Integer Programming

Branch & Bound – Foundations and Algorithm

English

Wolfgang Boehme

wolfgang.boehme@economic-modeling.de

www.economic-modeling.de

25 October 2026

This document provides a systematic introduction to integer programming. Prerequisites: basic knowledge of the Simplex method.

The documentation on this website was developed with the support of the AI Claude (Anthropic).

Table of Contents

1 Introduction and Motivation **2**

2 Fundamental Concepts **2**
- 2.1 Types of Integer Programs . . . 2
- 2.2 LP Relaxation . . . 2
- 2.3 Why Rounding Does Not Work . . . 3

3 The Branch & Bound Algorithm **3**
- 3.1 Basic Principle . . . 3
- 3.2 Complete Algorithm . . . 4
- 3.3 Node Selection and Variable Choice Strategies . . . 4

4 Complete Example **5**
- 4.1 Problem Statement . . . 5
- 4.2 Node 0: LP Relaxation . . . 5
- 4.3 Node 1: $x_1 \leq 1$ . . . 6
- 4.4 Node 2: $x_1 \geq 2$ . . . 6
- 4.5 Node 3: $x_1 \geq 2$, $x_2 \leq 2$ . . . 6
- 4.6 Node 4: $x_1 \geq 2$, $x_2 \geq 3$ . . . 6
- 4.7 Node 5: $x_1 \leq 2$, $x_2 \leq 2$ (and $x_1 \geq 2$) . . . 6
- 4.8 Node 6: $x_1 \geq 3$, $x_2 \leq 2$ . . . 6
- 4.9 Node 7: $x_1 \geq 3$, $x_2 = 0$ . . . 6
- 4.10 Node 8: $x_1 \geq 3$, $x_2 \geq 1$, $x_2 \leq 2$ . . . 7
- 4.11 Summary: Branch & Bound Tree . . . 7
- 4.12 Optimal Solution . . . 7

5 Correctness and Convergence **7**

6 Special Structures: Binary Programs **8**
- 6.1 0-1 Knapsack Problem . . . 8
- 6.2 Cutting Planes . . . 8

7 Branch & Bound + Cutting Planes: Branch & Cut **8**

8 Complexity and Practical Remarks **9**
- 8.1 NP-Hardness . . . 9
- 8.2 Practical Software . . . 9

9 Summary **10**

1 Introduction and Motivation

In vielen realen Optimierungsproblemen können die Entscheidungsvariablen keine beliebigen reellen Werte annehmen, sondern müssen **ganzzahlig** sein. Typische Beispiele:

Typical Applications of Integer Programming

- **Facility location:** Should a factory be built at a given site or not? ($x_i \in \{0, 1\}$)
- **Knapsack problem:** How many units of an item should be packed? ($x_i \in \mathbb{Z}_{\geq 0}$)
- **Workforce planning:** How many employees are assigned to shift i? ($x_i \in \mathbb{Z}_{\geq 0}$)
- **Production planning:** How many whole machines are purchased? ($x_i \in \mathbb{Z}_{\geq 0}$)
- **Vehicle routing:** Is route j used or not? ($x_j \in \{0, 1\}$)

Simply rounding the LP solution does generally **not** yield the optimal integer solution and can even produce infeasible solutions. A specialised algorithm is therefore required: **Branch & Bound**.

2 Fundamental Concepts

2.1 Types of Integer Programs

Definition: Integer Programs

- **Integer linear program (ILP):** All variables integer.

$$\max \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}, \quad \mathbf{x} \in \mathbb{Z}^n$$

- **Mixed-integer linear program (MILP):** Only some variables are integer.

$$\max \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \mathbf{y} \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{y} \leq \mathbf{b}, \quad \mathbf{x} \in \mathbb{Z}^p, \ \mathbf{y} \in \mathbb{R}^q$$

- **Binary integer program (BIP):** All variables binary ($x_i \in \{0, 1\}$).

2.2 LP Relaxation

Definition: LP Relaxation

The **LP relaxation** of an ILP is obtained by dropping the integrality constraints:

$$\text{(LP-R)} \quad \max \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}$$

Theorem 1: Bound from LP Relaxation

For every ILP: the optimal value of the LP relaxation z^*_{LP} is an **upper bound** for the ILP optimal value z^*_{ILP}:

$$z^*_{ILP} \leq z^*_{LP}$$

Proof. Every integer feasible solution is also LP-feasible (since integrality is a stronger requirement). The LP optimises over a larger feasible set, so its optimum is at least as large. □

Integrality gap: The difference $z^*_{LP} - z^*_{ILP}$ is called the **integrality gap**. The larger this gap, the harder the problem typically is.

2.3 Why Rounding Does Not Work

Counterexample: Rounding Yields Incorrect Results

$\max\ z = 5x_1+4x_2 \quad \text{s.t.} \quad 6x_1+4x_2 \leq 21, \quad x_1+2x_2 \leq 7, \quad x_1, x_2 \geq 0,\ x_1, x_2 \in \mathbb{Z}$

LP relaxation: $x_1^* = 2{,}5$, $x_2^* = 2{,}25$, $z^*_{LP} = 21{,}5$
Rounded: $\lfloor x_1 \rfloor = 2$, $\lfloor x_2 \rfloor = 2 \Rightarrow z = 18$ (zulässig, aber nicht optimal!)
True optimum: $x_1^* = 3$, $x_2^* = 0$, $z^*_{ILP} = 15$ *oder besser:* $x_1^* = 1$, $x_2^* = 3$, $z = 17$
Tatsächlich: $x_1^* = 3$, $x_2^* = 0$ mit $z = 15$ oder $x_1^* = 0$, $x_2^* = 3$ mit $z = 12$ – das gerundete Ergebnis $(2, 2)$ mit $z = 18$ ist zufällig besser, aber es ist nicht das wahre Optimum. Das wahre Optimum liegt bei $x_1 = 2$, $x_2 = 2$, $z = 18$ – in diesem Fall stimmt es zufällig, aber das ist nicht garantiert!

3 The Branch & Bound Algorithm

3.1 Basic Principle

Branch & Bound (B&B) combines three ideas:

1. **Relaxation (Bound):** Solve the LP to obtain an upper bound for the subtree.
2. **Branching (Branch):** Split the problem into two subproblems by adding a constraint.
3. **Pruning (Cut/Prune):** Discard subproblems whose upper bound does not exceed the best known integer solution.

Definition: Branching Variable and Branching

Let $\mathbf{x}^*$ be the LP-optimal solution of a subproblem and x_j^* a fractional variable ($x_j^* \notin \mathbb{Z}$). We **branch on** x_j:

- **Left subtree:** add $x_j \leq \lfloor x_j^* \rfloor$

- **Right subtree:** add $x_j \geq \lceil x_j^* \rceil$

Every integer solution satisfies exactly one of these two conditions.

3.2 Complete Algorithm

Algorithm: Branch & Bound (Maximization)

1. **Initialization:** Set $z^* = -\infty$, $\mathbf{x}^* = \emptyset$. Add the original LP as node 0 to the queue.
2. **Node selection:** Choose a node from the queue (e.g. Best-First or Depth-First).
3. **Solve LP:** Solve the LP relaxation of the node.
4. **Case analysis:**
   (a) **LP infeasible:** Discard node. *Pruned (infeasible).*
   (b) **LP bound $\leq z^*$:** Discard node. *Pruned (bound).*
   (c) **LP solution integer and LP bound $> z^*$:** Update z^* and $\mathbf{x}^*$. *New best solution.*
   (d) **LP solution fractional:** Choose fractional variable x_j, create two child nodes. *Branch.*
5. If queue empty: $\mathbf{x}^*$ is optimal. STOP. Otherwise: go to Step 2.

3.3 Node Selection and Variable Choice Strategies

Node selection:

- **Best-First Search:** Choose the node with the highest upper bound. Reaches the optimum quickly but requires more memory.
- **Depth-First Search:** Follow the deepest node. Memory-efficient, finds feasible solutions quickly.
- **Best-First + Backtracking:** Combination of both strategies.

Variable choice (branching rules):

- **Most fractional:** Choose the variable whose value is closest to 0.5.
- **Least fractional:** Choose the variable with the smallest fractional part.
- **Pseudo-cost branching:** Estimate improvement per branch from previous iterations.

4 Complete Example

4.1 Problem Statement

Example: Production Planning with Integer Quantities

A company produces two products P_1 and P_2.

$$\max\ z = 5x_1 + 4x_2$$

$$\begin{aligned} 6x_1 + 4x_2 &\leq 21 \quad \text{(machine capacity)} \\ x_1 + 2x_2 &\leq 7 \quad \text{(storage capacity)} \\ x_1, x_2 &\geq 0, \quad x_1, x_2 \in \mathbb{Z} \end{aligned}$$

4.2 Node 0: LP Relaxation

We first solve the LP relaxation (without integrality constraints).

Initial tableau (slack variables s_1, s_2):

Basis	x_1	x_2	s_1	s_2	b
s_1	6	4	1	0	21
s_2	1	2	0	1	7
z	-5	-4	0	0	0

Iteration 1: Pivot column x_1 ($\bar{c}_1 = -5$). Ratios: $21/6 = 3.5$, $7/1 = 7$. Pivot row: s_1. Pivot element: 6.

Tableau after Iteration 1:

Basis	x_1	x_2	s_1	s_2	b
x_1	1	2/3	1/6	0	7/2
s_2	0	4/3	$-1/6$	1	7/2
z	0	$-2/3$	5/6	0	35/2

Iteration 2: Pivot column x_2. Ratios: $(7/2)/(2/3) = 5.25$, $(7/2)/(4/3) = 2.625$. Pivot row: s_2. Pivot element: 4/3.

Optimal tableau (Node 0):

Basis	x_1	x_2	s_1	s_2	b
x_1	1	0	1/4	$-1/2$	7/4
x_2	0	1	$-1/8$	3/4	21/8
z	0	0	3/4	1/2	**21,5**

LP solution: $x_1^* = 7/4 = 1{,}75$, $x_2^* = 21/8 = 2{,}625$, $z_{LP}^* = 21{,}5$
Both variables are **fractional** $\Rightarrow$ Branch & Bound required.
We branch on x_1 (most fractional: $1.75 - 1 = 0.75 > 2.625 - 2 = 0.625$):

Knoten 1: $x_1 \leq 1$ Knoten 2: $x_1 \geq 2$

4.3 Node 1: $x_1 \leq 1$

Additional constraint: $x_1 \leq 1$, i.e. $x_1 + s_3 = 1$.
LP solution Node 1: With $x_1 \leq 1$ the Simplex yields: $x_1 = 1$, $x_2 = 3$, $z = 17$.
Both variables are integer!
New best solution: $z^* = 17$, $x_1^* = 1$, $x_2^* = 3$.

4.4 Node 2: $x_1 \geq 2$

Additional constraint: $x_1 \geq 2$.
LP solution Node 2: $x_1 = 2$, $x_2 = 2{,}25$, $z = 19$.
$x_2 = 2.25$ is fractional $\Rightarrow$ branch on x_2:

Knoten 3: $x_2 \leq 2$ Knoten 4: $x_2 \geq 3$

4.5 Node 3: $x_1 \geq 2$, $x_2 \leq 2$

LP solution: $x_1 = 2{,}5$, $x_2 = 2$, $z = 20{,}5$
$x_1 = 2.5$ fractional $\Rightarrow$ branch on x_1:

Knoten 5: $x_1 \leq 2$ Knoten 6: $x_1 \geq 3$

4.6 Node 4: $x_1 \geq 2$, $x_2 \geq 3$

LP solution: $x_1 = 5/4$, $x_2 = 3$ – aber $x_1 \geq 2$ und $x_1 = 1{,}25 < 2$ $\Rightarrow$ **Unzulässig!**
Node 4 is discarded.

4.7 Node 5: $x_1 \leq 2$, $x_2 \leq 2$ (and $x_1 \geq 2$)

Damit $x_1 = 2$ (fixiert). **LP solution:** $x_1 = 2$, $x_2 = 2$, $z = 18$.
Beide ganzzahlig! $z = 18 > 17 = z^*$ $\Rightarrow$ **New best solution:** $z^* = 18$.

4.8 Node 6: $x_1 \geq 3$, $x_2 \leq 2$

LP solution: $x_1 = 3$, $x_2 = 3/4$, $z = 18$.
$x_2 = 0.75$ fractional $\Rightarrow$ branch on x_2:

Knoten 7: $x_2 \leq 0$ Knoten 8: $x_2 \geq 1$

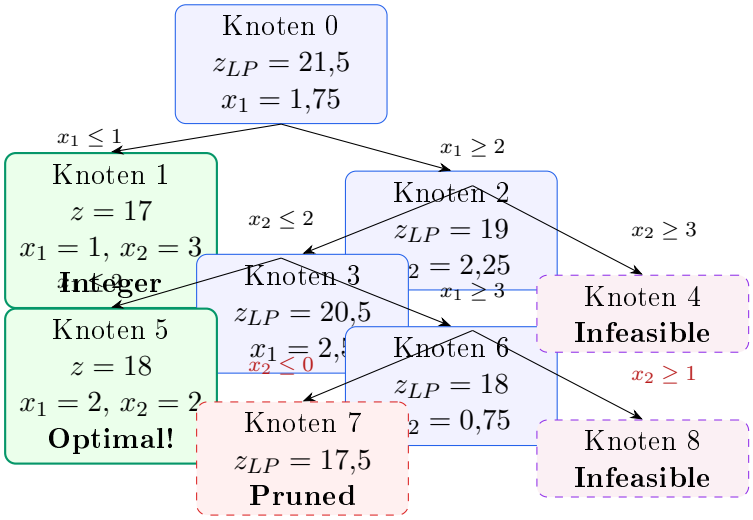
4.9 Node 7: $x_1 \geq 3$, $x_2 = 0$

LP solution: $x_1 = 3{,}5$, $x_2 = 0$, $z = 17{,}5$
$z = 17.5 \leq z^* = 18$ $\Rightarrow$ **Pruned!**

4.10 Node 8: $x_1 \geq 3$, $x_2 \geq 1$, $x_2 \leq 2$

Check: $6(3) + 4(1) = 22 > 21 \Rightarrow$ **Infeasible!**

4.11 Summary: Branch & Bound Tree



Colour code:

- Blue: LP solution, still fractional
- Green: Integer solution found
- Red dashed: Pruned (bound $\leq z^*$)
- Purple dashed: Infeasible

4.12 Optimal Solution

$$\boxed{x_1^* = 2, \quad x_2^* = 2, \quad z_{ILP}^* = 18}$$

Integrality gap: $z_{LP}^* - z_{ILP}^* = 21{,}5 - 18 = 3{,}5$ (16,3%)

5 Correctness and Convergence

Theorem 2: Correctness of Branch & Bound

The Branch-&-Bound algorithm finds the optimal integer solution (or establishes infeasibility), provided:

1. Every node is correctly evaluated by LP solution,
2. The branching is exhaustive (every integer solution lies in exactly one subtree),
3. The tree is fully evaluated (or correctly pruned).

Proof. By complete induction on the depth of the B&B tree. Every integer solution $\mathbf{x} \in \mathbb{Z}^n$ lies in exactly one leaf of the fully developed tree. Since $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lceil x_j^* \rceil$ cover all integer values disjointly, no feasible integer solution is missed. Pruning only discards nodes whose LP bound cannot exceed the best known integer solution – such nodes cannot contain a better solution. □

Theorem 3: Finiteness of Branch & Bound

For a bounded feasible region and integer coefficients, B&B terminates in finitely many steps.

Proof sketch. The tree depth is bounded by the number of branching variables. For a bounded feasible region there are only finitely many integer points. Pruning drastically reduces the number of nodes to examine. □

6 Special Structures: Binary Programs

6.1 0-1 Knapsack Problem

The classical **0-1 knapsack problem** is an important special case:

$$\max \sum_{j=1}^{n} c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{n} a_j x_j \leq b, \quad x_j \in \{0,1\}$$

Bound via LP relaxation: Sort items by c_j/a_j (descending) and fill the knapsack greedily – the last item is packed fractionally (greedy bound).

6.2 Cutting Planes

An important complement to B&B is **cutting planes**, which tighten the feasible region of the LP relaxation without excluding integer solutions.

Definition: Gomory Cut

Let row i of the optimal tableau be $x_{B_i} = \bar{b}_i - \sum_j \bar{a}_{ij} x_j$ with $\bar{b}_i \notin \mathbb{Z}$. The **Gomory cut** reads:

$$\sum_j \{\bar{a}_{ij}\} x_j \geq \{\bar{b}_i\}$$

where $\{v\} = v - \lfloor v \rfloor$ denotes the fractional part.

Gomory cuts guarantee that after finitely many cuts, the LP relaxation yields the optimal integer solution as its LP solution.

7 Branch & Bound + Cutting Planes: Branch & Cut

Modern ILP solvers combine B&B with cutting planes into the **Branch-&-Cut algorithm**:

1. Solve LP relaxation at the node.

2. Add cutting planes (Gomory, clique cuts, etc.).
3. If LP solution is integer: update best solution.
4. If LP bound is too weak: prune.
5. Otherwise: branch.

Branch & Cut is the foundation of all modern ILP solvers (CPLEX, Gurobi, SCIP).

8 Complexity and Practical Remarks

8.1 NP-Hardness

Integer linear programs are in general **NP-hard**: no polynomial-time algorithm is known. In the worst case the number of B&B nodes grows exponentially.

Nevertheless, modern solvers (CPLEX, Gurobi) solve problems with millions of variables in reasonable time – thanks to:

- **Strong LP relaxation** through many cutting planes
- **Heuristics** for quickly finding good starting solutions
- **Presolving:** simplification of the problem before solving
- **Parallelisation** of the B&B tree

8.2 Practical Software

- **CPLEX, Gurobi:** Commercial solvers, very fast
- **SCIP, CBC (COIN-OR):** Open-source alternatives
- **Python:** `scipy.optimize.milp`, `PuLP`, `OR-Tools`
- **MATLAB:** `intlinprog`

9 Summary

Key Points: Integer Programming

1. **Integrality constraints** arise when variables can only take integer values.
2. The **LP relaxation** provides an upper bound; simple rounding does not yield the optimum.
3. **Branch & Bound** solves the ILP by systematic branching and pruning.
4. **Bound:** LP relaxation of each subproblem provides an upper bound.
5. **Branch:** Branch on fractional variable by adding two new constraints.
6. **Prune:** Nodes with bound $\leq z^*$ are discarded.
7. **Integrality gap** $= z^*_{LP} - z^*_{ILP}$ measures the difficulty of the problem.
8. **Cutting planes** (Gomory) tighten the LP bound without excluding integer solutions.
9. **Branch & Cut** combines B&B with cutting planes – the foundation of modern ILP solvers.
10. ILP is in general **NP-hard**; modern solvers nonetheless solve large instances efficiently.
