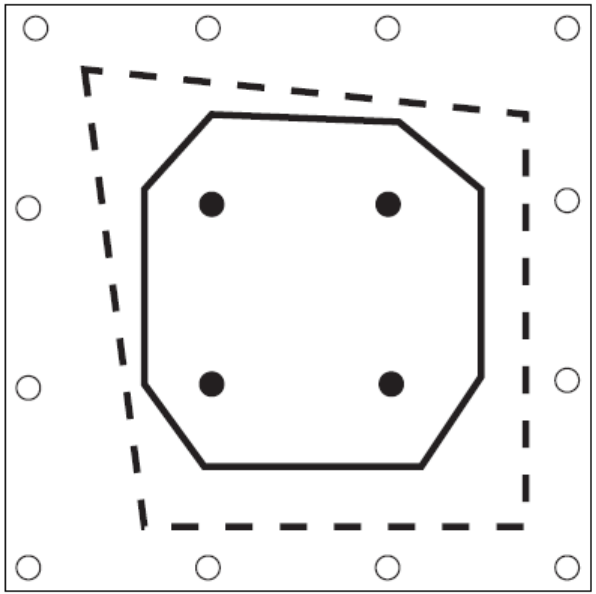
Optimization

Lecture 9.1

M.EIC - 2021.2022



Linear Programming

SOLVING INTEGER LINEAR PROBLEMS

Methods to solve integer programming problems

- Whereas the simplex method is effective for solving linear programs, there is no single technique for solving integer programs.
- Instead, several procedures have been developed, and the performance of any technique appears to be highly problem-dependent.
- Methods to date can be classified as following one of three approaches:
    i) rounding techniques
    ii) cutting-plane techniques;
    iii) group-theoretic techniques
    iv) enumeration techniques, including the branch-and-bound procedure
    v) heuristics and metaheuristics

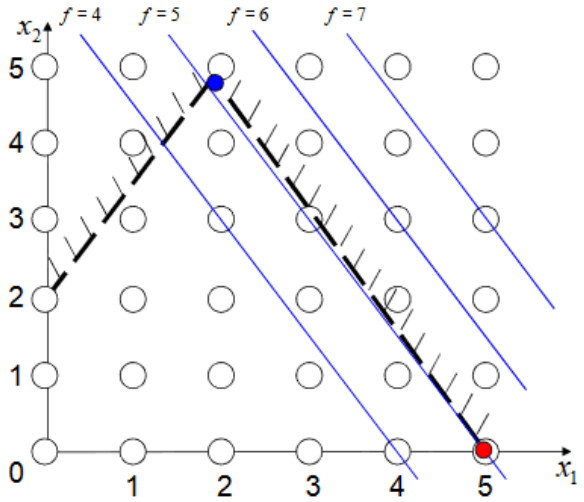
In addition, several composite procedures have been proposed, which combine techniques using several of these approaches.

IP Current Status

- Has become “dominant” over linear programming in past decade
- Saves billions of dollars/year in industry
- Provides benchmarks for TSP problems
- 1 million variable LP approximations
- Branch-and-bound, cutting planes, and separation all used in practice
- General purpose packages do not tend to work as well as with linear programming -- knowledge of the domain is critical.

Rounding

Rounding is a simple solution that forgets about discrete variables, solve the LP problem and use some round-off strategy



$$\max f(x_1, x_2) = x_1 + 0.64x_2$$
$$50x_1 + 31x_2 \leq 250$$
$$-3x_1 + 2x_2 \leq 4$$
$$x_1, x_2 \in Z$$

LP-solution: (1.95, 4.92)
where f=5.098

IP-solution: (5,0)
Where f=5

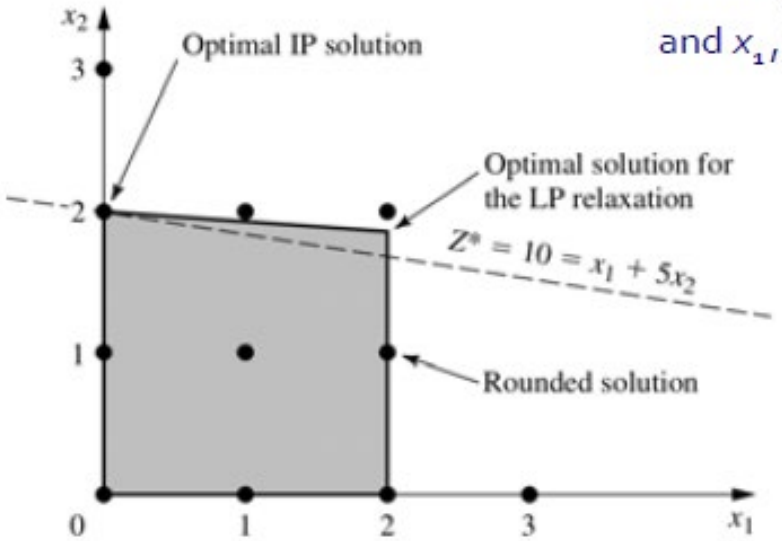
But the rounded LP-solution is not feasible…
and the IP solution is very different from the LP solution ☹

Just another example

Minimize $Z = x_1 + 5x_2$ subject to $x_1 + 10x_2 \leq 20$

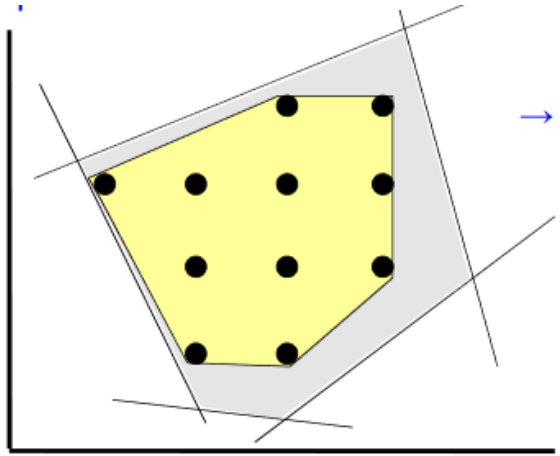
$x_1 \leq 2$

and $x_1, x_2 \geq 0$, integers.



The feasible solutions of a IP

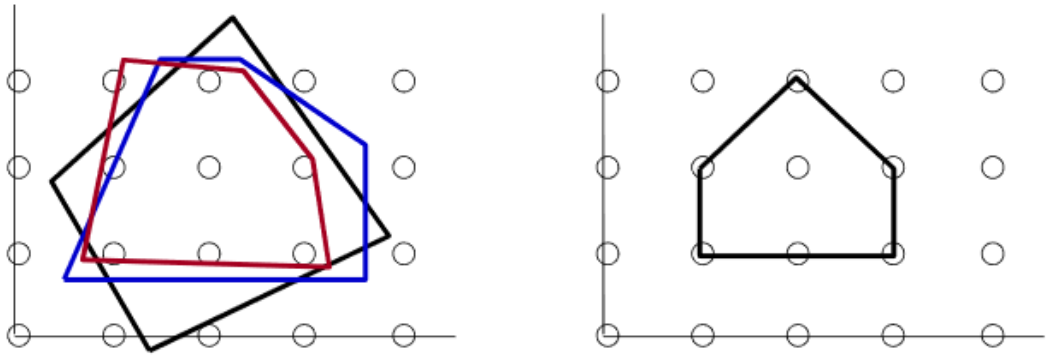
- The feasible region of an integer problem is a subset of the continuous one.
- MILP or IP = initial LP + integer variable constraints
- A solution can never improve if a constraint is added to the problem



→ The initial IP-solution is never better as the original LP

IP formulation and convex hulls

- An IP can in principle be formulated in several ways.
- A convex hull joins all feasible extreme points into a convex envelope.
- Ideal Convex hull = tightest possible IP formulation

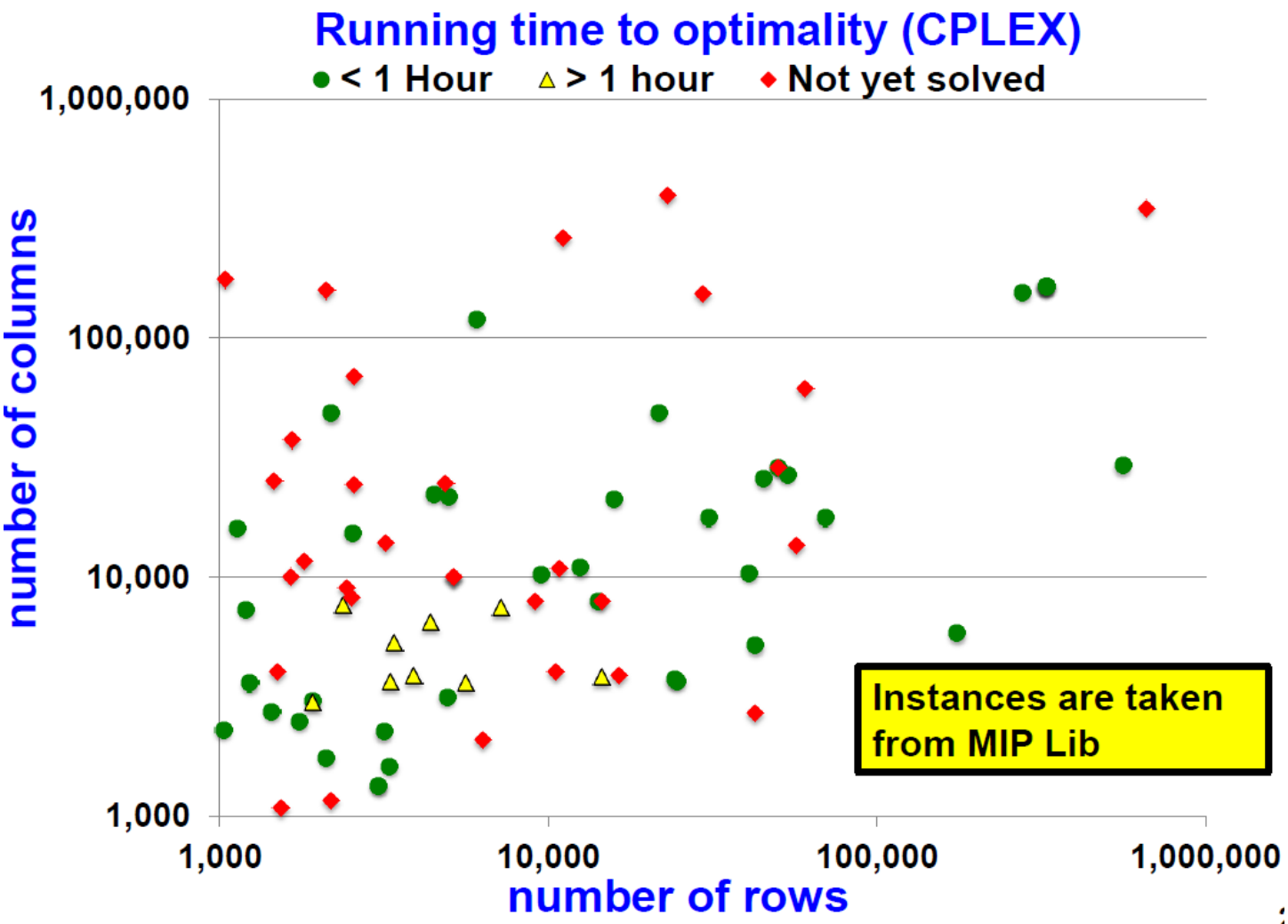


Are integer problems easy to solve?

- A difference to LP is that IP has fewer solutions.
- IP problems have a finite number of feasible solutions

However,

- Finite numbers can be very large!!! With n variables, a BIP problem has 2^n solutions, having exponential growth.
- LP assures that a continuous solution can be optimal, showing the efficiency of the Simplex method.
- LP problems are much easier to solve than IP problems!!!


Running time to optimality (CPLEX)
< 1 Hour
> 1 hour
Not yet solved
1,000,000
100,000
10,000
1,000
number of columns
1,000
10,000
100,000
1,000,000
number of rows
Instances are taken from MIP Lib

Using optimality conditions and bounds

- Optimality conditions allow us to evaluate the quality of a candidate solution even if we do not know the optimal solution (z).
- One method consists in finding a lower bound $\underline{z} \leq z$ and an upper bound $\overline{z} \geq z$ such that, if $\overline{z} = \underline{z}$ then z is optimal.
- Usually, we build sequences of upper and lower bounds:

$$\overline{z}_1 \geq \overline{z}_2 \geq \ldots \geq \overline{z}_k \geq z \qquad \underline{z}_1 \leq \underline{z}_2 \leq \ldots \leq \underline{z}_t \leq z$$

such that for any $\zeta > 0$, we can define an interval $\left|\overline{z}_k - \underline{z}_t\right| < \zeta$

This interval will be the stopping criterium for the algorithm.

Optimality conditions

- When the algorithm stops, we know that the candidate solution z is at most ζ units inferior to the optimal solution.

- Being able to estimate the quality of a candidate solution has a vital importance for the optimization process.

Relaxation:
Transformation of a given problem in a different, simpler, with a wider solution space than the original problem space.
In a maximization problem, a relaxation provides an upper bound. $\bar{z} \geq z$

Types of relaxation:
- ignore some constraints
- ignore the variable's integrality constraints (Linear relaxation)

LP relaxation

- Consequently, most IP algorithms incorporate the Simplex method. This is called LP relaxation.
- Sometimes the solution of the LP problem is also the solution of the IP problem, such as:
- transportation problem, assignment problem, shortest path problem, maximum flow problem,…
- Special structures, such as mutually exclusive constraints or contigent decisions may sometimes simplify the problem.

BRANCH & BOUND METHOD

Linear Relaxation, upper bounds and lower bounds

$$\max 4x_1 - x_2$$
$$s.to$$
$$7x_1 - 2x_2 \le 14$$
$$x_2 \le 3$$
$$2x_1 - 2x_2 \le 3$$
$$x_1 \ge 0, \text{integer}$$
$$x_2 \ge 0, \text{integer}$$

Lower bound

$x_1 = 2$ and $x_2 = 1$ is a feasible solution, so $z \geq 7$.

Upper bound

The optimal solution of the linear relaxation is $x_1 = 2.857$ and $x_2 = 3$, with $z_{LP} = 8.428$

Then, $z \leq 8 \leq 8.428$

Branch & Bound method (1)

The Branch & Bound is the most common method for the resolution of Integer Programming problems and Mixed Integer Programming problems.

It is a divide and conquer approach: we divide the original problem into several simpler problems.

1. Divide P problem into a set of sub-problems {Pk}.
2. Solve sub-problems Pk.
3. Obtain the solution of P through the solutions of the sub-problems.

Branch & Bound (2)

Consider the problem:

$$P : z = \max \{c^T x : x \in S\}$$

How to divide P into smaller sub-problems that will recombine in such a way that a solution for the original problem can be found?

Proposition

- Let $S = S1 \cup \ldots \cup SK$ be a decomposition of S into K smaller sets.
- Let $zk = \max\{c^T x : x \in Sk\}$ for $k = 1, \ldots, K$.
- Then, $z = \max \{z_k : k = 1, \ldots, K\}$.

Branch & Bound (3)

Solve the Linear Relaxation (LP) of the original problem (IP).
As a result, we can have the following situations (fathoming tests):

1. The LP does not have feasible solutions: The IP does not have feasible solutions.
2. The LP optimal solution is a feasible IP solution: we have found the optimal solution.
3. The LP optimal solution is not feasible for the IP: we have an upper bound (for a maximization problem).

In cases 1 and 2, the algorithm stops.
In case 3, we divide (or branch) the problem and solve the next sub-problems recursively.

Branch & Bound (4)

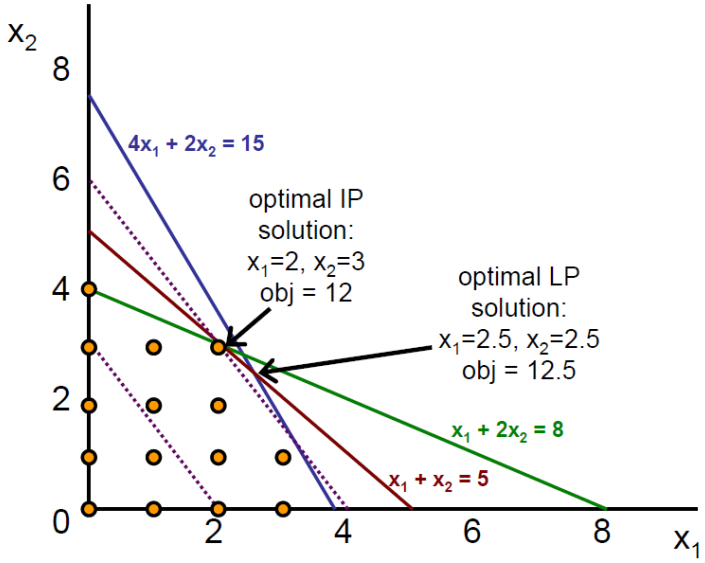
In order to divide a problem into sub-problems:

- Select a variable i, which value, x_i , is not integer in the LP solution.
- Create two subproblems:
  - ✓ *In one of the sub-problems, impose the constraint:* $x_i \leq \lfloor x_i \rfloor$
  - ✓ *In the other one, impose the constraint* $x_i \geq \lceil x_i \rceil$
- After the branching, solve the subproblems recursively..
- If the optimal solution of the linear relaxation is inferior to the current lower bound, it is not necessary to analyse the subproblem further.
- This is the key of success and efficiency of the method: implicit enumeration.

Example

P_1

$$\max 3x_1 + 2x_2$$
$$s.to$$
$$4x_1 + 2x_2 \leq 15$$
$$x_1 + 2x_2 \leq 8$$
$$x_1 + x_2 \leq 5$$
$$x_1 \geq 0, \text{integer}$$
$$x_2 \geq 0, \text{integer}$$



P1 optimal solution:	x1 = 2	x2 = 3	obj = 12
PL optimal solution:	x1 = 2.5	x2 = 2.5	obj = 12.5

Example

P1 divides into P11 and P12

PI optimal solution:	x1 = 2	x2 = 3	f.o = 12
LP optimal solution :	x1 = 2.5	x2 = 2.5	f.o = 12.5

Since both x1 and x2 are non integer, we can branch the problem in x1 or x2.
Branching the problem in x1 (we could also have chosen x2)

P_{11}

$$
\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \geq 3 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}
$$

P_{12}

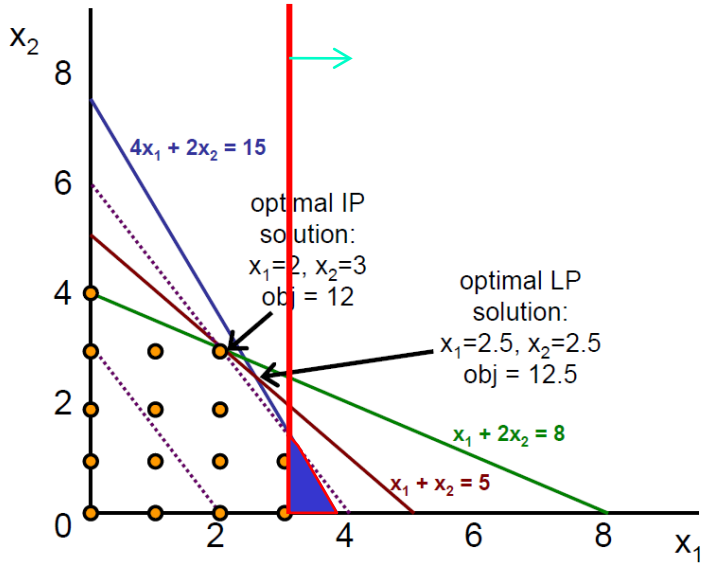
$$
\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \leq 2 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}
$$

Example

solving P11 and dividing into P111 and P112

P_{11}

$$\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \geq 3 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}$$



Solving the linear relaxation:

x1 = 3
x2 = 1.5
f.o = 12

Since x2 is not integer, we have to branch the problem again, adding the constraints x2 <=1 e x2 >=2

Example

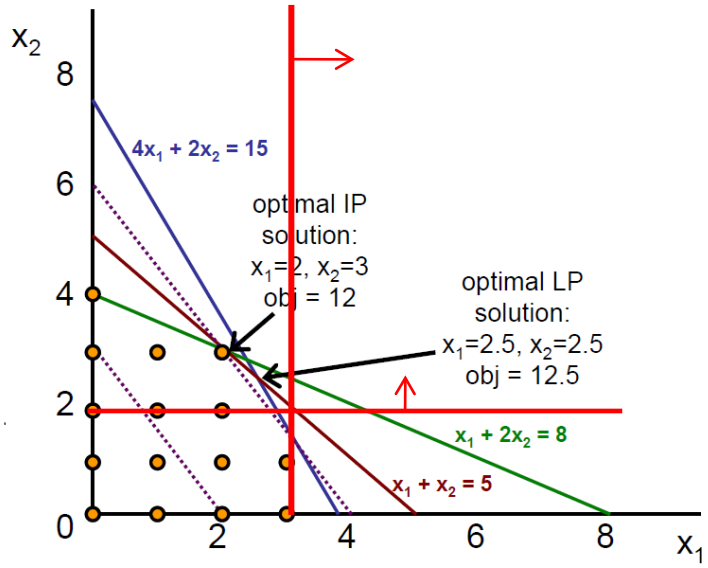
solving P111

P_{111}

$$\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \geq 3 \\
&x_2 \geq 2 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}$$

This problem does not have feasible solutions.

The search on this branch ends here

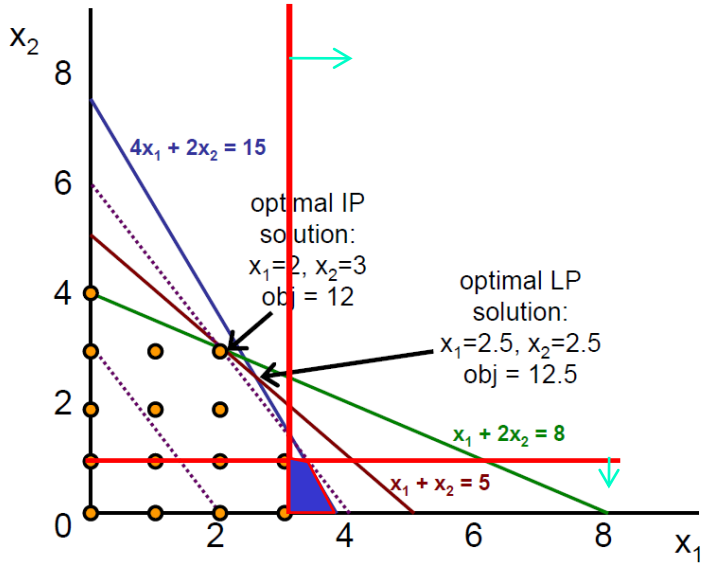


Example

solving P112 and dividing into P1121 and P1122

P_{112}

$$\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \geq 3 \\
&x_2 \leq 1 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}$$



Solving the linear relaxation:

x1 = 3.25
x2 = 1
f.o = 11.75

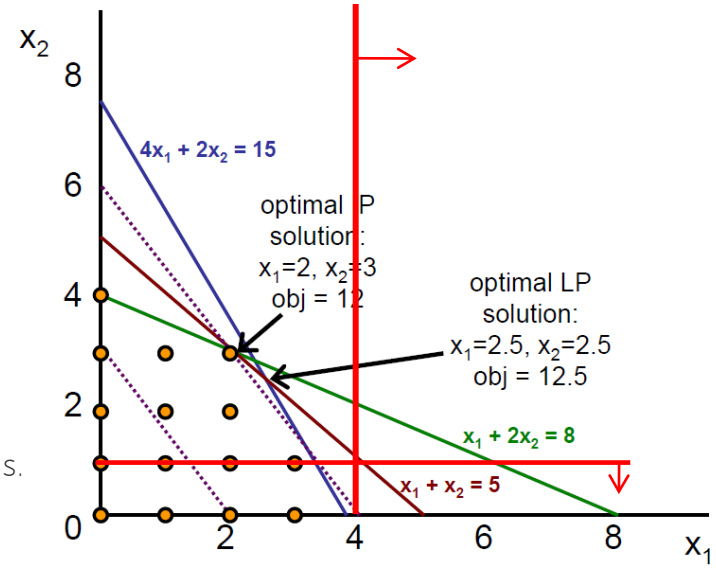
Since x1 is not integer, we divide the problem again, adding the constraints x1 >= 4 e x1<=3

Example

solving P1121

P_{1121}

$$\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \geq 3 \\
&x_2 \leq 1 \\
&x_1 \geq 4 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}$$



This problem does not have feasible solutions.

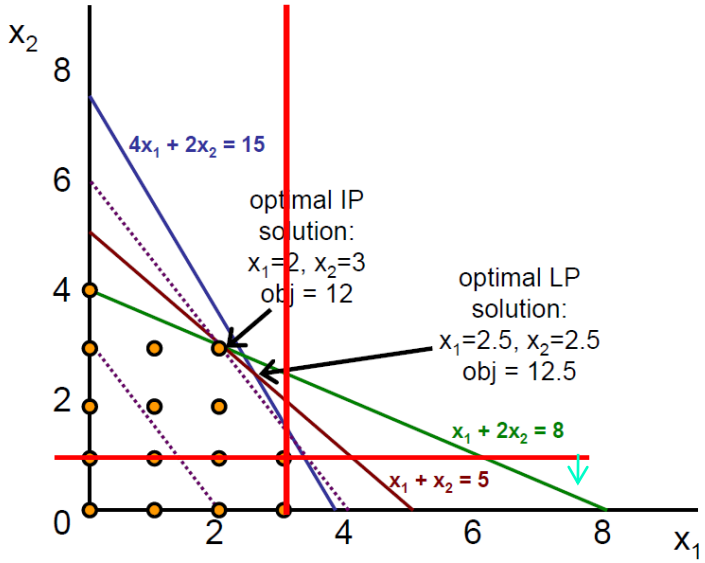
The search on this branch ends here.

Example

solving P1122- integer solution and lower bound

P_{1122}

$$\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \geq 3 \\
&x_2 \leq 1 \\
&x_1 \leq 3 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}$$



Solving the linear relaxation:

X1 = 3
X2 = 1
f.o. = 11

This solution is integer (by far, the only one), so we don't need to branch this problem further…however, the search process has not ended yet…☹

Example

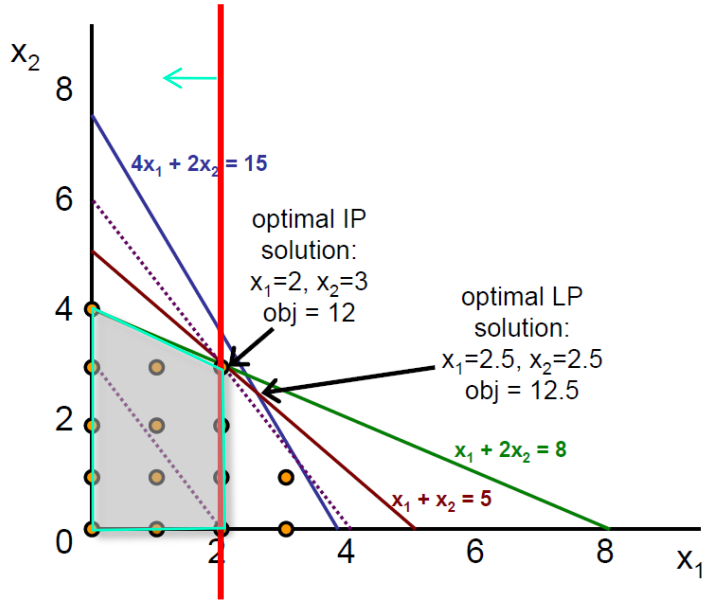
solving P12- optimal solution

P_{12}

$$\begin{aligned}
&\textbf{maximize } 3x_1 + 2x_2 \\
&\textbf{subject to} \\
&4x_1 + 2x_2 \leq 15 \\
&x_1 + 2x_2 \leq 8 \\
&x_1 + x_2 \leq 5 \\
&x_1 \leq 2 \\
&x_1 \geq 0, \text{ integer}; \; x_2 \geq 0, \text{ integer}
\end{aligned}$$

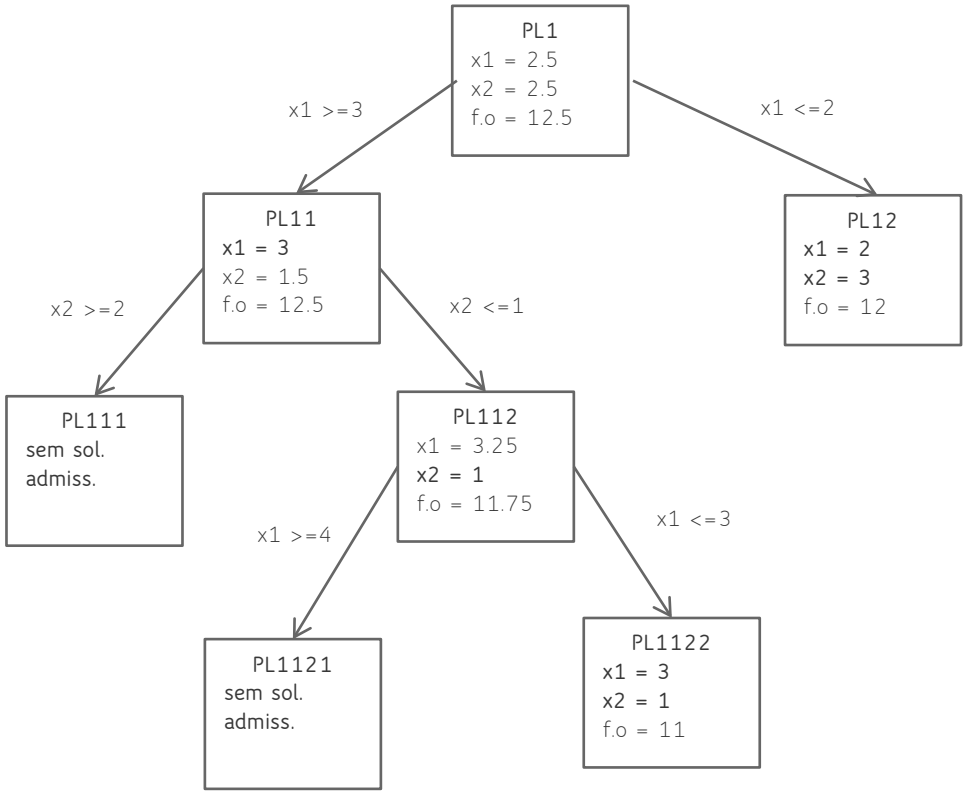
Solving the linear relaxation:

X1 = 2
X2 = 3
f.o = 12



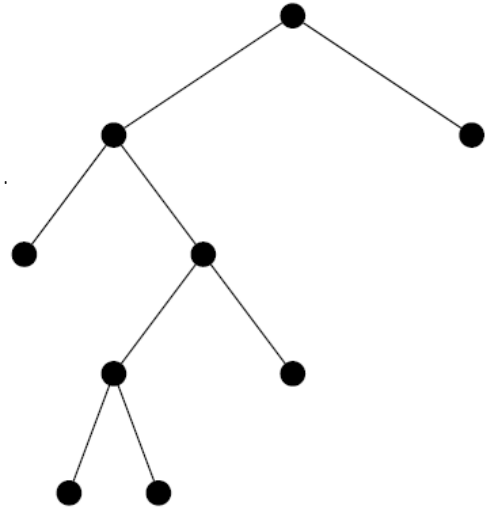
The LP solution is integer, so we don't need to further branch the problem. The solution is better than the other intger solution and we don't have more problems to analyse.
Hence this solution is optimal ☺

Branch & Bound search tree



Terminology

- If we picture the subproblems graphically, they form a search tree.
- Eliminating a problem from further consideration is called pruning.
- The act of bounding and then branching is called processing.
- A subproblem that has not yet been processed is called a candidate.
- The set of candidates is the candidate list.
- The incumbent solution is a feasible IP solution with the best objective function value obtained so far.



while there is some **active** nodes **do**

select an **active** node *j*

mark j as **inactive**

Solve LP(j): denote solution as x(j);

Case 1 -- **if** $z_{LP}(j) \le z_I$ **then** ***prune*** node j;

Case 2 -- **if** $z_{LP}(j) > z_I$ and

if x(j) is feasible for IP(j)

then Incumbent := x(j), and $z_I := z_{LP}(j)$;

then ***prune*** node j;

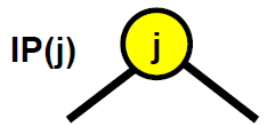
Case 3 -- If **if** $z_{LP}(j) > z_I$ and

if x(j) is not feasible for IP(j) **then**

mark the children of node *j* as **active**

endwhile

z_I: **incumbent obj. value**



Choices in Branch-and-Bound

Each of the steps in a branch-and-bound algorithm can be done in many different ways.

- Heuristics to find feasible solutions -> yields lower bounds (in maximization)
- Solving a relaxation -> yields upper bounds (in maximization)
- Node selection - which subproblem to look at next
- Branching - dividing the feasible region

- We can help an integer programming solver by telling it how it should do these steps.
- We can even implement our own better way to do one or more of these steps.
- We can do better because we know more about our problem.

How Long Does Branch-and-Bound Take?

Simplistic approximation:

Total time = (Time to process a node) x (Number of nodes)

When making choices in branch-and-bound, think about the effect on these separately

Choices in Branch-and-Bound: Heuristics

Practical perspective: finding good feasible solutions is most important

- Manager won't be happy if you tell her you have no solution, but you know the optimal solution is at most U … ☹

A heuristic is an algorithm that tries to find a good feasible solution

- No guarantees are given -> maybe it fails to find a solution, maybe it finds a poor one
- But, typically a heuristic runs fast
- Sometimes called "primal heuristics"

Good heuristics help find an optimal solution in branch-and-bound

- Key to success: Prune early and often
- We prune when $U(S_i) \leq L$, where L is the best lower bound
- Good heuristics => larger L => prune more

Heuristics - Examples

- LP relaxation can be interpreted as a heuristic. But often (usually) fails to give a feasible solution
- Rounding/Diving
  i. Round the fractional integer variables
  ii. With those fixed, solve LP to find continuous variable values
  iii. Diving: fix one fractional integer variable, solve LP, continue
- Metaheuristics (Simulated Annealing, Tabu Search, Genetic Algorithms, etc…)
- Optimization-based heuristics
  - Solve a heavily restricted version of the problem optimally
- Problem specific heuristics
  - This is often a very good way to help an IP solver

Choices in Branch-and-Bound: Choosing/solving the Relaxation

The relaxation is the most important factor for proving a solution is optimal

Optimal value of the relaxation yields the upper bound (maximization problem)

- Recall: we prune when $U(S_i) \leq L$
- Smaller (tighter) upper bounds => prune more
- So the formulation is very important!!

The time spent solving the relaxation at each node usually dominates the total solution time

- We want to solve it fast, but also want to solve fewer
- Potential trade-off: a formulation that yields a better upper bound may be larger (more time to solve relaxation)
- Usually, the formulation with a better upper bound wins

Solving the LP relaxation efficiently

Branching is usually done by changing bounds on a variable that is fractional in the current solution (for example, $x_j \leq 0 \; or \; x_j \geq 1$)

- The only difference in the LP relaxation for the new subproblem is this bound change
- LP dual solution remains feasible
- Reoptimize with dual simplex
- If choose to process the new subproblem next, can even avoid refactoring the basis

Choices in Branch-and-Bound: Node selection

Node selection: Strategy for selecting the next subproblem (node) to be processed.

- Important, but not as important as heuristics, relaxations, or branching
- Often called search strategy

Two different goals:

- Minimize overall solution time.
- Find a good feasible solution quickly.

Search Strategy: Best First

One way to minimize overall solution time is to try to minimize the size of the search tree.
We can achieve this if we choose the subproblem with the best bound (highest upper bound if we are maximizing).

Justification:

- A candidate node is said to be critical if its bound exceeds the value of an optimal solution to the IP.
- Every critical node will be processed no matter what the search order
- Best first is guaranteed to examine only critical nodes, thereby minimizing the size of the search tree (for a given fixed choice of branching decisions).

Drawbacks of Best First

Doesn't necessarily find feasible solutions quickly

- Feasible solutions are “more likely" to be found deep in the tree

Node setup costs are high

- The linear program being solved may change quite a bit from one node LP solve to the next

Memory usage is high

- It can require a lot of memory to store the candidate list, since the tree can become large Jim

Search Strategy: The Depth First Approach

Depth first: always choose the deepest node to process next. Dive until you prune, then back up and go the other way

Avoids most of the problems with best first:

- Number of candidate nodes is minimized (saving memory)
- Node set-up costs are minimized since LPs change very little from one iteration to the next
- Feasible solutions are usually found quickly

Unfortunately, if the initial lower bound is not very good, then we may end up processing many non-critical nodes. We want to avoid this extra expense if possible.