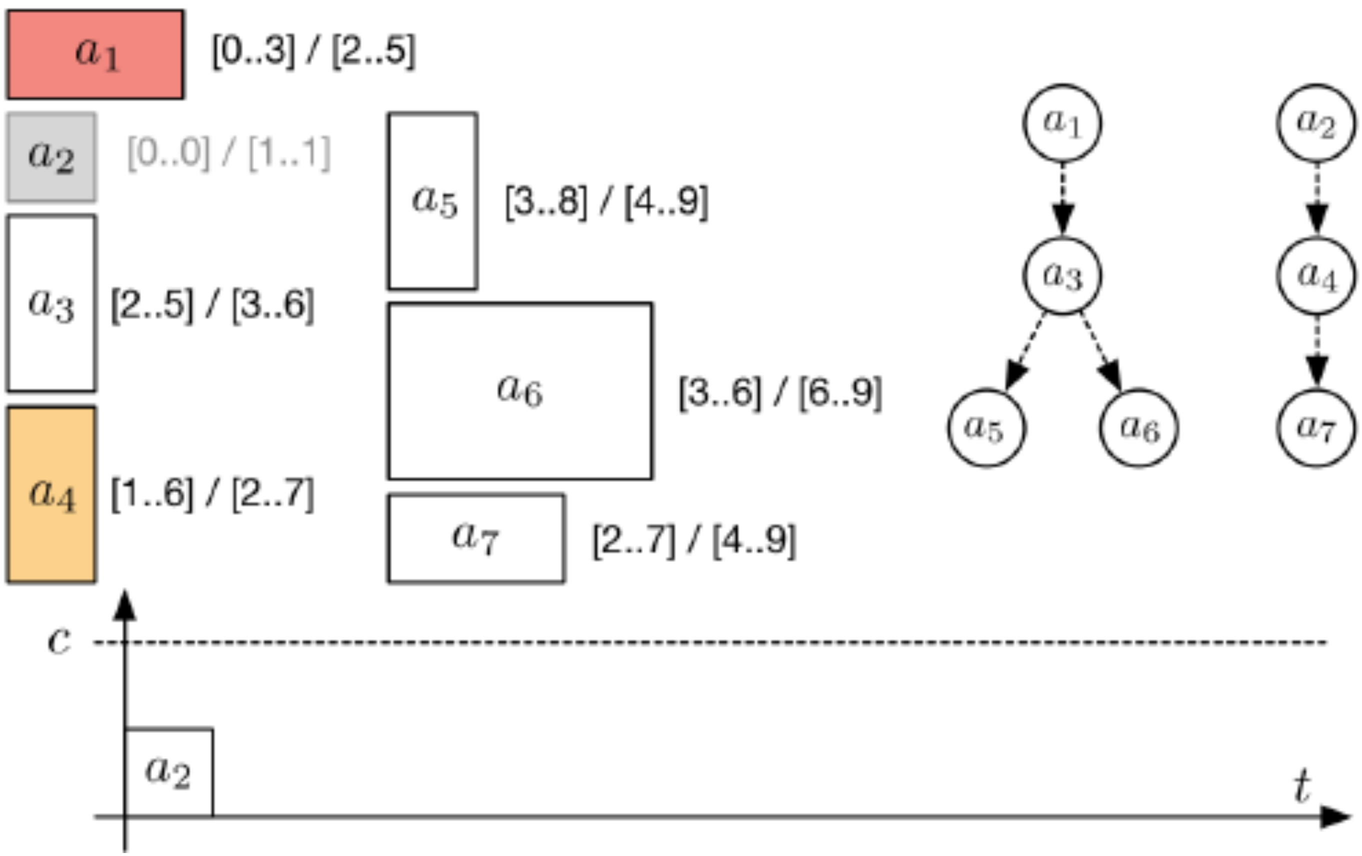
Backtracking for Proving Optimality



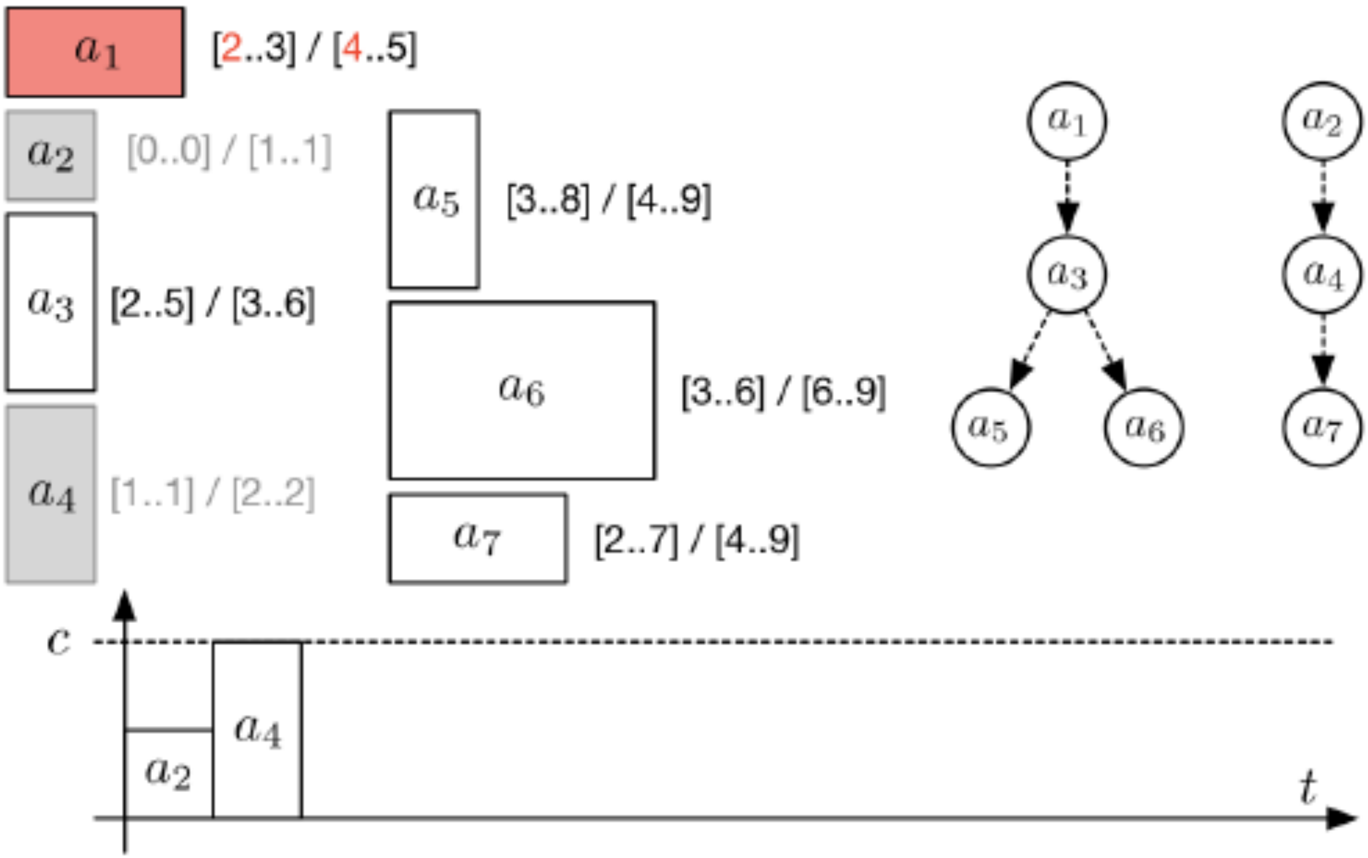
Backtracking for Proving Optimality



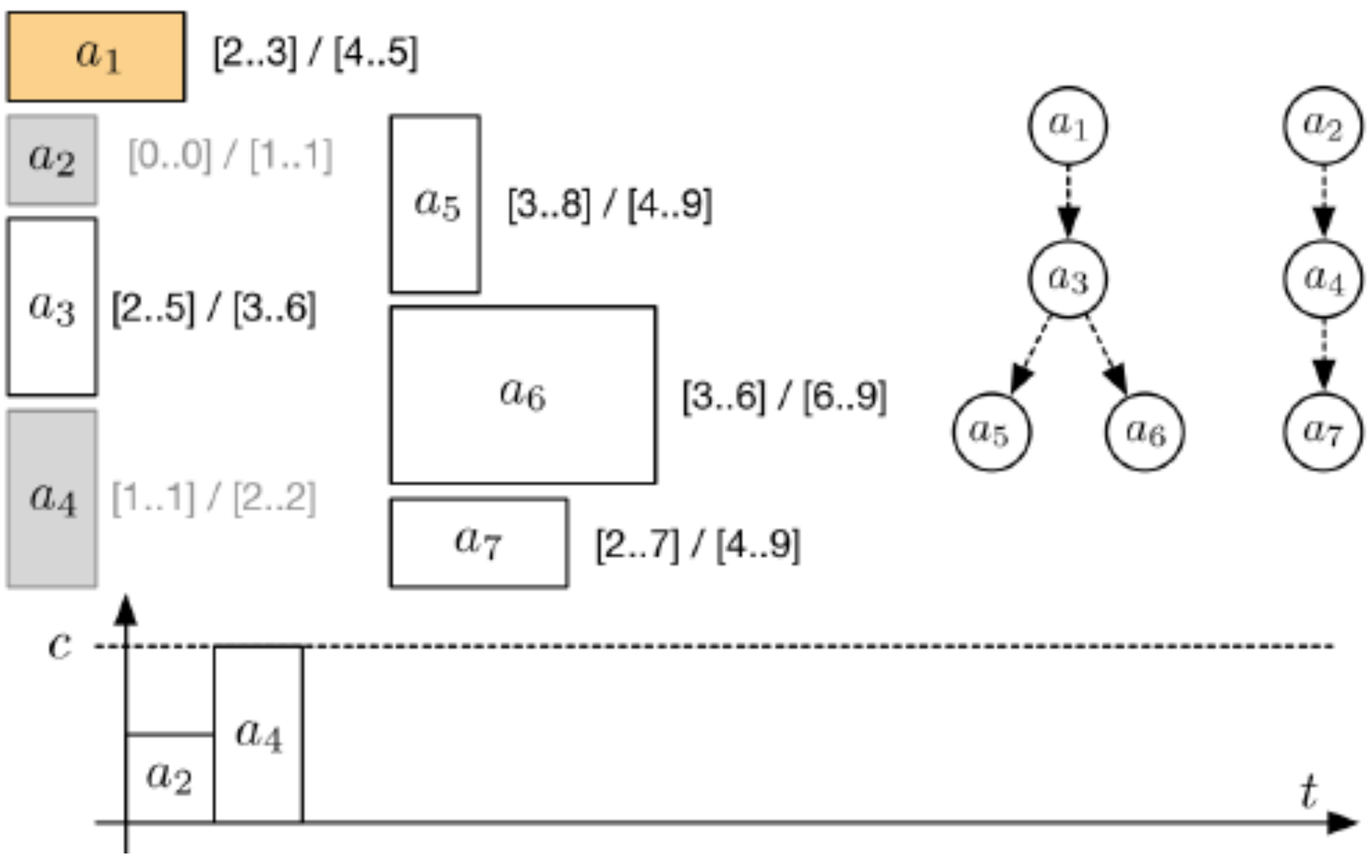
Backtracking for Proving Optimality



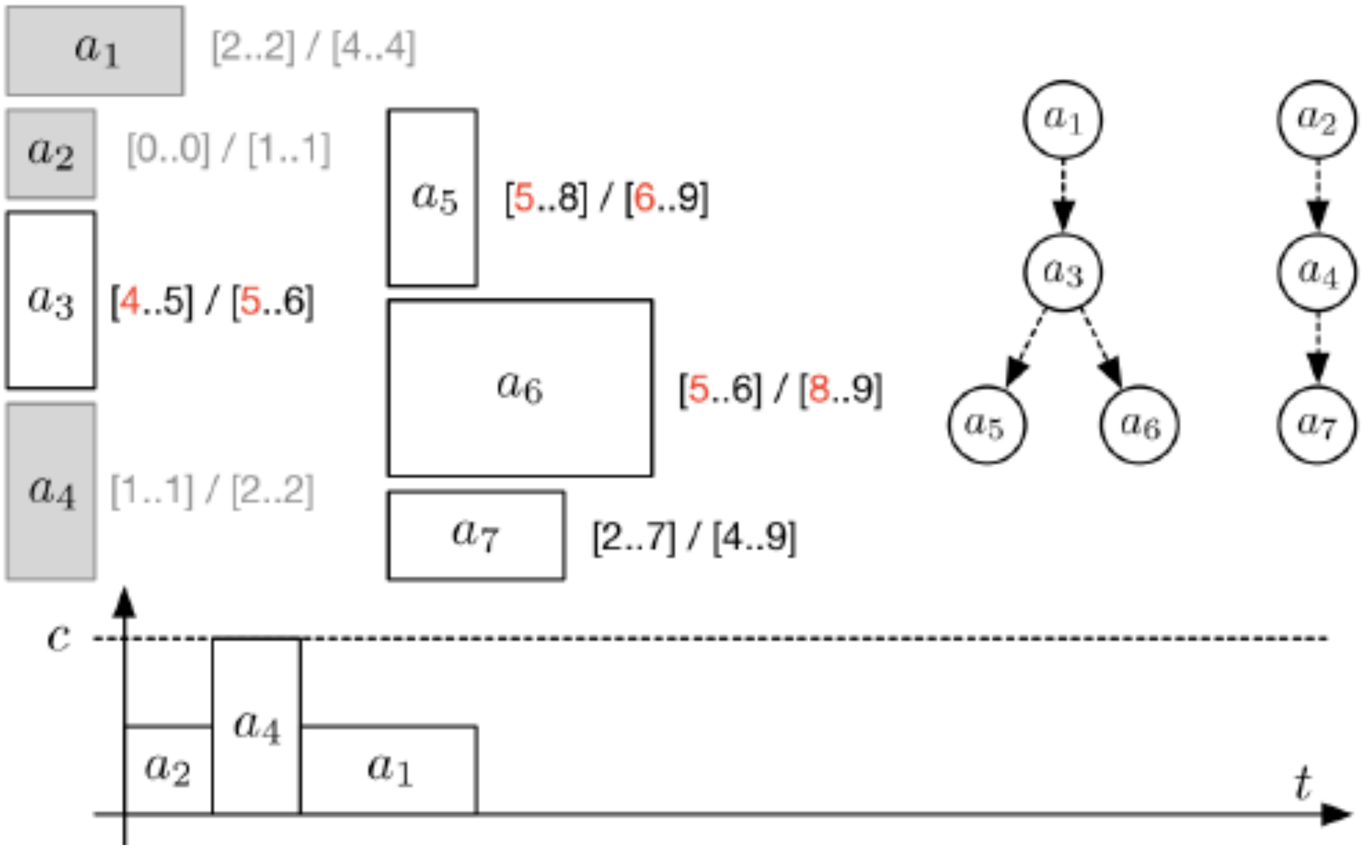
Backtracking for Proving Optimality



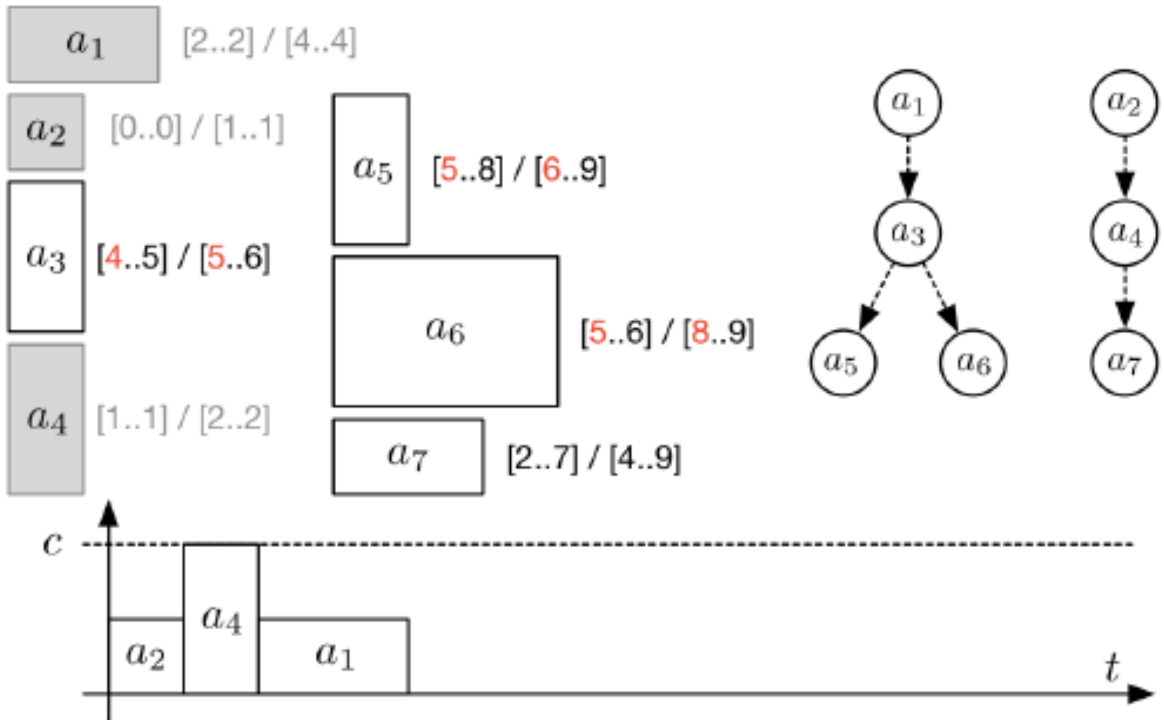
Backtracking for Proving Optimality



Backtracking for Proving Optimality

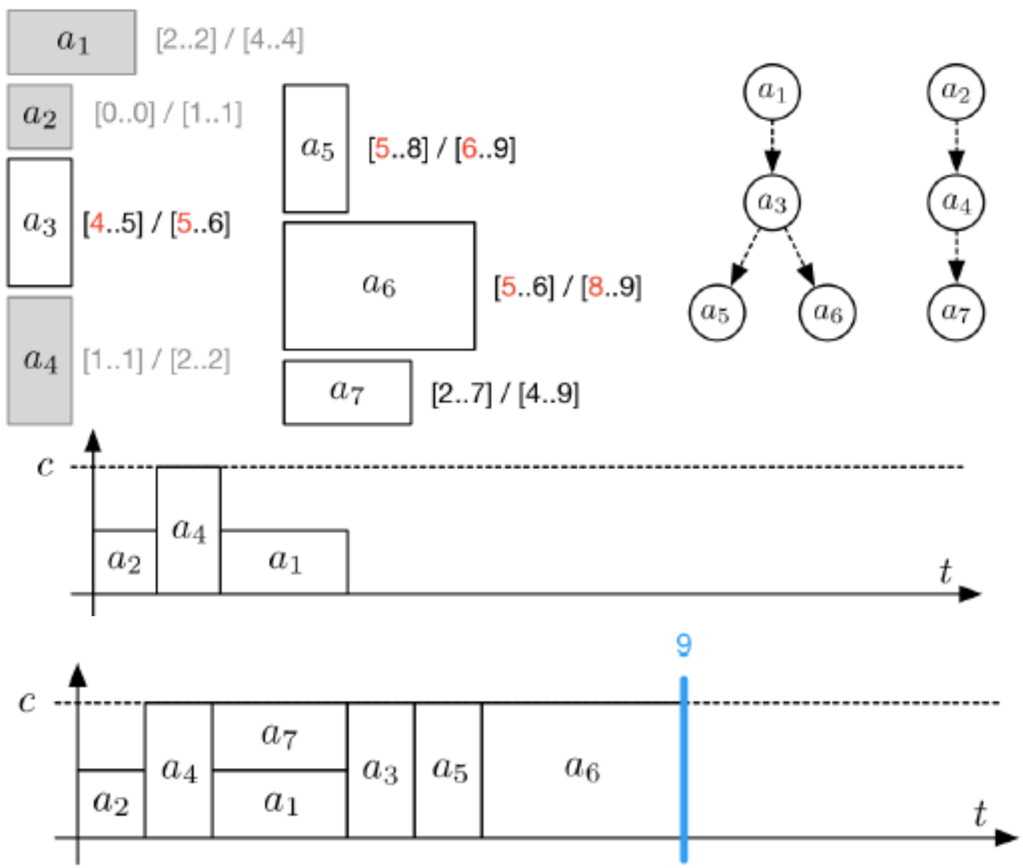


Backtracking for Proving Optimality



- By proceeding along this branch, we will find the optimal solution.

Backtracking for Proving Optimality



SetTimes Search Strategy

- Main idea
  - On the first branch schedule an activity a_i with minimum EST_i, schedule it at its EST_i.
    - Break ties according to any rule.
  - On backtracking, postpone a_i.
    - When propagation updates EST_i, schedule a_i.

SetTimes Search Strategy

- A very effective search strategy.
  - Based on PRB scheduling: finds good solutions early.
  - Effective branching choices (much better than posting $S_i \neq v$).
- Incomplete search strategy
  - At choice points, we do not partition the search space.
  - Either we schedule an activity **i** at EST_i or we make it wait.

SetTimes Search Strategy

- Why does it work?
  - The cost function is regular.
  - There is no point in not scheduling activities at their EST_i unless they are delayed by previous activities.
- When doesn't it work?
  - Non-regular cost functions.
    - E.g., costs for starting activities too early.
  - Side constraints that alter the problem structure.
    - E.g., maximal time legs.
- Other strategies are becoming more popular. E.g., domain splitting.