Nested Active-Time Scheduling

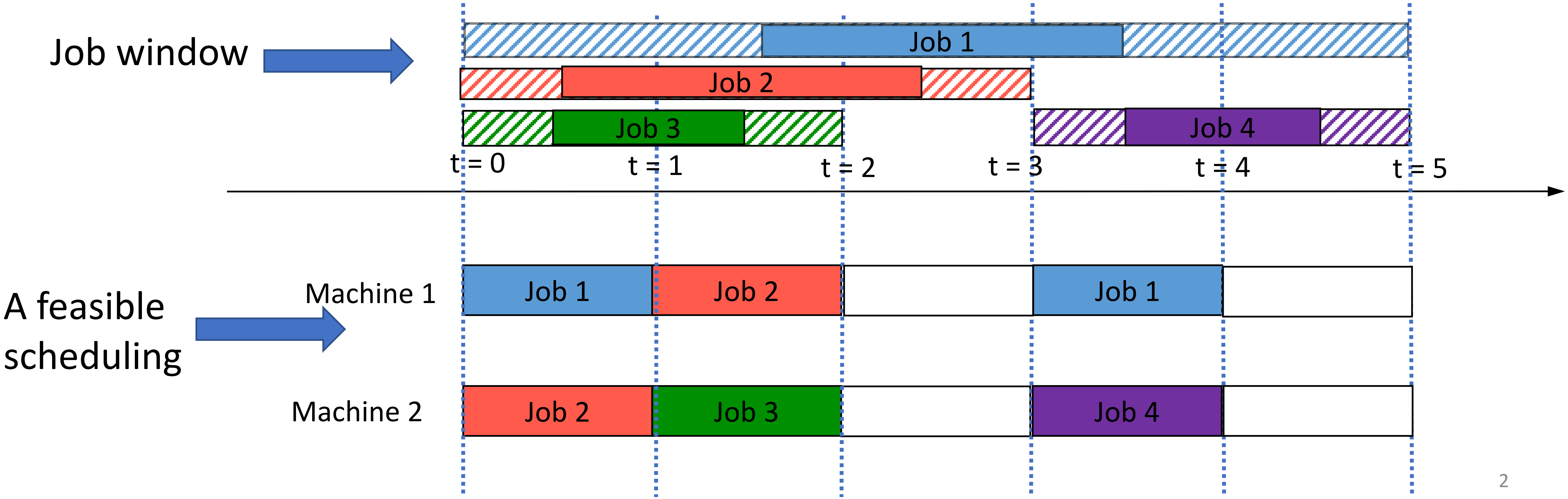
Nairen Cao, Jeremy T. Fineman, Shi Li,

Julián Mestre, Katina Russell, Seeun William Umboh

ISSAC 2022

Example of active time problem: g = 2

- Job 1: window: $[0,5)$; length 2
- Job 2: window: $[0,3)$; length 2
- Job 3: window: $[0,2)$; length 1
- Job 4: window: $[3, 5)$; length 1



Active-time problem: definition

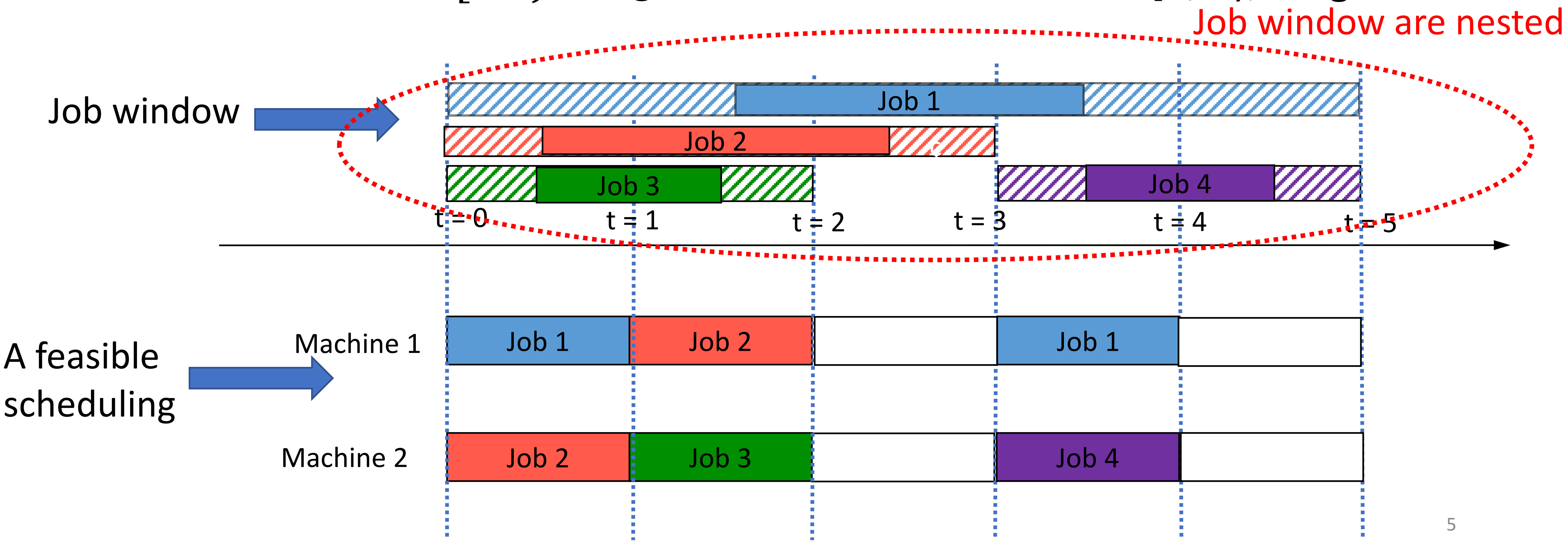
- Given $g > 0$ machines.
- Given a set of jobs J, where each job $j \in J$ has release time r_j, deadline d_j, and length p_j. We call $[r_j, d_j)$ the job j's window.
- Time is organized into discrete (integer) steps or slots, and preemption is allowed but only at slot boundaries. A time slot is active if there is a job scheduled inside.
- Target: find a schedule with minimum number of active steps that schedule all jobs within their windows

Nested Active-time problem: definition

- The same as active time problem. In addition,
- For each pair job, either their windows are disjoint, or one is fully contained in the other.

Example of active time problem: g = 2

- Job 1: window: $[0,5)$; length 2
- Job 2: window: $[0,3)$; length 2
- Job 3: window: $[0,2)$; length 1
- Job 4: window: $[3, 5)$; length 1



Related work

Paper	result	Method	Remark
Chang, Gabow, and Khuller 2014	2-approximate	Linear programming	General case
Kumar and Khuller 2018	2-approximate	Greedy algorithm	General case
Călinescu and Wang 2021	2-approximate	Linear programming	General case
Sagnik and Manish 2021	NP complete		General case

Related work

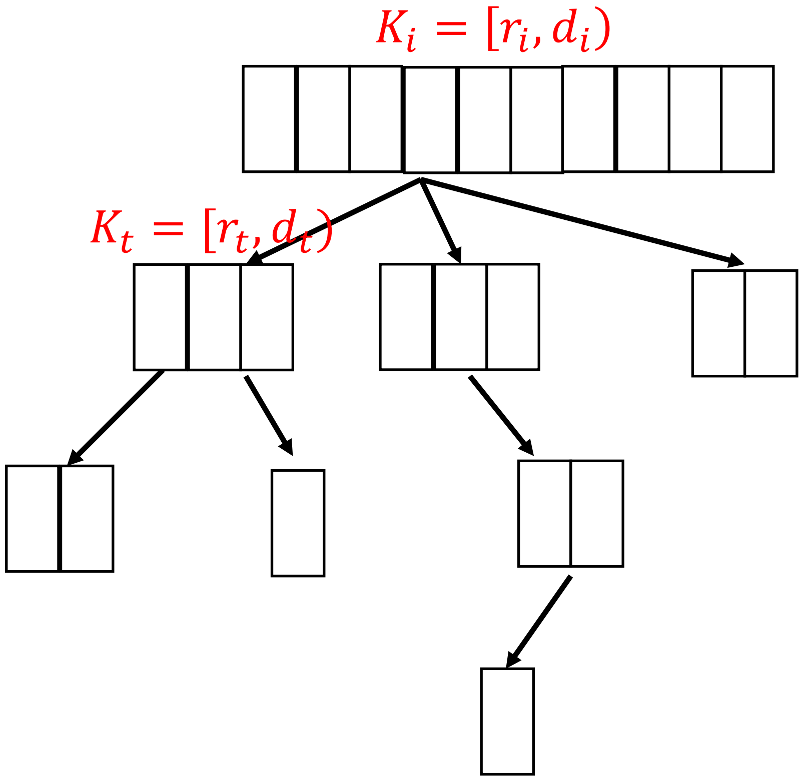
Paper	result	Method	Remark
Chang, Gabow, and Khuller 2014	2-approximate	Linear programming	General case
Kumar and Khuller 2018	2-approximate	Greedy algorithm	General case
Călinescu and Wang 2021	2-approximate	Linear programming	General case
Sagnik and Manish 2021	NP complete		General case
Our result	NP complete		Nested case
Our result	1.8 approximate	Linear programming rounding	Nested case

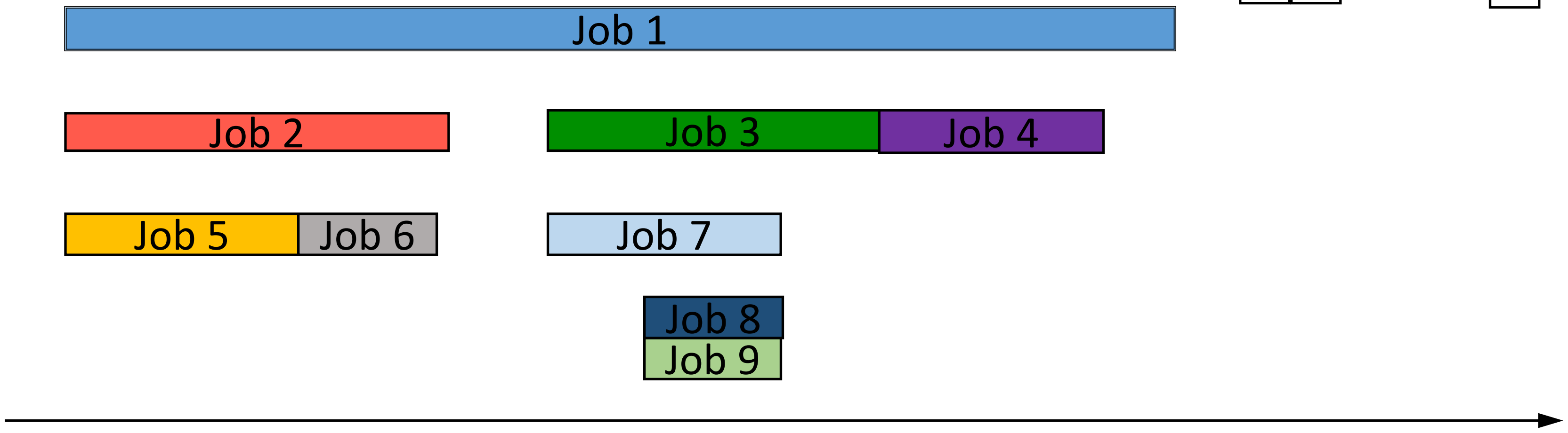
Outline

- Linear programming preprocessing
- Linear programming rounding algorithm
- The feasibility of our method

Linear programming: preprocessing

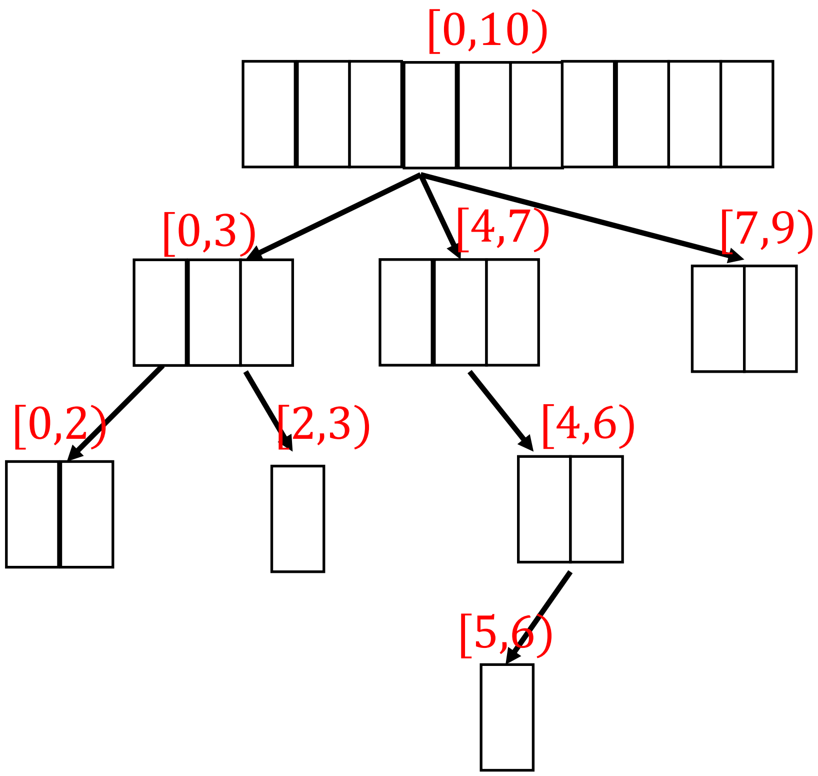
- Given an instance (g, J), we can construct a tree based on the job window and time slot.
- For node i and its child node t, we have $\mathrm{K}_t \subset K_i$.

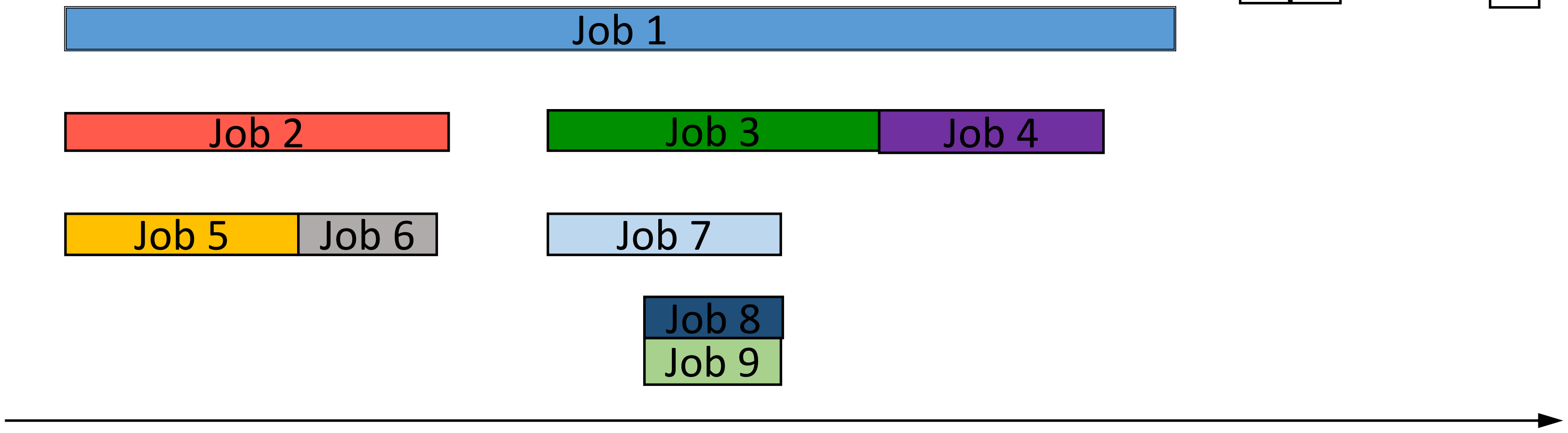




Linear programming: preprocessing example

- Given an instance (g, J), we can construct a tree based on the job window and time slot.
- For node i and its child node t, we have $\mathrm{K}_t \subset K_i$.



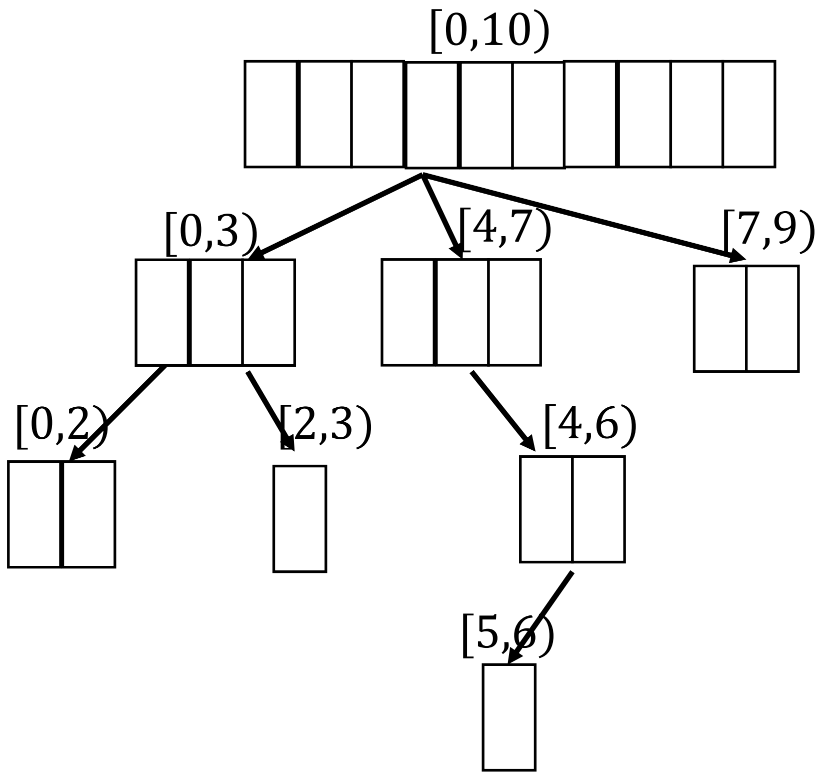


Linear programming preprocessing: Why?

- Specify the ownership of a time slot
- Extra constraint for linear programming

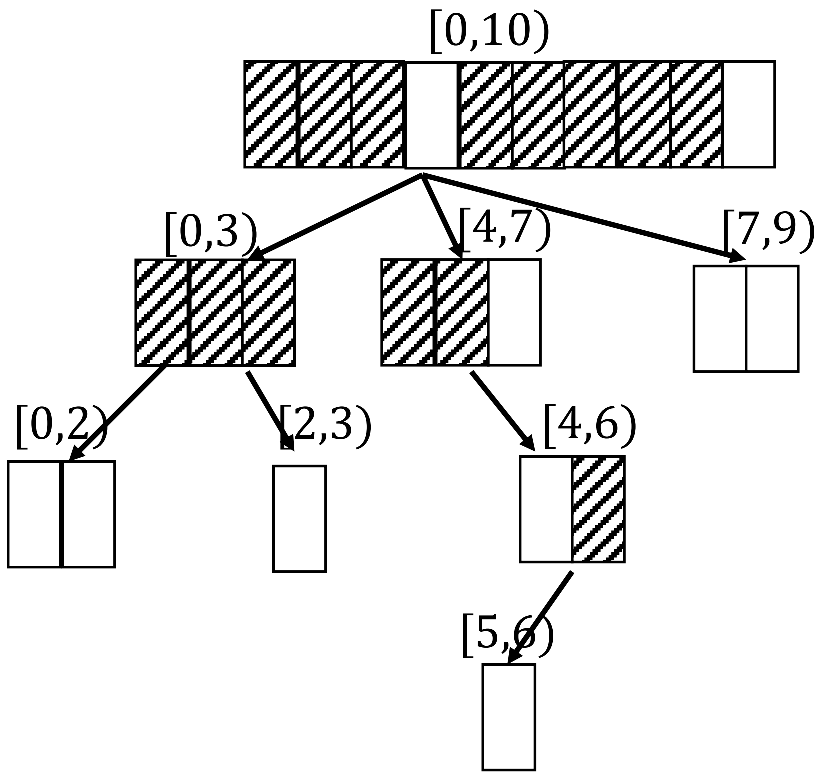
Linear programming: preprocessing

- We don't want to double count the time slot, so we only consider a time slot if it doesn't appear in the descendant node.



Linear programming: variable

- We remvoe a time slot if it appears in its descendant node.
- Variable:
  - $x(i)$ be the number of active time slots in node i and
  - $y(i,j)$ be the number of time slot job j placed in node i.



Linear programming: formulation

- Variable:
  - $x(i)$: # active time slots in node i
  - $y(i,j)$: # time slot job j placed in node i.
- Constraints:

Each job j must be scheduled.

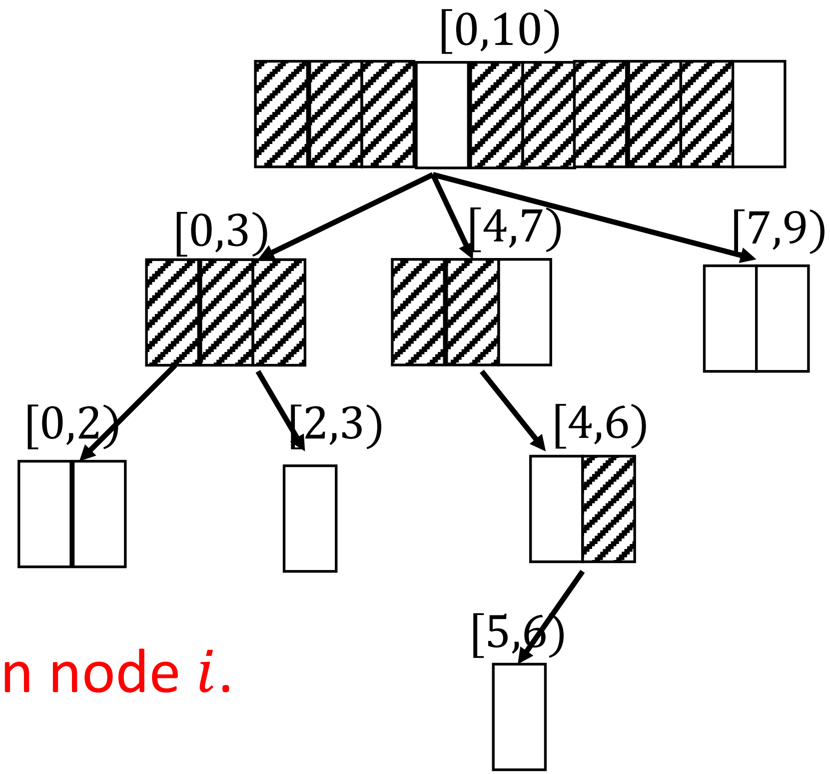
$$\sum_i y(i,j) \geq p_j$$

For each job j, we can at most schedule $x(i)$ in node i.

$$y(i,j) \leq x(i)$$

We have only g machines.

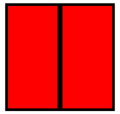
$$\sum_j y(i,j) \leq g \cdot x(i)$$



Linear programming: integer gap

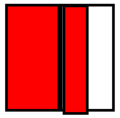
$J = \{1,2,\dots,g+1\}$ with length 1

$[0,2)$



Optimal integer solution: 2

$[0,2)$

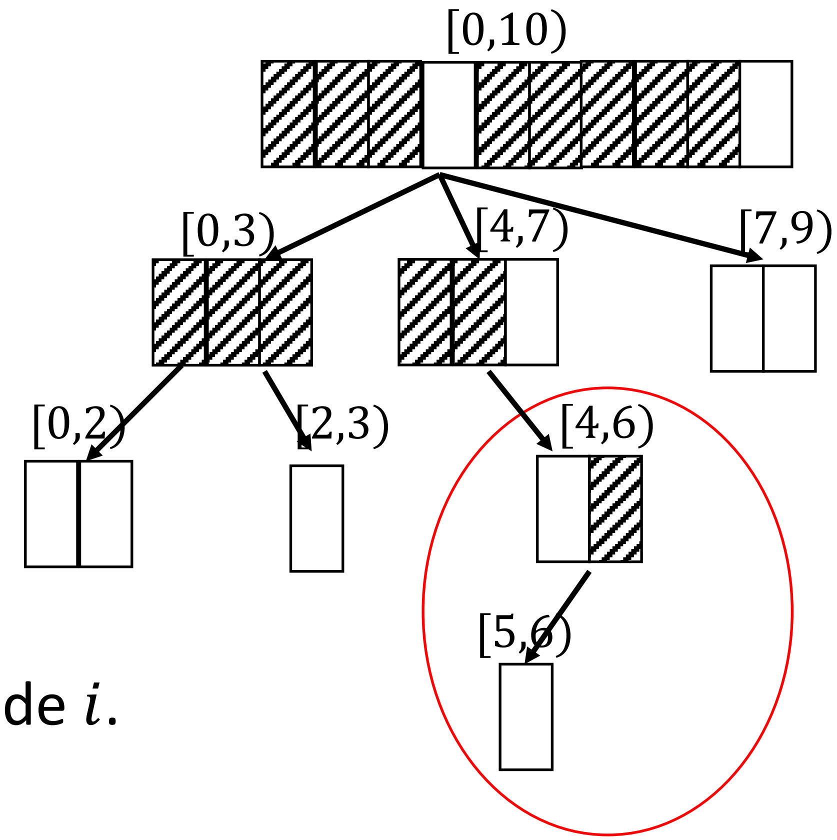


Linear programming solution:

$$\frac{g+1}{g} \to 1 (when\ g\ is\ large)$$

Linear programming processing: enhance lp

- Variable:
  - $x(i)$: # active time slots in node i
  - $y(i, j)$: # time slot job j placed in node i.
- Constraints:



Each job j must be scheduled.

For each job j, we can at most schedule x_i in node i.

We have only g machines.

For all subtree, check if we have to open at least 1,2,3 time slot

Linear programming processing: enhance lp

For subtree $\mathcal{T}$, $i \in \{1,2\}$:

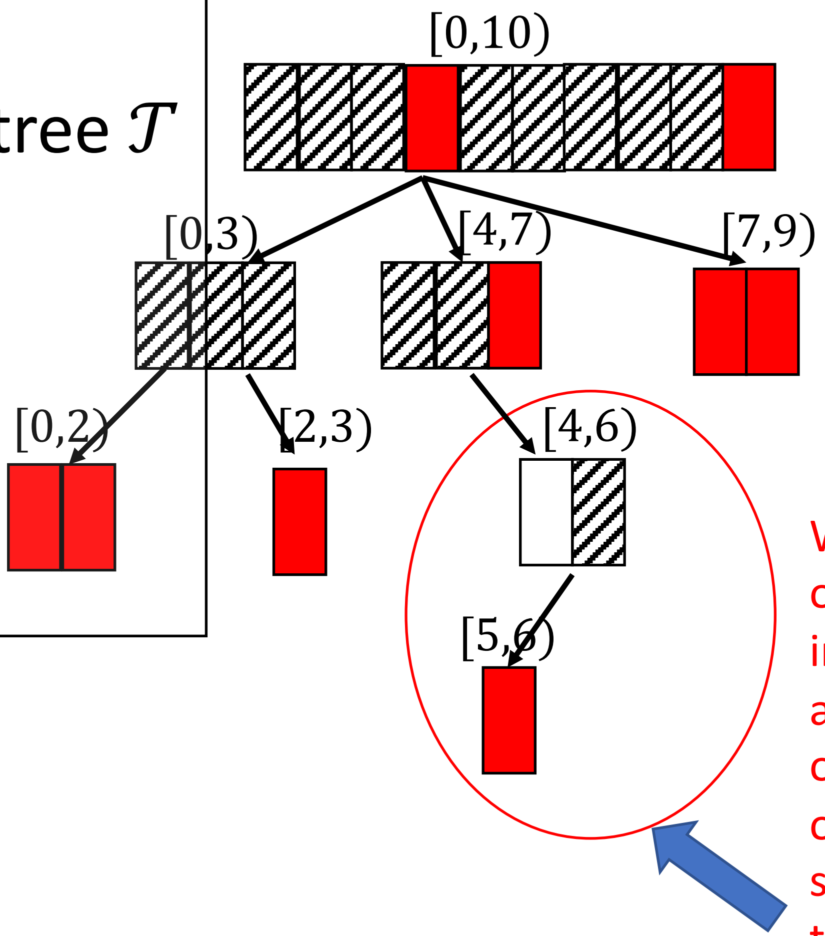
Open all possible i time slot in each subtree $\mathcal{T}$

Open all time slots outside of $\mathcal{T}$

Check if we can schedule all jobs inside

If not, set up a constraint in the linear programming:

open i+1 time slot in $\mathcal{T}$



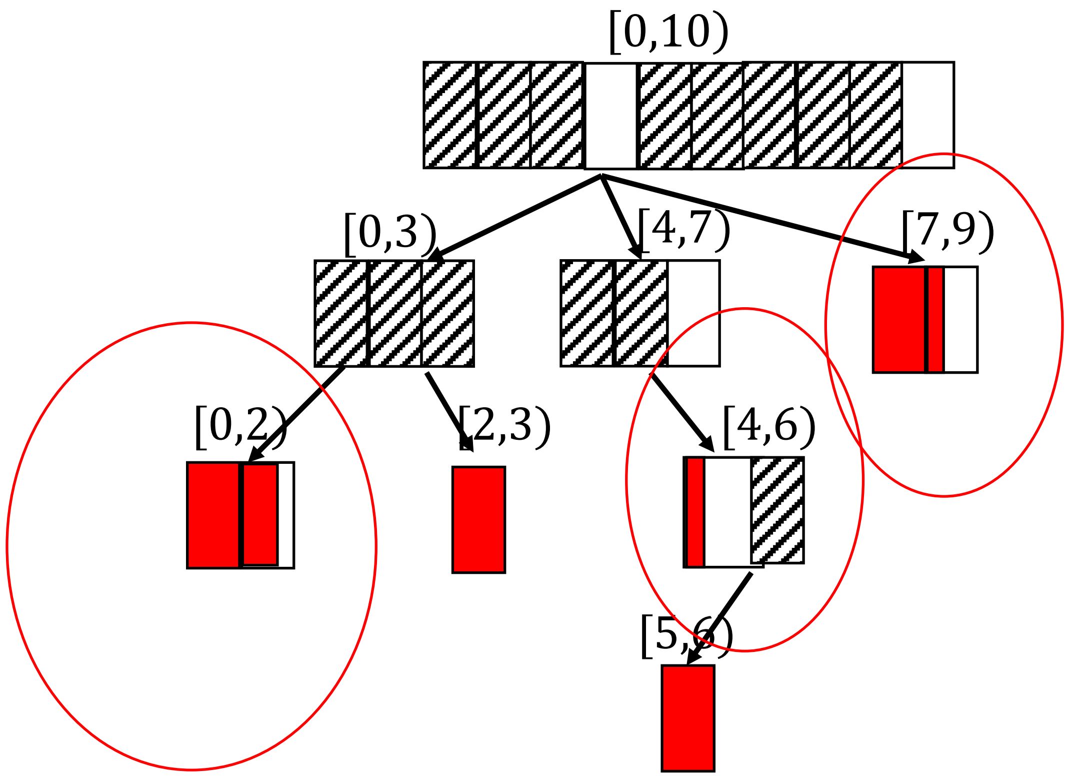
We will try to open 1 time slot in the subtree and open all other time slot outside the subtree to check the feasibility.

Outline

- Linear programming preprocessing
- Linear programming rounding algorithm
- The feasibility of our method

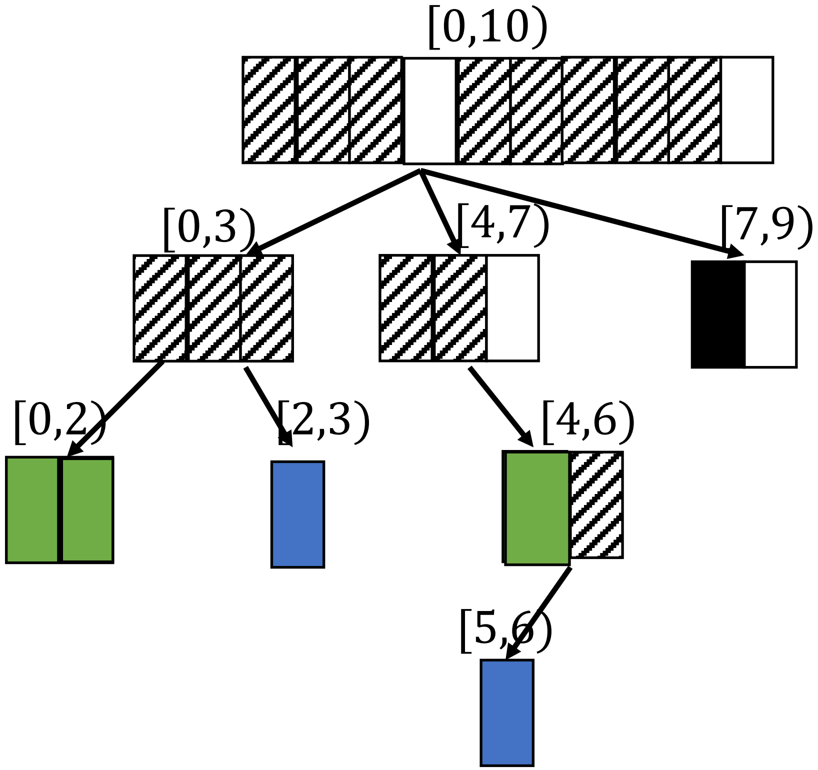
Linear programming: lp rounding

- If we round up all nodes, then we will get 2-approximate ratio.
- Round up some fractional nodes and round down some fractional nodes.



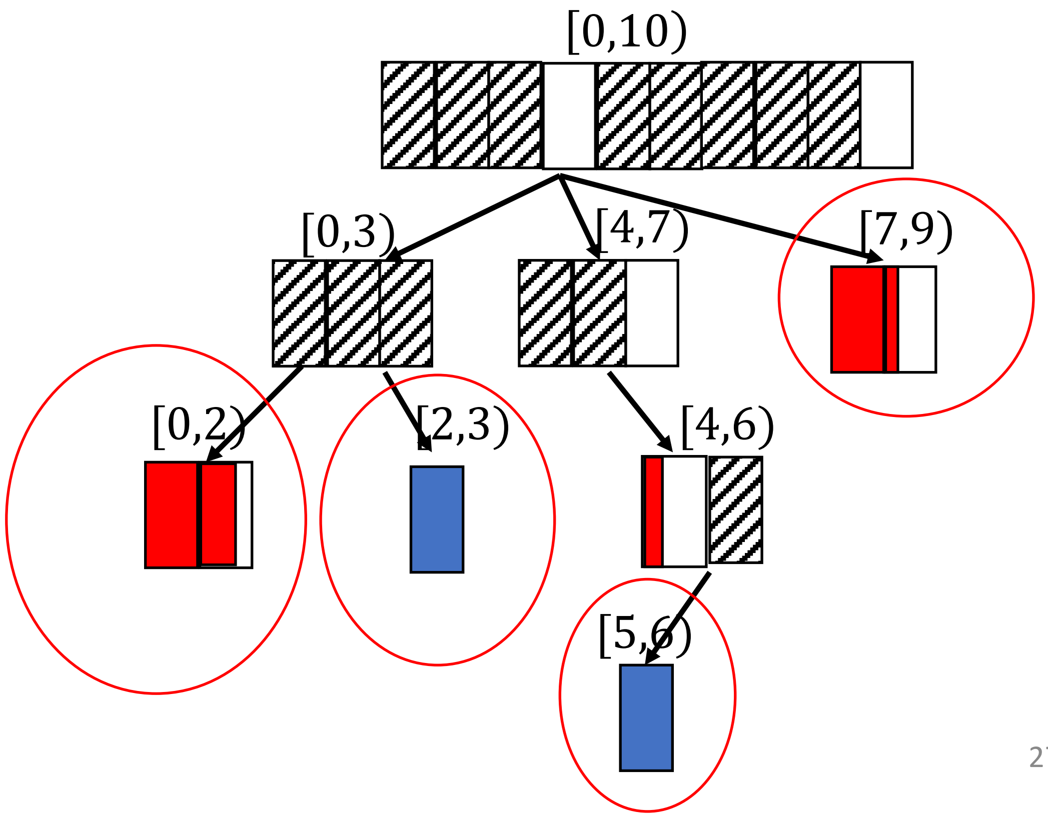
Linear programming: Hardness

- We round down black node.
- Difficulty: there might be jobs placed in [7, 9) in the linear programming. However, we need to move those jobs into other time slot.
- Our enhanced constraint can ensure those jobs can be scheduled.



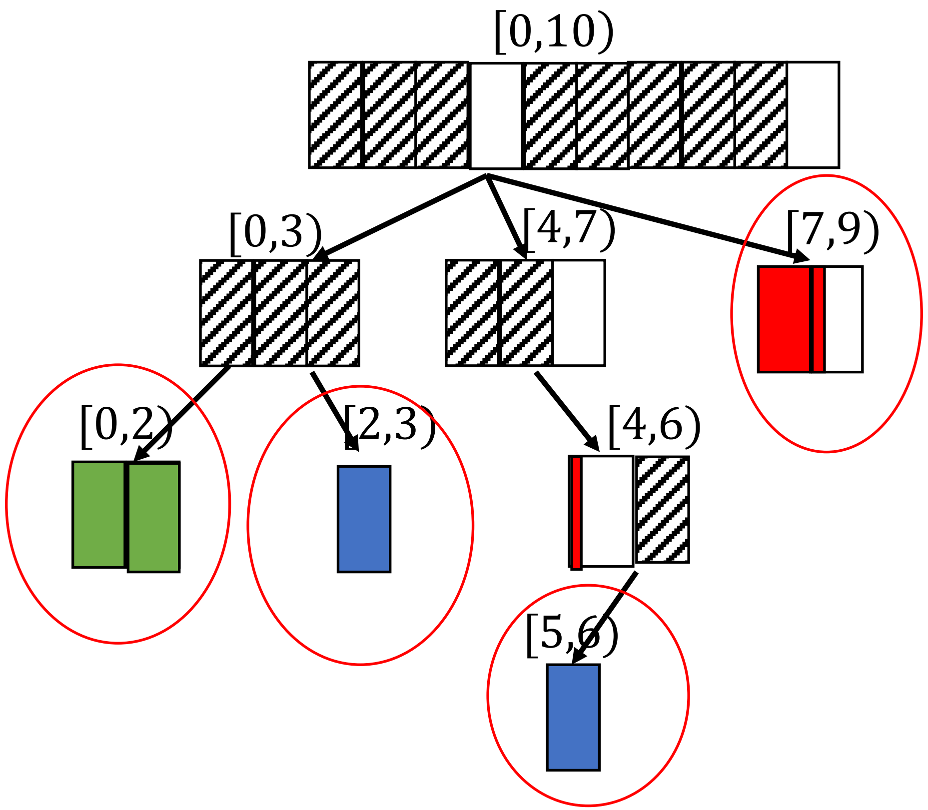
Linear programming: lp rounding

- Round up some fractional nodes.
- process from bottom to top, at any node, we consider the subtree rooted at this node.



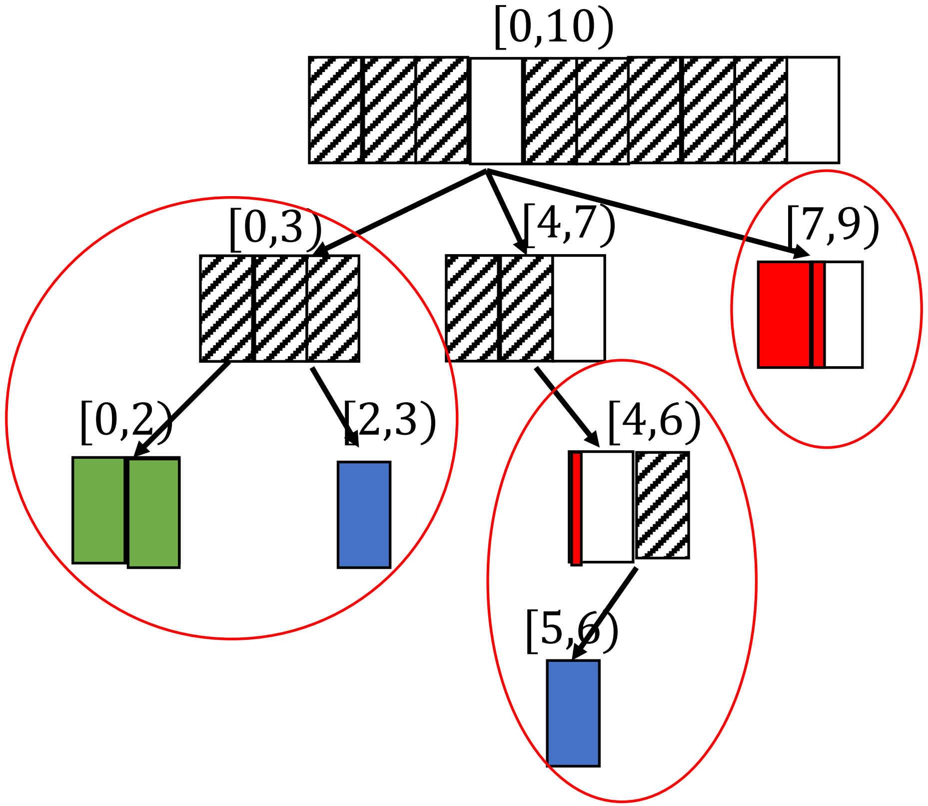
Linear programming: lp rounding example

- Blue node are integer time slot node.
- For any subtree, if rounding up doesn’t violates 1.8 approximate ratio respect to the subtree, we choose a random node to round up.



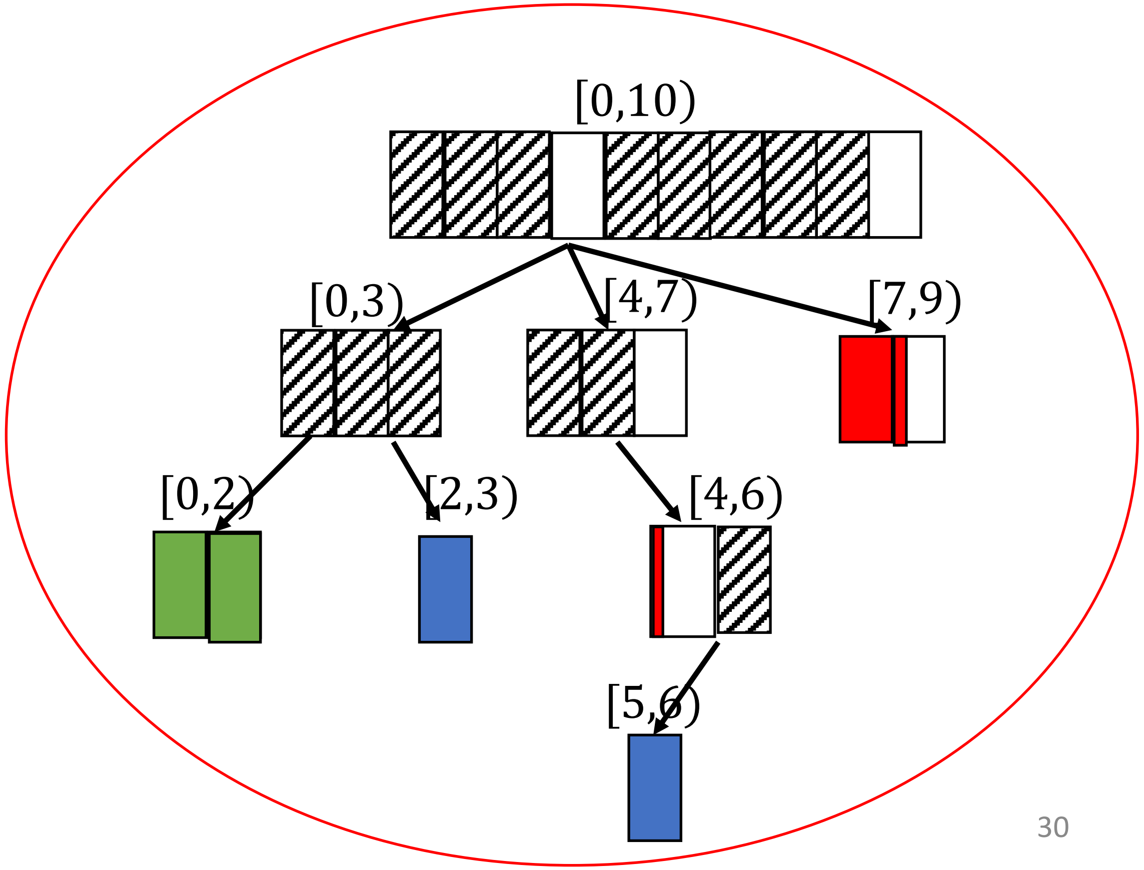
Linear programming: lp rounding example

- Blue node are integer time slot node.
- For any subtree, if rounding up doesn’t violates 1.8 approximate ratio respect to the subtree, we choose a random node to round up.



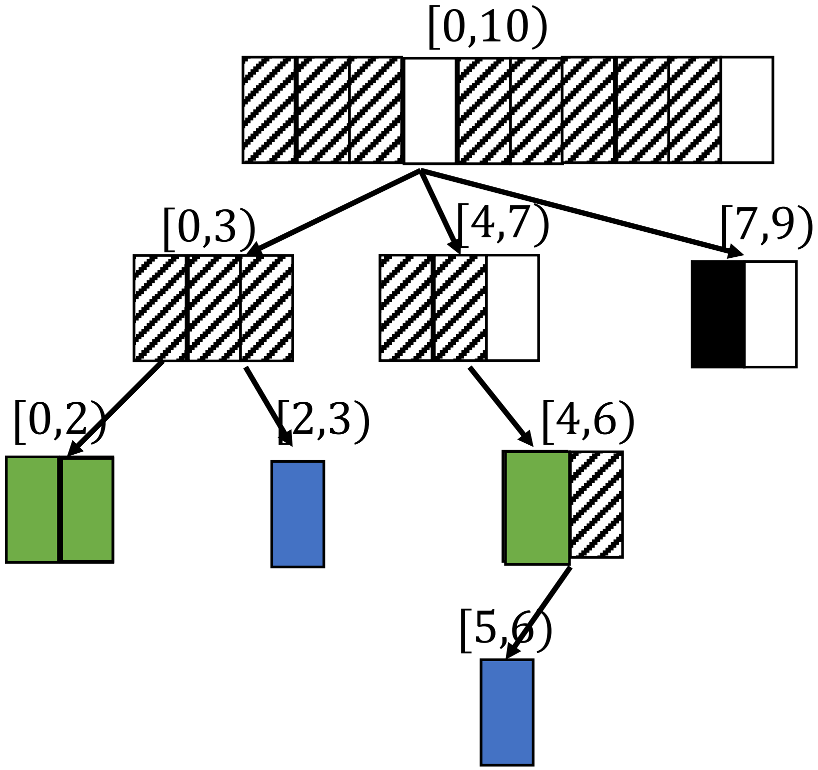
Linear programming: lp rounding example

- When we process the root node, we round another red node up.



Linear programming: lp rounding example

- Round up green node
- Round down black node



Outline

- Linear programming preprocessing
- Linear programming rounding algorithm
- The feasibility of our method

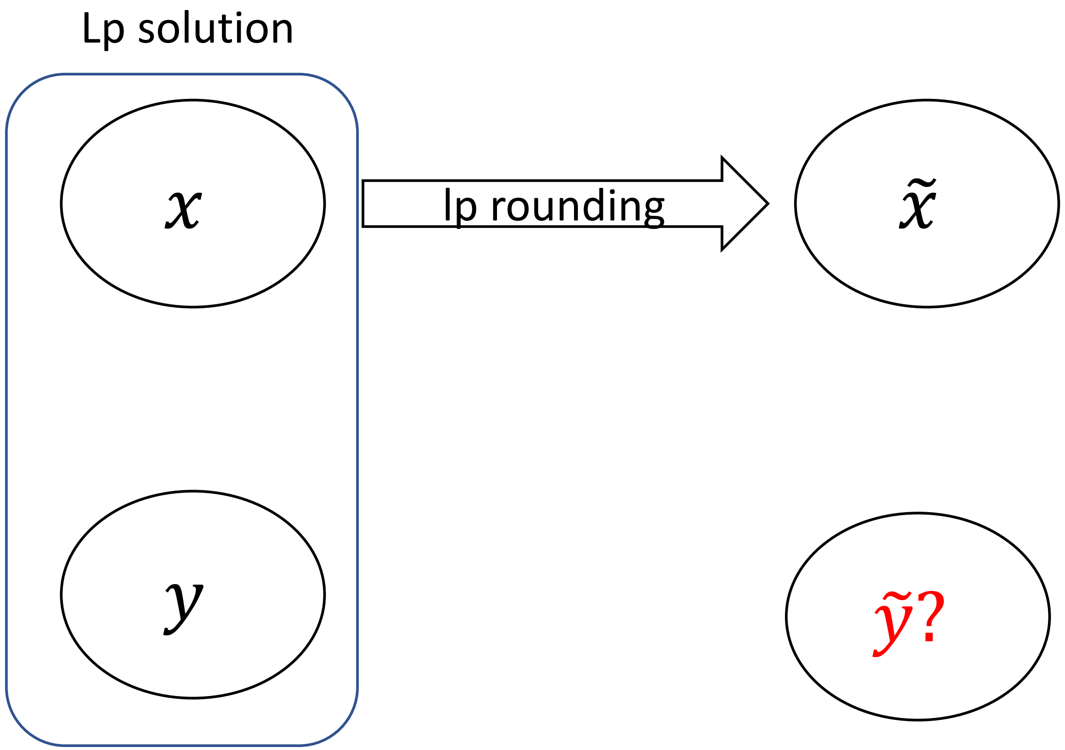
Feasibility of the lp rounding scheme

What we have:

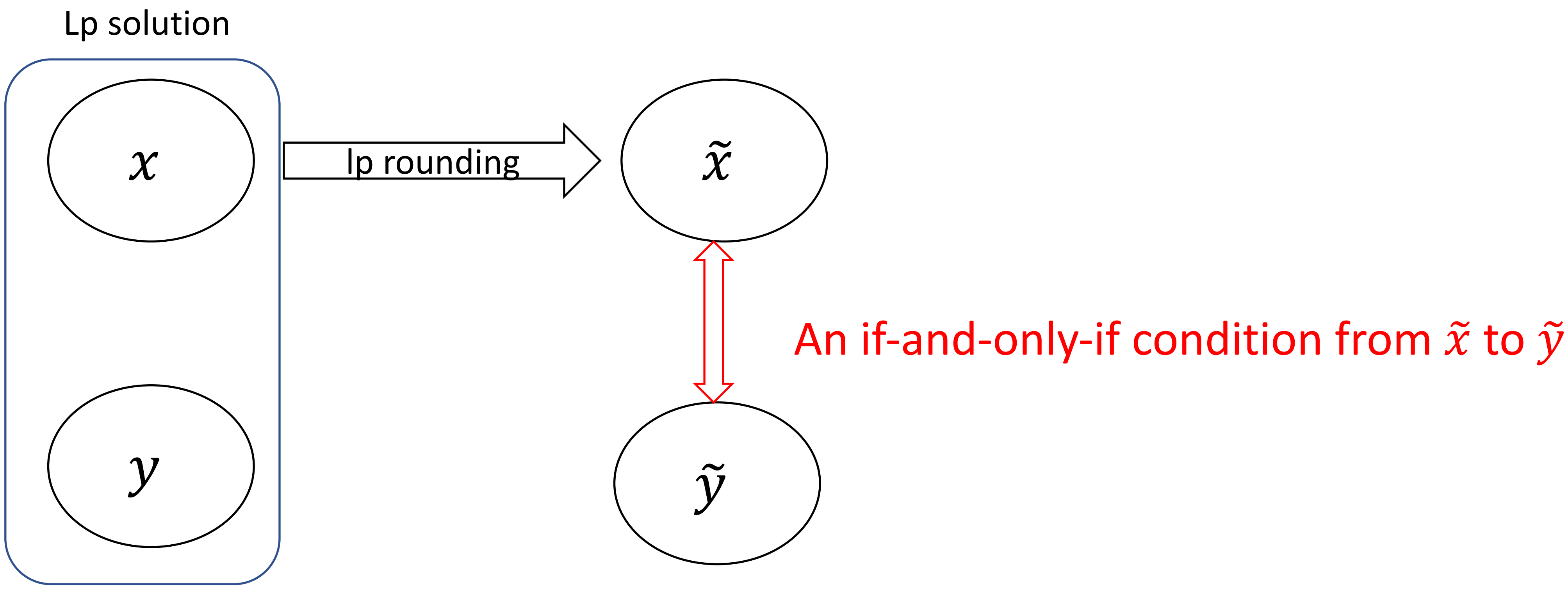
- The lp active time slot x
- The lp scheduling y
- The rounding active time slot $\tilde{x}$

We have to show

- The existence of scheduling $\tilde{y}$



Feasibility of the lp rounding scheme



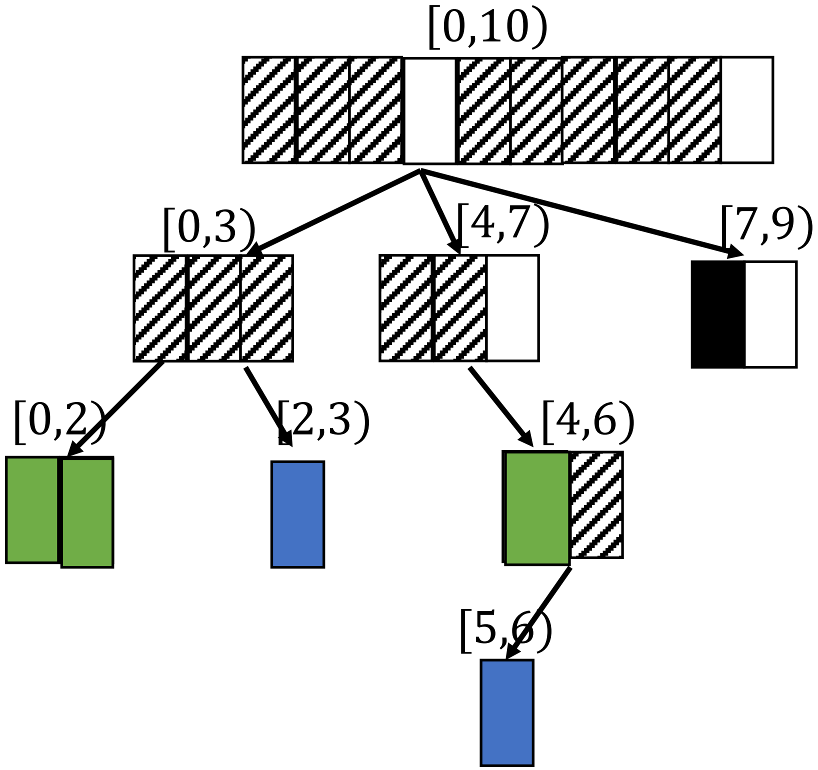
If-and-Only-If condition for feasibility

- Given a solution $\tilde{x}$
- Consider an arbitrary job subset $J' \subset J$, for any node i in the tree, we can schedule at most

$$\min(|J'(Anc(i))|, g) \cdot \tilde{x}(i)$$

job volume inside.

Jobs in J' which can be scheduled in node i



If-and-Only-If condition for feasibility

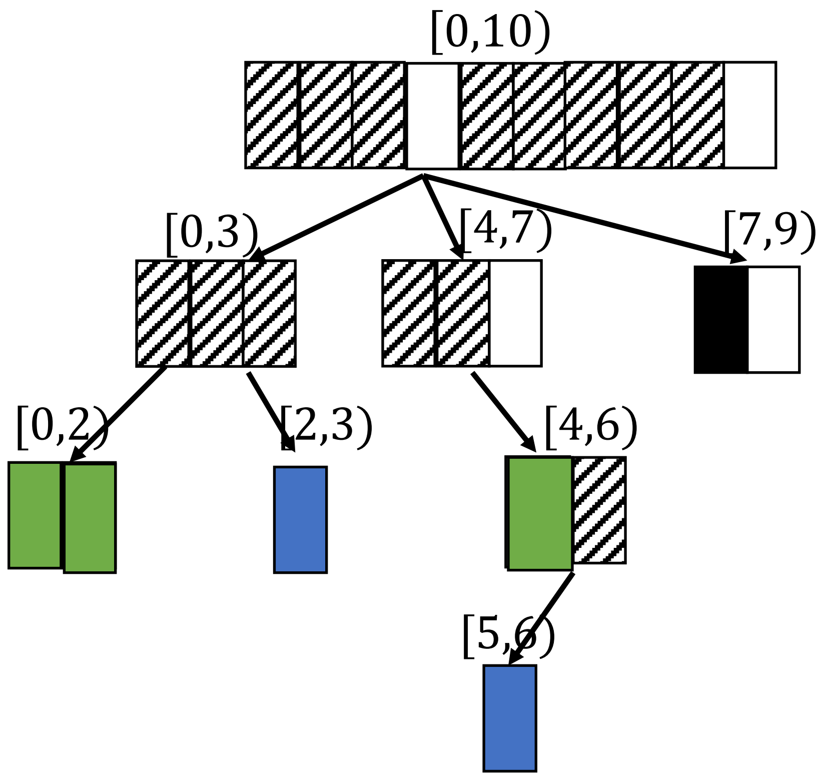
- Given a solution $\tilde{x}$
- Consider an arbitrary job subset $J' \subset J$, for any node i in the tree, we can schedule at most

$$\min(J'(Anc(i)), g) \cdot \tilde{x}(i)$$

job inside.

- In total, we can schedule at most

$$\sum \min(J'(Anc(i)), g) \cdot \tilde{x}(i)$$



If-and-Only-If condition for feasibility

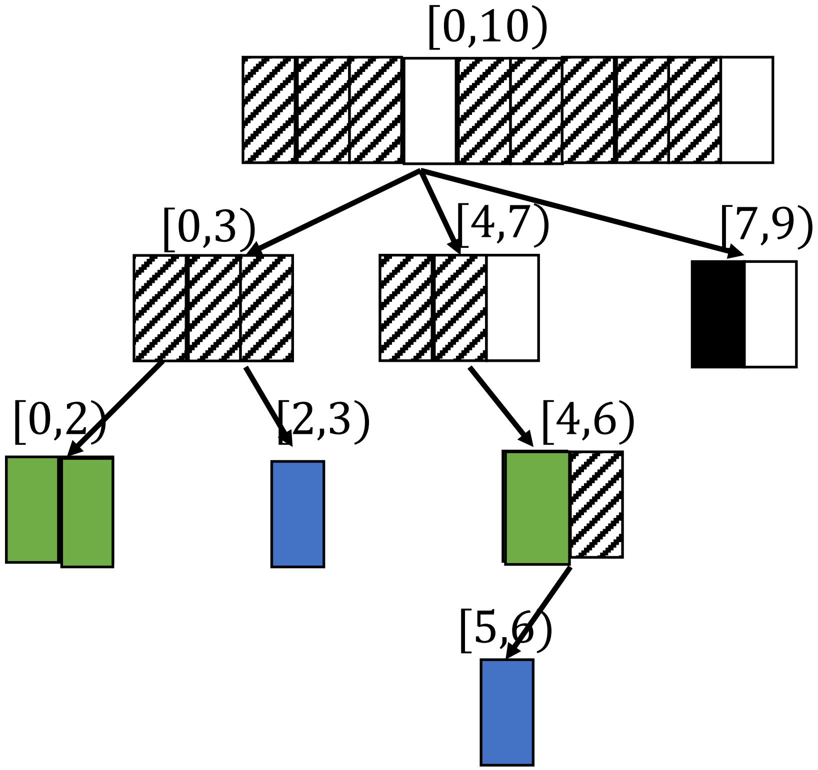
- Given a solution $\tilde{x}$
- Consider an arbitrary job subset $J' \subset J$, for any node i in the tree, we can schedule at most

$$\min(J'(Anc(i)), g) \cdot \tilde{x}(i)$$

job inside.

- In total, we can schedule at most

$$\sum \min(J'(Anc(i)), g) \cdot \tilde{x}(i) \ \geq p(J')$$



If-and-Only-If condition for feasibility

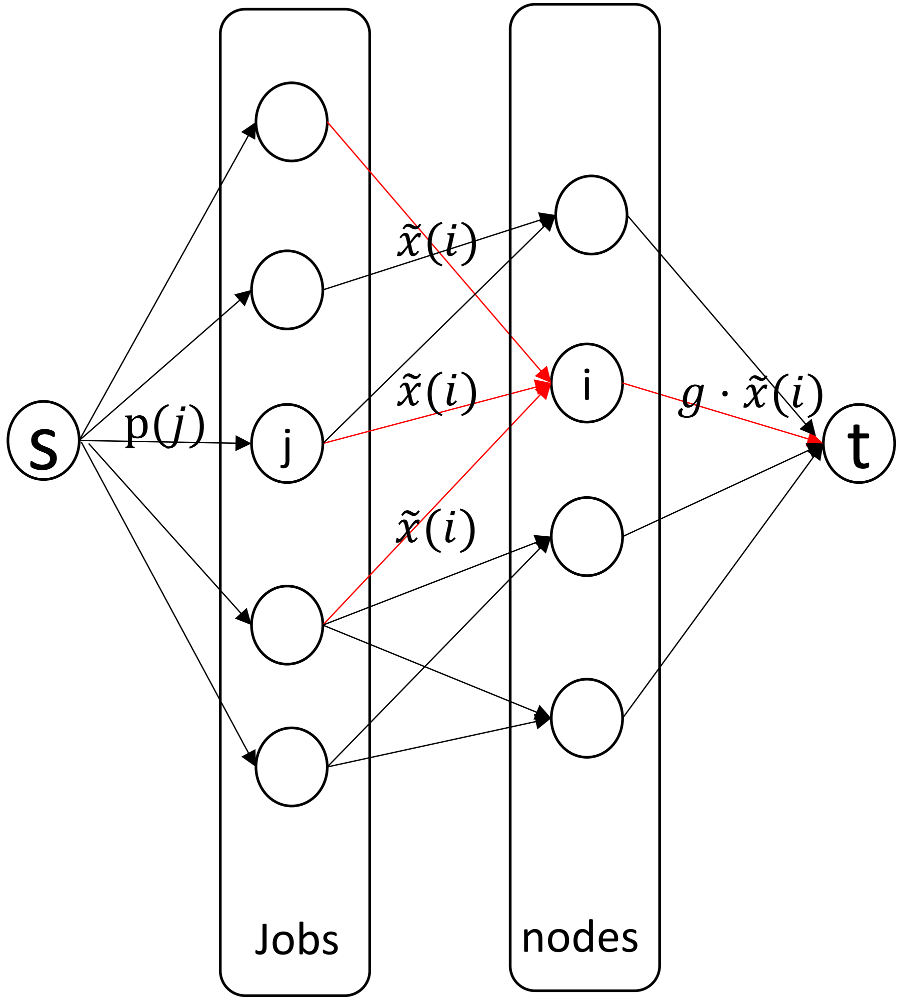
- For a job subset J', if

$$\sum \min(J'(Anc(i)), g) \cdot \tilde{x}(i) \geq p(J')$$

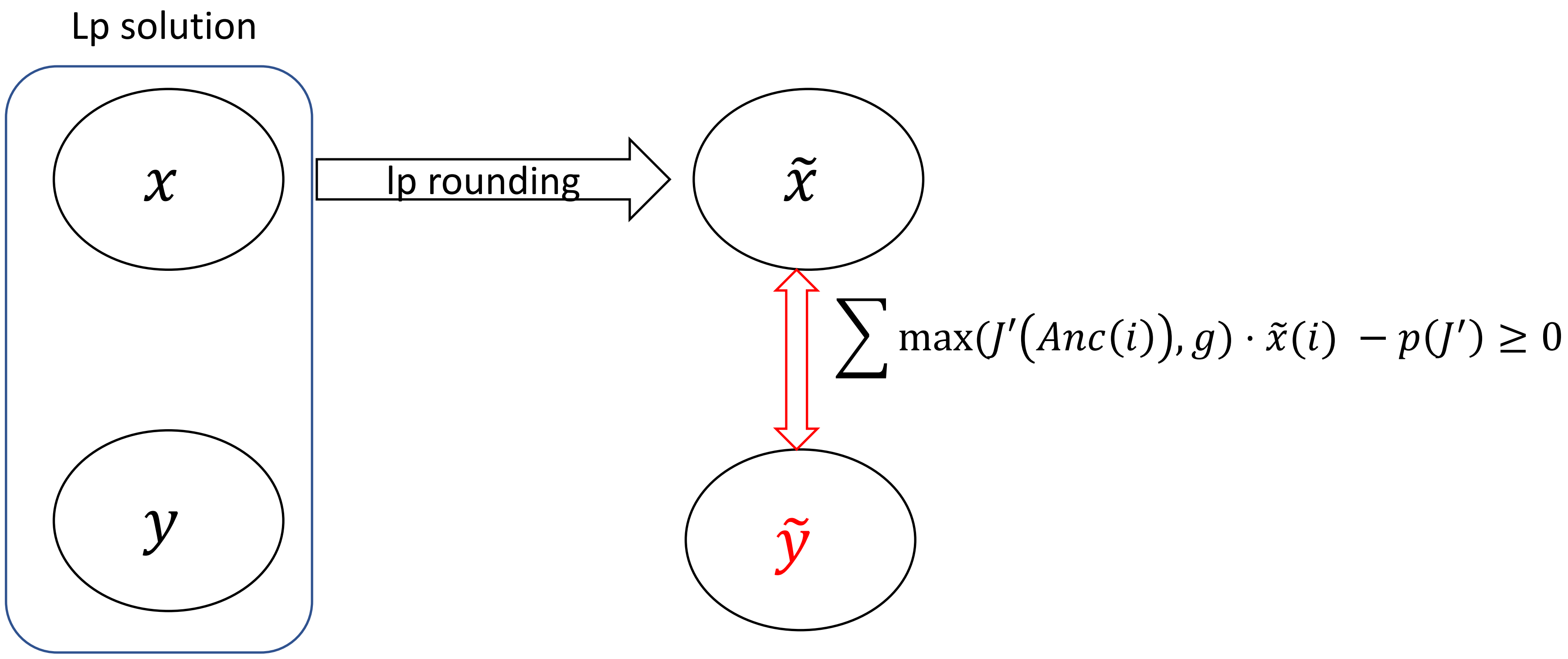
- Maximum flow minimum cut theorem
- The cut is at least

$$\sum \min(J'(Anc(i)), g) \cdot \tilde{x}(i)$$

- The maximum flow is at least $p(J')$



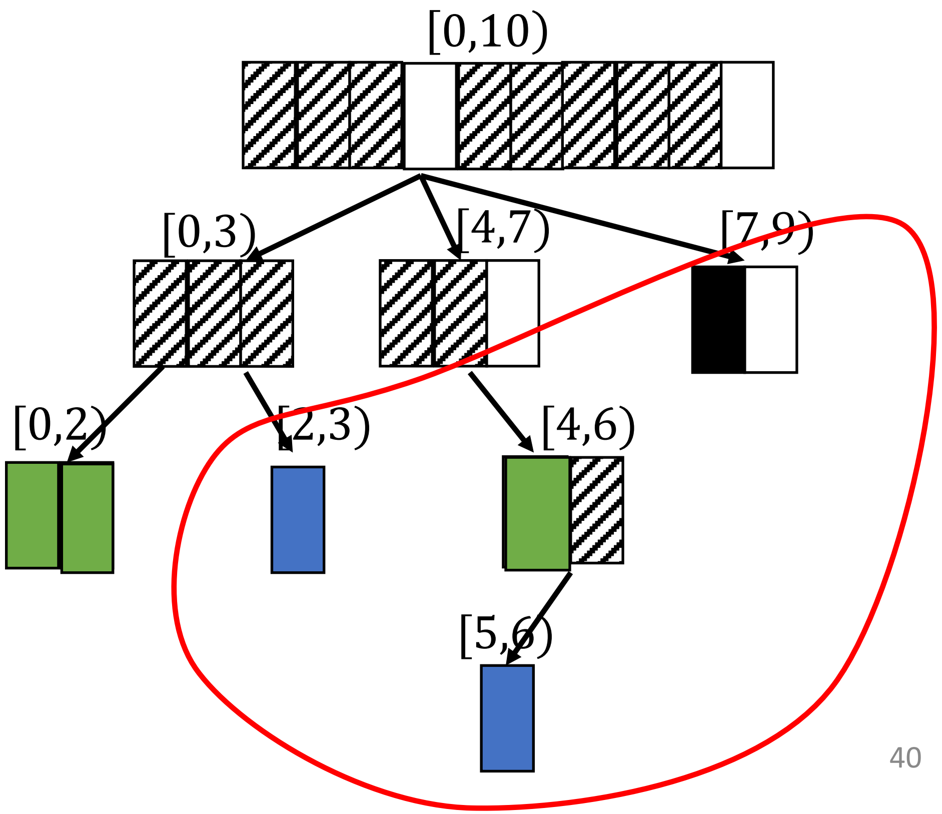
Feasibility of the lp rounding scheme



Feasibility

- We partition the subtrees into group of size at most 3 such that one rounding down node has two rounding up nodes.
- We show the partition always exists.

$$\sum \max(J'(Anc(i)), g) \cdot \tilde{x}(i) \ - p(J')$$



Nested Active-time problem: Summary

- There exists an 1.8 approximate algorithm for nested active time problem.

Q & A