Integer Programming Solving

Author | Ishank Juneja

May 22, 2018

1 Overview

Notes discuss Algorithms to Solve IP formulations.

2 Solving 0-1 IP problems

Every IP problem with an upper bound on variables can be reduced to a 0-1 problem
example Max $6X_1 + 3X_2$
$X_1 \leq 8$, $X_2 \leq 7$, both are integers, then we can simply write the variables in binary form
$X_1 = y_0 + 2y_1 + 4y_2 = 8y_3$ where y_i are binary.
An important idea is to explore solution space, identify feasible solutions and discover new upper or lower bounds for the function to be optimized.
Example minimize

$$Z = 5X_1 + 6X_2 + 10X_3 + 7X_4 + 19X_5$$

Subject to
$5X_1 + X_2 + 3X_3 - 4X_4 + 3X_5 \geq 2 \quad (1)$
$-2X_1 + 5X_2 - 2X_3 - 3X_4 + 4X_5 \geq 0 \quad (2)$
$X_1 - 2X_2 - 5X_3 + 3X_4 + 4X_5 \geq 2 \quad (3)$
$X_j \quad \in \quad \{0, 1\}$
Some observations are:

- When we discover a feasible solution we can be sure that the optimal binary vector of inputs will certainly not have a 1 where the already discovered feasible solution has a 0. Since any such input will lead to a higher Z.
- If the problem is a minimization problem and all the constraints are grater than equal to type, then for any infeasible solution, the constraint that fails to be satisfied, can be 'improved' by flipping some 'helpful variables' from 0's to 1's (helpful if coefficients are +ve). Even after flipping all helpful variables, if we don't get a feasible solution even by the end of this then we can reject all partially flipped input bin. vectors in between.

An algorithm can be obtained using the above idea.
The sets involved in this algorithms are S : The set of indices i such thatX_i has been forced to 1.
In absence of a var. being forced we initialize it to be 0 since we have a Minimization problem and begin algorithm with $\vec{0}$.
V : Is the set of violated constraints for the current vector whose feasibility is being checked.
T : is the set of helpful variables (Their Indices). A variable can only be in the set of helpful variables if it has not already been forced to a value.

Additive Algorithm / Implicit Enumeration Algorithm (1959)

1. $S_1 = \{\}, V_1 = \{1, 3\}, T_1 = \{1, 2, 3, 4, 5\}$. Since all are helpful, we can pickup anyone, fix it to 1 and then determine V and T for the new input vector with one index flipped. Pick 5 in this case.

2. $S_2 = \{5\}, V_2 = \{\}, T_2$ d.n.e since V is a null set, we get a new feasible solution Z = 19, X_5 = 1 (Rest = 0).

3. Whenever we find a new feasible solution (No constraint violated) we perform backtracking. $S3_{=}\{\bar{5}\}$ (means X5 is fixed at 0, other variables are not fixed to 0 but are 0), $V_4 = \{1, 3\}$, $T_3 = \{1, 2, 3, 4\}$ since for a variable to be helpful, an additional requirement is that the variable not be in S.

4. $S_4 = \{\bar{5}, 1\}$ which is an ordered set, $V_4 = \{2, 3\}, T_4 = \{2, 4\}$

5. $S_5 = \{\bar{5}, 1, 2\}, V_5 = \{3\}, T_5 = \{4\}$

6. $S_6 = \{\bar{5}, 1, 2, 4\}$, $V_6 = \{\}$. hence another (better) solution is Z = 18, $X_5 = 0, X_4 = 1, X_3 = 0, X_2 = 1, X_1 = 1$.

7. Backtrack again $S_7 = \{\bar{5}, 1, 2, \bar{4}\}$, $V_7 = \{3\}$, $T_7 = \{\}$. Since there are no helpful variables, we will backtrack in next step.

8. $S_8 = \{\bar{5}, 1, \bar{2}\}$. We remove all elements to the right of the element we back track on. $V_8 = \{2, 3\}$ (Like V_4), $T_4 = \{4\}$. But what is going to happen is, we flip 4, cons. 2 is still violated, and T is empty.
   So better to backtrack straight away.

9. $S_9 = \{\bar{5}, \bar{1}\}$, $V_9 = \{1, 3\}$, $T_9 = \{2, 3, 4\}$

10. $S_{10} = \{\bar{5}, \bar{1}, 4\}$, $V_{10} = \{1, 2\}$, $T_{10} = \{2, 3\}$. in the next step we will back track since even after using the helpful variables we cannot get feasibility. (Check!)

11. $S_{11} = \{\bar{5}, \bar{1}, \bar{4}\}$, $V_{11} = \{1, 3\}$, $T_{11} = \{2, 3\}$. Since all the elements of S are fixed at 0, No backtracking is possible and the algorithm terminates. Best solution is Z = 18, $X_5 = 0, X_4 = 1, X_3 = 0, X_2 = 1, X_1 = 1$

To choose the helpful variable that will be flipped then we can define an 'Index of helpfulness'. This can be simply adding all the coefficients of a variable in all the constraints.
If we have n constraints and m variables. The memory required is 2n + m.
Whenever we backtrack it may be for two reasons. Case 1 is that we find a feasible solution and since we are performing a minimization problem there is no point exploring cases with more 1's. Hence we backtrack onto alternate possibilities.
Or, exploring deeper down makes no sense since either T is empty (No helpful variables). Or even flipping helpful variables will not lead to feasibility.
Figure 1 illustrates the step wise working of the algorithms. The node indices are the same as the ones used above, straight arrows represent usual progress while curved arrows are back-tracking.

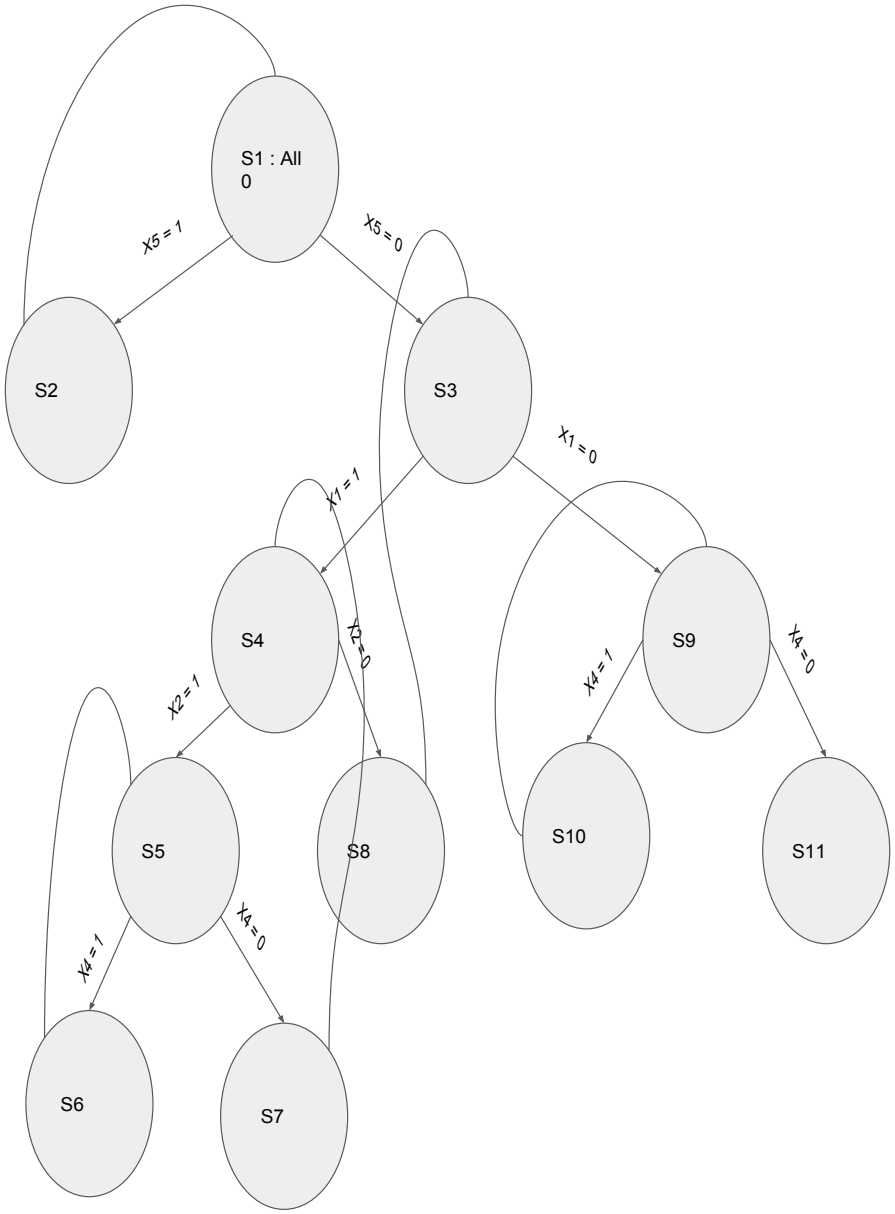

Figure 1: Figure illustrating How algorithm works

The standard 0-1 problem

Minimization problem : All constraints are $\geq$ type, all variables lie in $\{0, 1\}$, the minimization obj. function Z, has +ve coefficients for all variables.
Simple modifications can be made to convert any problem to the std. form

- $\leq$ type constraints can be made into $\geq$by multiplying with -1 on both sides
- A maximization problem can be made into a minimization problem by multiplying objective function by -1
- If there is a variable X_j with a negative coefficient in Z, then replace it by $X_j^{'} = 1 - X_j$. This may introduce a constant in Z which can be ignored and added in the end.
- **Handling Equations** : Say, $X_1+2X_2-X_3 = 1$, then we can write X_1 as $X_1 = 1-2X_2+X_3$ We can convert this into constraints using $X_1 \geq 0$, $X_1 \leq 1$. This gives the two constraints $1 - 2X_2 + X_3 \geq 0$ and $1 - 2X_2 + X_3 \leq 1$.
  In general, n equations will lead to n constraints of geq type and n of leq type.
- **Product Terms** : Say Z = $X_2X_3 + X_2^2 + X_3^3$. Same as $X_2X_3 + X_2 + X_3$.
  We can introduce a new variable $y = X_2X_3$ and convert the relationship to constraints.
  $X_2 + X_3 - y \leq 1$ and $-X_2 - X_3 + 2y \leq 0$
  Here y is also a 0-1 variable. Another method could have been to not define y as a 0-1 variable.

3 General All Integer Programming problem

Example : Maximize $X_1 + X_2$, subject to the constraints
$7X_1 - 5X_2 \leq 7$ and
$-12X_1 + 15X_2 \leq 7$
Where $X_1, X_2 \geq 0, X_1, X_2 \in \mathcal{Z}$
The steps in the general **Branch and Bound Algorithm** is

3.1 Solve the associate Linear Programming Problem

We get the solution $X_1^* = 28/9, X_2^* = 133/45, Z^* = 273/45$

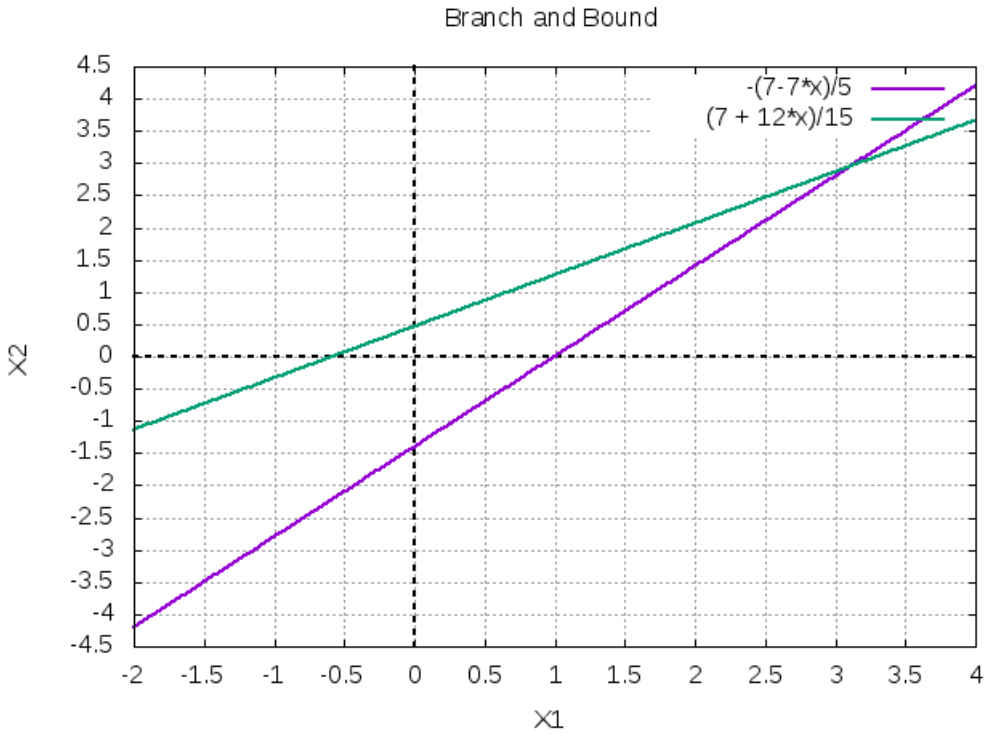

Figure 2: LPP type plot

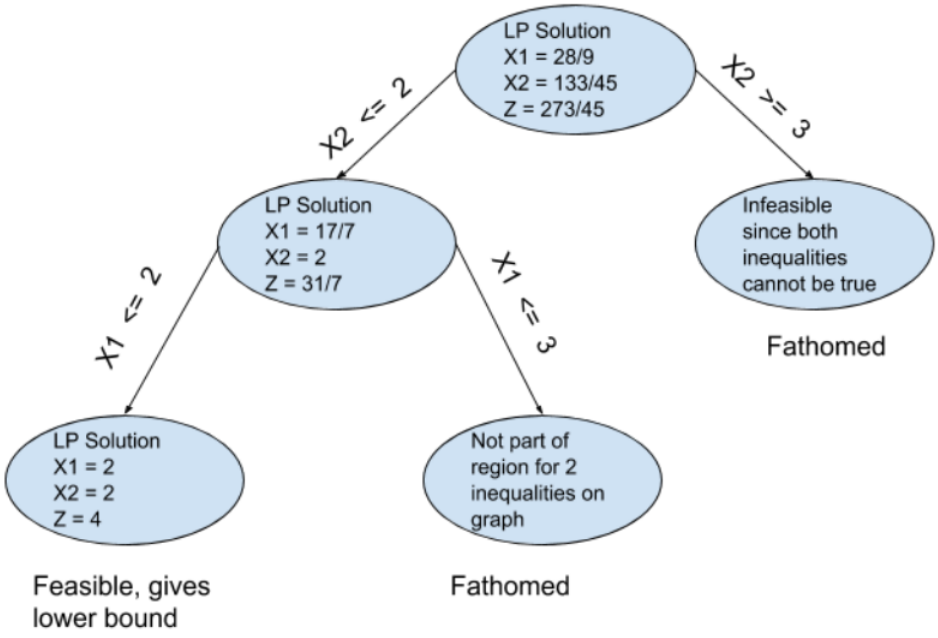

Figure 3: Branch and Bound Tree example

Description of a Branch and Bound Algorithm

Some ideas which affect the memory and time requirement of the IP problem.

- Bounding Strategy : Any LP optimum is an upper bound to the IP optimum that we can get by branching further on that node. Also we bound by feasibility and infeasibility.
- Branching Strategy : To branch out we need to pick a variable to branch over. Typically (No proven rule) we branch on the variable that has the largest fractional part. (greedy app.)
- Node Selection Strategy : We may need to choose between several 'active nodes'.
  In this example that was not the case there was only a single flow. But if this is not the case then we would proceed with the node with the higher LP optimum value, in the hope that it would give a higher IP optimum (Greedy Algorithm approach).

Memory requirement : Unlike 0-1 solving algorithm, the variables can take many values and we can have a branching on the same variable more than once. Hence we need to store many active nodes.
A solution is to sort active nodes in decreasing order of LP optimums.
Mixed Integer programming : The same approach can be used. It will be easier since we do not need to branch on the continuous variables.

Finding LP optimum quickly : We can use sensitivity analysis and upper bounded simplex to reduce computation (Since the problem is the same albeit with additional constraints)required to solve the associated LP problems as we go deeper.
The procedure is to solve every problem that comes up one by one using the simplex algorithm.

4 Gomory's Cutting plane Algorithm

Based on simplex ideas. Not clear as of now. Keeps chopping areas in feasible region until LP optimum is IP feasible.

5 All Integer Primal Algorithm

Based on simplex ideas. Not clear as of now. Solves maximization problem and leq constraints.

6 All Integer Dual Algorithm

Based on simplex ideas. Not clear as of now. Solves minimization problem and geq constraints.