LINEAR INTEGER PROGRAMMING

BRANCH AND BOUND

Internship

Professor Giovanni Finocchio

Report by Mujibullah Rabin

JULY 26, 2024

UNIVERSITY OF MESSINA

Contents:

Linear integer programming (LIP)

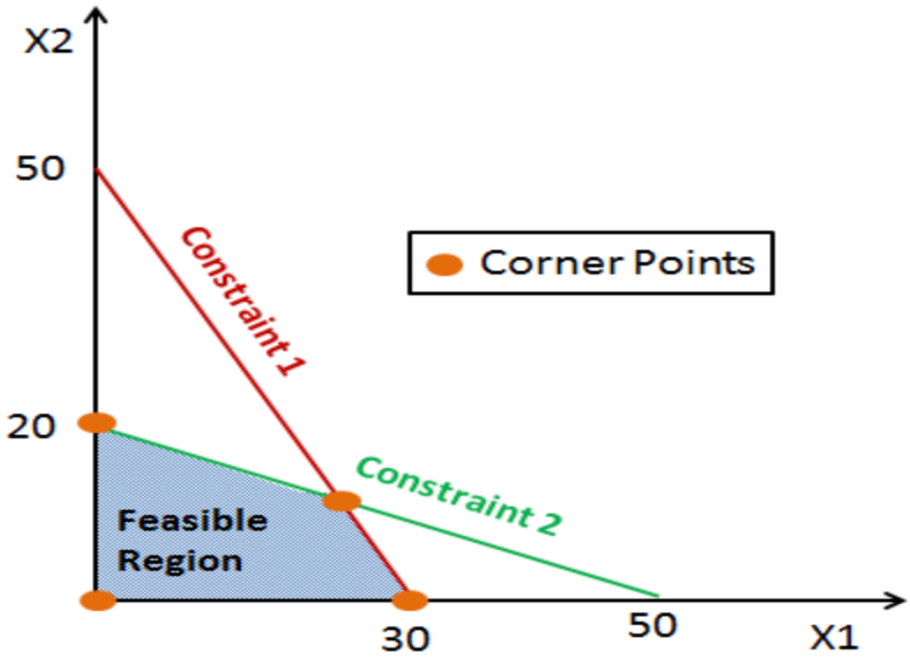
Linear programming (LP)

Linear programming (**LP**), also called **linear optimization**, is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements and objective are represented by linear relationship

Important terminologies of the LP

- **Decision Variables:** The decision variables are the variables that will decide my output. They represent my ultimate solution. To solve any problem, we first need to identify the decision variables.
- **Objective Function:** It is defined as the objective of making decisions.
- **Constraints:** The constraints are the restrictions or limitations on the decision variables. They usually limit the value of the decision variables.

Here is an example of linear programming



We can solve the linear programming in two methods:

1. Graphical Method
2. Simplex method

And we can solve the linear programming by Python, R, Matlap and some other programming languages.

Integer programming

An integer programming problem is a mathematical optimization or feasibility program in which some or all the variables are restricted to be integers. In many settings the term refers to integer linear programming (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

And integer programming is a powerful problem-solving tool used in fields such as economics and operations research. It involves the optimisation of linear equations by assigning integer values to each variable, thus allowing for optimal solutions that are integral rather than fractional.

What is difference between linear programming and integer programming:

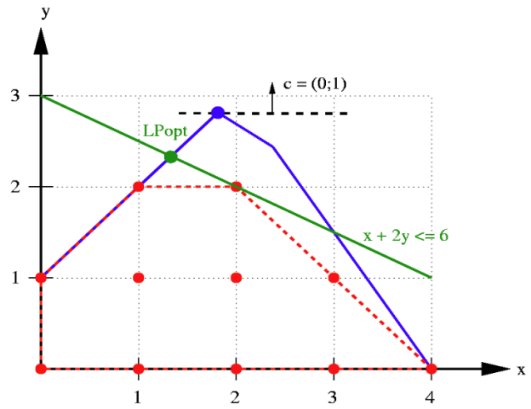
- The primary difference between integer programming and linear programming is the constraint on the variables. In linear programming, the variables are allowed to take on any real number value that satisfies the constraints. In integer programming, some or all the variables are constrained to take on only integer values.

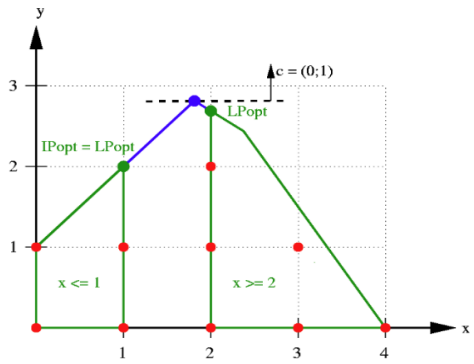
  Linear programming involves optimizing a linear objective function subject to linear constraints, where the variables can be any real numbers.

  Integer programming involves optimizing an objective function (which may be linear or nonlinear) subject to constraints, where some or all the variables must be integers.

Linear integer programming (LIP)

Since we know in LP where variables can be any number and, in the IP, where the variables should be integer. And in linear integer programming the objective function and constraints are linear, and variables are integers.





As The diagrams shows illustrate solutions to a linear integer programming (LIP) problem, typically involving the optimization of a linear objective function subject to linear equality and inequality constraints, where some or all variables are constrained to be integers.

- **Objective Function**: Both diagrams are optimizing the objective function in the direction of c=(0,1)c = (0,1)c=(0,1), which means they are maximizing y.
- **Constraints**: Linear and integer constraints define the feasible region.
- Integer Solutions: Only integer points within the feasible region are considered as potential solutions for the integer programming problem.
- **Optimal Solutions**: The optimal integer points (LIPopt) are highlighted and compared with the linear program's optimal solutions.

What can for all these can be used in our real life:

Linear programming is used in business and industry in production planning, transportation and routing, and various types of scheduling. Airlines use linear programs to schedule their flights, considering both scheduling aircraft and scheduling staff. Delivery services use linear programs to schedule and route shipments to minimize shipment time or minimize cost. Retailers use linear programs to determine how to order products from manufacturers and organize deliveries with their stores. Manufacturing companies use linear programming to plan and schedule production. Financial institutions use linear programming to determine the mix of financial products they offer, or to schedule payments transferring funds between institutions. Health care institutions use linear programming to ensure the proper supplies are available when needed. And as we'll see below, linear programming has also been used to organize and coordinate lifesaving health care procedures.

Linear integer programming with python

For this we can use different libraries where pulp is one of the most used and easy libraries.

```
from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable
```

Importing necessary libraries

And here is the example of the equation where we will find the maximum using the LP with python

$$
\begin{aligned}
\text{maximize} \quad & z = x + 2y \\
\text{subject to:} \quad & 2x + y \leq 20 \\
& -4x + 5y \leq 10 \\
& -x + 2y \geq -2 \\
& -x + 5y = 15 \\
& x \geq 0 \\
& y \geq 0
\end{aligned}
$$

Here is the example equation

Now the The first step is to initialize an instance of LpProblem to represent your model:

Python

```
# Create the model
model = LpProblem(name="small-problem", sense=LpMaximize)
```

Once we have the model, you can define the decision variables as instances of the LpVariable class:

Python

```
# Initialize the decision variables
x = LpVariable(name="x", lowBound=0)
y = LpVariable(name="y", lowBound=0)
```

Now we will get the instances by multiplying a decision variable with a scalar or build a linear combination of multiple decision variables.

```
Python
>>> expression = 2 * x + 4 * y
>>> type(expression)
<class 'pulp.pulp.LpAffineExpression'>

>>> constraint = 2 * x + 4 * y >= 8
>>> type(constraint)
<class 'pulp.pulp.LpConstraint'>
```

And now we will create the constraints and objective function as well as to assign them to our model

```
Python
# Add the constraints to the model
model += (2 * x + y <= 20, "red_constraint")
model += (4 * x - 5 * y >= -10, "blue_constraint")
model += (-x + 2 * y >= -2, "yellow_constraint")
model += (-x + 5 * y == 15, "green_constraint")
```

And the objective function:

```
Python
# Add the objective function to the model
obj_func = x + 2 * y
model += obj_func
```

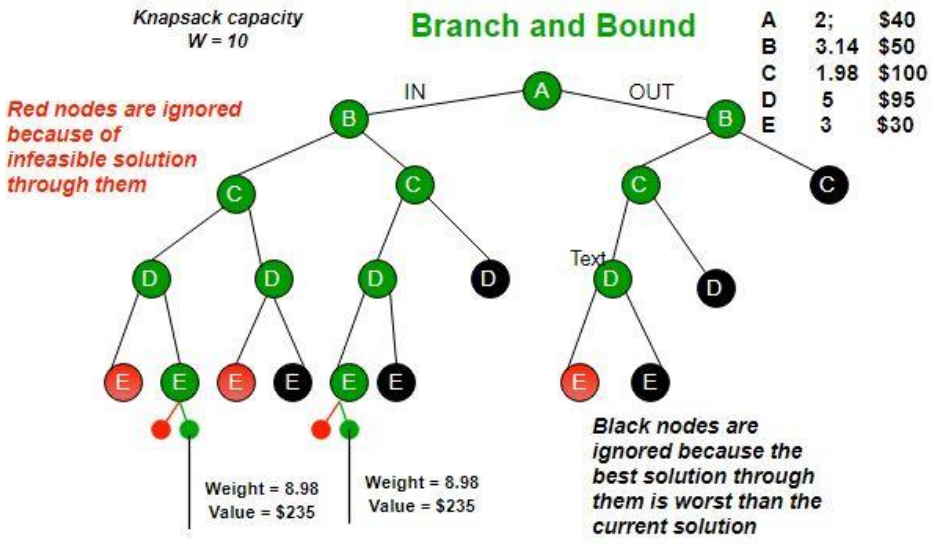
Now we put the constraint and objectives in the model so we can solve them:

```
Python
# Solve the problem
status = model.solve()
```

This is a way of solving the LP with the python.

Branch and bound

Branch and bound is an algorithm for solving optimization problems by breaking them down into smaller sub-problems and using a bounding function to eliminate sub-problems that cannot contain the optimal solution. It is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization.



A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set.

And here is an example of the solving branch and bound algorithm using python:

```
import numpy as np
from scipy.optimize import linprog

c = np.array([-5.0, -4.0])

A_ub = np.array(
    [[2.0, 3.0],
     [2.0, 1.0]]
)

b_ub = np.array([12.0, 6.0])

sol_relaxed = linprog(c, A_ub=A_ub, b_ub=b_ub)
```

- Which here it should give the output of:
  x: [1.5, 3.0]
- *fun*: -19.5

how this this branch and bound works:

- **Branching**: Splitting the problem into smaller subproblems (e.g., setting a variable to specific integer values).
  Branching Process: Divide the problem into subproblems by fixing the values of certain variables.
- **Bounding**: Calculating an upper or lower bound for the objective function in each subproblem to determine if further exploration is necessary.
  bounding for each subproblem, calculate the bound (using LP relaxation) to check if it can potentially lead to a better solution.

Steps branch and bound goes:

1. Solve the LP relaxation of the original problem (ignore integer constraints).
2. If the solution is integer, it's optimal.
3. If not, branch by setting a variable to its nearest integer values.
4. Calculate bounds for each subproblem.
5. Prune subproblems with bounds worse than the current best solution.
6. Repeat until all branches are explored or pruned.

Implementation of branch and bound using python

Here is the equation where we wil solve it using python.

Objective function = 40x+30y subject to the constraints = 2x+y≤20 second constraints = 4x+3y≤60

So here in this equation we have the constraints and objective where we will find the maximum using PULP.

```
[ ]  from pulp import LpMaximize, LpProblem, LpVariable, lpSum, PULP_CBC_CMD

[ ]  problem = LpProblem("Maximize_Profit", LpMaximize)

[ ]  x = LpVariable('x', lowBound=0, cat='Integer')
     y = LpVariable('y', lowBound=0, cat='Integer')
```

As here we installed the libraries then we specify the objective into maximum and define the decision variables.

```
[ ]  problem += 40 * x + 30 * y

     problem += 2 * x + y <= 20
     problem += 4 * x + 3 * y <= 60

[ ]  problem.solve(PULP_CBC_CMD(msg=0))
```

And here we see the objective function and add the constraints to the problem.

And by using CBC we solve the linear programming problem

```
[ ] print(f"Optimal value of x: {x.varValue}")
    print(f"Optimal value of y: {y.varValue}")
    print(f"Maximum profit: {problem.objective.value()}")

    Optimal value of x: 0.0
    Optimal value of y: 20.0
    Maximum profit: 600.0
```

This cell prints the optimal values of x and y, and the maximum profit obtained from the objective function.

```
with open("LIP_solution.txt", "w") as file:
    file.write(f"Optimal value of x: {x.varValue}\n")
    file.write(f"Optimal value of y: {y.varValue}\n")
    file.write(f"Maximum profit: {problem.objective.value()}\n")
```

This cell writes the optimal values of x and y, and the maximum profit to a text file named "LIP_solution.txt".

Solve linear integer programming using AI

Except branch and bound we can solve the linear integer programming problems by using AI (machine learning, AI and machine learning offer alternative and sometimes more efficient approaches, especially for large-scale or complex problems.

1. **Genetic Algorithms (GA)**: These are inspired by the process of natural selection and are used to find approximate solutions to optimization and search problems.
2. **Reinforcement Learning (RL)**: Reinforcement learning can be used to train an agent to make a sequence of decisions that maximize a reward function. For LIP, the RL agent can learn to find solutions by interacting with the problem environment.
3. **Neural Networks**: Neural networks can be trained to predict approximate solutions to LIP problems based on past data or problem characteristics. This approach is useful when exact solutions are not necessary, or when speed is crucial.
4. **Hybrid Approaches**: **Combining ML with Traditional Methods**: Machine learning can be used to enhance traditional optimization methods. For instance, machine learning models can predict good initial solutions or parameter settings for traditional algorithms, speeding up the convergence.
5. **Constraint Learning**: **Learning Constraints**: Machine learning can help in learning the constraints from data in scenarios where constraints are not explicitly given but can be inferred from the data.

Solving linear integer programming using machine learning:

```
[2]  import numpy as np

[3]  # Define the objective function
     def objective_function(x, y):
         return 40 * x + 30 * y
```

This line imports the NumPy library, which is used for numerical operations in Python.

And the other line defines the objective function we want to maximize: 40x+30y40x + 30y40x+30y.

```
[4]  def is_feasible(x, y):
         return (2 * x + y <= 20) and (4 * x + 3 * y <= 60)

[5]  def initialize_population(pop_size, n_vars):
         return np.random.randint(0, 20, size=(pop_size, n_vars))
```

This function checks if a solution (x,y)(x, y)(x,y) satisfies both constraints:

And the other line initializes the population with random integer values between 0 and 19.

```
[7]  def selection(population, fitness):
         indices = np.argsort(fitness)[-len(population)//2:]
         return population[indices]
```

This function selects the top 50% of the population based on fitness.

```
def crossover(parents):
    offspring = []
    for i in range(len(parents) // 2):
        parent1, parent2 = parents[i], parents[len(parents) - i - 1]
        child1 = np.array([parent1[0], parent2[1]])
        child2 = np.array([parent2[0], parent1[1]])
        offspring.extend([child1, child2])
    return np.array(offspring)
```

This function performs crossover to generate offspring. Pairs of parents are combined to create two children. For each pair, a child is created by taking the first variable from one parent and the second variable from the other parent.

```
# Main Genetic Algorithm Loop
pop_size = 100
n_vars = 2
n_generations = 1000
mutation_rate = 0.1
```

```
[11] population = initialize_population(pop_size, n_vars)
     for generation in range(n_generations):
         fitness = evaluate_population(population)
         parents = selection(population, fitness)
         offspring = crossover(parents)
         offspring = mutate(offspring, mutation_rate)
         population[:len(offspring)] = offspring
```

pop_size, n_vars, n_generations, and mutation_rate are parameters for the genetic algorithm.

```
[12] # Find the best solution
     best_solution = population[np.argmax(evaluate_population(population))]
     print("Best solution found:", best_solution)
     print("Objective value:", objective_function(best_solution[0], best_solution[1]))
```

```
Best solution found: [ 1 17]
Objective value: 550
```

After the final generation, the best solution is found by evaluating the fitness of the population and selecting the individual with the highest fitness. The best solution and its objective value are printed.

Which is 550.

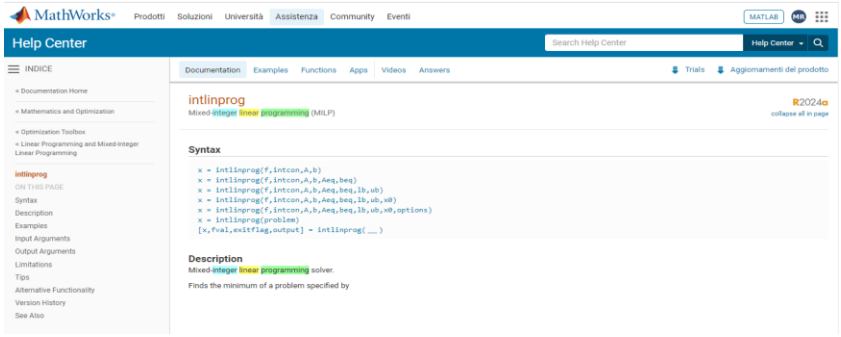
Solving linear integer programming using MATLAB

can be solved using MATLAB. MATLAB provides several tools for optimization, including functions specifically designed for integer programming problems. The primary function for solving linear integer programming problems in MATLAB is intlinprog.

Here is how we can solve using MATLAB

1. **Define the objective function**: Specify the coefficients of the linear objective function.
2. **Define the constraints**: Specify the coefficients of the linear inequality and equality constraints, as well as the bounds on the variables.
3. **Call the solver**: Use intlinprog to solve the problem.

Here in the MATLAB official website we can find about linear integer solving instruction.



The same example we solved in the python we will solve in MATLAB

First, we will open the MATLAB workspace.

```
% Define the objective function coefficients (since we want to maximize, we'll use negative coefficients for minimization)
f = [-40, -30];

% Define the inequality constraints (Ax <= b)
A = [2, 1; 4, 3];
b = [20; 60];

% Variable bounds (x >= 0)
lb = [0; 0];

% Specify integer variables
intcon = [1, 2];
```

Here **Objective Function**: In linear programming, maximization problems can be converted into minimization problems by negating the objective function coefficients. Here, the objective is to maximize 40x+30y40x + 30y40x+30y, so we negate the coefficients to get [−40,−30][-40, -30][−40,−30].

And the **Constraints**: The matrix A and vector b define the inequality constraints.

The the third line **Bounds**: The vector lb defines the lower bounds for the variables. Here, both xxx and yyy are non-negative, so the lower bounds are [0;0][0; 0][0;0].

```
% Solve the integer linear programming problem
[x, fval, exitflag, output] = intlinprog(f, intcon, A, b, [], [], lb);

% Display the results
disp('Optimal solution:')
disp(x)
disp('Maximum profit at the optimal solution:')
disp(-fval)
```

Solver: The intlinprog function is used to solve the integer linear programming problem.

Display Results: The disp function is used to display the results.

- The optimal solution x is displayed.
- The value of the objective function at the optimal solution is displayed (negated back to represent the original maximization problem).

CONCLUSION

This report has explored Linear Integer Programming (LIP), highlighting its importance in solving optimization problems with integer solutions. Combining linear programming (LP) and integer programming (IP), LIP optimizes linear objective functions under linear constraints, crucial for applications in business, manufacturing, transportation, and more.

Using Python libraries like PuLP, we demonstrated how LIP can be practically implemented. The Branch and Bound algorithm were also discussed as a powerful method for discrete optimization problems.

Advanced techniques such as AI and Machine Learning, including Genetic Algorithms and Neural Networks, offer promising solutions for complex LIP problems, emphasizing the potential of integrating traditional methods with modern computational advancements.

In summary, LIP is a vital tool in optimization, with broad applications and ongoing enhancements through innovative computational techniques.

References:

- https://web.mit.edu/15.053/www/AMP-Chapter-09.pdf
- https://en.wikipedia.org/wiki/Linear_programming
- https://www.youtube.com/watch?v=K7TL5NMlKIk
- https://www.complexica.com/narrow-ai-glossary/integer-programming
- https://www.analyticsvidhya.com/blog/2017/02/lintroductory-guide-on-linear-programming-explained-in-simple-english/
- https://realpython.com/linear-programming-python/
- https://math.libretexts.org/Bookshelves/Applied_Mathematics/Applied_Finite_Mathematics_(Sekhon_and_Bloom)/04%3A_Linear_Programming_The_Simplex_Method/4.01%3A_Introduction_to_Linear_Programming_Applications_in_Business_Finance_Medicine_and_Social_Science#:~:text=Linear%20programming%20is%20used%20in,scheduling%20aircraft%20and%20scheduling%20staff.
- https://towardsdatascience.com/a-gentle-introduction-to-branch-bound-d00a4ee1cad