Integer Programming Methods
Week 1

Remy Spliet

Erasmus University Rotterdam

Course set-up

Contact

- Teachers:

Matthias Walter
University of Twente
m.walter@utwente.nl

Remy Spliet
Erasmus University Rotterdam
spliet@ese.eur.nl

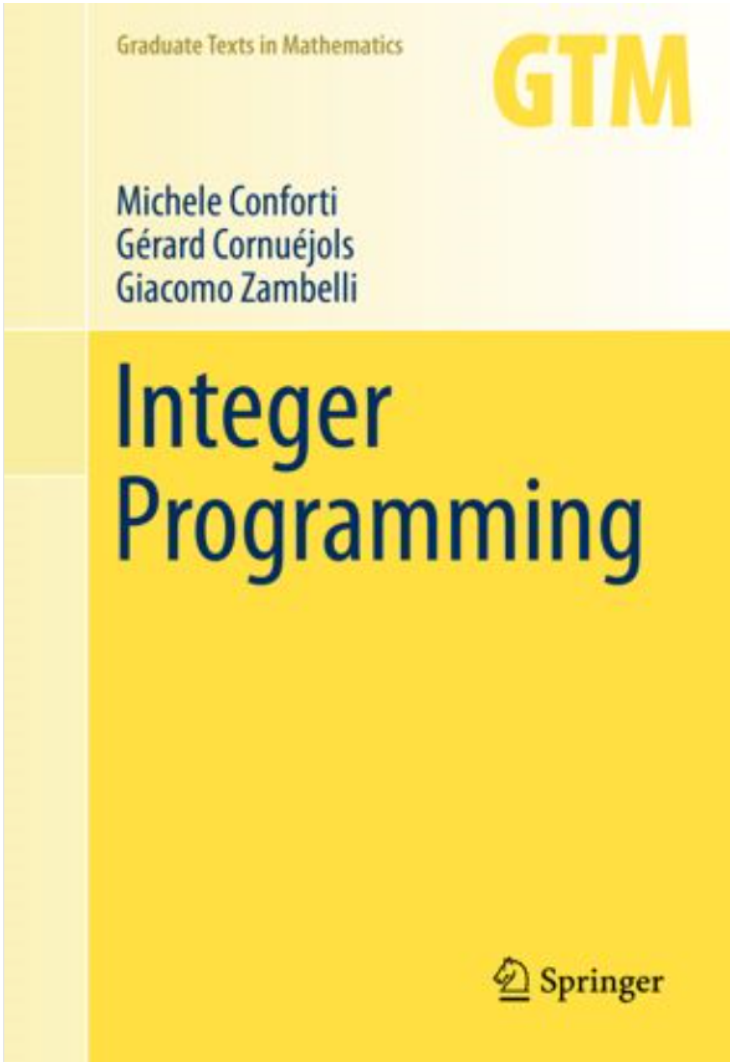
- Website: http://matthiaswalter.org/intpm/

Prerequisite knowledge

- Linear algebra
- Linear programming
- Basic combinatorial optimization
  (shortest path problem, matching, maximum flow, etc.)
- Basic complexity theory
- Programming

Content

Conforti, Cornuéjols, and Zambelli, *Integer programming*, Springer 2014, ISBN: 978-3-319-11008-0



Format

- 11 weeks.
- Per week:
  - Lecture
  - Read book
  - Homework
- Only 9 lectures, 2 spare lectures
- A **course schedule** can be found on the website.

Grading

- Grade is based on homework.
- Submit homework
  - in groups of 2-3,
  - by e-mail,
  - as pdf,
  - with title "IntPM_Homework_Set_<X>_<groupname>.pdf",
  - and be sure to put your names in the file.

Homework Set	**Deadline**	**Send to**
1	December 4, 11:00	spliet@ese.eur.nl
2	January 22, 11:00	spliet@ese.eur.nl
3	February 5, 11:00	m.walter@utwente.nl
4	February 19, 11:00	m.walter@utwente.nl
5	March 4, 11:00	m.walter@utwente.nl

- Each submission is awarded a score $\in \{0, 0.5, 1, 1.5, 2\}$.
- The final grade is the sum of the scores.

Integer linear programming

Integer linear program

Definition

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $\mathbf{c} \in \mathbb{Q}^n$. An **integer linear program** *(ILP) is*

$$\begin{aligned} \max \mathbf{c}^T \mathbf{x} & \\ A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\in \mathbb{N}_+. \end{aligned}$$

- The above ILP is in **cannonical form**.
- ILP also includes
  - minimization, since $\min \mathbf{c}^T \mathbf{x} = -\max -\mathbf{c}^T \mathbf{x}$,
  - $\geq$ constraints, since $A\mathbf{x} \geq \mathbf{b} \Leftrightarrow -A\mathbf{x} \leq -\mathbf{b}$,
  - $=$ constraints, since $A\mathbf{x} = \mathbf{b} \Leftrightarrow A\mathbf{x} \leq \mathbf{b}$ and $A\mathbf{x} \geq \mathbf{b}$,
  - negative valued variables, by using $\mathbf{x} = \mathbf{x}^+ - \mathbf{x}^-$ for $\mathbf{x}^+, \mathbf{x}^- \in \mathbb{N}_+$.

Related problems

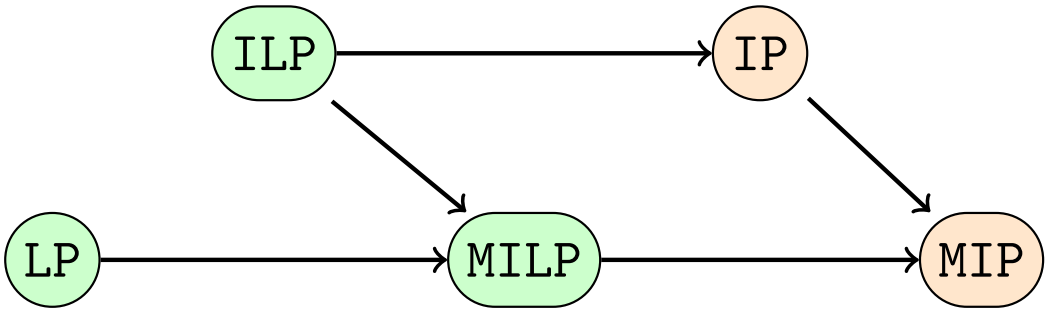
$$\begin{aligned} \max \ & f(\mathbf{x}) \\ & \mathbf{x} \in X \end{aligned}$$

- **Linear programming (LP):**
  f is linear, $X = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$.
- **Integer linear programming (ILP):**
  f is linear, $X = \{\mathbf{x} \in \mathbb{N}^m : A\mathbf{x} \leq \mathbf{b}\}$.
- **Mixed integer linear programming (MILP):**
  f is linear, $X = \{\mathbf{x} \in \mathbb{R}^n \times \mathbb{N}^m : A\mathbf{x} \leq \mathbf{b}\}$.

(Non-linear problems:)

- **Integer programming (IP):** $X \subseteq \mathbb{N}^n$.
- **Mixed integer programming (MIP):** $X \subseteq \mathbb{R}^n \times \mathbb{N}^m$.

Related problems



Special cases

✓ Many problems can be modelled as an ILP:

- the knapsack problem,
- the set packing problem,
- the set covering problem,
- the set partitioning problem,
- the traveling salesman problem,
- the generalized assignment problem,
- ...

! An algorithm for ILP would **solve all** these problems.

Complexity

Theorem

ILP is NP-hard

Proof.

- We create a polynomial time reduction of an NP-hard problem.
- **Subset sum** is the following NP-complete problem:

  Given $\{w_1, \ldots, w_n\} \subset \mathbb{N}$, *does any subset sum to* W*?*

- Subset sum can be formulated as an ILP:

$$\max 0$$
$$\sum_{i=1}^{n} w_i x_i = W$$
$$x_i \in \{0, 1\} \; \forall i \in \{1, \ldots, n\}$$

□

Complexity

- ✗ Never say:

  I can formulate my problem as an ILP, so it is NP-hard.

- E.g. the shortest path problem is polynomially solvable and can be formulated as an ILP.

Branch-and-Bound

Branch-and-Bound

- **Branch-and-Bound** is an algorithm to solve an ILP.
- **Branching** is used to go over all (relevant) solutions.
- **Bounding** is used to avoid evaluating all solutions.

! We consider **maximization**.
Branch-and-Bound similarly applies to minimization.

- Upper bounds are typically found using **relaxations**
- Lower bounds might be found using **heuristics**.

Branch-and-Bound:

Relaxations

Relaxations

Definition

A **Relaxation** *of a problem P, is a problem P' such that, every feasible solution of P is also feasible for P', and has the same solution value.*

- The solution value of a relaxation provides an **upper bound**.

Theorem

Consider a problem P and its relaxation P'. If the optimal solution to P' is feasible for P, it is also optimal for P.

- Classical relaxations are:
  - Lagrangian relaxation
  - LP relaxation

Lagrangian relaxation

- ILP:

$$\begin{aligned} \max \mathbf{c}^T \mathbf{x} & \\ A\mathbf{x} &\leq \mathbf{b} \\ D\mathbf{x} &\leq \mathbf{d} \\ \mathbf{x} &\in \mathbb{N}^n. \end{aligned}$$

- Lagrangian function:

$$\begin{aligned} \theta(\lambda) = \max \mathbf{c}^T \mathbf{x} &+ \lambda^T(\mathbf{d} - D\mathbf{x}) \\ A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\in \mathbb{N}^n. \end{aligned}$$

✓ The Lagrangian relaxation is the upper bound

$$\min_{\lambda \geq 0} \theta(\lambda).$$

LP relaxation

- ILP:

$$\begin{aligned} \max \mathbf{c}^T \mathbf{x} & \\ A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\in \mathbb{N}^n. \end{aligned}$$

- LP relaxation:

$$\begin{aligned} \max \mathbf{c}^T \mathbf{x} & \\ A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\in \mathbb{R}^n. \end{aligned}$$

✓ The LP relaxation provides an upper bound.

Branch-and-Bound:

Branching

Branching

Branching is the splitting up of feasible space as follows:

- Define the **feasible space** $X = \{\mathbf{x} \in \mathbb{N}^n : A\mathbf{x} \leq \mathbf{b}\}$.
- **Split up** the feasible space into $\{X_1, \ldots, X_k\}$, such that
  - $X_i \subset X$, for all $i \in \{1, \ldots, k\}$,
  - $X_i \bigcap X_j = \emptyset$, for all $i, j \in \{1, \ldots, k\}$, $i \neq j$,
  - $\bigcup_{i=1}^{n} X_i = X$.
- Denote
$$z_i = \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X_i\}$$
- It follows that
$$z = \max_{i \in \{1,\ldots,k\}} \{z_i\}.$$
- The solution to the relaxation using X should be **infeasible** for all relaxations using X_i, $i \in \{1, \ldots, k\}$.

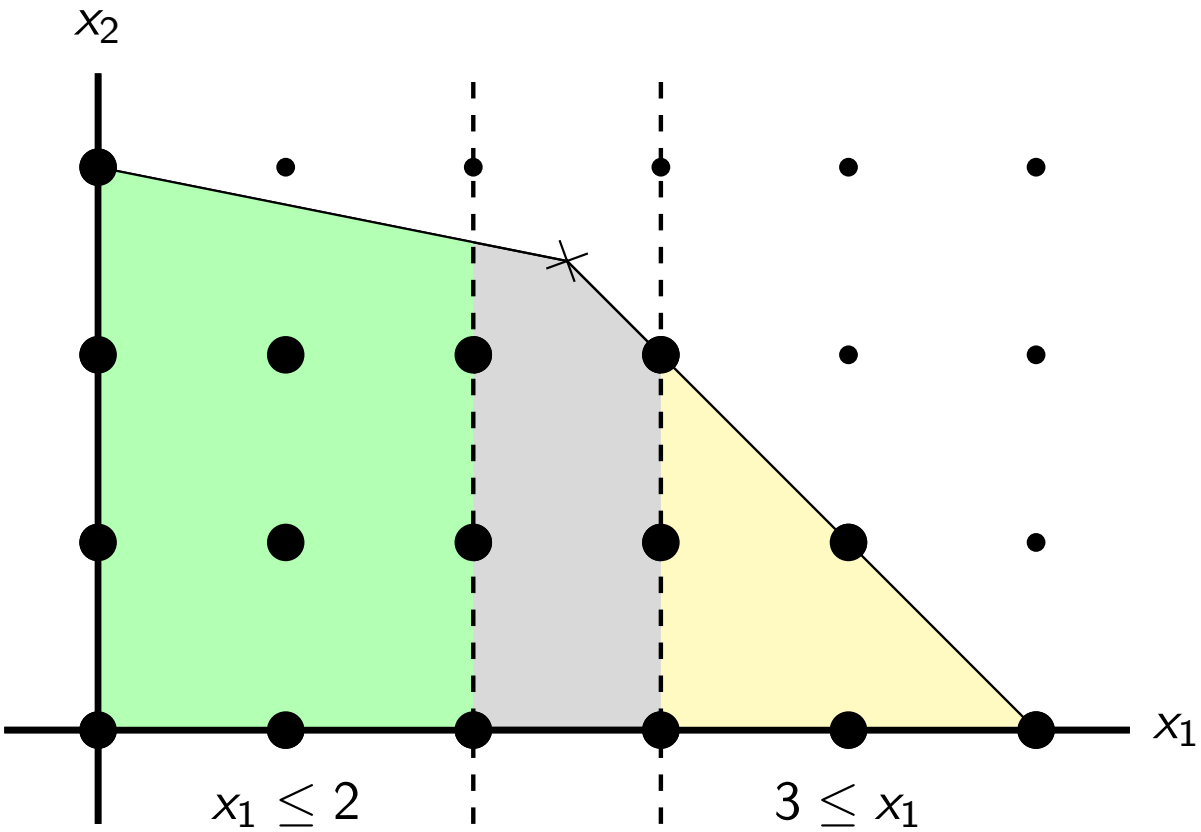
Branching - example

$$\begin{array}{rrrrrr} \max & x_1 & + & 3x_2 & & \\ & x_1 & + & x_2 & \leq & 5 \\ & x_1 & + & 5x_2 & \leq & 15 \\ & x_1 & , & x_2 & \in & \mathbb{N}_+ \end{array}$$

- The LP relaxation has solution $x_1 = 2.5$, $x_2 = 2.5$.
- This solution is not feasible (it is not integer).
- We could branch as follows:

$$X_1 = \{x \in \mathbb{N}_+^2 : x_1 + x_2 \leq 5, x_1 + 5x_2 \leq 15, x_1 \leq 2\}$$
$$X_2 = \{x \in \mathbb{N}_+^2 : x_1 + x_2 \leq 5, x_1 + 5x_2 \leq 15, x_1 \geq 3\}$$

Branching - example



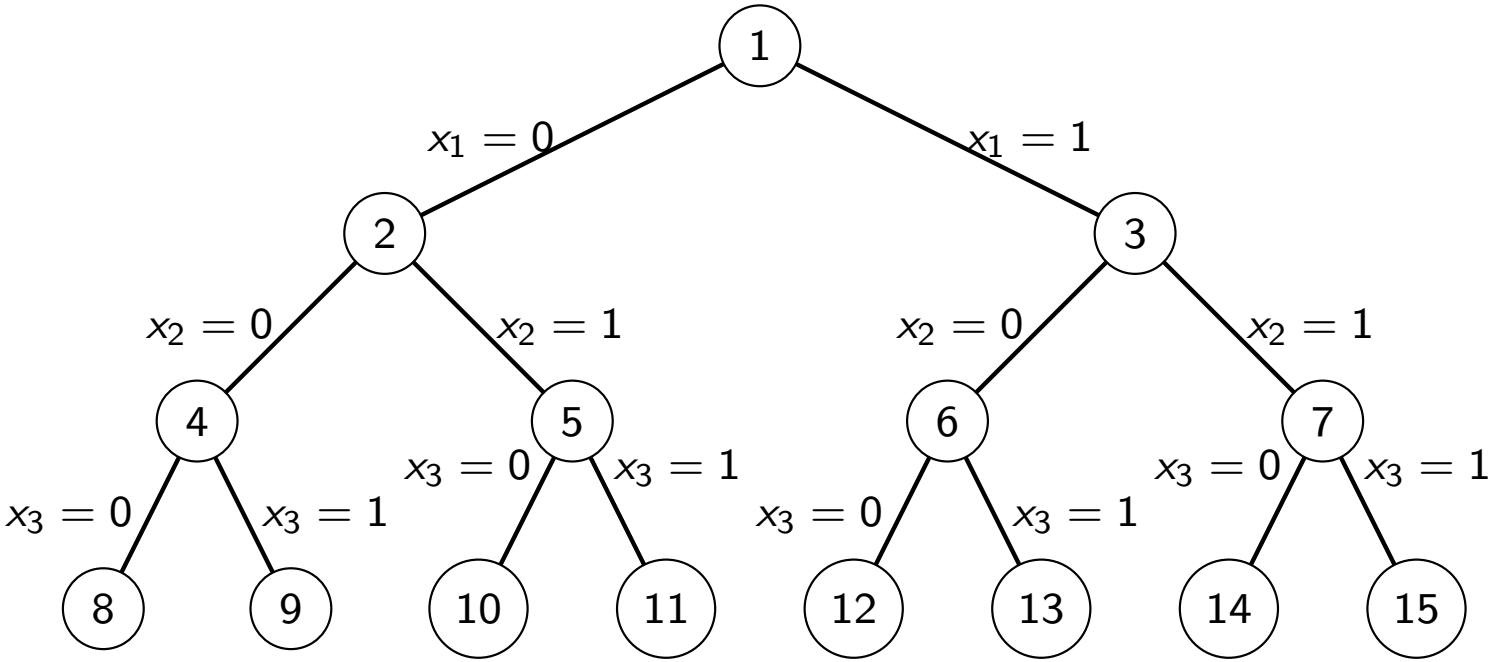
Branching

Branching tree:
(Also known as a search tree.)

- It is a **tree**.
- The **rootnode** corresponds to the ILP.
- The **child nodes** of each node are obtained by branching, i.e., they split up the corresponding problem.

Branching

Branching tree example: $x \in \{0,1\}^3$



Branching

Branching rule:

- is a rule that specifies how to **branch**,
- is used to generate a branching **tree**,
- guarantees that the **optimal** solution is found in one of the **leaf nodes** of the tree.

Example:

- In each node solve the LP relaxation.
- Branching rule:
  - Select a variable with fractional value $x = \alpha$.
  - Create two children in which $x \leq \lfloor \alpha \rfloor$ and $\lceil \alpha \rceil \leq x$ are imposed.

Branch-and-Bound:

Bounding

Bounding

- ✗ Creating the entire branching tree is time consuming.
- ✗ Evaluating all nodes in the branching tree time is consuming.
- ✓ We can limit the size of the branching tree by **pruning**:
  - Prune by infeasibility.
  - Prune by optimality.
  - Prune by bound.
- Typically we:
  - solve relaxations to get **upper bounds**,
  - use 'integer' solutions to relaxations as **lower bounds**,
  - or use heuristics to get **lower bounds**.

Branch-and-Bound:

Node selection

Node selection

- **Processing** refers to solving the relaxation of a node.
- In each iteration of Branch-and-Bound, select an unprocessed node to be processed.
- Common **node selection strategies** are:
  - **Largest upper bound first**
    Select the unprocessed node with the largest upper bound.
  - **Deepest node first**
    Select the unprocessed node that is deepest in the tree, (farthest from the rootnode).

Summary of Branch-and-Bound

Branch-and-Bound is a framework.

A Branch-and-Bound algorithm is obtained by choosing:

- A relaxation,
- A branching rule,
- A node selection strategy,
- A heuristic for lower bounds (not necessary).

Summary of Branch-and-Bound

Step 0 **Initialize** branching tree.

Step 1 **Terminate** if no unprocessed nodes remain.

Step 2 **Select** an unprocessed node.

Step 3 **Bound**.

Step 4 **Prune**

- If the current node is pruned by infeasibility or optimality, return to step 1.

Step 5 **Branch** and return to step 1.

Outlook

Outlook

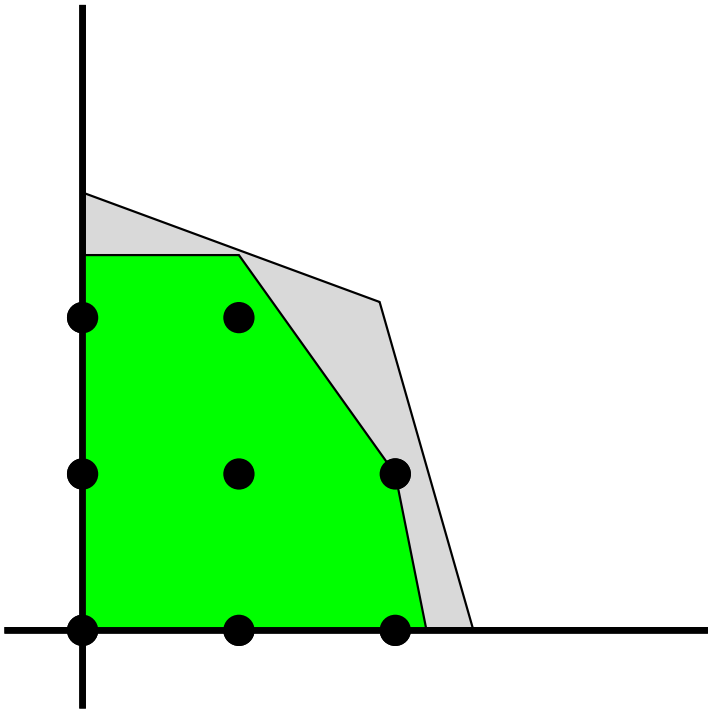
! The relaxation impacts the effectiveness of branch-and-bound:
- the **speed** of computing the relaxed solution,
- the **strength** of the bound.

x Never say:

I made some ILP and my solver cannot find a solution fast, so heuristics are necessary.

Outlook

- For (general) ILPs, the LP relaxation is important.
- The bound of the LP relaxation is called **LP bound**.



Outlook

In this course we study

- the strength of LP relaxations,
- techniques to strengthen LP relaxations,
- algorithms to efficiently compute strong LP bounds,
- easy and difficult ILPs.