Integer Programming

A Introduction

Integer programming (IP) is a branch of mathematical optimization where the variables are restricted to take integer values. This distinguishes it from linear programming (LP), where variables are allowed to be real numbers. Integer programming problems can be formulated to solve a wide range of real-world decision-making problems where decisions need to be made in whole units, such as selecting projects, scheduling tasks, or assigning resources.

Types of Integer Variables:

- **Binary Variables:** Take values 0 or 1 (e.g., yes/no decisions).
- **Integer Variables:** Take integer values within a specified range.
- **Mixed Integer Variables:** A combination of both binary and integer variables.

Types of Integer Programming problems:

- **Total (or Pure) Integer Model:** All the decision variables are integer.
- **0-1 Integer Model (or Binary Model):** The decision variables can only take the values 0 or 1.
- **Mixed Integer Model:** Some decision variables are integer and others can be non-integer (fractional).

Integer programming problems belong to the class of NP-hard problems, meaning that finding an optimal solution is computationally intractable in the worst case. As the number of variables and constraints increases, the computational complexity grows exponentially. This limits the size of problems that can be solved within reasonable time and resource constraints. Also, verifying the feasibility of integer solutions can be difficult, especially in problems where feasibility is not straightforwardly deducible from the structure of the problem. In summary, Integer programming problems are challenging to solve due to several computational complexities and theoretical reasons:

- Discrete Nature of Variables: Unlike linear programming (LP), where variables can take any real value, IP restricts variables to integer values. This discrete nature significantly complicates the search for optimal solutions because feasible solutions are no longer continuous.
- Combinatorial Explosion: The number of possible combinations of integer solutions increases exponentially with the number of integer variables. This leads to a vast search space that grows rapidly, making it computationally expensive to explore all possible solutions.
- Non-Convexity: Integer programming problems are generally non-convex due to the presence of integer constraints. This non-convexity introduces challenges in finding globally optimal solutions, as local optima may exist that are not globally optimal.

- Branch and Bound Complexity: The branch and bound algorithm, commonly used to solve IP problems, involves recursively partitioning the solution space into smaller regions (branches). Each branch requires solving an LP relaxation and potentially adding cutting planes, which adds computational overhead.

The combination of discrete variables, combinatorial explosion, non-convexity, and NP-hardness makes integer programming problems inherently challenging to solve. Effective solutions often require a blend of algorithmic sophistication, computational power, and problem-specific insight.

B Some Examples

B.1 An Assignment Problem

A company has 4 machines available for assignment to 4 tasks. Any machine can be assigned to any task, and each task requires to be processed by one machine. The time required to set up each machine for the processing of each task is given in the table below.

	Time (hrs)			
	Task 1	Task 2	Task 3	Task 4
Machine 1	13	4	7	6
Machine 2	1	11	5	4
Machine 3	6	7	2	8
Machine 4	1	3	5	9

The company wants to minimize the total setup time needed for the processing of all four tasks.

The Model:

$$\text{Min } z = 13x_{11} + 4x_{12} + 7x_{13} + 6x_{14} + 1x_{21} + 11x_{22} + 5x_{23} + 4x_{24} + 6x_{31} + 7x_{32} + 2x_{33} + 8x_{34} + x_{41} + 3x_{42} + 5x_{43} + 9x_{44}$$

s.t.

$$\left.\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} &= 1 \\ x_{21} + x_{22} + x_{23} + x_{24} &= 1 \\ x_{31} + x_{32} + x_{33} + x_{34} &= 1 \\ x_{41} + x_{42} + x_{43} + x_{44} &= 1 \end{aligned}\right\} \text{(Assignment of machines to tasks)}$$

$$\left.\begin{aligned} x_{11} + x_{21} + x_{31} + x_{41} &= 1 \\ x_{12} + x_{22} + x_{32} + x_{42} &= 1 \\ x_{13} + x_{23} + x_{33} + x_{43} &= 1 \\ x_{14} + x_{24} + x_{34} + x_{44} &= 1 \end{aligned}\right\} \text{(Assignment of tasks to machines)}$$

$x_{ij} \in \{0, 1\}$, for $1 \leq i \leq 4$ and $1 \leq j \leq 4$

where, for $1 \leq i \leq 4$ and $1 \leq j \leq 4$, x_{ij}= 1 if machine i is assigned to task j, 0 otherwise.

Suppose that n represents the number of machines (in our example, $n = 4$. The number of integer solutions is $n!$. The table below shows how fast the number of solutions grows with the number of machines:

n	$n!$
3	6
5	120
10	3628800
100	9.33×10^{157}
1000	4.02×10^{2567}

B.2 A Knapsack Problem

Consider a set of items each of which has a predefined weight and a monetary value (profit) as described in the table below. These items have to be packed into a container whose maximum weight is limited to 11.3 tons. The problem consists of choosing a subset of items that:

- Fit into the bag considering the weight limit and
- Yield a maximum total profit

Product	Weight	Profit
1	3.2 tons	$727
2	4.0 tons	$763
3	4.4 tons	$60
4	2.0 tons	$606
5	0.1 tons	$45
6	2.9 tons	$370
7	0.3 tons	$414
8	1.3 tons	$880
9	0.6 tons	$133
10	3.9 tons	$820

The Model:

$$\begin{aligned} \text{Max } z = \quad & 727x_1 + 763x_2 + 60x_3 + 606x_4 + 45x_5 + 370x_6 + 414x_7 + 880x_8 + 133x_9 + 820x_{10} \\ \text{s.t.} \quad & 3.2x_1 + 4.0x_2 + 4.4x_3 + 2.0x_4 + 0.1x_5 + 2.9x_6 + 0.3x_7 + 1.3x_8 + 0.6x_9 + 3.9x_{10} \leq 11.3 \\ & x_i \in \{0, 1\} \quad \text{for } i = 1, 2, \ldots, 10 \end{aligned}$$

where, for $i = 1, 2, \ldots, 10$, x_j= 1 if item j is chosen, 0 otherwise.

n	2^n
10	1024
20	1048576
30	1073741824
40	1099511627776
50	1125899906842624
60	1152921504606846976
70	1180591620717411303424
80	$1208925819614629174706176$
90	$1237940039285380274899124224$
100	$1267650600228229401496703205376$