INTEGER LINEAR PROGRAMMING AND ITS EXTENSIONS

SUHAAN MOBHANI

Abstract. Integer Linear Programming (ILP) is a fundamental optimization technique widely used in various fields. This paper explores the basics of this optimization technique, its mathematical formulation, solution methods, and some of its notable extensions, including mixed integer linear programming, binary integer linear programming and integer nonlinear programming. We include detailed examples and step by step solutions to illustrate the application of these methods.

1. Introduction

Integer Linear Programming (ILP) is a mathematical optimization technique that involves finding the best solution from a set of feasible solutions, where some (Mixed Integer Linear Programming) or all of the variables are constrained to take integer values. It has robust applications in fields such as operations research, economics, and computer science due to its ability to efficiently optimize tasks. While there are many more complicated algorithms to solve the problem with lesser time complexity, we stick to using the simplex method for the purpose of this paper, as well as provide an introduction to the linear programming methods that form the base for figuring integer solutions.

2. Note on Time Complexity

Time complexity plays a vital role in the efficiency of tasks. Using algorithms with a "larger" time complexity can lead to wastage of resources and loss of both money and time (for obvious reasons). While in this paper, we use a smaller amount of variable factors, real life applications in fields like economics and computer science might introduce a large amount of variables of exponential category, which are more computation heavy. In such a scenario, the methods presented in this paper might not be the most efficient.

3. Defining the problem

An Integer Linear Programming problem can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \zeta = \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\ & x_j \in \mathbb{Z}, \\ & x_j \geq 0 \quad \forall j. \end{aligned}$$

Where c_i, a_{ij}, and b_i are constants, and x_i are integer variables.

3.1. **The technique for solving minimization problems.**

We can use the same maximization technique presented in this paper to solve minimization problems as well.

Consider if you want to minimize an objective function ζ. The problem can be adapted to a maximization problem by simply negating the objective function. So, now instead we need to maximize $-\zeta$.

The idea behind this is intuitive and needs little justification.

4. Ignoring the Integer Condition

Before trying to solve the problem with integer conditions, we try solving the problem ignoring any such conditions, that is, we eliminate the integer constraint and solve the regular linear programming instance. We call the solution that we obtain through this (that may not necessarily be integers) as the Linear Programming relaxation (LP Relaxation).

An Linear Programming problem can be formulated as:

$$\begin{aligned} \text{maximize} \quad & \zeta = \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\ & x_j \geq 0 \end{aligned}$$

Where c_i, a_{ij}, and b_i are constants, and x_i are nondiscrete variables. Notice how this is the same as our definition for an ILP problem, except that the integer constraint is removed.

The LP relaxation is particularly useful in the context of understanding the range

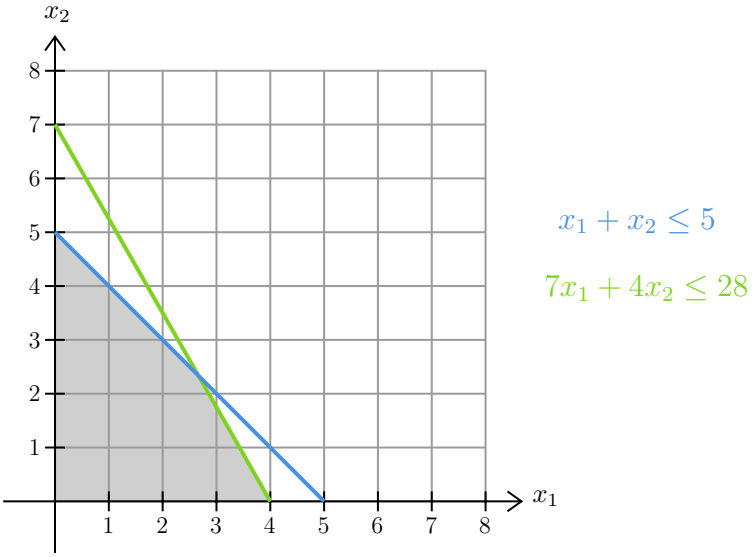
that the integer constraint objective function can obtain. Particularly, we get the following:

$$\text{Integer Problem } \zeta \leq \text{ LP Relaxation } \zeta$$

The proof of the above is trivial and can easily be demonstrated by graphical means as well.
Consider the following example of a LP:

$$\begin{aligned} \text{maximize} \quad & \zeta = 4x_1 + 3x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 5, \\ & 7x_1 + 4x_2 \leq 28, \\ & x_1, x_2 \geq 0. \end{aligned}$$

There are two inequalities which bound the problem.
Here is a graphical representation of the problem:



Feasible solutions are the set of values for the decision variables that satisfy all the constraints. These solutions lie within the feasible region defined by the intersection of all the constraints. In the graph above, the shaded region represents the feasible region.
On the other hand, any solution which is not considered feasible is an infeasible solution. Infeasible solutions lie outside the feasible region and do not represent valid solutions to the problem.
For example, in our problem, any solution that satisfies $x_1 + x_2 \leq 5$ AND $7x_1 + 4x_2 \leq 28$ with $x_1, x_2 \geq 0$ is a feasible solution. However, if a solution doesn't satisfy both of the conditions, then it is an infeasible solution.

By simple inspection of the graph, we can figure that the maximum lies at the intersection of the two inequalities.
So, we get that $\zeta = \frac{53}{3}$ when $x_1 = \frac{8}{3}$ and $x_2 = \frac{7}{3}$.
We define this particular solution that disregards the integer condition as the LP relaxation.

However, there is a more sophisticated way than the graphical method that can be used for a larger number of variables, known as the simplex tableau.

We convert our problem into standard form by introducing slack variables s_1 and s_2. The idea of their implementation is to convert the inequalities into equalities so that we can carry operations on them by rearranging.

$$\begin{aligned} x_1 + x_2 + s_1 &= 5, \\ 7x_1 + 4x_2 + s_2 &= 28, \\ x_1, x_2, s_1, s_2 &\geq 0. \end{aligned}$$

Now, we present a more organized way of going through these operations. It involves setting up a simplex tableau to use the simplex method to solve the LP problem.
Here is the Initial Simplex Tableau for this problem:

		x_1	x_2	s_1	s_2	
Basis	c_B	4	3	0	0	b
s_1	0	1	1	1	0	5
s_2	0	7	4	0	1	28
	z_j	0	0	0	0	0
	$c_j - z_j$	4	3	0	0	

We call the c_j row as the row which stores the objective function coefficients for each variable. This is directly underneath the variable heading row in the initial simplex tableau.

We store the coefficients of the variables in the constraints in the A matrix, which is located directly under the c_j row.

To the right of the A matrix is the b column, which stores the quantity.

Unit columns are columns which consist of one value being 1 and the others being 0. Since the part of the A matrix under the variables s_1 and s_2 are unit columns, they are known as basic variables.

All other variables are known as nonbasic variables. So, in this case, x_1 and x_2 are nonbasic variables.

The solutions that is obtained by setting the nonbasic variables to zero is called basic feasible solution.

Below the A matrix lies the z_j row. For basic variables, the z_j row represents their unit contribution to the objective function value. However, for nonbasic variables, it represents the value lost if a unit of the variable is added to the current solution.

These basic variables are stored in the column known as the basis, with their objective function coefficients placed next to them in the c_B column.

Finally, the $c_j - z_j$ row represents the net change per unit in the objective function value if that variable is added to the solution.

Next, we identify the entering variable which is the variable with the largest $c_j - z_j$) value. We also identify the leaving variable which is the variable in the basis with the smallest non-negative ratio of b_i/a_{ij}. In this case the entering variable is x_1 and the leaving variable is s_2.

If in any scenario, the ratios for variables in the basis is the same, we usually take the one with the lower index and swap that out. A similar rule is applied if the largest $c_j - z_j$) value is repeated between multiple variables.
Generally, we define the pivot column as the column of the entering variable, and the pivot row as the row in the basis of the leaving variable. The Pivot Element is located at the intersection of the pivot column (of the entering variable) and the pivot row (of the leaving variable). In this case, the pivot element has a value of 7.

		$\mathbf{x_1}$	x_2	s_1	s_2		
Basis	c_B	**4**	3	0	0	b	Ratio
s_1	0	**1**	1	1	0	5	$\frac{5}{1} = 5$
s_2	0	**7**	4	0	1	28	$\frac{28}{7} = 4$
	z_j	**0**	0	0	0	0	
	$c_j - z_j$	**4**	3	0	0		

Now, we wish to transform the pivot element to have a unit value. To do this, we shall divide all values in the pivot row by the unit element. In doing so, our pivot row looks like this:

		x_1	x_2	s_1	s_2	
Basis	c_B	4	3	0	0	b
s_1	0	1	1	1	0	5
x_1	4	7/7	4/7	0/7	1/7	28/7
	z_j	0	0	0	0	0
	$c_j - z_j$	4	3	0	0	

Now, we must find a value k such that in the pivot column:

$$s_1 - (k)(x_1) = 0.$$

Then, for every value in the s_1 row, we replace the original value s_1 with $s_1-(k)(x_1)$ for the corresponding x_1 value.
In this case:

$$1 - (k)(1) = 0 \implies k = 1.$$

Using $k = 1$ and carrying out the elementary row operations, our simplex tableau should now look like this:

		x_1	x_2	s_1	s_2	
Basis	c_B	4	3	0	0	b
s_1	0	0	3/7	1	-1/7	1
x_1	4	1	4/7	0	1/7	4
	z_j	0	0	0	0	0
	$c_j - z_j$	4	3	0	0	

Now, we populate the z_j row using the following for each column one at a time:

$$z_j = \overrightarrow{c_B} \cdot \begin{pmatrix} s_1 \\ x_1 \end{pmatrix}.$$

Then, we recalculate the $c_j - z_j$ row using the new z_j values. After doing so, our tableau should now look like this:

		x_1	x_2	s_1	s_2	
Basis	c_B	4	3	0	0	b
s_1	0	0	3/7	1	-1/7	1
x_1	4	1	4/7	0	1/7	4
	z_j	4	16/7	0	4/7	16
	$c_j - z_j$	0	5/7	0	-4/7	

Now, our first iteration of elementary row operations is complete. However, we must continue iterating until all the $c_j - z_j$ values are nonpositive. So, we continue with the second iteration.
To identify changes to the basis, we recalculate the ratios for our rows:

So, for the second iteration, the leaving variable is s_1 and the entering variable is x_2. The pivot element is $\frac{3}{7}$.
Now, we divide all values in the x_2 row by the pivot element. Then, we use $k = \frac{4}{7}$ and modify the x_1 row using $x_1^* = x_1 - (4/7)x_2$, which results in the table looking like this:

Basis	c_B	x_1 4	x_2 3	s_1 0	s_2 0	b	Ratio
s_1	0	0	3/7	1	-1/7	1	7/3
x_1	4	1	4/7	0	1/7	4	7
	z_j	4	16/7	0	4/7	16	
	$c_j - z_j$	0	5/7	0	-4/7		

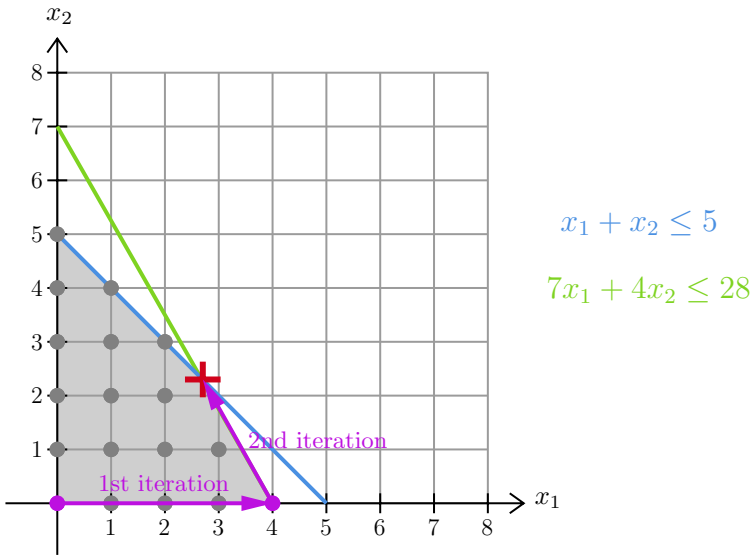
Basis	c_B	x_1 4	x_2 3	s_1 0	s_2 0	b
x_2	3	0	1	7/3	-1/3	7/3
x_1	4	1	0	-4/3	1/3	8/3
	z_j	4	16/7	0	4/7	16
	$c_j - z_j$	0	5/7	0	-4/7	

Now, once again, we recalculate our z_j and $c_j - z_j$ values:

Basis	c_B	x_1 4	x_2 3	s_1 0	s_2 0	b
x_2	3	0	1	7/3	-1/3	7/3
x_1	4	1	0	-4/3	1/3	8/3
	z_j	4	3	5/3	1/3	$\boxed{53/3}$
	$c_j - z_j$	0	0	-5/3	-1/3	

Since all our $c_j - z_j$ values are nonpositive, we terminate the process and no further iterations needed to be carried out. We have found our maximum $\zeta = \frac{53}{3}$ when $x_1 = \frac{8}{3}$ and $x_2 = \frac{7}{3}$. This is our LP relaxation.

It is interesting to see how the process of carrying our each iteration moves along an edge of the feasible region graph until it comes to the final solution. This is illustrated on the graph below:



Also note how each iteration resulted in a point on the graph, while also giving the objective function value at that point.

Notice how this process of using the simplex tableau was significantly longer than the graphical method. However, for a larger number of variables (more than 3), the graphical method would prove useless, and although it would increase the number of iterations for even the simplex tableau method, the use of machines to carry out such advanced computation significantly saves time and improves accuracy.

The simplex method was created by G.B. Dantzig in 1949. However, Chvátal (1983) introduced a more intuitive dictionary notation to develop the algorithm. We present the generalized dictionary notation below:

An Linear Programming problem can be formulated as:

$$\begin{aligned} \text{maximize} \quad & \zeta = \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\ & x_j \geq 0 \end{aligned}$$

Where c_i, a_{ij}, and b_i are constants, and x_i are nondiscrete variables. We similarly define our slack variables generally:

$$w_i = b_i - \sum_{j=1}^{n} a_{ij} x_j.$$

As we saw from our simplex tableau method, we treat the slack variables similarly to the original variables. So, to make them indistinguishable, we treat $w_i = x_{n+i}$. So our new list of x variables is now:

$$(x_1, \ldots, x_n, w_1, \ldots, w_m) = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m}).$$

So, we can rewrite our slack variables as:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j.$$

Adding this to our objective function definition of ζ, we get the starting dictionary. As we continue carrying out operations throughout the simplex method, the state will change from one dictionary to the next.

In general, each dictionary will contain m basic variables and n nonbasic variables. This is fairly straightforward to notice, by referring back to the simplex tableau. To simplify things further, we define $\mathcal{N}$ as the collection of the nonbasic variable indices. Similarly, we define $\mathcal{B}$ as the collection of the basic variable indices. At the beginning, each of them are:

$$\mathcal{N} = 1, 2, \cdots, n,$$

$$\mathcal{B} = n+1, n+2, \cdots, n+m.$$

With each iteration, exactly one variable goes from being non basic to basic and exactly one variable goes from basic to nonbasic. Note how this relates to the entering and leaving variable from the basis in the simplex tableau method. The naming process and the criteria for selection for each of these variables remains the same as the simplex tableau method.

Once the entering and leaving variables have been identified, row operations are carried out on the dictionary. This process of converting one dictionary to the next is known as a pivot. After carrying out pivots, the current dictionary will change to look like this:

$$\zeta = \bar{\zeta} + \sum_{j \in \mathcal{N}} \bar{c}_j x_j$$

$$x_i = \bar{b}_i - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \quad i \in \mathcal{B}.$$

In the pivot, we effectively carry out the procedure similar to what was presented in the simplex tableau of row operations. While it is more algebraic in the case of the dictionary method, the essence remains the same, and the same type of manipulation is occurring on the system to end up with the same result.

5. Solution Methods

In this paper, we shall present 3 separate methods of solving such an integer linear programming problem. The 2 primary ones are the branch and bound method and the cutting planes method. The third one is compound method that uses the other two primary ones to find an optimal solution.

5.1. **Branch and Bound.**

Branch and Bound is a method for solving ILP problems. It involves dividing the problem into smaller subproblems (branching) and calculating bounds for the objective function in these subproblems. In essence, we can boil down this method to a bunch of categorized steps:

(1) **Branching:**
- Select a variable with a noninteger value in the solution of the LP relaxation.
- Create two subproblems or branches by adding constraints to the selected variable: $x \leq \lfloor x^* \rfloor$ and $x \geq \lceil x^* \rceil$. This is to effectively split the integer lattice points to either side of the point, which allows for swift categorization.

(2) **Bounding:**
- Solve the LP relaxation of the new subproblems.
- If a subproblem is infeasible, discard it.
- If a subproblem has an integer solution, compare it with the best solution and update if it is better.

(3) **Pruning:**
- Discard subproblems that cannot produce a better solution than the best solution.

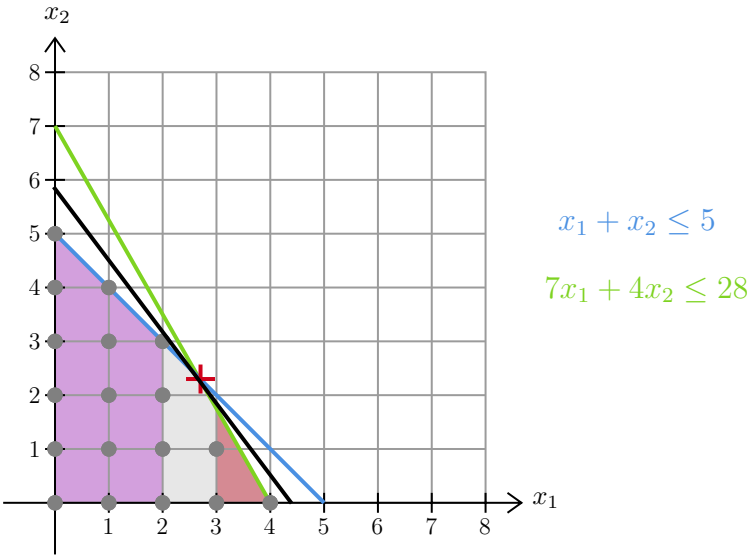
(4) **Iteration:**
- Repeat branching, bounding, and pruning steps until all the subproblems are either solved or discarded.

To understand this process further, lets consider the same LP relaxation that we found previously, where $\zeta = \frac{53}{3}$ when $x_1 = \frac{8}{3}$ and $x_2 = \frac{7}{3}$.
To apply the Branch and Bound method, we first choose a variable to branch on. For uniformity sake, we usually go in a sequential order unless there is a special circumstance. For this problem, we branch on x_1 to get the following branches:

$$x_1 \leq 2 \quad \& \quad x_1 \geq 3.$$

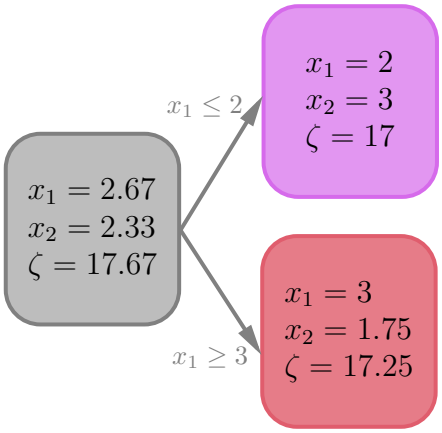
To understand this better, lets consider each of the branches graphically. The $x_1 \leq 2$ branch is colored pink, and the $x_1 \geq 3$ branch is colored red. Note that

our target solutions are only lattice points within the branches, and not the entire branch.



Notice how the subproblems have effectively split the points to the right and the left of the original LP relaxation (marked as a red cross in the diagram). For context, the objective function from the LP relaxation is plotted as the black line passing through the red cross.

To keep a better track of the individual branches, we create a tree noting down our new subproblems. This is called an *enumeration tree*. Each element in the tree will consist of its relation to other elements as well as the best solution offered by that particular branch. Here is how our tree looks after the first branch on x_1:



Since the best solution in our $x_1 \leq 2$ branch is an integer solution, we can update the variable BEST to a value of 17. Since we got an integer value from this branch, there is no need to branch it further.
Looking at our $x_1 \geq 3$ branch, we see that our best solution results in a noninteger

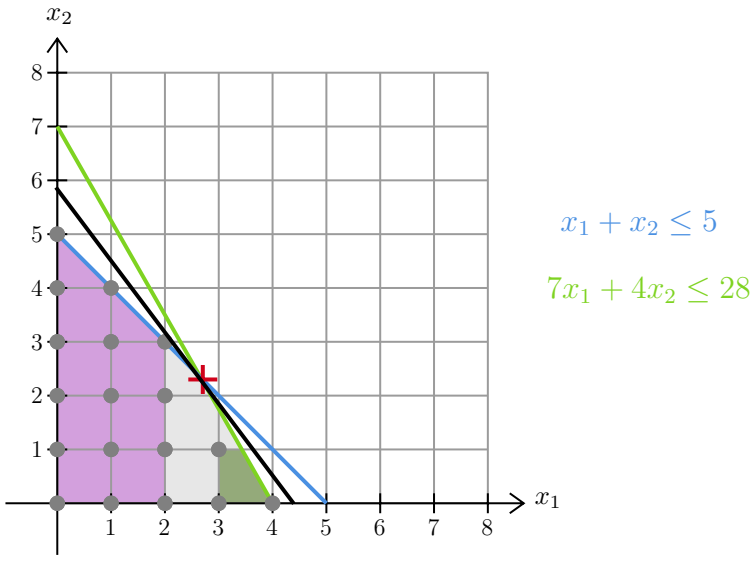
value for x_2. This means that we must branch it further.

In this case, our value of ζ from the $x_1 \geq 3$ branch is larger than our BEST variable ($17.25 > 17$). This means that there is potential for a "better" solution to exist in our unsolved branch, hence we must branch it further. However, if any branch results in a ζ value lesser than the current BEST variable, then there is no potential for a better solution from that branch, so we discard processing it entirely. A simple observation such this saves us from doing a lot more excessive computation, which is really significant when there are a lot more variables.

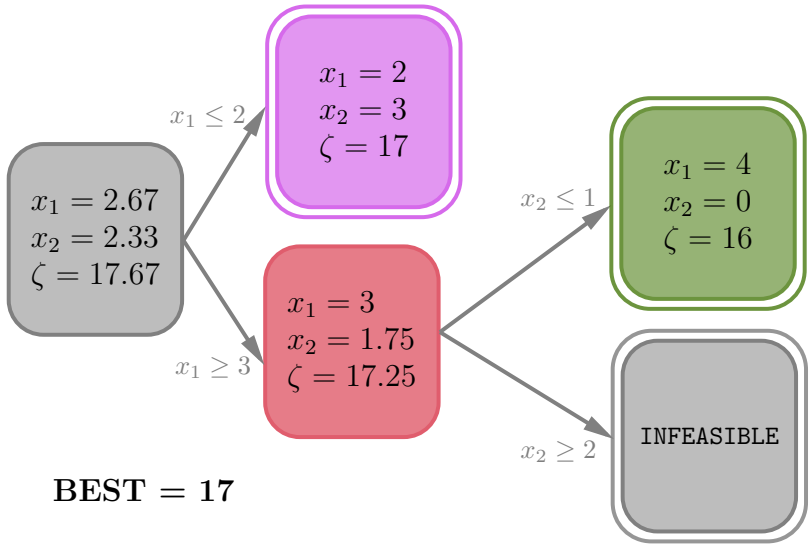
So, now, we branch further on the $x_1 \geq 3$ branch by branching on x_2, until we get an integer solution. We get the following branches:

$$x_2 \leq 1 \qquad x_2 \geq 2.$$

Plotting these new branches graphically, we obtain:



Our $x_2 \leq 1$ branch is shaded green, and the $x_2 \geq 2$ branch lies entirely outside our feasible region, hence it is unplotted. We call such a branch infeasible.
We now update our enumeration tree by adding the new branches.



Even though our green subproblem results in an integer solution, but since its value of ζ is lesser than the BEST variable ($16 < 17$), we do not update the BEST variable. Since both the branches were definitive (one resulting in an integer solution and the other being infeasible), they do not branch further. This means our enumeration tree is complete. Notice how the definitive branches of the tree are boxed twice. This is a standard convention used to keep better track of unsolved branches when larger problems have to be solved.

So our optimal solution to the ILP is:

$$\zeta = 17, \quad x_1 = 2, \quad x_2 = 3.$$

Note how the ζ value from our integer solution is lesser than that found in the LP relaxation.

5.1.1. *Searching using Enumeration Trees*

When searching using an enumeration tree, we could use either Breadth-first search or Depth-first search. In Breadth-first search, we would solve all the problems on a given level of the tree before going deeper to a further level. On the other hand, in depth-first search, we go deep by solving all subproblems on a branch until the branch is definitive before going to another branch on the same level.

Typically, carrying out a depth-first search is preferred for several reasons:

(1) Integer solutions are usually found deep in the tree. Finding these early helps in pruning the tree, by comparing branches with the BEST variable, hence improving efficiency.
(2) Depth-first search is easily implemented using recursion, making the algorithm simpler to code.
(3) Moving deeper in the tree only requires adding or refining bounds on variables, simplifying further LP problems. This also helps with pruning.

5.2. **Cutting Planes.** Cutting Planes is another method used to solve ILP problems by iteratively adding linear constraints (cuts) to exclude fractional solutions, thereby moving closer to the integer solution.

First, we must identify a constraint (cut) that excludes the current non-integer solution $\mathbf{x}^*$ but does not exclude any feasible integer solutions. We choose to use Gomory cuts.

For a basic non-integer variable x_i^*, we derive a Gomory fractional cut:

$$\sum_{j \in \mathcal{N}} (a_{ij} - \lfloor a_{ij} \rfloor) x_j \geq 1 - (x_i^* - \lfloor x_i^* \rfloor),$$

where $\mathcal{N}$ is the set of non-basic variables.

We can then add the cut to the original set of constraints, creating a new linear programming problem. We then solve the new LP relaxation with the added cut. If the solution is an integer, it is the optimal solution to the ILP. If not, we must repeat the process by generating additional cuts until we find an integer solution.

The cutting planes method iteratively refines the feasible region by adding linear constraints, systematically converging to the integer optimal solution while ensuring all feasible integer solutions remain within the feasible region.

5.3. **Branch and Cut.** The Branch and Cut method combines two powerful techniques: branch and bound, and cutting planes, to solve integer linear programming (ILP) problems. This method iteratively refines the feasible region by branching on fractional variables and adding cutting planes to exclude noninteger solutions.

Since this is a compound method, it is typically used for much larger problems consisting of larger numbers or a larger number of variables. At each iteration, a

subproblem is created, similar to the Branch and Bound method presented before, and for larger subproblems, the cutting planes method can be used to simplify things further. At each iteration, a choice is made on the usage of each depending on a number of factors. Using both of these methods together is much more effective than using any one of them by themselves.

6. Extensions of ILP

6.1. **Mixed Integer Linear Programming (MILP).** Mixed Integer Linear Programming involves problems where only some of the variables are constrained to be integers, while others can be continuous. By being continuous, we mean that they can take on analogous values, including fractional values or even irrational real values that might not necessarily be an integer.

Altering the formulation of an ILP, A Mixed Integer Linear Programming (MILP) problem can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \zeta = \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\ & x_j \in \mathbb{Z} \quad \forall j \in \mathcal{I}, \\ & x_j \geq 0, \end{aligned}$$

where c_j, a_{ij}, and b_i are constants, x_j are decision variables, and $\mathcal{I} \subseteq \{1, 2, \ldots, n\}$ is the index set of variables that are required to be integers. The remaining variables can take any non-negative real values.

The process for solving MILP problems is similar to the Branch and Bound process we used for ILP problems, but simply tweaking it so that variables that must not be integers are not ever branched. To understand it better, lets consider the same problem as before, but with the condition that ONLY x_1 must be an integer, whereas x_2 can take on any real value. Hence, we get our new definition for the

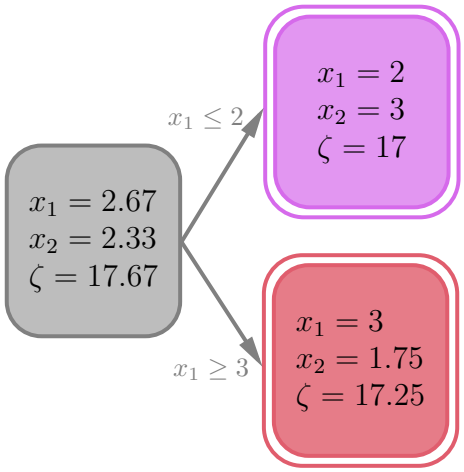
problem as this:

$$\begin{aligned} \text{maximize} \quad & \zeta = 4x_1 + 3x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 5, \\ & 7x_1 + 4x_2 \leq 28, \\ & x_1 \in \mathbb{Z}, \\ & x_1, x_2 \geq 0. \end{aligned}$$

If we recollect, our LP relaxation was: $\zeta = \frac{53}{3}$ when $x_1 = \frac{8}{3}$ and $x_2 = \frac{7}{3}$.
We set up slack variables similarly to how we did before, and carry out the first process where we branch on x_1:

$$x_1 \leq 2 \qquad x_1 \geq 3.$$

This results in the following enumeration tree:



Notice how both the subproblems are double boxed. This means that they've both produced a definitive value. Unlike ILP, we don't need to branch the $x_1 \geq 3$ subproblem any further since there is no condition on x_2 having to be an integer.

Through obvious inspection, we see that we get our optimal solution from the $x_1 \geq 3$ branch. Notice how this is different to what we had obtained using ILP.
So, our final solution to this MILP problem is:

$$\zeta = 17.25, \quad x_1 = 3, \quad x_2 = 1.75$$

This process can be adapted for any MILP problem with any number of variables, as we simply use the Branch and Bound method presented for ILP's. For MILPs, there are significantly lesser number of branches than ILPs due to the lesser number of conditions. Generically, the lesser number of variables that are required to be

integers, the lesser computation and branches will be required to find an optimal solution on average.

6.2. **Integer Nonlinear Programming (INLP).** Integer Nonlinear Programming involves optimization problems with nonlinear objective functions or constraints. (ie: degrees more than one)

We can adapt a similar method of branching and bounding to solve Integer Nonlinear Programming problems, using graphical methods. However, as numbers get larger, and the number of variables increase, using such methods can become quite difficult and computation heavy, apart from time consuming. The exact method has not been covered in this paper, but using the previously presented methods and with a little extension can be figured out as an exercise by the reader.

6.3. **Binary (0-1) Integer Linear Programming.** Binary (0-1) variables are decision variables that can only take on the values 0 or 1. These determine whether to include a certain non-decision variable in a solution (1) or not (0).

A Binary (0-1) Integer Linear Programming (BILP) problem can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \zeta = \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \\ & x_j \in \{0,1\} \quad \forall j, \end{aligned}$$

where c_j, a_{ij}, and b_i are constants, and x_j are binary decision variables that can only take values of 0 or 1.

BILP is particularly useful for fixed cost problems. Fixed cost problems often involve scenarios where there are substantial initial costs incurred if a particular decision is made, and these costs must be incorporated into the optimization model. They usually involve binary decision variables linked with other nonbinary regular variables.

This approach is widely used in various industries, including logistics, manufacturing, and project management.

7. Applications

Integer linear programming, Integer nonlinear Programming and Mixed Integer Linear Programming have a wide range of applications in the real world in several fields. Some of these applications are below:

- **Optimization of supply chains in economics**: ILP is used to optimize production, inventory, and distribution in supply chains.
- **Operations Scheduling**: ILP and MILP are applied to schedule tasks, jobs, or resources efficiently.
- **Designing Network**: ILP and INLP are used to design optimal networks for telecommunications, transportation, and utilities for optimized connectivity.
- **Optimization of Returns on Financial Portfolios**: MILP helps in selecting the optimal mix of investment assets to maximize returns and minimize risks.

With the advent of quantum computing, linear programming methods has a larger range of use cases. As such methods are further developed, ILP methods will be used in a larger variety of scenarios across all fields.

8. Conclusion

The simplex method, while being one of the more "simple" methods, is a strong but powerful tool used for solving linear programming instances. Recently, increased computational power and algorithmic improvements have significantly enhanced its practicality, allowing for the efficient solving of larger scale linear programming problems that were previously unreachable.

Integer Linear Programming is a versatile and powerful optimization technique with numerous applications. Its extensions, such as Mixed Integer Linear Programming and Integer Nonlinear Programming allow for the modeling and solving of even more complex problems. While all of these follow virtually similar methods, it is very important to identify which technique is the most appropriate for that scenario and implement it accordingly.

Due to its adaptability and robust nature, Integer Linear Programming and its extensions will undoubtedly remain a critical component in the toolkits of a wide range of people striving to achieve optimal solutions in their respective fields with greater precision and confidence.

References

G. B. Dantzig, *Programming in a Linear Structure*, Rand Corporation, P-392, Santa Monica, 1949.
T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, 2009.
A. H. Land and A. G. Doig, *An Automatic Method of Solving Discrete Programming Problems*, Econometrica, 28(3), pp. 497-520, 1960.
R. E. Gomory, *Outline of an Algorithm for Integer Solutions to Linear Programs*, Bulletin of the American Mathematical Society, 64(5), pp. 275-278, 1958.
C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, 1998.
W. L. Winston, *Operations Research: Applications and Algorithms*, 4th ed., Brooks/Cole, 2004.
R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, 3rd ed., Springer, 2008.
H. Paul Williams, *Model Building in Mathematical Programming*, 5th ed., Wiley, 2013.
Laurence A. Wolsey, *Integer Programming*, Wiley-Interscience, 1998.
Richard Kipp Martin, *Large Scale Linear and Integer Optimization: A Unified Approach*, Springer, 1999.
Fred Glover, *Handbook of Metaheuristics*, Springer, 2003.
John N. Hooker, *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, Wiley-In
Nemhauser, G. & Wolsey, L. (1988), *Integer and Combinatorial Optimization*, Wiley, New York.
Garfinkel, R. & Nemhauser, G. (1972), *Integer Programming*, John Wiley and Sons, New York.