Operations Research (Master's Degree Course)

6. Integer Linear Programming

Silvano Martello
DEI "Guglielmo Marconi", Università di Bologna, Italy

Integer Linear Programming (ILP)

- In many real-world situations the variables cannot assume fractional values: assignment problem (candidates to be assigned to duties), transportation problems ($x_j =$ number of vehicles to be used on route j), ...
- $A(m \times n)$, $b(m)$, $c(n)$ integer (like for LP); **ILP in standard form**:

$$\begin{aligned} \min\ & c'x \\ Ax &= b \\ x &\geq 0 \\ x & \text{ integer} \end{aligned}$$

- Canonical form, general form, transformations: like for LP (integer slack and surplus variables).
- By removing the integrality constraint we obtain the **continuous relaxation of the ILP**.
- Let z(ILP) and z(LP) be the solution values of an ILP and of its continuous relaxation:
  - **Property** z(LP) $\leq z$(ILP)

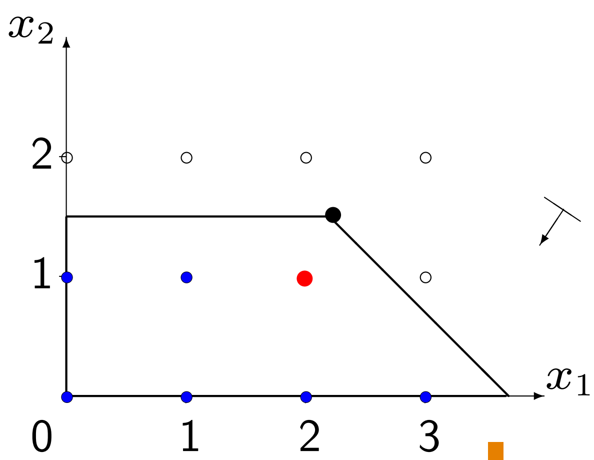
    Proof We look for a minimum in a larger feasible set. $\square$
  - We say that z(LP) is a **lower bound** on z(ILP).
  - $\Rightarrow$ **If z(LP) corresponds to an integer point $x \in R^n$ then x is optimal for the ILP**.

Geometrical interpretation

- The constraints

$$\begin{aligned} Ax &\geq b \\ x &\geq 0 \end{aligned}$$

define a polyhedron (like in LP):



- The integrality constraint x integer imposes that the **feasible region** only consists of the points • with integer coordinates within the polyhedron.
- The **optimal solution** • is the best among such points.
- By removing the integrality constraint (**Continuous Relaxation**), the LP provides a **fractional** solution (•) and a lower bound on the optimal ILP solution.
- Could we solve ILP by rounding the fractional solution provided by the LP?

A naïve Algorithm for ILP

- **procedure LIGABUE**:

  begin

  find the optimal solution x to the continuous relaxation LP of the ILP;

  if LP is impossible **then** ILP is impossible (**comment:** certainly true)

  else

  if LP is unbounded **then** ILP is unbounded (**comment:** "normally" true)

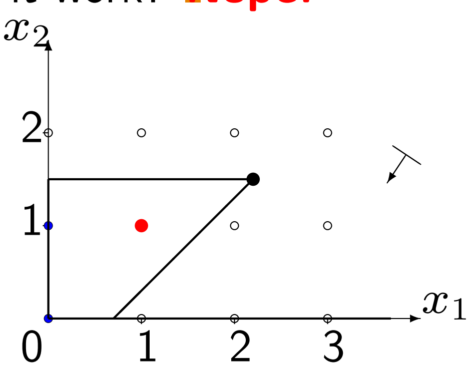
  else

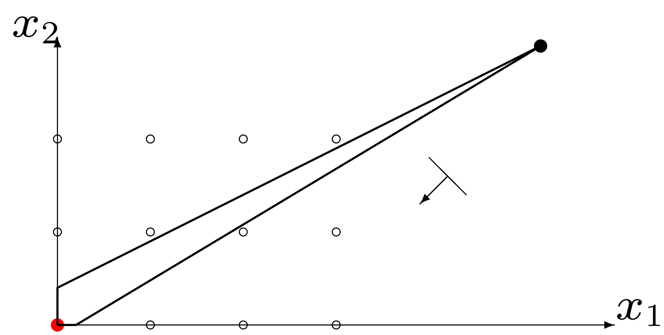
  if x is integer **then** $x^* := x$ (**comment:** certainly true)

  else round each fractional x_j to its closest integer

  end.

- Does it work? **Nope!**





- rounded points can all be unfeasible;
- integer and continuous solution can be very "far" from each other.
- Are there cases where it works, i.e., the LP relaxation always has an integer solution?

Unimodularity

- An integer square matrix B is **unimodular (UM)** if $\det(B) = \pm 1$.
- An integer rectangular $m \times n$ matrix A is **totally unimodular (TUM)** if every non-singular square submatrix of A is UM.
- **Property 1** *If A is TUM then the vertices of $\{x : Ax = b, x \geq 0\}$ are integer $\forall$ integer b.*

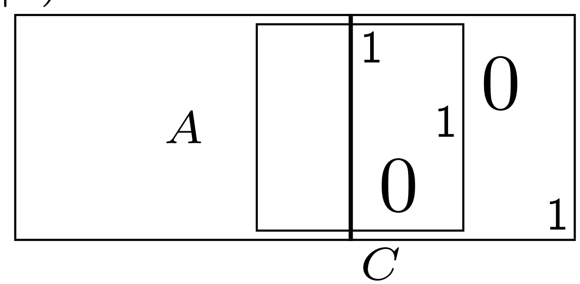
  Proof B = matrix corresponding to a base of A. $\Rightarrow$ basic solution: $x_\beta = B^{-1}b = \dfrac{B^a}{\det(B)}b$,

  with B^a = adjoint of B ($b^a_{ij} = (-1)^{i+j} \cdot (j,i)$-minor).
  A is TUM $\Rightarrow$ B is UM $\Rightarrow$ x is integer. $\square$

- **Property 2** *If A is TUM then the vertices of $\{x : Ax \leq b, x \geq 0\}$ are integer $\forall$ integer b.*

  Proof It is enough to show that if A is TUM then $(A|I)$ is TUM.

  Let C be a square non-singular submatrix of $(A|I)$:



  Permute the rows of C so that $\widetilde{C} = \left(\begin{array}{c|c} B & 0 \\ \hline D & I \end{array}\right)$:

  $\det(\widetilde{C}) = \det(B) \Rightarrow \det(C) = \pm\det(\widetilde{C}) = \pm 1.$ $\square$

- Hence: **if A is TUM then the simplex algorithm solves ILP.**
- **Sufficient conditions** exist to check if a matrix is TUM.

General methods for ILP

Two main methods: **cutting planes** and **branch-and-bound** (combined in **branch-and-cut**).

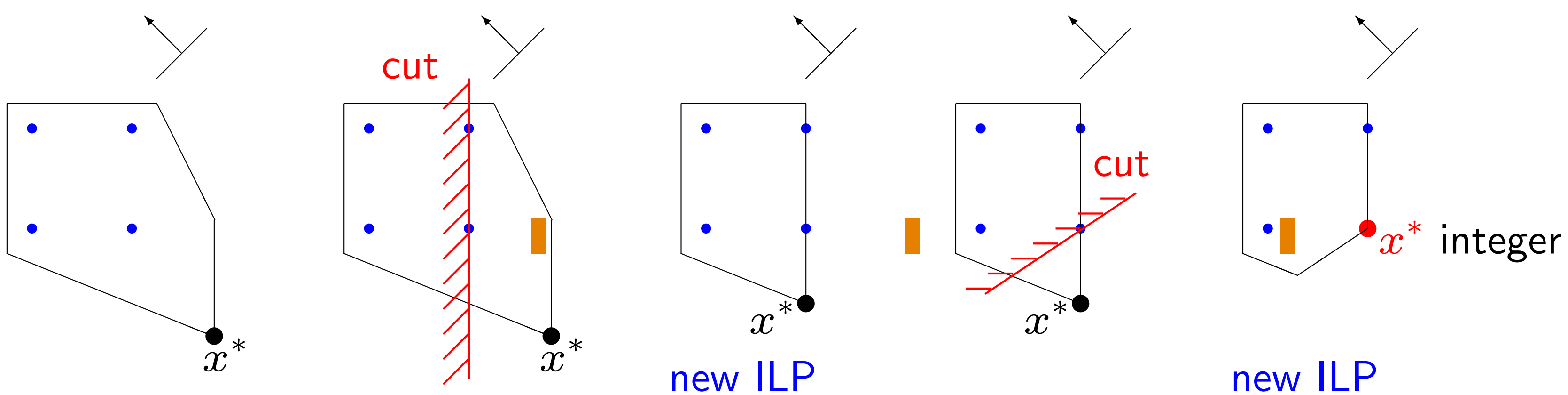
Cutting plane algorithms

Basic scheme (suppose the continuous relaxation is bounded and non-empty):

1. solve the continuous relaxation, and let x^* be the optimal solution;
2. if x^* is an integer point, then the problem is solved. Otherwise
3. **add** to the ILP a **linear constraint (cut)** which:

   (i) eliminates a part of the feasible region **containing x^*, but**

   (ii) does not eliminate any feasible integer solution;
4. solve the continuous relaxation of the new ILP ($\rightarrow$ new x^*) and go to 2.



Cutting plane algorithms (cont'd): Gomory cuts (1958)

- $\forall\ y \in R^1$, the **integer part** of y is $\lfloor y \rfloor = \max$ integer $q : q \leq y$.
- Y = final LP tableau; $\mathcal{B}$ = optimal base ($\leftrightarrow B$; $x_{\beta(0)} = (-z)$);
- $\forall\ i(i = 0, \ldots, m)$ we have: $$x_{\beta(i)} + \sum_{A_j \notin \mathcal{B}} y_{ij} x_j = y_{i0} \qquad (\alpha)$$
- $x \geq 0 \Rightarrow \sum_{A_j \notin \mathcal{B}} \lfloor y_{ij} \rfloor x_j \leq \sum_{A_j \notin \mathcal{B}} y_{ij} x_j \Rightarrow \underbrace{x_{\beta(i)} + \sum_{A_j \notin \mathcal{B}} \lfloor y_{ij} \rfloor x_j}_{\text{integer } \forall \text{ integer } x \implies} \leq y_{i0}$
  $$\implies x_{\beta(i)} + \sum_{A_j \notin \mathcal{B}} \lfloor y_{ij} \rfloor x_j \leq \lfloor y_{i0} \rfloor \qquad (\beta)$$
- $(\alpha) - (\beta)$: $\sum_{A_j \notin \mathcal{B}} (y_{ij} - \lfloor y_{ij} \rfloor) x_j \geq (y_{i0} - \lfloor y_{i0} \rfloor)$.
- The **fractional part** of y_{ij} is $f_{ij} = y_{ij} - \lfloor y_{ij} \rfloor$; $0 \leq f_{ij} < 1$.
- $\sum_{A_j \notin \mathcal{B}} f_{ij} x_j \geq f_{i0}$ (**Gomory cut** corresponding to row i).
- Multiply by -1 and add a slack variable: $-\sum_{A_j \notin \mathcal{B}} f_{ij} x_j + s = -f_{i0}$.

Cutting plane algorithms: Gomory cuts (cont'd)

- **Theorem** *By adding to a final* LP *tableau the cut*

$$-\sum_{A_j \notin \mathcal{B}} f_{ij} x_j + s = -f_{i0},$$

  1. *no feasible integer point is eliminated;*
  2. *the resulting tableau contains a base which is:*
     (i) *primal infeasible (i.e., outside the new feasible region) if y_{i0} is not integer;*
     (ii) *dual feasible.*

  Proof

  1. the cut was obtained by just imposing the integrality constraints to the LP.

  2.(i) s is a new basic variable which, when added to $\mathcal{B}$, gives a base whose solution includes $s = -f_{i0} \leq 0$, i.e., it is primal infeasible, if y_{i0} is not integer;

  2.(ii) in row 0, the column corresponding to s has 0 $\Rightarrow$ the solution remains dual feasible. $\square$

- It is thus convenient to continue with the dual simplex algorithm.
- The selected row i is called the **generating row**.
- Up to 1958 it was widely considered **impossible** to solve ILP through the Simplex algorithm.
- Widely used in the Sixties in spite of their slow convergence (the cuts become weak after few iterations), the Gomory cuts were progressively abandoned in favor of branch and bound (next).
- Starting from the late Nineties the usefulness of (a variant of) Gomory cuts has been rediscovered $\Longrightarrow$ all modern (branch-and-cut) solvers include them.

Cutting plane algorithms: Gomory cuts (cont'd)

procedure GOMORY:
begin
  remove the integrality constraints from the ILP thus obtaining an LP;
  call TWO_PHASE for LP, and let Y be the final tableau;
  if $\mathit{infeasible}$ = false **and** $\mathit{unbounded}$ = false **then**
    begin
      $\mathit{feasible}$:= true; $k := 0$ (**comment:** cut counter);
      while $\exists$ fractional y_{i0} **and** $\mathit{feasible}$ = true **do**
        begin
          select an $i : y_{i0}$ is fractional, and set $k := k + 1$;
          add the equation $-\sum_{A_j \notin \mathcal{B}} f_{ij} x_j + s_k = -f_{i0}$ to the tableau;
          call DUAL_SIMPLEX;
          if $\mathit{infeasible}$ = true **then** $\mathit{feasible}$:= false (**comment:** unbounded dual, impossible ILP)
        end
    end
end.

Convergence: If there is no degeneration, each iteration considers a base that is different from the previous ones. It can be proved that, even in case of degeneration, the method converges if:
(i) the generating row is selected as the first fractional row;
(ii) the dual simplex pivot is selected according to a method analogous to Bland's rule.

Example:

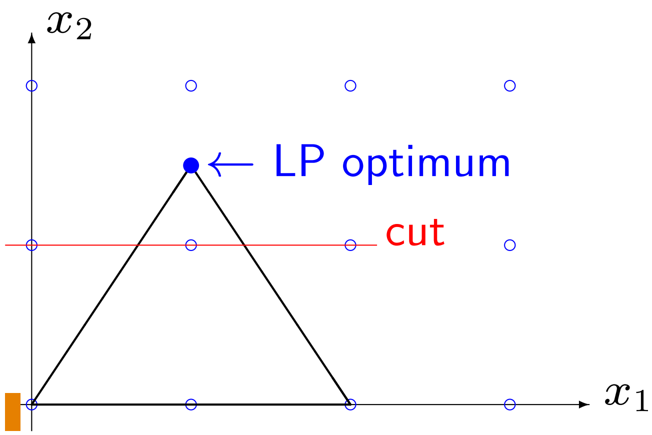
$$\begin{array}{lrrcll}
\min & & -\,x_2 & & & \\
& 3x_1 & +2x_2 & \le & 6 & \to \quad 3x_1 + 2x_2 + x_3 = 6 \\
& -3x_1 & +2x_2 & \le & 0 & \to \quad -3x_1 + 2x_2 + x_4 = 0 \\
& x_1 & ,\ x_2 & \ge & 0 \text{ integer} & \qquad x_3, x_4 \ge 0 \text{ integer}
\end{array}$$

		x_1	x_2	x_3	x_4
$-z$	0	0	−1	0	0
x_3	6	3	2	1	0
x_4	0	−3	2	0	1

		x_1	x_2	x_3	x_4
$-z$	0	$-\frac{3}{2}$	0	0	$\frac{1}{2}$
x_3	6	6	0	1	−1
x_2	0	$-\frac{3}{2}$	1	0	$\frac{1}{2}$

		x_1	x_2	x_3	x_4
$-z$	$\frac{3}{2}$	0	0	$\frac{1}{4}$	$\frac{1}{4}$
x_1	1	1	0	$\frac{1}{6}$	$-\frac{1}{6}$
x_2	$\frac{3}{2}$	0	1	$\frac{1}{4}$	$\frac{1}{4}$

generating row :	**0** (fractional)	**1** (integer)	**2** (fractional)
cut :	$\frac{1}{4}x_3 + \frac{1}{4}x_4 \ge \frac{1}{2}$	$\frac{1}{6}x_3 + \frac{5}{6}x_4 \ge 0$	$\frac{1}{4}x_3 + \frac{1}{4}x_4 \ge \frac{1}{2}$
	$\Downarrow$	$\Downarrow$	$\Downarrow$
by substitution :	$x_2 \le 1$	$x_1 - x_2 \ge -\frac{1}{2}$	$x_2 \le 1$



Selecting row 0:

$$-\tfrac{1}{4}x_3 - \tfrac{1}{4}x_4 + s_1 = -\tfrac{1}{2}$$

		x_1	x_2	x_3	x_4	s_1
$-z$	$\frac{3}{2}$	0	0	$\frac{1}{4}$	$\frac{1}{4}$	0
x_1	1	1	0	$\frac{1}{6}$	$-\frac{1}{6}$	0
x_2	$\frac{3}{2}$	0	1	$\frac{1}{4}$	$\frac{1}{4}$	0
s_1	$-\frac{1}{2}$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$	1

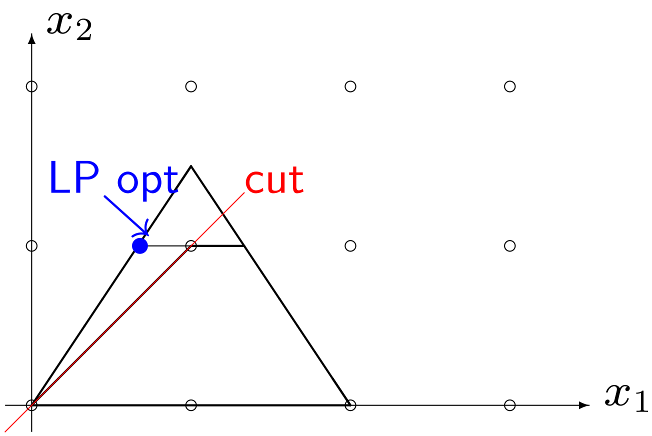
		x_1	x_2	x_3	x_4	s_1
$-z$	1	0	0	0	0	1
x_1	$\frac{2}{3}$	1	0	0	$-\frac{1}{3}$	$\frac{2}{3}$
x_2	1	0	1	0	0	1
x_3	2	0	0	1	1	-4

generating row : $\mathbf{1}$ $(\mathbf{0}, \mathbf{2}, \mathbf{3}$ integer)

cut : $\frac{2}{3}x_4 + \frac{2}{3}s_1 \geq \frac{2}{3}$

$\Downarrow$

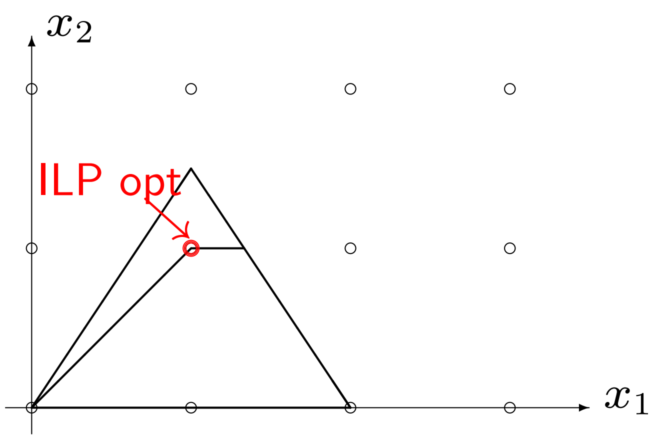
by substitution : $x_1 \geq x_2$



cut equation: $-\frac{2}{3}x_4 - \frac{2}{3}s_1 + s_2 = -\frac{2}{3}$

		x_1	x_2	x_3	x_4	s_1	s_2
$-z$	1	0	0	0	0	1	0
x_1	$\frac{2}{3}$	1	0	0	$-\frac{1}{3}$	$\frac{2}{3}$	0
x_2	1	0	1	0	0	1	0
x_3	2	0	0	1	1	-4	0
s_2	$-\frac{2}{3}$	0	0	0	$-\frac{2}{3}$	$-\frac{2}{3}$	1

		x_1	x_2	x_3	x_4	s_1	s_2
$-z$	1	0	0	0	0	1	0
x_1	1	1	0	0	0	1	$-\frac{1}{2}$
x_2	1	0	1	0	0	1	0
x_3	1	0	0	1	0	-5	$\frac{3}{2}$
x_4	1	0	0	0	1	1	$-\frac{3}{2}$



Optimal integer solution: $x_1 = x_2 = 1$; $z = -1$.

Example:

$$\begin{array}{llllllll}
\min & x_1 & + x_2 & & & & & \\
& 6x_1 & + x_2 & \leq 4 & \rightarrow & 6x_1 + x_2 + x_3 & = & 4 \\
& 3x_1 & & \geq 1 & \rightarrow & 3x_1 - x_4 & = & 1 \\
& x_1 & , x_2 & \geq 0 \text{ integer} & & x_3, x_4 & \geq & 0 \text{ integer}
\end{array}$$

Phase 1:

	0	1	0	0	0	0
		x_1^a	x_1	x_2	x_3	x_4
$-\xi$	-1	0	-3	0	0	1
x_2	4	0	6	1	1	0
x_1^a	1	1	3	0	0	-1

		x_1^a	x_1	x_2	x_3	x_4
$-\xi$	0	1	0	0	0	0
x_2	2	-2	0	1	1	2
x_1	$\frac{1}{3}$	$\frac{1}{3}$	1	0	0	$-\frac{1}{3}$

Phase 2:

$$
\begin{array}{c|c|cccc|}
 & 0 & 1 & 1 & 0 & 0 \\
 & & x_1 & x_2 & x_3 & x_4 \\
\hline
-z & -\frac{7}{3} & 0 & 0 & -1 & -\frac{5}{3} \\
\hline
x_2 & 2 & 0 & 1 & 1 & 2 \\
x_1 & \frac{1}{3} & 1 & 0 & 0 & -\frac{1}{3} \\
\hline
\end{array}
\qquad
\begin{array}{c|c|cccc|}
 & & x_1 & x_2 & x_3 & x_4 \\
\hline
-z & -\frac{1}{3} & 0 & 1 & 0 & \frac{1}{3} \\
\hline
x_3 & 2 & 0 & 1 & 1 & 2 \\
x_1 & \frac{1}{3} & 1 & 0 & 0 & -\frac{1}{3} \\
\hline
\end{array}
$$

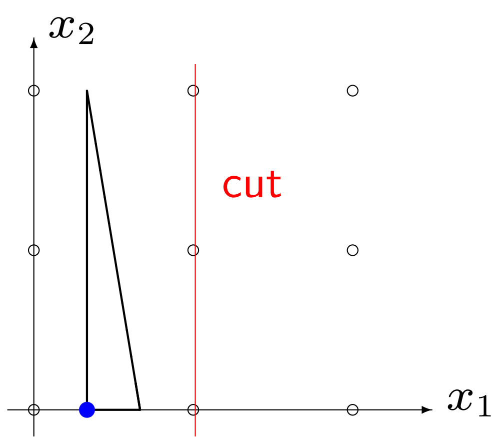
Generating row **0**: cut $\frac{1}{3}x_4 \geq \frac{2}{3}$ $(\Rightarrow -\frac{1}{3}x_4 + s_1 = -\frac{2}{3})$

$$
\begin{array}{c|c|ccccc|}
 & & x_1 & x_2 & x_3 & x_4 & s_1 \\
\hline
-z & -\frac{1}{3} & 0 & 1 & 0 & \frac{1}{3} & 0 \\
\hline
x_3 & 2 & 0 & 1 & 1 & 2 & 0 \\
x_1 & \frac{1}{3} & 1 & 0 & 0 & -\frac{1}{3} & 0 \\
s_1 & -\frac{2}{3} & 0 & 0 & 0 & -\frac{1}{3} & 1 \\
\hline
\end{array}
$$

		x_1	x_2	x_3	x_4	s_1
$-z$	-1	0	1	0	0	1
x_3	-2	0	1	1	0	6
x_1	1	1	0	0	0	-1
x_4	2	0	0	0	1	-3

unlimited dual $\Rightarrow$

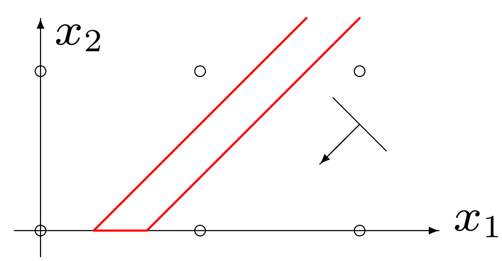
impossible primal.



- **Observation:**

When the continuous relaxation of the ILP is unbounded, "normally" the ILP is unbounded as well, but in very particular cases it can be impossible.

Example:



Branch-and-bound (Land and Doig, 1960)

1. $\mathbf{P^0}$ = problem to be solved;
2. solve the continuous relaxation (x^* = optimal solution);
3. if x^* is an integer point, then the problem is solved. Otherwise
4. select a fractional component, x_j^*, of x^* and impose

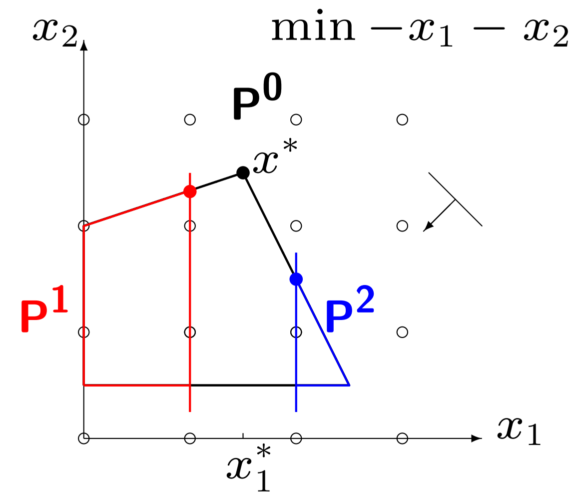
   two **mutually exclusive** and **exhaustive** constraints (**Branching**):

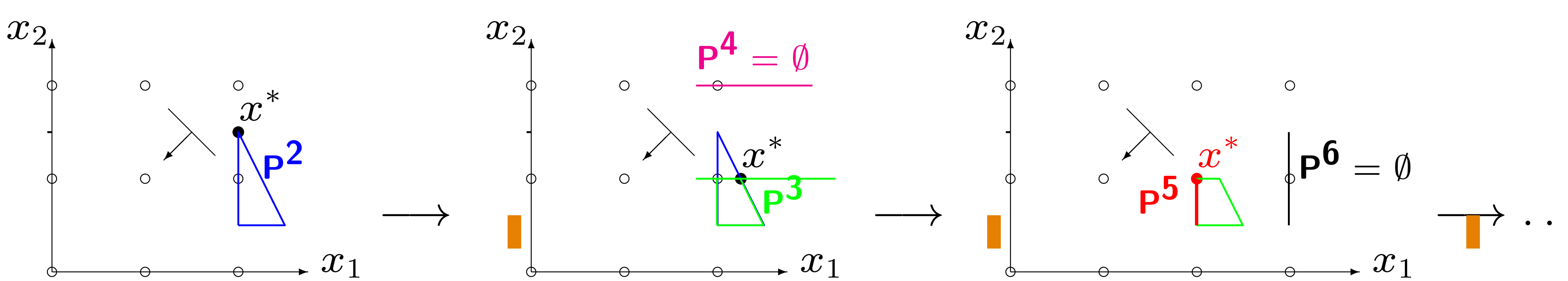
   $$x_j \leq \lfloor x_j^* \rfloor \quad \textbf{or} \quad x_j \geq \lfloor x_j^* \rfloor + 1 \quad (\lfloor a \rfloor = \text{largest integer} \leq a);$$

5. we obtain two new ILP problems: $\mathbf{P^1} = \mathbf{P^0}$ & $(x_j \leq \lfloor x_j^* \rfloor)$,

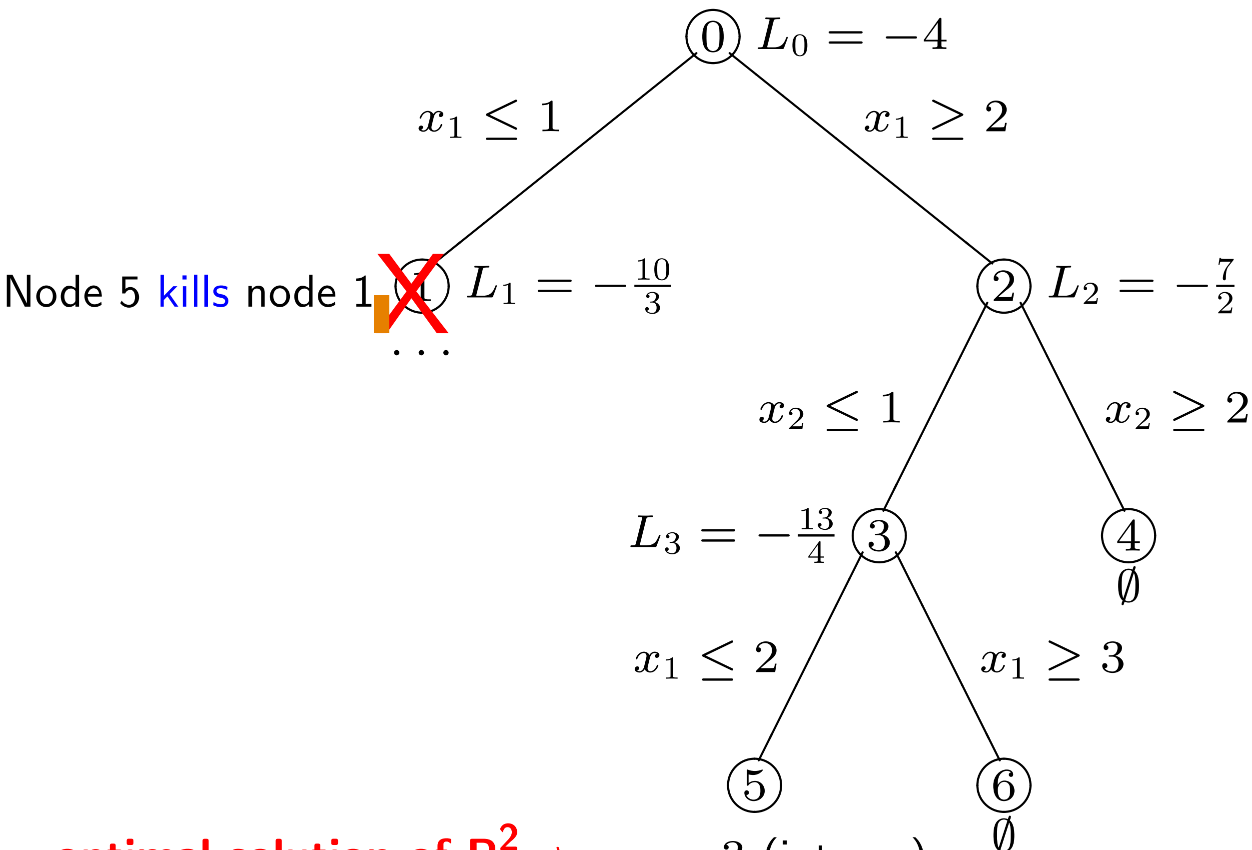
   $\mathbf{P^2} = \mathbf{P^0}$ & $(x_j \geq \lfloor x_j^* \rfloor + 1)$;

6. **solution of** $\mathbf{P^0}$ = best solution between that of $\mathbf{P^1}$ and that of $\mathbf{P^2}$;
7. solve $\mathbf{P^1}$ and $\mathbf{P^2}$ in the same way (recursively). Example: take $\mathbf{P^2}$:

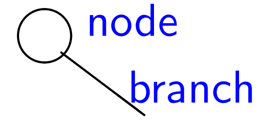




- Let L_i = solution value of the continuous relaxation of $\mathbf{P}^i$ (**Lower bound**);
  Representation through **branch-decision tree:**



Terminology:

- 

- 0 root
- 4, 5, 6 leaves
- 2 parent of 3 and 4
- 3, 4 children of 2
- 2 ancestor of 3, 4, 5 e 6
- 3, 4, 5, 6 descendants of 0 and 2

optimal solution of $\mathbf{P}^2$ $\rightarrow z = -3$ (integer)

- We should now find in the same way the optimal solution of $\mathbf{P}^1$, **BUT** (**Bounding**):
- the solution of $\mathbf{P}^i$ cannot have a smaller value than L_i (computed on a larger feasible region);
- as c is integer, $\lceil L_i \rceil$ is a **Lower bound** for problem $\mathbf{P}^i$ ($\lceil a \rceil$ = smallest integer $\geq a$);
- if we have already found an integer solution of value $z \leq \lceil L_i \rceil$, **we don't need to solve $\mathbf{P}^i$**.
- Solution of $\mathbf{P}^0$ = best integer solution found when no new branching is possible.

How to add the constraints to the tableau

$x_i = a$ **fractional (in base)**:

1. $x_i \le \lfloor a \rfloor \quad \rightarrow \quad x_i + s = \lfloor a \rfloor$

$$
\begin{array}{c|c|c|ccc|c|}
 & & & & & & s \\
\hline
 & -z_0 & \overline{c}_j & 0 & \dots & 0 & 0 \\
\hline
 & & & 1 & & 0 & 0 \\
x_i & a & [y_{ij}] & & 1 & & 0 \\
 & & & 0 & & 1 & 0 \\
\hline
s & \lfloor a \rfloor & 0 \quad \dots & 0 & 1 & \dots 0 & 1 \\
\hline
\end{array}
\quad \text{subtract the row of } x_i \Rightarrow
$$

$$
\begin{array}{c|c|c|ccc|c|}
\hline
s & r & & 0 & \dots & 0 & 1 \\
\hline
\end{array}
\quad r = \lfloor a \rfloor - a < 0
$$

The relative costs remain non-negative, the current solution becomes unfeasible $\Rightarrow$ dual simplex.

How to add the constraints to the tableau (cont'd)

$x_i = a$ **fractional (in base)**:

2. $x_i \geq \lfloor a \rfloor + 1 \quad \rightarrow \quad -x_i + s = -\lfloor a \rfloor - 1 = t$

$$
\begin{array}{c|c|c|ccc|c|}
 & & & & & & s \\
\hline
 & -z_0 & \overline{c}_j & 0 & \ldots & 0 & 0 \\
\hline
 & & & 1 & & & 0 \\
 & & & & & 0 & \\
x_i & a & [y_{ij}] & & 1 & & 0 \\
 & & & 0 & & & \\
 & & & & & 1 & 0 \\
\hline
s & t & 0 \quad \ldots & 0 & -1 & \ldots 0 & 1 \\
\hline
\end{array}
$$

sum the row of $x_i \Rightarrow$

$$
\begin{array}{c|c|c|ccc|c|}
\hline
s & r & & 0 & \ldots & 0 & 1 \\
\hline
\end{array}
$$

$$r = \underbrace{a - \lfloor a \rfloor}_{< 1} - 1 < 0$$

The relative costs remain non-negative, the current solution becomes unfeasible $\Rightarrow$ dual simplex.

Example:
$$\begin{array}{rrrrrl} \max\ z = & x_1 & +\,4\,x_2 & & & \\ & x_1 & +\,3\,x_2 & \le & 9 & \\ & 2\,x_1 & -\ \ x_2 & \ge & 0 & \\ & x_1, & x_2 & \ge & 0, & \text{integer.} \end{array}$$

$$\begin{array}{rrrrrrrl} \min\ -z = & -\ x_1 & -\,4\,x_2 & & & & & \\ & x_1 & +\,3\,x_2 & +\ x_3 & & = & 9 & \\ & -\,2\,x_1 & +\ \ x_2 & & +\ x_4 & = & 0 & \\ & x_1, & x_2, & x_3, & x_4 & \ge & 0, & \text{integer.} \end{array}$$

The simplex algorithm gives, in two pivoting operations (Dantzig rule),

		x_1	x_2	x_3	x_4
z	$\frac{81}{7}$	0	0	$\frac{9}{7}$	$\frac{1}{7}$
x_1	$\frac{9}{7}$	1	0	$\frac{1}{7}$	$-\frac{3}{7}$
x_2	$\frac{18}{7}$	0	1	$\frac{2}{7}$	$\frac{1}{7}$

Lower bound value $L = \lceil -\frac{81}{7} \rceil = -11.$

We select x_1 for branching:

1. $x_1 \le 1 \rightarrow x_1 + x_5 = 1.$

1. $x_1 \leq 1 \rightarrow x_1 + x_5 = 1$:

		x_1	x_2	x_3	x_4	x_5
z	$\frac{81}{7}$	0	0	$\frac{9}{7}$	$\frac{1}{7}$	0
x_1	$\frac{9}{7}$	1	0	$\frac{1}{7}$	$-\frac{3}{7}$	0
x_2	$\frac{18}{7}$	0	1	$\frac{2}{7}$	$\frac{1}{7}$	0
x_5	$-\frac{2}{7}$	0	0	$-\frac{1}{7}$	$\frac{3}{7}$	1

		x_1	x_2	x_3	x_4	x_5
z	9	0	0	0	4	9
x_1	1	1	0	0	0	1
x_2	2	0	1	0	1	2
x_3	2	0	0	1	-3	-7

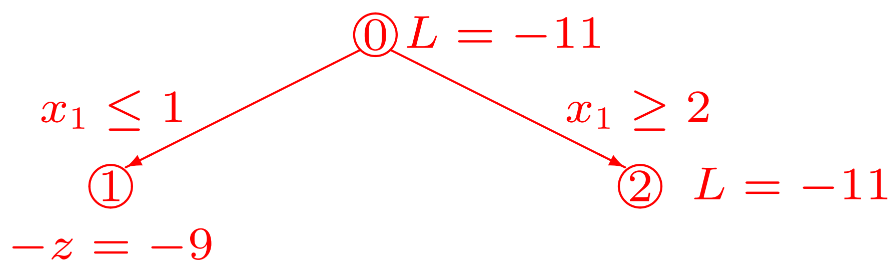
Integer solution of value $-9 > L$ (not provably optimal).

2. $x_1 \geq 2 \rightarrow -x_1 + x_6 = -2$:

		x_1	x_2	x_3	x_4	x_6
z	$\frac{81}{7}$	0	0	$\frac{9}{7}$	$\frac{1}{7}$	0
x_1	$\frac{9}{7}$	1	0	$\frac{1}{7}$	$-\frac{3}{7}$	0
x_2	$\frac{18}{7}$	0	1	$\frac{2}{7}$	$\frac{1}{7}$	0
x_6	$-\frac{5}{7}$	0	0	$\frac{1}{7}$	$-\frac{3}{7}$	1

		x_1	x_2	x_3	x_4	x_6
z	$\frac{34}{3}$	0	0	$\frac{4}{3}$	0	$\frac{1}{3}$
x_1	2	1	0	0	0	-1
x_2	$\frac{7}{3}$	0	1	$\frac{1}{3}$	0	$\frac{1}{3}$
x_4	$\frac{5}{3}$	0	0	$-\frac{1}{3}$	1	$-\frac{7}{3}$

The LP solution (fractional) provides a lower bound $\lceil -\frac{34}{3} \rceil = -11$ for the current node.



The lower bound is better than the incumbent solution $\Rightarrow$ new branching.

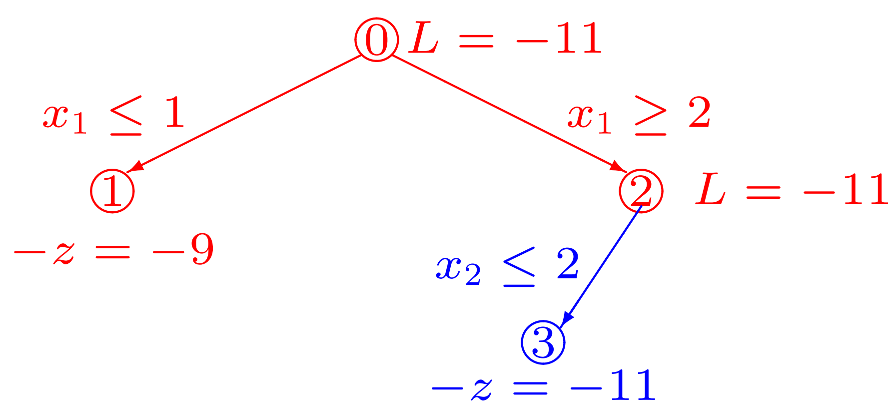
We select x_2 for branching:

3. $x_2 \leq 2 \rightarrow x_2 + x_7 = 2.$

3. $x_2 \leq 2 \rightarrow x_2 + x_7 = 2$:

		x_1	x_2	x_3	x_4	x_6	x_7
z	$\frac{34}{3}$	0	0	$\frac{4}{3}$	0	$\frac{1}{3}$	0
x_1	2	1	0	0	0	-1	0
x_2	$\frac{7}{3}$	0	1	$\frac{1}{3}$	0	$\frac{1}{3}$	0
x_4	$\frac{5}{3}$	0	0	$-\frac{1}{3}$	1	$-\frac{7}{3}$	0
x_7	$-\frac{1}{3}$	0	0	$-\frac{1}{3}$	0	$-\frac{1}{3}$	1

		x_1	x_2	x_3	x_4	x_6	x_7
z	11	0	0	1	0	0	1
x_1	3	1	0	1	0	0	-3
x_2	2	0	1	0	0	0	1
x_4	4	0	0	2	1	0	-7
x_6	1	0	0	1	0	1	-3



Optimal LP solution (integer) of value -11 = lower bound of the parent node

$\Rightarrow$ no need to explore the second child.

Final optimal solution: $x_1 = 3$, $x_2 = 2$.

Exploration strategies in branch-and-bound

- Two main issues for designing a branch-and-bound algorithm:
  - explorations strategy (which node has to be explored next);
  - how to compute bounds (will be seen later).
- the branch-decision tree can be **binary** (two children per node)
  or **multiple** ($q > 2$ children per node, using q **exhaustive** conditions).
- the branching conditions are preferably **mutually exclusive**, but not necessarily.
- **Notation:**
  - z = value of the incumbent solution;
  - $L(P^k)$ = lower bound value for node P^k.

Exploration strategies in branch-and-bound (cont'd)

Depth-first:

- compute $L(P^0)$; generate the first child (P^1) of P^0, and compute $L(P^1)$;
- generate the first child of P^1, and so on. **Rule:**
- **Forward step:** generate one child of the last generated node, P^k, until
  - P^k is immediately solvable (e.g., integer LP solution), hence possibly update z, or
  - $L(P^k) \geq z$, or
  - P^k does not have feasible non-explored nodes. In such cases:
- **Backtracking:** Backtrack to the parent of P^k, say P^p, and,
  - if $L(P^p) < z$, generate the next child of $L(P^p)$, then its first child, and so on;
  - otherwise (or if all children of P^p have already been explored) backtrack to the parent of P^p;
- terminate when trying to backtrack from P^0.
- **Pros:**
  - small number of active nodes;
  - each node is parent or child of the previous node;
  - easy to implement;
  - feasible incumbent solutions are quickly produced.

Exploration strategies in branch-and-bound (cont'd)

Lowest-first (Highest-first for maximization problems):

- compute $L(P^0)$, and initialize $\Pi := \{P^0\}$ (active nodes);
- at each iteration:
  - remove the node with smallest lower bound from Π;
  - generate all its children, and compute their lower bounds;
  - if there are immediately solvable nodes, possibly update z;
  - add to Π the non immediately solvable nodes P^k for which $L(P^k) < z$;
- terminate when $\Pi = \emptyset$ or all nodes $P^k \in \Pi$ have $L(P^k) \geq z$.
- **Pros:**
  - the most promising node is explored first;
  - small number of nodes globally explored to obtain the solution;
  - no node is explored in vain (but in case of ties):

    if we explore P^r, with $L(P^r) > L(P^\ell)$, P^ℓ will have to be explored in any case.
- **Cons:**
  - higher computing time for exploring a node;
  - high number of active nodes;
  - no relationship between the current node and the previous node;
  - not easy to implement.

Exploration strategies in branch-and-bound (cont'd)

Depth-first (revisited):

- Same structure as depth-first, but
  - in the **forward step** all children of the current node are generated, their lower bounds are computed, and exploration continues with the child having minimum lower bound;
  - when **backtracking** the exploration continues with the non-explored child having minimum lower bound.
  - given two children, P^ℓ and P^r with $L(P^\ell) < L(P^r)$, P^ℓ will have to be explored in any case.

Breadth-first:

- all children of P^0 are generated;
- all children of all children of P^0 that are not non immediately solvable are generated, and so on.
- Rarely used;
- adopted when all (or a large subset of) feasible solutions have to be generated.

Mixed Integer Linear Programming

- In a general case: a subset of variables can only take **integer values**;

  the other variables can take **fractional values**.

- **Example:** transportation problems in which the variables represent:
  - numbers of vehicles to be used on the various routes;
  - quantities of goods (in tons) to be loaded on the vehicles .

- **Mixed Integer Linear Programming (MILP):**

$$\min c'x$$

$$\begin{array}{rcl} Ax & = & b \\ x & \geq & 0 \\ x_j & & \text{integer } (j = 1, \ldots, \overline{n}) \\ x_j & & \text{fractional } (j = \overline{n} + 1, \ldots, n) \end{array}$$

- **Branch-and-bound** works: we only branch on a fractional x_j $(j = 1, \ldots, \overline{n})$;
- **Gomory cuts** do not work. Modified cuts (*Gomory Mixed-Integer Cuts*, GMICs) work, but
- they were not used until the end of the 90s ($\Longleftarrow$ branch-and-bound is faster), until
- Recent discovery: they work very well in **branch-and-cut** algorithms,
  both for MILP and for ILP! All current software is based on branch-and-cut with GMICs.

0-1 (or Binary) Linear Programming

- Special case of ILP: $$\begin{array}{rcl} \min c'x & & \\ Ax & = & b \\ x_j & \in & \{0,1\} \quad \forall\, j\,. \end{array}$$
- The simplest case occurs when A has just one row: $a'x = \bar{b}$;
- usually expressed in maximization canonical form (**0-1 Knapsack Problem (KP01)**):

$$\begin{array}{rcl} \max \displaystyle\sum_{j=1}^{n} p_j x_j & & \\ \displaystyle\sum_{j=1}^{n} w_j x_j & \leq & c \\ x_j & \in & \{0,1\} \quad (j = 1, \ldots, n)\,. \end{array}$$

  - n items, each having a value p_j (**profit**) and a **weight** w_j;
  - a container (**knapsack**) having **capacity** c;
  - select a subset of items having maximum profit and a total weight not exceeding c;
  - applications in *cargo loading*, *capital budgeting*, ...
  - widely studied because of its simple structure.
  - We will assume that p_j, w_j and c are positive integers, $w_j \leq c \;\forall\, j$, $\displaystyle\sum_{j=1}^{n} w_j > c$.

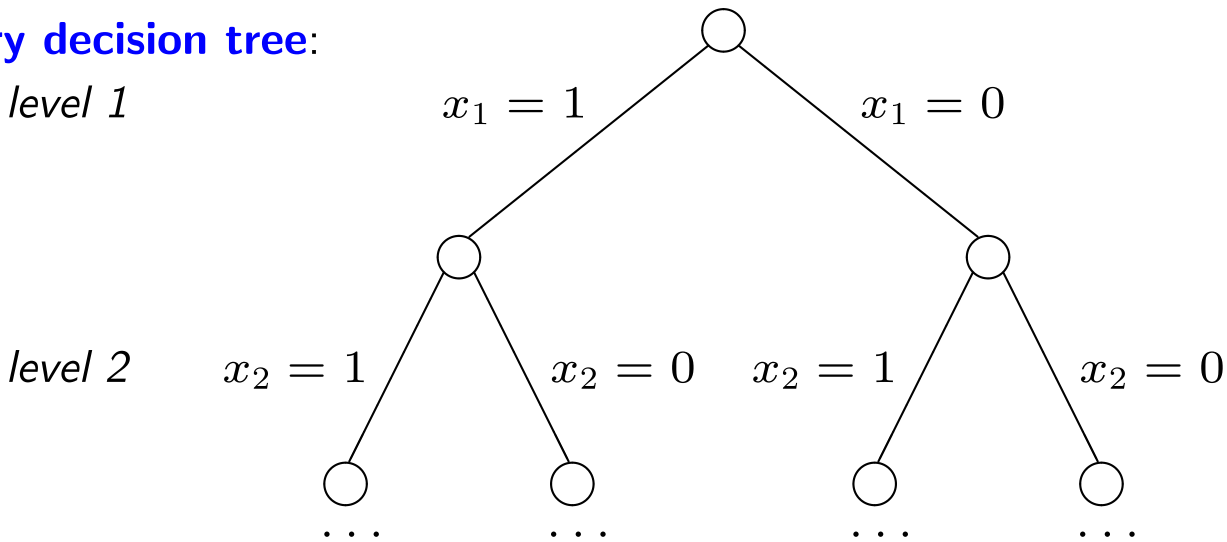
Branch-and-bound algorithm for the 0-1 Knapsack Problem

- It is convenient to preliminary sort the items by **non-increasing profit per unit weight**:

$$\frac{p_j}{w_j} \geq \frac{p_{j+1}}{w_{j+1}} \ \forall \ j$$

.

- **Binary decision tree**:



- **Exploration strategy**: at the first level $x_1 = 1$, $x_1 = 0$; at each iteration,
  if $\exists$ active node generated by $(x_j = 1)$, then branch from it
  otherwise branch from the last node generated by $(x_j = 0)$.

- $\Rightarrow$ the set of active nodes contains:
  at most one node generated by $(x_j = 1)$;
  one or more nodes generated by $(x_j = 0)$.

Branch-and-bound algorithm for the 0-1 Knapsack Problem (cont'd)

- **Upper bound:** continuous relaxation:

  $x_j \in \{0,1\}$ $(j = 1, \ldots, n)$ replaced by $0 \le x_j \le 1$ $(j = 1, \ldots, n)$.

- Solution of the continuous relaxation (Dantzig, 1957):

  consecutively insert the best element, taking a fraction of the first item, s, that does not fit:

  $s := \min\{i \; : \sum_{j=1}^{i} w_j > c\}$(critical item); $\bar{c} := c - \sum_{j=1}^{s-1} w_j$(residual capacity) :

  $$U := \left\lfloor \sum_{j=1}^{s-1} p_j + \bar{c}\,\frac{p_s}{w_s} \right\rfloor.$$

- **Example:**

$$\begin{array}{rcl} n &=& 5 \\ p' &=& (\;12, \;\; 12, \;\; 7, \;\; 6, \;\; 2) \\ w' &=& (\;\;4, \;\;\; 5, \;\; 3, \;\; 3, \;\; 2) \\ c &=& 10 \\ U &=& \left\lfloor 12 + 12 + 1 \cdot \frac{7}{3} \right\rfloor = 26 \end{array}$$

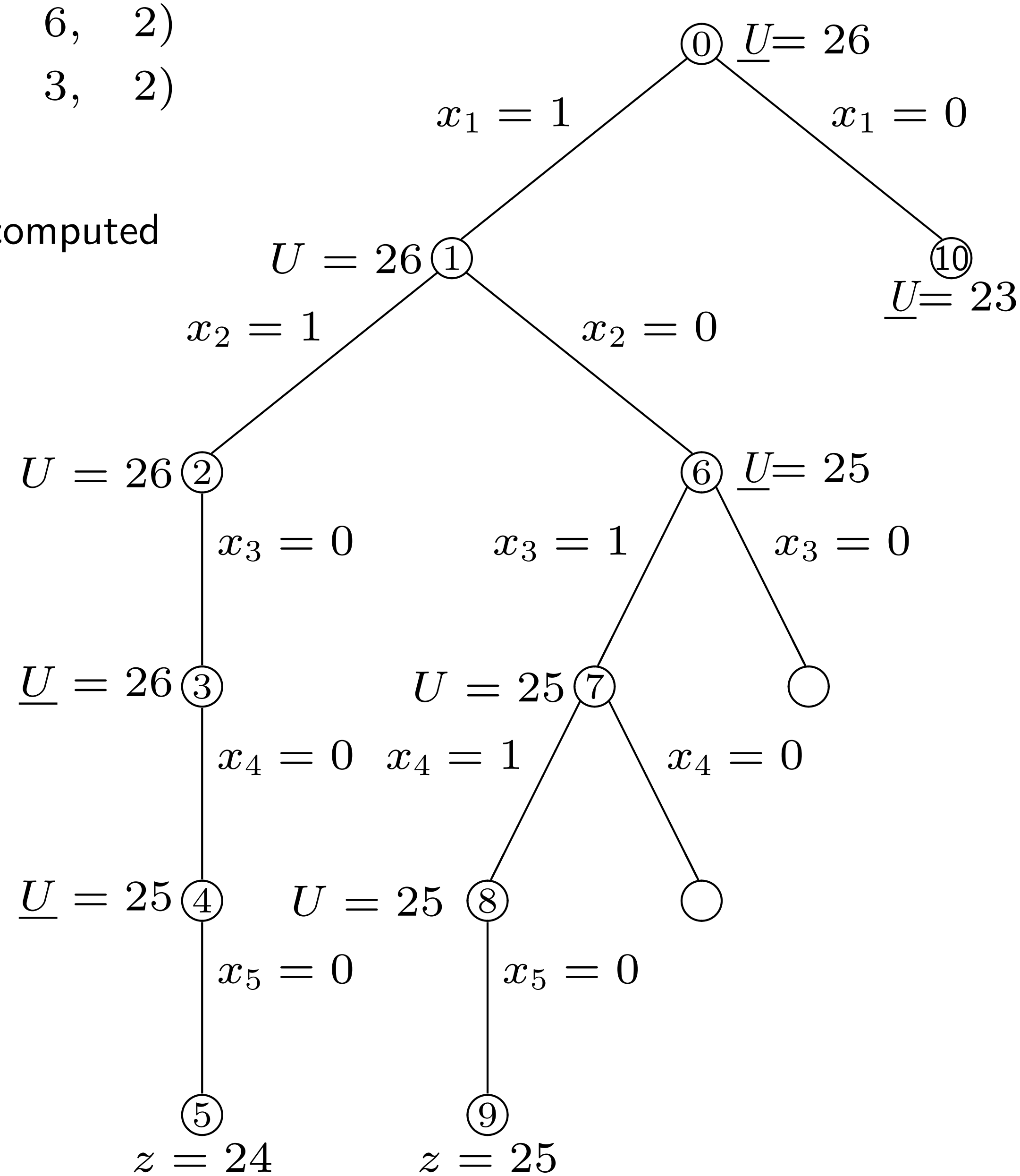
- **Note:** Upper bound of a node generated by $(x_j = 1)$ = upper bound of the parent node;

  $\forall$ node, upper bound of the left son $(x_j = 1) \ge$ upper bound of the right son $(x_j = 0)$.

Example:

$$
\begin{array}{rcllllll}
n & = & & 5 \\
p' & = & (& 12, & 12, & 7, & 6, & 2) \\
w' & = & (& 4, & 5, & 3, & 3, & 2) \\
c & = & & 10
\end{array}
$$

Upper bound U underlined if it has to be computed



Branch-and-bound algorithm for the 0-1 Knapsack Problem (cont'd)

Implementation: k =current level;

$x' = (1,\ 1,\ 0,\ 1,\ 0,\ \underbrace{1}_{k},\ -,\ -,\ -,\ -)$ = current values in the tree;

$$\left.\begin{array}{l} k := k+1 \text{ (next level);} \\ \textbf{if } w_k \le c - \sum_{j=1}^{k-1} w_j x_j \textbf{ then } x_k := 1 \\ \textbf{else} \\ \quad \textbf{begin} \\ \quad\quad x_k := 0; \end{array}\right\} \textit{forward step}$$

U := upper bound for the current node;

while $U \le z$ (= incumbent solution value) **do**

begin

$$\left.\begin{array}{l} k := \max\{j < k : x_j = 1\}; \\ x_k := 0; \end{array}\right\} \textit{backtracking}$$

U := upper bound for the current node

end

end

Branch-and-bound algorithm for the 0-1 Knapsack Problem (cont'd)

Upper bound: A good upper bound should be **tight** (i.e., have a low value)

Improving the Dantzig bound:

- in the optimal solution the critical item s is either excluded or included:
- if $x_s = 0$ the bound is $B^1 = \left\lfloor \sum_{j=1}^{s-1} p_j + \overline{c}\, \frac{p_{s+1}}{w_{s+1}} \right\rfloor$;
- if $x_s = 1$ the bound is $B^2 = \left\lfloor \sum_{j=1}^{s} p_j - \underbrace{(w_s - \overline{c})}_{\text{missing capacity}} \frac{p_{s-1}}{\underbrace{w_{s-1}}_{\text{worst ratio} \Rightarrow \text{minimum loss}}} \right\rfloor$;
- **new bound:** $\overline{U} = \max(B^1, B^2)$.
- **Example:** $n = 5$, $p' = (15, 8, 8, 7, 5)$,
  $w' = (5, 3, 4, 4, 5)$, $c = 10$.
  $U(\text{Dantzig}) = 27$; $B^1 = 26$, $B^2 = 25$ $\Rightarrow \overline{U} = 26$.
- $B^1 \leq U$ (obvious); easy to algebraically prove that $B^2 \leq U$; $\Rightarrow$ $\overline{U} \leq U$
- $\overline{U}$ is tighter and requires few additional operations $\Rightarrow$ convenient.
- In general, compromise between tightness and computing time.

Branch-and-cut algorithms

- **Branch-and-cut** = (branch-and-bound) "+" (cutting-planes).
- Branch-and-bound algorithm which, at each decision node, generates cuts in an attempt to find an integer solution, or at least to improve the bound.
- Gomory cuts depend on the conditions imposed by the ancestor nodes;
- in branch-and-cut it is common to generate **weaker** cuts that are valid for the whole branch-decision tree (**global cuts**). The cuts are stored in a special data base (**cut pool**).
- At each branch-decision node:

  x^* := solution of the continuous relaxation of the current problem;

  while x^* not integer **and** iteration limit not reached **do**

      if the pool contains cuts that are violated by x^* **then**

          begin

              select one or more cuts not yet used, and add them to the current problem;

              x^* := solution of the continuous relaxation

          end

      else generate new cuts, and add them to the pool;
- Main difficulty: method for generating global cuts.
- Gomory mixed-integer cuts are also always added.

Some considerations on time bounds

- **How many time (steps, iterations ...)** requires the branch-and-bound algorithm for KP01?
- **If we are lucky**, the first (leftmost) **n** branches will find the optimal solution,

  and the bounds will kill all other nodes: the algorithm will take a time proportional to **n**.
- **If we are unlucky**, the bounds will kill no node: time proportional to $\mathbf{2^n}$.
- We say that, **in the worst case**, the algorithm solves the problem in $\mathbf{O(2^n)}$ time, or that the algorithm has **time complexity** $\mathbf{O(2^n)}$.
- Other problems: consider a **graph** having **n vertices** (**Introduction**)
- <u>Problem</u>: find the **shortest path** that connects two vertices of a **graph**: there is an algorithm (studied in **Network Optimization**) that solves the problem in $\mathbf{O(n^2)}$ time.
- <u>Problem</u>: find the **longest path** that connects two vertices of a graph: a branch-and-bound type algorithm (studied in **Network Optimization**) solves the problem in $\mathbf{O((n-1)!)}$ time.
- <u>Problem</u>: find the **shortest tour** to deliver products from a depot to clients and return to the depot: a branch-and-bound type algorithm (studied in **Network Optimization**) solves the problem in $\mathbf{O((n-1)!)}$ time.
- Great difference. For **n=100**: $100^2 \approxeq 10^4$, $2^{100} \approxeq 10^{30}$, $99! \approxeq 10^{156}$.
  (The number of atoms in the universe is estimated to be $\approxeq 10^{80}$.)
- Do we know faster algorithms for the longest path or the shortest tour in a graph? **No!**.
- These issues will be examined in the next section.

Software and freeware for LP, ILP, and MILP

Main issues for a practical implementation of the simplex algorithm:

1. **Numerical stability:**
   - floating point operations produce errors that can propagate;
   - example: $a := \frac{1}{3}(= 0.333\ldots)$; $b := 1 - 3 * a (= 0.000\ldots01)$; is b a valid pivot?
2. **CPU time:**
   - for large-size instances the tableau can include billions of entries, and pivoting operations can be prohibitive;
   - implementations adopt the **Revised Simplex Algorithm**, which only uses the inverse, B^{-1}, of the base sub-matrix:
     - column 0 of the tableau (rows 1–m) $= x_\beta = B^{-1}b$;
     - relative cost vector $= \overline{c}' = c' - c'_\beta B^{-1} A$;
     - jth column of the tableau $= B^{-1} A_j$.
3. **Sparsity:**
   - in real-world large instances matrix A is frequently very "sparse" (most values are 0);
   - special decomposition techniques are used for efficiently handling B^{-1}.

4. Column Generation:

- industrial problems may have up to many thousands of constraints and variables, so even the execution of the revised simplex algorithm can be impossible in practice.
- The most widely method to handle such cases is *Column Generation*:

(i) solve a *"restricted" LP* in which the tableau only includes a small subset of columns (i.e., variables) but is such that a feasible solution exists.

Once the optimal basis has been obtained,

(ii) check if there exists a non-included column with negative relative cost;

(iii) **if** no such column exists

then the restricted LP solution is optimal for the complete problem

else add such column to the tableau and iterate.

Software and freeware for LP, ILP, and MILP (cont'd)

Didactic software:

- Web page: `http://www.or.deis.unibo.it/staff_pages/martello/cvitae.html`
- → Courses → Didactic Tools. **Applets** to execute (student implementations):
  - Phase 1 and Phase 2 of the simplex algorithm;
  - Gomory algorithm;
  - Branch-and-bound algorithm for the 0-1 knapsack problem;
  - Dynamic programming algorithm for the 0-1 knapsack problem (to be seen later).

Commercial software (branch-and-cut + local cuts + GMICs + column generation + ...):

- **CPLEX Optimizer** (ILOG → IBM)
- **Gurobi Optimizer** (Gurobi Optimization)
- **LINDO** and **LINGO** (LINDO Systems)
- **XPRESS Optimizer** (FICO)
- **MPL** (Maximal Software): high level language, independent on the platform (Windows, Unix, Mac, OSF), capable of interfacing with all solvers.

Demos

- All commercial software provide free *demos* and/or *student versions* with a limit (200-300) on the number of variables/constraints.

Excel (Microsoft) includes an LP/ILP solver (☹ not recommended)

Open source solvers (they can require a licence for commercial use):

- **lpsolve**: simplex, branch-and-bound; ANSI C; `http://sourceforge.net/projects/lpsolve`
- **Clp** and **Cbc**: simplex, branch-and-cut; C++; `http://www.coin-or.org`
- **SCIP**: simplex, branch-cut-and-price + constraint programming; C callable library, very efficient; `http://scip.zib.de/`
- **GLPK**: simplex + interior point (to be defined later), branch-and-cut; GNU; ANSI C callable library; `https://www.gnu.org/software/glpk/`

Nonlinear Programming

- Most softwares can also solve, with reasonable efficiency, **Convex Quadratic Programming:** (special) quadratic objective function, linear constraints.
- No efficient, reliable software exists for general **Nonlinear Programing**. Best code: **Baron**.