REPUBLIC OF TURKEY

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING

TARIK ÇELIK, ALPEREN DURAN

SUPERVISOR
ASSOC. PROF. TÜLAY AYYILDIZ

GEBZE
2025

REPUBLIC OF TURKEY

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

ILP SOLVER

TARIK ÇELIK, ALPEREN DURAN

COMPUTER ENGINEERING

SUPERVISOR
ASSOC. PROF. TÜLAY AYYILDIZ

GEBZE
2025

	GRADUATION PROJECT JURY APPROVAL FORM

This project has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering in 12/01/2025 by the following jury.

JURY

Member (Supervisor) : Assoc. Prof. Tülay AYYILDIZ

Member : Prof. Didem GÖZÜPEK KOCAMAN

ABSTRACT

Integer Linear Programming (ILP) is a mathematical optimization technique used in solving problems involving objective function, decision variables and constraints with integer coefficients. This report about theoretical background, methodologies, and practical implementation of an ILP solver, providing a detailed information about the techniques used, their mathematical formulations,the challenges faced during development, tests and results of implementation and the future work.

Keywords: Optimization Problems, Integer Linear Programming

ACKNOWLEDGEMENT

We would like to thank to our supervisor Assoc. Prof. Tülay AYYILDIZ, for her invaluable guidance, continuous support, and weekly feedbacks throughout progress of this project. Her knowledge about linear programming and some other topics and her encouragement have been played important role successful completion of our work.

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
ILP	:	Integer Linear Programming
$\mathbb{R}$	:	Real Number Set
$\in$	:	Element of Symbol
$\forall$	:	Every Symbol
$\sum$	:	Summation Symbol
$\mathbb{Z}$	:	Integer Number Set

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The optimization problem means to reach optimal value of an objective. These problems involve finding the optimal solution for an objective function subject to given constraints. We can faced these type of problems in various fields in real life such as scheduling, manufacturing, resource management and finance.
This project focuses on solving these type of problems.It uses different methods for this process. The objective of this project is find the optimal solution in a possible short time.

1.1. Project Description

In this project we developed a solver program for Integer Linear Programming problems. The Integer Linear Programming (ILP) Problem Solver is a tool that helps optimize linear objectives. The solver program uses two main integer optimization problem solving methods these are Branch and Bound method and Gomory Cuts method. The program takes input from user which is problem data and it solves that problem using both Branch and Bound method and Gomory Cuts method then shows the optimal result of the problem if it is a bounded problem.The architecture of program is shown in Figure 3.2 .

2. PROBLEM DEFINITION

An optimization problem is a type of problem where the goal is to find the best solution from a set of possible solutions under given constraints. Integer Linear Programming (ILP) is a specific kind of optimization problem where the relationships between variables are linear, and some or all variables must take integer values.

ILP is used in many real-life situations, like scheduling, resource planning, and logistics. However, solving ILP problems can be difficult, especially when the problems are large or have complex constraints.

This project focuses on creating an ILP solver that can handle different types of ILP problems efficiently. The solver should find the best solution while managing time, accuracy, and usability.

2.1. Optimization Problems

An optimization problem is about finding the best solution from a group of possible solutions. It involves either maximizing or minimizing something, like profit or cost, while following certain rules or limits.There are different types of optimization problems such as linear optimization(linear programming), non-linear optimization and integer optimization(integer linear programming).In linear programming both objective function and constraints are linear.In nonlinear optimization either the objective function or constraints are nonlinear.In integer linear optimization the decision variables must be integer.

2.1.1. Main Parts of an Optimization Problem

- **Objective Function:** This is the goal wanted to achieve, like making the most profit or reducing the cost as much as possible. It is the value to be maximized or minimized.
  Example:
  - Maximize profit: $C = 10x_1 + 5x_2$
  - Minimize cost: $C = x_1^2 + 2x_2$
- **Decision Variables:** These are the controllable variables, such as the number of products to produce or the allocation of resources. The solution represents the optimal values for these variables that achieve the desired objective.

  - Example: Variables x_1 and x_2 might represent the number of two types of products to produce.

- **Constraints:** These represent the limitations or conditions that must be adhered to, such as a budget or maximum production capacity. Constraints can be expressed as equalities or inequalities, ensuring that the solution remains feasible and practical.
  Example:

  - $x_1 + x_2 \leq 100$ (limited resources)
  - $x_1 \geq 0, x_2 \geq 0$ (can not produce negative products)

- **Feasible Region:** This is the set of all possible solutions that satisfy the constraints.The optimal solution must be within this region.

2.2. Integer Linear Programming (ILP)

The name linear integer programming refers to the class of combinatorial constrained optimization problems with integer variables, where the objective function is a linear function and the constraints are linear inequalities. The Linear Integer Programming (LIP) optimization problem can be stated in the following general form[1]:

- Maximize: $\mathbf{cx}$

- subject to: $\mathbf{Ax} \leq \mathbf{b}$, $\mathbf{x} \in \mathbb{Z}^{\mathbf{n}}$

where the solution $\mathbf{x} \in \mathbb{Z}^{\mathbf{n}}$ is a vector of n integer variables:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$$

and the data are rational and are given by the m×n matrix **A**, the 1×n matrix **c**, and the 3 m×1 matrix **b**.
ILP is a specific class of optimization problems where the objective function and constraints are linear, and the decision variables must be integers. ILP problems are categorized into:

1. **Pure ILP**: All variables are integers.

2. **Mixed ILP**: Some variables are integers, while others are continuous.

3. **Binary ILP**: Variables are restricted to binary values (0 or 1), commonly used in yes/no decisions.[1]

3. METHODOLOGIES

3.1. Simplex Method

The Simplex method is an approach to solving linear programming models by hand using slack variables, tableaus, and pivot variables as a means to finding the optimal solution of an optimization problem. A linear program is a method of achieving the best outcome given a maximum or minimum equation with linear constraints[2].

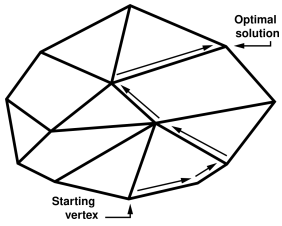

Figure 3.1: Simplex Algorithm visualization.[3]

To solve a linear programming model using the Simplex method the following steps are necessary:

1. Standard form
2. Adding slack variables
3. Creating the tableau
4. Pivot variables
5. Creating a new tableau
6. Checking for optimality
7. Identify optimal values

These requirements can always be satisfied by transforming any given linear program using basic algebra and substitution.To transform a minimization linear program model into a maximization linear program model, simply multiply both the left and the right sides of the objective function by -1.
The optimal solution of a maximization linear programming model are the values assigned to the variables in the objective function to give the largest zeta value. The optimal solution would exist on the corner points of the graph of the entire model. To check optimality using the tableau, all values in the last row must contain values greater

than or equal to zero. If a value is less than zero, it means that variable has not reached its optimal value.[2]

3.1.1. Mathematical Formulation of the Simplex Method

3.1.1.1. Problem Formulation (Standard Form)

The *Linear Programming (LP)* problem in standard form is:

$$\text{Maximize} \quad z = \sum_{j=1}^{n} c_j x_j \quad \text{or} \quad z = c^T x$$

subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, m, \quad x_j \geq 0, \quad j = 1, 2, \ldots, n.$$

To convert the inequalities $\leq$ into equalities, we introduce *slack variables* s_i:

$$\sum_{j=1}^{n} a_{ij} x_j + s_i = b_i, \quad s_i \geq 0.$$

The LP problem becomes:

$$\text{Maximize} \quad z = \sum_{j=1}^{n} c_j x_j$$

subject to:

$$Ax + Is = b,$$

where:

- $A \in \mathbb{R}^{m \times n}$ is the matrix of constraint coefficients.
- $x \in \mathbb{R}^n$ are the decision variables.
- $s \in \mathbb{R}^m$ are the slack variables.
- I is the $m \times m$ identity matrix.

3.1.1.2. Construct the Initial Tableau

The *initial simplex tableau* is formed as:

$$\begin{bmatrix} 1 & -c^T & 0 & \\ 0 & A & I & b \end{bmatrix},$$

where:

- The first row corresponds to the *objective function* z.
- The remaining rows correspond to the *constraints* $Ax + Is = b$.
- The identity matrix I corresponds to the slack variables.

3.1.1.3. Check for Optimality

- Look at the coefficients of the objective row (excluding the RHS column).
- If all coefficients are *non-negative*, the solution is *optimal*:

$$c_j \geq 0 \quad \forall j \implies \text{optimal solution found.}$$

- If any coefficient is *negative*, proceed to the *pivoting step*.

3.1.1.4. Pivoting Step (Entering and Leaving Variables)

3.1.1.4.1 Select the Entering Variable : The *entering variable* corresponds to the column with the most negative coefficient in the objective row:

$$j^* = \arg\min_j \{c_j : c_j < 0\}.$$

3.1.1.4.2 Select the Leaving Variable : Compute the *ratios* for each constraint:

$$\text{Ratio}i = \frac{b_i}{aij, \quad \text{where } a_{ij>0.}}$$

- The *leaving variable* corresponds to the row with the *smallest positive ratio*:

$$i^* = \arg\min_i \left\{ \frac{b_i}{a_{ij} : a_{ij>0}} \right\}.$$

3.1.1.5. Perform the Pivot Operation

- *Pivot element*: The intersection of the *pivot column* j^* and *pivot row* i^*. - *Row operation* to make the pivot element equal to 1:

$$\text{New Pivot Row} = \frac{\text{Pivot Row}}{\text{Pivot Element}}.$$

- Update all other rows to make the entries in the pivot column equal to 0:

$$\text{New Row}_i = \text{Old Row}_i - (\text{Pivot Column Coefficient} \times \text{New Pivot Row}).$$

3.1.1.6. Iterate

- Repeat steps *3–5* until the objective row has no negative coefficients.

3.1.1.7. Extract the Solution

- Identify the *basic variables* from the tableau (columns that form the identity matrix).
- The values of the basic variables are taken from the RHS of the tableau:

$$x_j = \text{RHS value for } j \in \text{basic variables}.$$

- The *non-basic variables* (not part of the identity matrix) are set to *0*:

$$x_j = 0 \quad \text{for non-basic variables}.$$

3.1.1.8. Termination Condition

The algorithm terminates when: - All coefficients in the objective row (except for the RHS) are *non-negative*:

$$c_j \geq 0 \quad \forall j \implies \text{optimal solution found}.$$

3.2. Branch and Bound Method

The branch and bound method is a solving method for Integer Linear Problems . It uses divide and conquer concept. It works by dividing the problem into smaller subproblems (this process is called ”branching”) and then calculating bounds on the best possible solution for each subproblem. Firstly it sends the problem to simplex

method and take the result of Linear Relaxation of problem.Then it checks the result if it is integer then the solution process is done but if it has fractional values then it creates subproblems to eliminate the fractional result.It calculates all possible solutions until reach the optimal solution while preventing not necessary calculations.

If the bound for a subproblem shows that it can not have a better solution than the current best solution which is obtained from previous subproblems, that subproblem is eliminated (this is called ”pruning”). This provides restriction of the solution space. The process continues until all possible subproblems are solved or eliminated.And then it finally returns optimal solution of the given problem.

3.2.1. Mathematical Representation of the Branch and Bound Method

3.2.1.1. Problem Formulation

Given an *Integer Programming (IP)* problem:

$$\text{Minimize (or Maximize)} \quad z = c^T x$$

subject to

$$Ax \leq b, \quad x \in \mathbb{Z}^n, \quad x \geq 0,$$

where:

- x is the vector of decision variables,
- $c \in \mathbb{R}^n$ is the cost vector,
- $A \in \mathbb{R}^{m \times n}$ is the matrix of constraints,
- $b \in \mathbb{R}^m$ is the right-hand side vector,
- $\mathbb{Z}^n$ denotes the set of integers.

3.2.1.2. Solve the Relaxed Linear Programming (LP) Problem

- Relax the integer constraints $x \in \mathbb{Z}^n$ to $x \in \mathbb{R}^n$.
- Solve the *relaxed LP problem*:

$$z_{\text{LP}} = \min c^T x \quad \text{subject to} \quad Ax \leq b, \quad x \in \mathbb{R}^n.$$

- Let x_{LP} be the optimal solution:
  - If $x_{\text{LP}} \in \mathbb{Z}^n$, the solution is already optimal.

- Otherwise, proceed to *branching*.

3.2.1.3. Branching

- Identify a non-integer component x_i of x_{LP}.
- Create two subproblems:
  - **Subproblem 1:** Add the constraint $x_i \leq \lfloor x_{\text{LP},i} \rfloor$ (floor).
  - **Subproblem 2:** Add the constraint $x_i \geq \lceil x_{\text{LP},i} \rceil$ (ceil).

3.2.1.4. Bound Calculation

- Solve the relaxed LP for each subproblem.
- Let z_{sub} be the objective value of the subproblem.
- Compare:
  - **Lower Bound (Minimization):** The smallest z_{sub} from any subproblem.
  - **Upper Bound (Minimization):** The best known integer feasible solution.

  For maximization problems:
  - **Lower Bound:** The best feasible integer solution.
  - **Upper Bound:** The largest z_{sub}.

3.2.1.5. Pruning (Eliminating Subproblems)

Prune a subproblem if:

- The LP relaxation is infeasible.
- The objective value z_{sub} is worse than the current best solution.
- The bounds indicate that the subproblem cannot yield a better solution.

3.2.1.6. Iterate Branching and Bounding

Repeat *branching, bounding, and pruning* until all subproblems have been processed or pruned.

3.2.1.7. Stopping Condition

The algorithm terminates when no subproblems remain. The best feasible solution x^* is the optimal integer solution.

3.2.2. Pseudo-code of Branch and Bound Method

Algorithm 1 Branch-and-Bound Algorithm

Input: Objective function c, Constraints A, b, Senses, Maximum depth d_{max}
Output: Best solution and value
procedure BRANCH-AND-BOUND
    Normalize the constraints using A, b, and senses.
    Initialize best_solution $\leftarrow$ None, best_value $\leftarrow -\infty$, visited_states $\leftarrow \emptyset$
    Call BRANCH$(c, A, b, best_solution, best_value, 0, d_{max}, visited_states)$
    Return best_solution, best_value
end procedure
procedure BRANCH$(c, A, b, best_solution, best_value, depth, d_{max}, visited_states)$
    if $depth > d_{max}$ or ($depth > 10$ and $best\_value \geq 0$) **then**
        Return best_solution, best_value
    end if
    Solve the LP relaxation using Simplex to get solution and value.
    if solution is None **then**
        Return best_solution, best_value
    end if
    Generate state_key using the solution, A, and b (rounded).
    if state_key $\in$ visited_states **then**
        Return best_solution, best_value
    end if
    Add state_key to visited_states.
    if value $\leq$ best_value **then**
        Return best_solution, best_value
    end if
    Find fractional variables in the solution.
    if no fractional variables **then**
        Update best_solution and best_value with the current solution and value.
        Return best_solution, best_value
    end if
    Choose a variable to branch on based on fractional values.
    Create new constraints for lower and upper bounds.
    Add constraints to form new subproblems.
    Recursively call BRANCH on both subproblems.
    Return best_solution, best_value
end procedure

3.3. Gomory Cuts Method

The Gomory Cuts method is a solving method for Integer Linear Problems. Firstly it sends the problem to Simplex method for getting the result of Linear relaxation of that problem.If the result of Linear Relaxation problem has fractional values then it creates gomory cuts (creating a new constraint with using speicific row of simplex tableau) and adds that cuts to tableau after that it sends the updated tableau to simplex method and gets the result.This process goes until finding a solution that does not have fractional values.

3.3.1. Mathematical Representation of the Gomory Cuts Algorithm

3.3.1.1. Problem Formulation

Given an *Integer Programming (IP)* problem:

$$\text{Minimize (or Maximize)} \quad z = c^T x$$

subject to

$$Ax \leq b, \quad x \in \mathbb{Z}^n, \quad x \geq 0,$$

where:

- x is the matrix of decision variables,
- $c \in \mathbb{R}^n$ is the cost matrix,
- $A \in \mathbb{R}^{m \times n}$ is the matrix of constraints,
- $b \in \mathbb{R}^m$ is the right-hand side matrix,
- $\mathbb{Z}^n$ the set of integers.

3.3.1.2. Solve the Relaxed Linear Programming (LP) Problem

1. Relax the integer constraints $x \in \mathbb{Z}^n$ to $x \in \mathbb{R}^n$.
2. Solve the *relaxed LP problem*:

$$z_{\text{LP}} = \min c^T x \quad \text{subject to} \quad Ax \leq b, \quad x \in \mathbb{R}^n.$$

3. Let x_{LP} be the optimal solution:

- If $x_{\text{LP}} \in \mathbb{Z}^n$, the solution is already optimal.
- Otherwise, proceed to *cut generation*.

3.3.1.3. Gomory Fractional Cut

Let the optimal solution x_{LP} to the relaxed LP problem be:

$$x_{\text{LP}} = (x_1^, x_2^, \ldots, x_n^*)$$

where some components x_i^* are non-integers. Identify a constraint in the final simplex tableau of the form:

$$x_i + \sum_{j \in N} a_{ij} x_j = b_i,$$

where b_i is not an integer. Let $f(b_i) = b_i - \lfloor b_i \rfloor$ be the fractional part of b_i, and for each a_{ij}, let $f(a_{ij}) = a_{ij} - \lfloor a_{ij} \rfloor$.

The Gomory fractional cut is given by:

$$\sum_{j \in N} f(a_{ij}) x_j \leq f(b_i).$$

This cut eliminates the current fractional solution without removing any feasible integer solutions.

3.3.1.4. Add the Cut and Solve Again

1. Add the Gomory cut to the LP formulation.
2. Re-solve the LP problem.
3. Repeat the process until all components of x are integers.

3.3.1.5. Stopping Condition

The algorithm terminates when all components of the solution x^* satisfy $x^* \in \mathbb{Z}^n$. The optimal integer solution is then found.

3.3.2. Pseudo-code of Gomory Cuts Method

Algorithm 2 Gomory Cut Algorithm for Integer Linear Programming

Input: Constraints matrix A, constraints vector b, objective coefficients c, constraint types.
Output: Optimal integer solution and value (if exists).
procedure SolveWithGomory(A, b, c, constraint_types)
    Initialize:
    Convert A and b into standard form for simplex by handling $\geq$ constraints.
    Set $precision \leftarrow 10^{-6}$, $pdigits \leftarrow 6$.
    Build the initial simplex tableau with A, b, and c.
    Initialize basis variables for the simplex method.
    while True do
        Primal Simplex Step:
        while there exists a column with a negative reduced cost in the objective row **do**
            Identify the entering column.
            Compute minimum ratio test to select the leaving row.
            Perform pivoting to update the tableau.
            if no entering column or leaving row found **then**
                Return `None, None`.
            end if
        end while
        Check Integrality:
        for all basic variables in the solution **do**
            Compute the fractional part of each variable.
            if fractional part $> precision$ and $1-$ fractional part $> precision$ **then**
                Break (non-integer solution found).
            end if
        end for
        if all variables are integers **then**
            Extract integer solution from the tableau.
            Compute the optimal objective value.
            Return the integer solution and optimal value.
        end if
        Add Gomory Cut:
        Identify the row containing a fractional solution.
        Compute the Gomory cut coefficients from the fractional parts.
        Append the new constraint (cut) to the tableau.
        Update the basis to include the new constraint.
        Dual Simplex Step:
        while there exists a negative value in the RHS of the tableau **do**
            Identify the leaving row.
            Compute minimum ratio test to select the entering column.
            Perform pivoting to update the tableau.
            if no leaving row or entering column found **then**
                Return `None, None`.
            end if
        end while
    end while
end procedure

3.4. Project Structure

The user enters the objctive function and constraints of ILP problem which he/she wants to solve.Then the input is sent to both Branch and Bound and Gomory Cuts methods and then the problem is solved by this two methods and the solution printed to user interface.

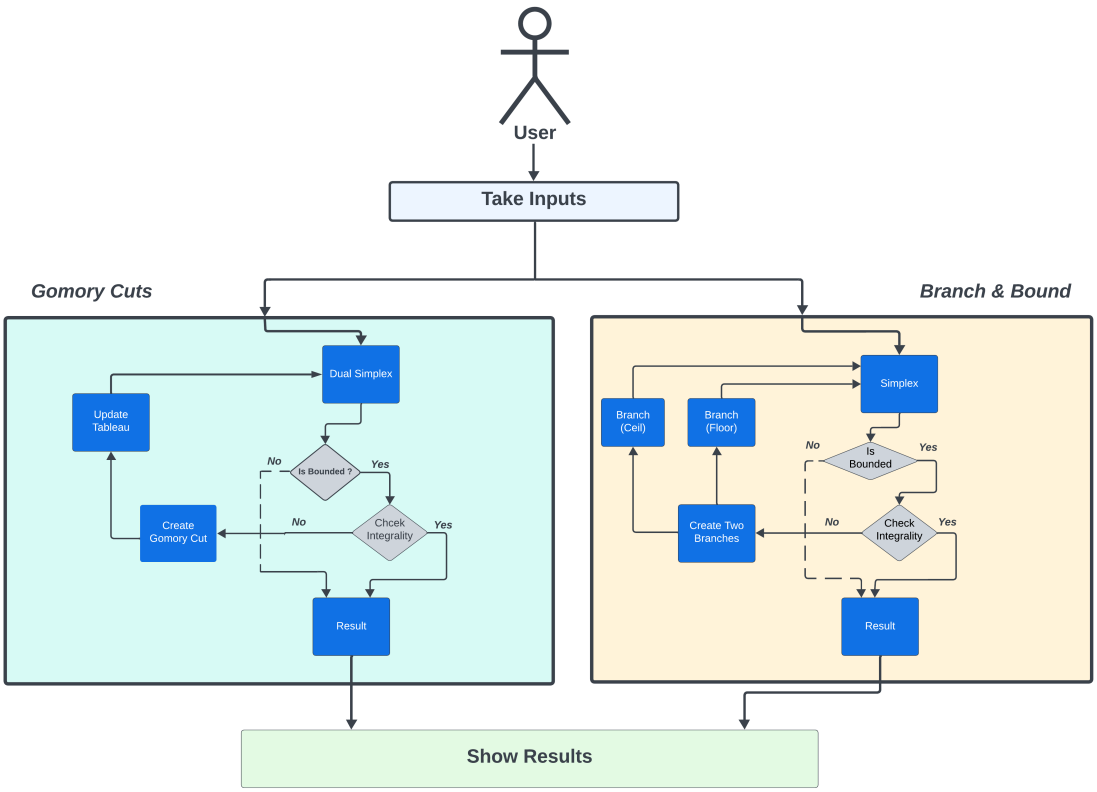

Figure 3.2: The structure of the project.You can access the full size of diagram on Github Repository[4]

4. IMPLEMENTATION AND TESTING

4.1. Implementation

4.1.1. Research

We made research about optimization problems,Linear Programming,Integer Linear Programming, Simplex method,duality, Branch and Bound, Gomory Cuts and Branch and Cut methods. First, we try to understand general concepts of these topics. Then we made research about Simplex method in detail and we inspect the steps of Simplex methods for future implementation. Then we do comprehensive research about Branch and Bound and Gomory Cuts and we try to understand solving steps of these to method in the future when implementing to help us.

4.1.2. Implementation of Simplex Method

In the implementation of Simplex method first, we convert the input matrices c, A and b into arrays for numerical operations. Then we construct a tableau from this arrays. Then we check the last row of the tableau for optimality check, if the all coefficients in the last row are non-negative the current solution is optimal and algorithm terminates. If the solution is not optimal, we select the pivot column, which is most negative coefficient in the last row. If all elements in the selected pivot column are non-positive, the problem is unbounded means that there is no feasible solution exists. After that, we select the pivot row, by computing the ratios between the right most column and we select the smallest positivie ratio as pivot row.The intersection pivot row and pivot column is the pivot element. Then we make pivot operations by making pivot element 1 and all other entries in the pivot column zero by substraction. We repeat this process until the there is no negative coefficients in the last row. We extract the optimal value and solution vector from the identify matrix and we return these values.

4.1.3. Implementation of Branch and Bound Method

In the implementation of Branch and Bound Method first, we call the Simplex Method to solve relaxed problem, if simplex function returns None, the problem is infeasible for that branch and the branch terminates. If the solution is integer feasible, we check the solution value is better than the current best value, if it is we update best

solution, if not we prune the branch means that no need to explore further this branch. Then we identify variables that have fractional variables, we choose branching variable and the we create two new sub-problems adding constraint as lower branch (floor value) and upper branch (ceil value). We recursively solve these two branches. If the recursion depth exceeds we prune the branch to prevent infinite recursion. Once all branches ae processed or pruned, we return the best integer solution and corresponding objective function value.

4.1.4. Implementation of Gomory Cuts Method

In the implementation of Gomory Cuts method first, we convert the input matrices c, A and b into arrays for numerical operations. Then we construct a tableau from this arrays. We use primal simplex method to solve relaxed Linear Problem without integer constraints, if the solution is integer feasible and returns optimal solution. If not we proceed to cut generation step. We identify a row in the tableau where right hand side value is fractional. For identified row we compute fractional parts of the coefficients and construct the gomory cut. This cut eliminates the current fractional solution without excluding any feasible integer solution. We add the gomory cut to the tableau as a new row, for that row we add new variable to represent slack variable. Then we solve the updated problem using dual simplex method to re-solve the tableau. If new solution is integer feasible, method returns the solution, if not method repeats checking fractional rows and generates gomory cuts until an integer solution is found. Method terminates when all basic varriables are integer or if the problem becomes infeasible.

4.2. Test with a Real Life Problem

4.2.1. Problem: Treasure Hunt in the Jungle

You are on a treasure hunt deep in the jungle and can carry only **20 kg** in your backpack. There are **8 valuable items** in front of you, each with different **weights** and **values**. Your goal is to **maximize the total value** of the items you carry without exceeding the backpack's **20 kg** limit.

4.2.2. Items List

Table 4.1: Items List of Treasure Hunt in the Jungle Problem

Item	**Weight (kg)**	**Value (points)**
Gold Coins	5	60
Healing Potion	3	40
Rope	4	30
Food Supplies	2	20
Map	1	15
Compass	2	10
Knife	6	35
Water Flask	7	50

4.2.3. Problem Statement

What is the **maximum value** of items you can carry in your backpack without exceeding the **20 kg** limit?

4.2.4. Mathematical Formulation

Let x_i be a binary variable that represents whether you include the i-th item:

$$x_i = \begin{cases} 1, & \text{if the item is included,} \\ 0, & \text{if the item is excluded.} \end{cases}$$

The mathematical formulation is:

$$\text{Maximize } z = 60x_1 + 40x_2 + 30x_3 + 20x_4 + 15x_5 + 10x_6 + 35x_7 + 50x_8$$

subject to:

$$5x_1 + 3x_2 + 4x_3 + 2x_4 + 1x_5 + 2x_6 + 6x_7 + 7x_8 \leq 20 \quad \text{(weight constraint)}$$

$$x_i \in \{0, 1\}, \quad \forall i = 1, 2, \ldots, 8.$$

4.2.5. Solution :

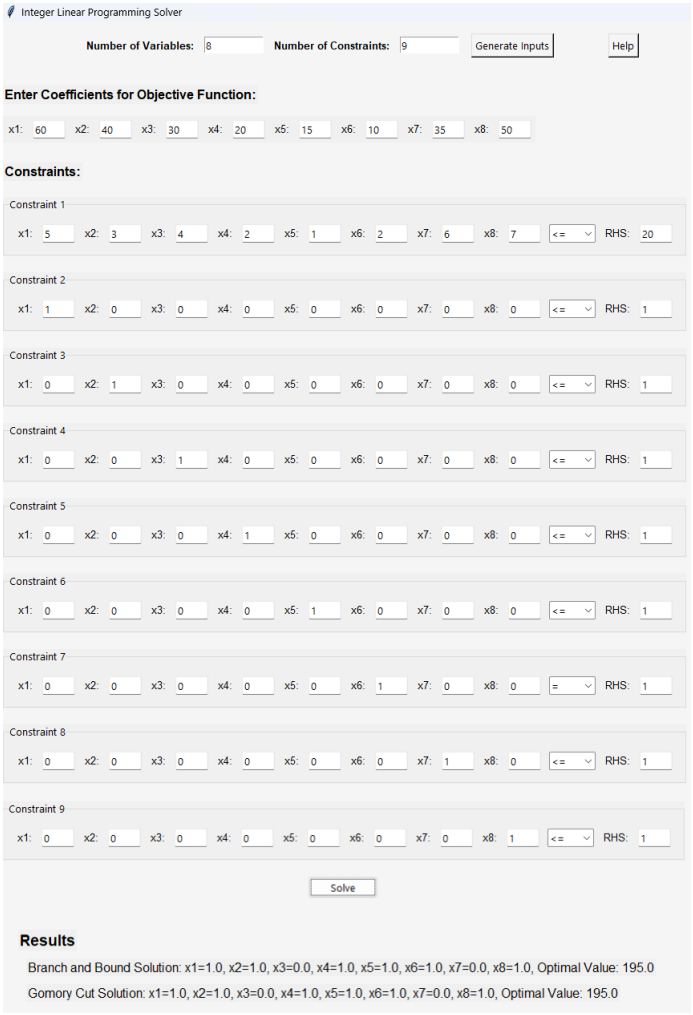

Figure 4.1: The solution of the Treasure Hunt in the Jungle Problem with ILP Solver

4.2.6. Total Weight and Value

Total weight:

$$5 + 3 + 2 + 1 + 2 + 7 = 20 \text{ kg}$$

Total value:

$$60 + 40 + 20 + 15 + 10 + 50 = 195 \text{ points.}$$

4.3. Efficiency

We tested our project by solving 30 different problems for each problem size and computed the average solution time of both Branch and Bound and Gomory Cuts Method.Then plotted a graph as shown Figure 4.2 using this values. The average solution time depends on problem size and also depends the complexity of the problem (computational hardness).This is why the average solution time does not increase with every increase in problem size.

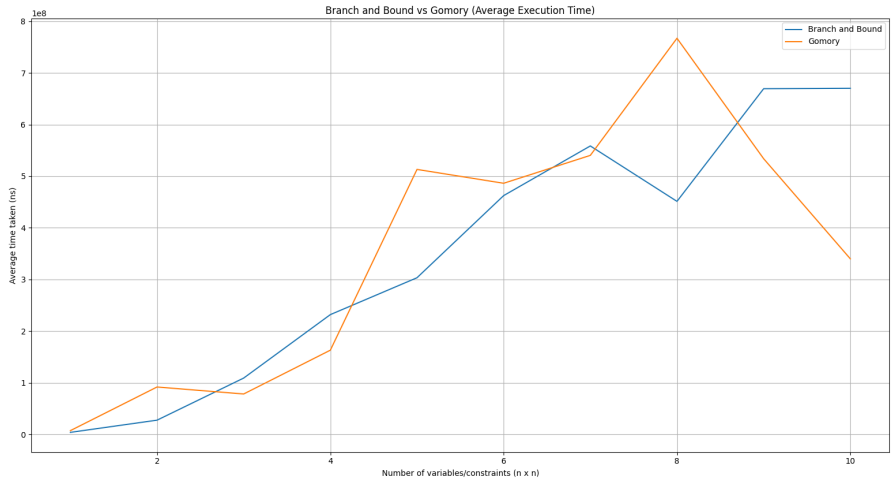

Figure 4.2: The graph of average solving times in different problem sizes.

5. CHALLENGES AND FUTURE WORK

5.1. Challenges

- **Simplex Method Implementation :** We could not handle all constraint types, we can only solve for "$\leq$" type.Because in our Simplex method implementation there is no part for adding surplus variables and artificial variables.The surplus variables and artificial variables are required for "=", "$\geq$" type of constraints.
- **Branch and Bound Method Implementation :** In our first implementation of Branch and Bound method the program does not stops because the pruning part is not implemented correctly.We solved this problem by re-implementing that part of the method.
- **Gomory Cuts Method Implementation :** When we implmementing the Gomory Cuts method firstly we faced a problem creating cuts beacuse we could not handle the simplex tableau correctly.We solved this problem by taking correct row from tableau which is used for creating ctus.
- **Testing :** When testing both methods with large number of problem, sometimes memory is stucked, so we could not get average solve time of problems with specific sizes. We solve this problem testing methods with smaller number of problems especially for Gomory Cuts method.

5.2. Future Work

- **Adding Pre-Conditioning :** We are considering adding pre-conditioning process to our project in the future, based on the advice of our project supervisor.This provides low computaitonal cost because it normalizes the coefficitens of problem data.There are several methods for this process such as Row Normalization and Jacobi Pre-Conditioner.
- **Implementation of General Form of Simplex Method :** According advice of our project supervisor, there is a general form of simplex method, that handles all types of constraints and all types of linear optimization problems (both minimization and maximization) . In the future this form of Simplex method should be implemented to solve all the types of ILP problems.
- **Implementing Branch and Cut Method :** In future, the Branch and Cut method should be added to this project as a option of solution algorithm because both

Branch and Bound and Gomory Cuts method can be inefficient for specific types or large size of problems.In this situations the Branch and Cut method can be efficient.

6. CONCLUSION

In this project, we developed an Integer Linear Programming (ILP) Problem Solver designed to optimize linear objectives within given constraints with integer coefficients. The solver contains two main Integer Linear problem solving methods these are the Branch and Bound method and the Gomory Cuts method. By implementing this methods, we provided an application for solving integer optimization problems efficiently.

We understand that the strengths and limitations of each method when implementing and testing them.Also we researched which method is suitable for which type or size of problems. The program successfully outputs optimal solutions for bounded problems, for real-life applications, such as scheduling, resource allocation. In conclusion, this project shows the importance of optimization problem solvers in decision-making processes in various fields. It also has future work, including integrating hybrid methods (Branch and Cut Method) or improving computational efficiency to handle larger and more complex ILP problems. Future researchs and improvements can improve the solver's capabilities, making it even more efficient and useful tool for real-world optimization problems.

BIBLIOGRAPHY

[1] V. G. Krasimira Genova, *Linear integer programming methods and approaches*, https://cit.iict.bas.bg/CIT_2011/v11-1/3-25.pdf, 2011.

[2] S. Bandgar, *Explanation of simplex method for minimization*, https://medium.com/analytics-vidhya/explanation-of-simplex-method-for-minimization-e32def1ef214.

[3] *Simplex algorithm visual figure*, https://en.wikipedia.org/wiki/Simplex_algorithm#/media/File:Simplex-description-en.svg.

[4] *Github repository*, https://github.com/tarikcelik1/Integer-Linear-Programming-Problem-Solver.

[5] *Trailer video*, https://www.youtube.com/watch?v=OlHA40_WwIQ.

APPENDICES

Appendix 1: Github Repo

You can find our project source codes and application on GitHub Repository[4]

Appendix 2: Youtube Link of Trailer

You can find the trailer video on Youtube[5]