Introduction to Optimisation:

Integer Programming

Lectures 10-11

Lecture notes by Dr. Julia Memar and Dr. Hanyu Gu and with an acknowledgement to Dr.FJ Hwang and Dr.Van Ha Do

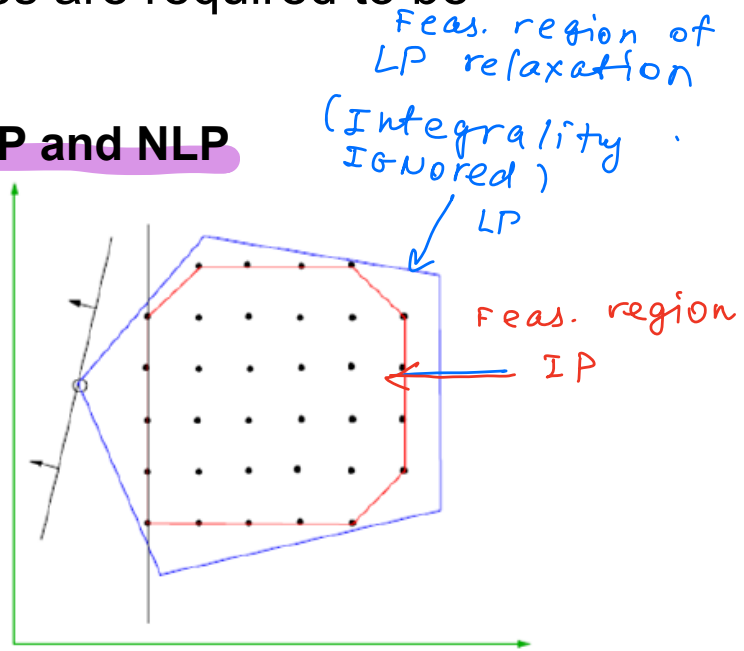
Introduction

- An integer program (IP) is a mathematical programming problem in which some or all of the variables are required to be integers.
- An IP is called
  - a pure IP if all variables are required to be integers,
  - a 0-1 IP or binary IP if all variables must be 0 or 1, and
  - a mixed IP (MIP) if some of the variables are required to be integers.
- There is no "nice" optimality conditions **as for LP and NLP**

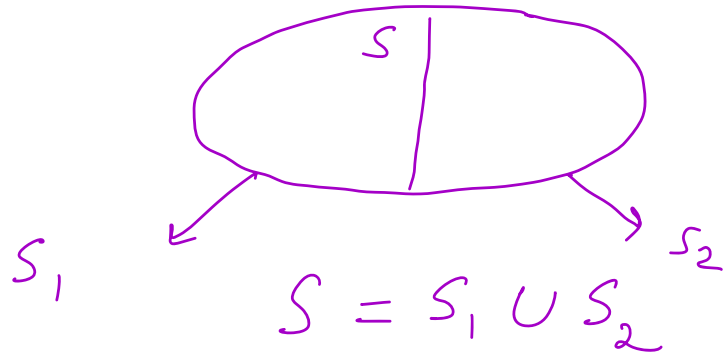
$$\min \quad \sum_{i}^{n} c_i x_i$$

$$s.t. \quad \sum_{j}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \cdots, m$$

$$x \quad \text{integer}$$



Branch-and-bound algorithm

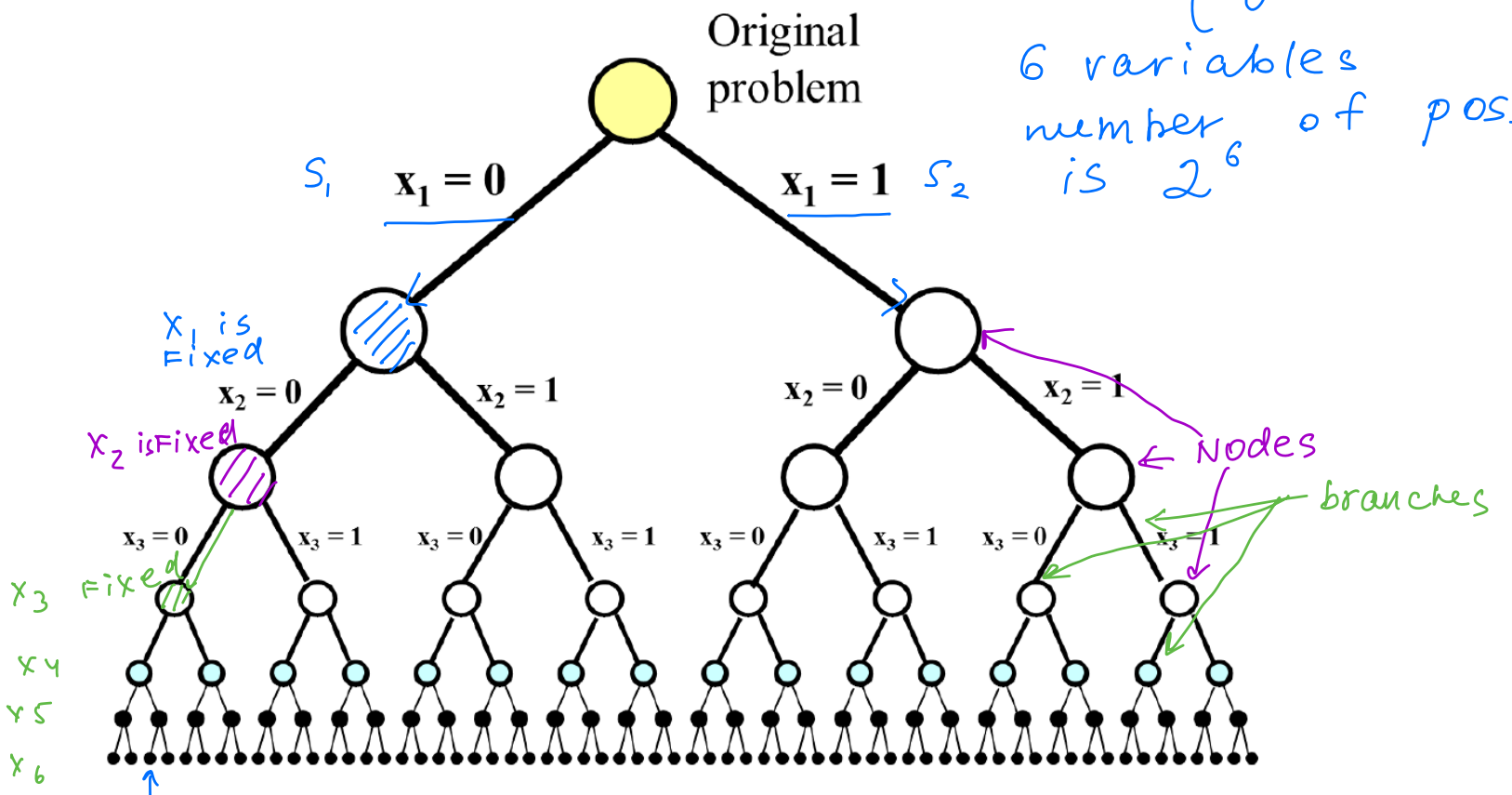


Branch-and-bound method:

- The feasible region S is replaced by smaller problems S_i- branching.
- Each S_i is a basis of another branching

$$x_i = \begin{cases} 1 \\ 0 \end{cases} \rightarrow \text{binary variable}$$

6 variables
number of poss. s-n is 2^6



Branch-and-bound algorithm

Pruning:

To avoid enumeration of all solutions, the search tree is ”pruned” and the following assumptions are made:

- there is an algorithm to calculate lower bound L_i on objective values of feasible solutions of a subproblem S_i
- the upper bound *U* of objective value on *S* can be obtained as an objective value on some solution

min problem

Assumed that

- LB_i can be obtained for each node i
- Then $LB_i \leq z^* \leq UB$ (optimal)

max problem

Assumed that

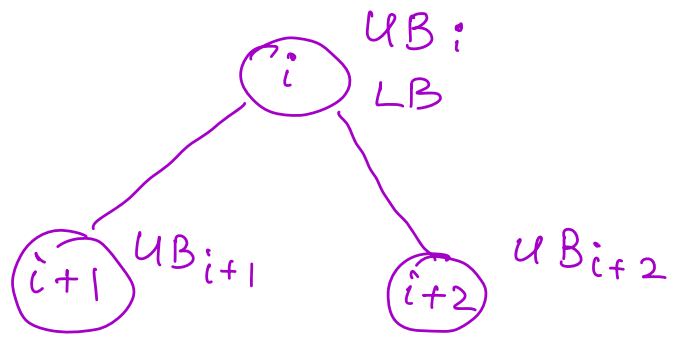
- UB_i can be obtained for each node
- LB - best feas. s-n so for

$LB \leq w^* \leq UB_i$

some feas. s-n best so far

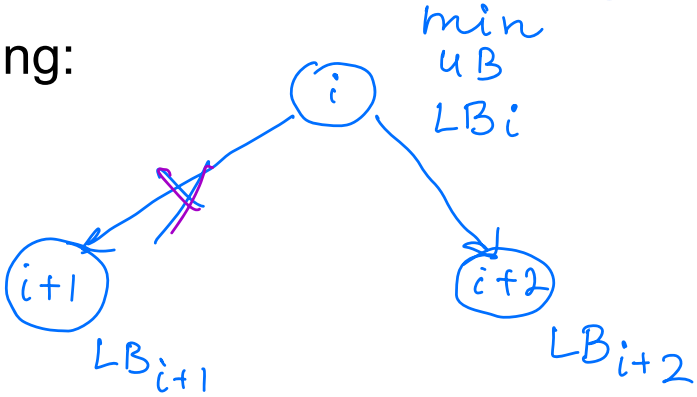
Branch-and-bound algorithm

max problem



if $UB_{i+1} \leq LB \rightarrow$ STOP, no further branching

Pruning:



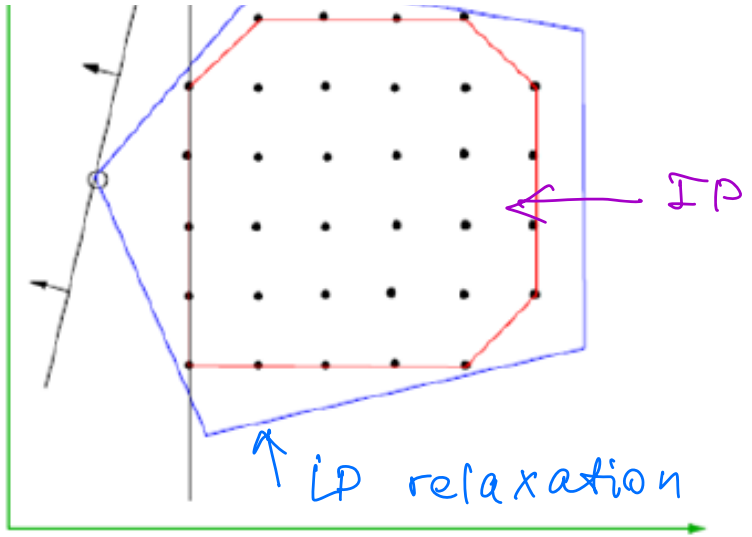
IF $LB_{i+1} \geq UB \rightarrow$ as any further branching on $(i+1)^{st}$ node can not produce OF value $< LB_{i+1} \rightarrow$ do not Branch further, prune

- If for a subproblem S_i

$$L_i \geq U$$

then S_i can/can not improve the objective value on S, hence

- LP relaxation – the integer requirement is ignored/relaxed



max problem $z^{*IP} \leq z^{*LP}$ can be used as UB

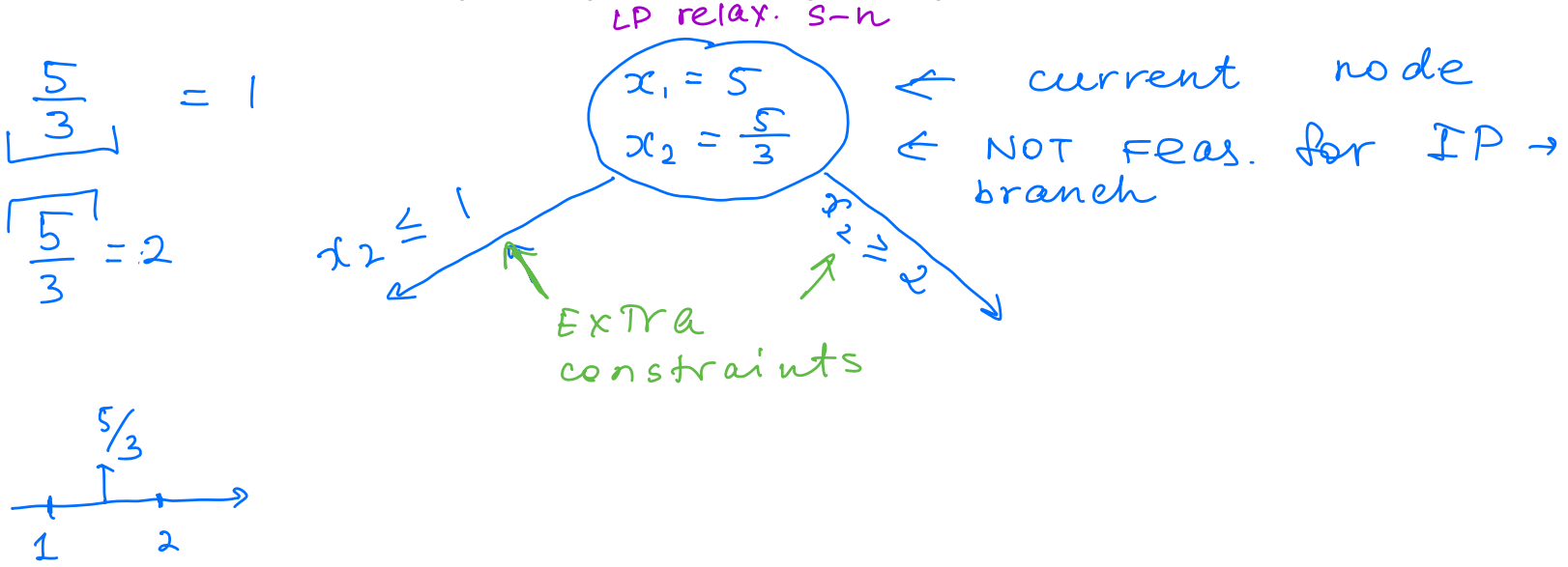
min problem $z^{*IP} \geq z^{*LP}$ can be used as LB

Branch-and-bound algorithm

Branching:

- each LP relaxation will represent a node on the solution tree;
- solve the LP relaxation corresponding to the current node.
  - If in resultant solution $x' = (x_1', x_2', \ldots, x_n')$ component x_j' is not integer, then branch on this node by adding constraint

    either $x_j \geq \lfloor x_j' \rfloor + 1$ or $x_j \leq \lfloor x_j' \rfloor$;



Branch-and-bound algorithm

Bounding:

- If for the current node the LP relaxation provides solution with OF value not better than the incumbent solution, no further branching required, and the node is
  - Optimality gap is useful in practice for large problems
- If the node's LP is infeasible, the node is

Rules to travel the solution tree:

- LIFO rule : Last-In-First- Out (LIFO) rule (depth-first)
- The Jumptracking rule: solves all the problems created by a branching and then branches on the node with the best OF-value (best-first).
- Many other rules for variable order and value order
  - Active research area to make use of machine learning

Example

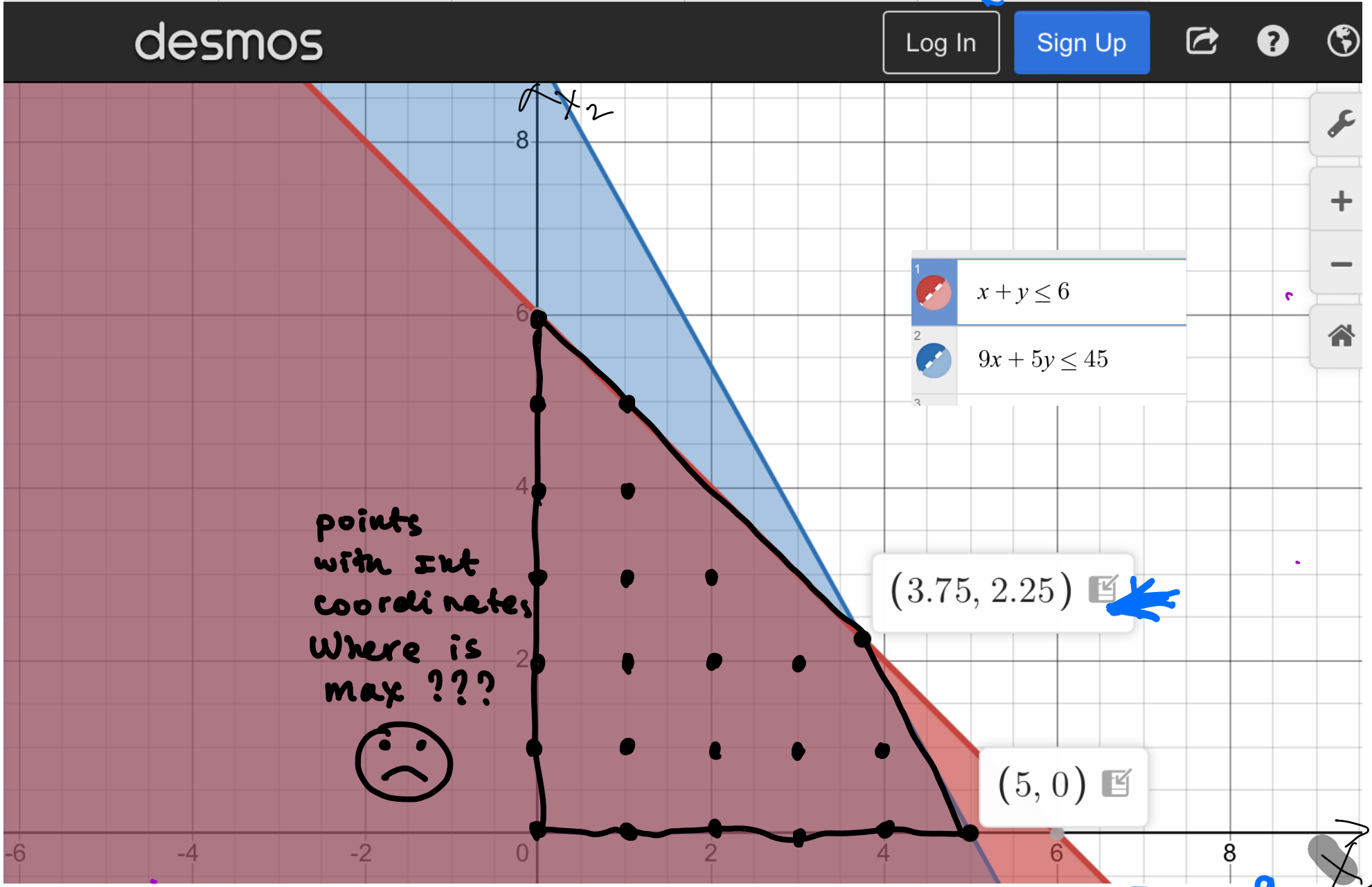
- $max\ z = 8x_1 + 5x_2$ *(1)*

  $s.t.$ $x_1 + x_2 \leq 6$

  $9x_1 + 5x_2 \leq 45$

  $x_1, x_2 \geq 0$, integer

- Solve the problem graphically:



For LP relaxation $\max z = z\left(\frac{15}{4}, \frac{9}{4}\right)$

$z^* = 41.25$

Example

- $max\ z = 8x_1 + 5x_2$ (*)

  $s.t.$ $x_1 + x_2 \leq 6$

  $9x_1 + 5x_2 \leq 45$

  $x_1, x_2 \geq 0$, integer

Solve the problem with branch-n-bound algorithm:

Node 1 (Subproblem 1): Solve (*) ignoring integer constraint:

- $max\ z = 8x_1 + 5x_2$ → From the graph. s-n

  $s.t.$ $x_1 + x_2 \leq 6$

  $9x_1 + 5x_2 \leq 45$

  $x_1, x_2 \geq 0$, ~~integer~~

(*) $z^* = 41.25$ → UB_1

$x^{*LP} = \begin{pmatrix} \frac{15}{4} \\ \frac{9}{4} \end{pmatrix}$ not feas. for IP

Subproblem 2

$x_1 \geq \lceil \frac{15}{4} \rceil = 4$

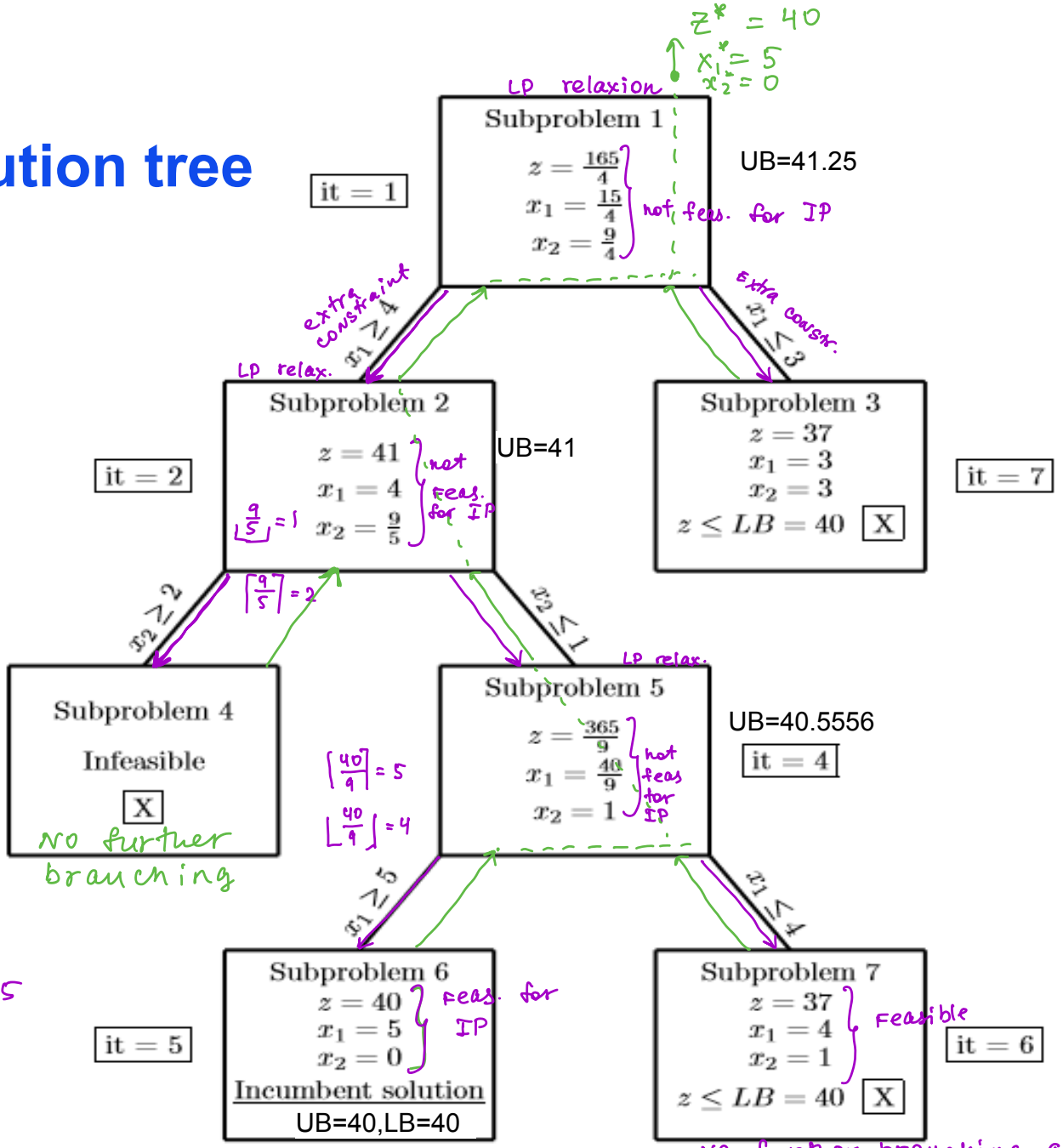
add the constraint to original (*) constraints

Subproblem 3

$x_1 \leq \lfloor \frac{15}{4} \rfloor = 3$

Example – solution tree

$$\begin{aligned} max\ z &= 8x_1 + 5x_2 \\ s.t. \quad & x_1 + x_2 \le 6 \\ & 9x_1 + 5x_2 \le 45 \\ & x_1, x_2 \ge 0, \text{integer} \end{aligned}$$



Subproblem 4:
extra constraints:
$x_1 \ge 4$ } contradict
$x_2 \ge 2$ } + constraint 2*

i) $x_1 + x_2 \le 6$
ii) $9x_1 + 5x_2 \le 45$ } original

1) $4 + 2 \ge 6 \rightarrow x_1 + x_2 = 6$
$x_1 + x_2 \ge 6$

2) $9x_1 + 5x_2 \ge 36 + 10 = 46 > 45$

No further branching, as UB = LB

- Solution of LP relax. gives UB
- As the LP relax-n S-n is integer → gives LB
- On this node UB = LB → no further branchin

on this node

- Depth first → 7 nodes
- Best first → 3 nodes

Combinatorial Optimisation problems

- Solution has a combinatorial structure, e.g., an object on a graph; the number of solutions increase exponentially fast
- Can be solved as Mixed Integer Program(MIP), but not efficient for large problems; active research area:
  - Knapsack problem
  - Assignment problem
  - Travelling salesman problem

5 Minimization of the maximum weighted lateness

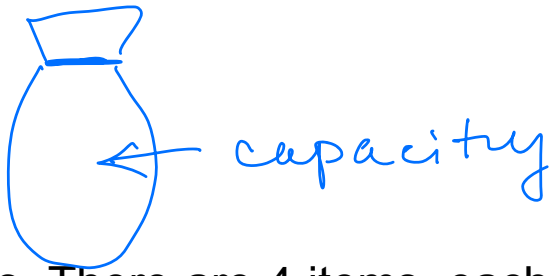
Consider the $P|prec, p_j = 1|F$ scheduling problem with the objective function

$$F(x_1, \ldots, x_n) = \max_{1 \le j \le n} w_j(x_j - d_j), \tag{6}$$

where integer d_j is a due date for completion of task j (the desired time by which task j needs to be completed) and w_j is a positive weight. Given this interpretation and the notation introduced in Sect. 3.1, the considered problem requires to minimize the maximum weighted lateness for the set of tasks with zero release times which are to be scheduled on m parallel identical machines without any restriction on the machines' availability. An approach similar to one discussed below was briefly outlined in Zinder (2007).

$L_i = w_j x_j - w_j d_j \le F$

Knapsack problem



There is a knapsack with a capacity of 14 units. There are 4 items, each of which retains a size and a value, e.g. item 1 has a size of 5 units and a value of 16 dollars, item 2 has a size of 7 units and a value of 22 dollars, item 3 has a size of 4 units and a value of 12 dollars, and item 4 has a size of 3 units and a value of 8 dollars. The objective is to decide which items shall be packed in the knapsack so as to maximise the total value of items in the knapsack. Then the knapsack problem can be formulated as follows:

$x_i = \begin{cases} 1, & \text{if item } i \text{ is "in"} \\ 0 & \text{otherwise} \end{cases}$

	1	2	3	4
value	16	22	12	8
size	5	7	4	3
ratio	3.2	$\frac{22}{7}$ $3\frac{1}{3}$	3	$\frac{8}{3}$ $2\frac{2}{3}$

- $max\ z = 16x_1 + 22x_2 + 12x_3 + 8x_4$

$s.t. \quad 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$

$x_1, x_2, x_3, x_4 \geq 0$ and binary

List 1, 2, 3, 4

	1	2	3	4
value	16	22	12	8
size	5	7	4	3
ratio	3.2	$\frac{22}{7}$ $3\frac{1}{3}$	3	$\frac{8}{3}$ $2\frac{2}{3}$

Knapsack problem

There is no need to use Simplex algorithm ☺

- Finding ratio $\frac{\text{value}}{\text{size}}$ and placing first the item with highest ratio

then with 2nd best ratio till only part fits

Solution: List : 1, 2, 3, 4

- Branching:

Subproblem 2 or Subproblem 3

$max\ z = 16x_1 + 22x_2 + 8x_4 +$

$s.t.\ \ 5x_1 + 7x_2 + 3x_4 \leq 14 -$

$x_1, x_2, x_4\ are\ binary\ variables$

Subproblem 1:

List 1, 2, 3, 4

IN	Capacity
$x_1 = 1$	5
$x_2 = 1$	7
$x_3 = \frac{2}{4}$ partial	2
$x_4 = 0$	14

$z^* = 16 + 22 + \frac{1}{2}12 + 8 \times 0 = 44$ → upper bounds

List: 1, 2, 3, 4

as we relaxed "binary" req-t for x_3

➤ Branching: on x_3 → Fix its value

Subproblem 2 ($x_3 = 1$) or Subproblem 3 ($x_3 = 0$)

Subproblem 2

$$\max z = 16x_1 + 22x_2 + 8x_4 + 12 \times 1$$

$$s.t.\ 5x_1 + 7x_2 + 3x_4 \le 14 - 4 \text{ (size of } x_3\text{)}$$

x_1, x_2, x_4 are binary variables

IN	Capacity
$x_3 = 1$	4
$x_1 = 1$	5
$x_2 = \frac{5}{7}$	5
$x_4 = 0$	12

$z = 43\frac{5}{7}$

need 7 units, have only 5

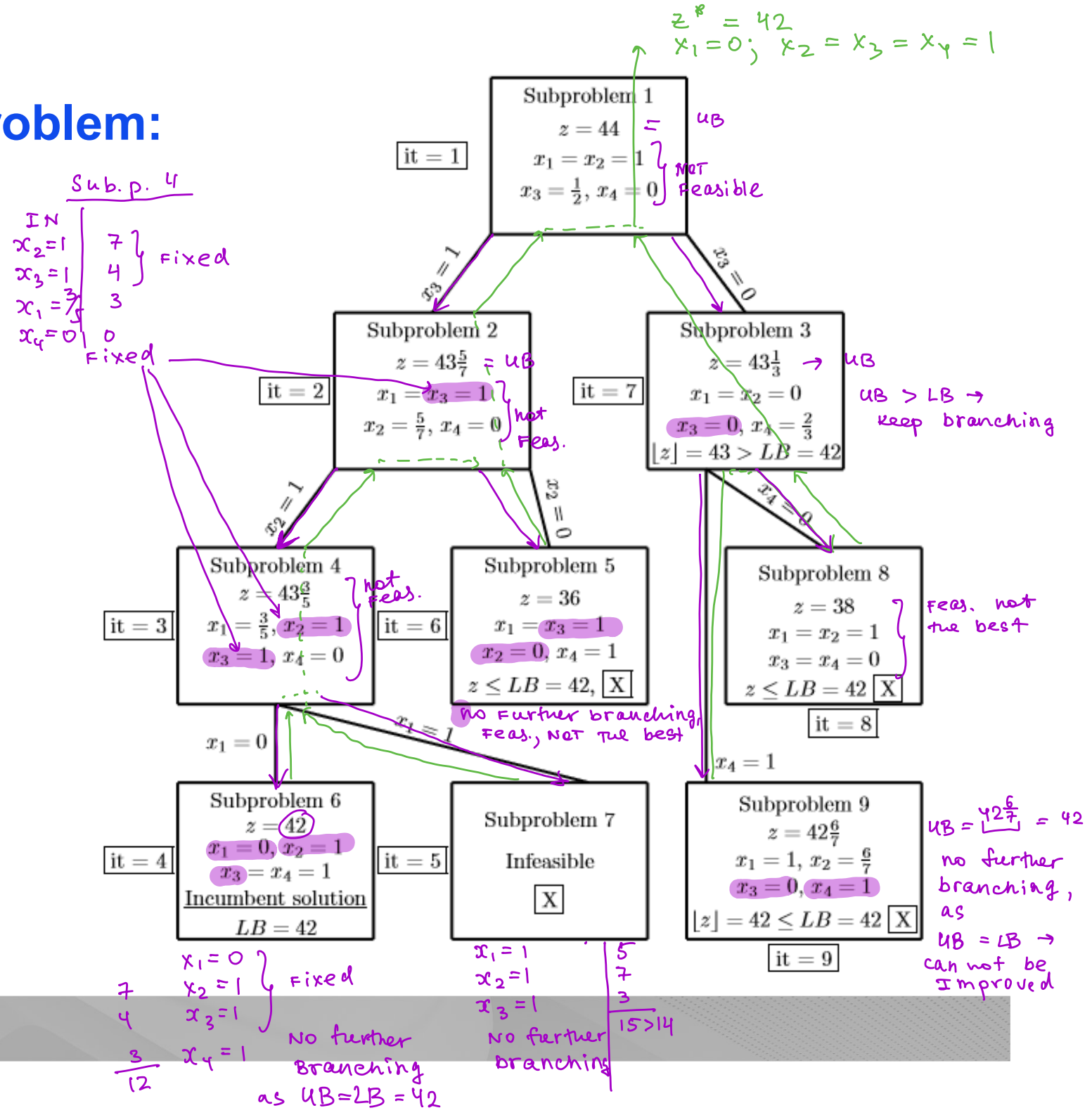
Subproblem 3

$$\max z = 16x_1 + 22x_2 + 8x_4$$

$$s.t.\ 5x_1 + 7x_2 + 3x_4 \le 14$$

IN	Capacity
$x_1 = 1$	5
$x_2 = 1$	7
$x_3 = 0$	~~0~~
$x_4 = \frac{2}{3}$	2
	12

Knapsack problem: solution tree



Assignment problem - example

A manufacturing factory has four machines which can process four sorts of tasks to be completed. The time required to set up each machine for completing each job is shown in the table below. Any machine which has been assigned to process a task cannot be reassigned to another task. The factory manager aims to minimise the total setup time needed to complete four jobs with these four machines.

	Time (Hours)			
Machine	Task 1	Task 2	Task 3	Task 4
1	14	5	8	7
2	2	12	6	5
3	7	8	3	9
4	2	4	6	10

How many possible assignments are there? $4! = 1 \times 2 \times 3 \times 4$

In general

$n!$

Assignment problem - example

Formulation: Let $x_{ij} = \begin{cases} 1, & \text{if machine } i \text{ does job } j \\ 0 & \text{otherwise} \end{cases}$

c_{ij} - set up time on machine i for job j

$$\min z = \sum_{i=1}^{4} \sum_{j=1}^{4} c_{ij} x_{ij}$$

s.t.

s.t. For each machine assign one job only

for $i = 1..4$ $\sum_{j=1}^{4} x_{ij} = 1$ • (*)

each task can be assegned to one machine only (**)

for $j = 1..4$ $\sum_{i=1}^{4} x_{ij} = 1$

Assignment problem - example

- *Totally unimodular matrix:* matrix A is totally unimodular (TU) if every square submatrix of A has determinant −1, 0 or +1

matrix of constraints (*)

A =

(*)

(**)

	x_11	x_12	x_13	x_14	x_21	x_22	x_23	x_24	x_31	x_32	x_33	x_34	x_41	x_42	x_43	x_44
i=1	1	1	1	1												
i=2					1	1	1	1								
i=3									1	1	1	1				
i=4													1	1	1	1
j=1	1				1				1				1			
j=2		1				1				1				1		
j=3			1				1				1				1	
j=4				1				1				1				1

$i = 1$; $x_{11} + x_{12} + x_{13} + x_{14} = 1$

$j = 1$ $x_{11} + x_{21} + x_{31} + x_{41} = 1$

Assignment problem - example

- Extreme points of LP are integral if A is *unimodular* and b is integral
  - Network flow problem, shortest path
  - Simplex algorithm can find optimal solution for the assignment problem, i.e., no need for B&B!
  - Simplex algorithm is/is not polynomial…

  - Can be solved by “Hungarian method” in polynomial time

$O(n^3)$

Hungarian method

Theorem: If a number is added to or subtracted from all of the entries of any one row or column of a cost matrix, then an optimal assignment for the resulting cost matrix is also an optimal assignment for the original cost matrix.

What is cost matrix?

14	5	8	7
2	12	6	5
7	8	3	9
2	4	6	10

matrix m x m

Hungarian method

1. Find the minimum element in each row of the cost matrix. Construct a new matrix by subtracting from each cost the minimum cost in its row. For this new matrix, find the minimum cost in each column. Construct another new matrix (called the **reduced cost matrix**) by subtracting from each cost the minimum cost in its column.

2. Draw the minimum number of lines (horizontal, vertical, or both) that are needed to cover all the zeros in the reduced cost matrix. If m lines are required, then an optimal solution is available among the covered zeros in the matrix. If fewer than m lines are needed, then go to Step 3.

3. Among all elements in the reduced cost matrix that are un-covered by the lines drawn in Step 2, find the smallest nonzero element, say c_0. Construct a new reduced cost matrix by
   - subtracting c_0 from each uncovered element of the reduced cost matrix;
   - adding c_0 to each element of the reduced cost matrix that is covered by two lines.

4. Go to step 2

$m = 4$

Hungarian method

14	5	8	7
2	12	6	5
7	8	3	9
2	4	6	10

Row min

14	5	8	7	5
2	12	6	5	2
7	8	3	9	3
2	4	6	10	2

9	0	3	2
0	10	4	3
4	5	0	6
0	2	4	8
0	0	0	2

Column min

Hungarian method

reduced cost-matrix

Row min

9	0	3	0
0	10	4	1
4	5	0	4
0	2	4	6

3 lines < 4

↓

min NOT-covered (1)

Tasks / machines	1	2	3	4
1	10	0	4	0
2	0	9	4	0
3	4	4	0	3
4	0	1	4	5

4 lines are required to cover all zeros

Column min

Among all elements in the reduced cost matrix that are un-covered by the lines drawn in Step 2, find the smallest nonzero element, say c_0. Construct a new reduced cost matrix by

- subtracting c_0 from each uncovered element of the reduced cost matrix;
- adding c_0 to each element of the reduced cost matrix that is covered by two lines.

$x_{12} = 1$ $x_{33} = 1$ $x_{24} = 1$ $x_{41} = 1$; all other $x_{ij} = 0$

Hungarian method

reduced cost-matrix

Row min

9	0	3	0
0	10	4	1
4	5	0	4
0	2	4	6

3 lines < 4

↓

min NOT-covered (1)

9 10	0	3 4	0
-1 0	9	3 4	0
3 4	4	-1 0	3
-1 0	1	3 4	5

another way to do step 3 → same result

Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row, and then add it to each covered column. Return to Step 3.