

Question 1.

(100pts)

Program milp.m function in Matlab or some other scripting language equivalent to built-in intlinprog.m using branch and bound method. milp.m should use the same calling convention taking the same arguments and returning the same output except maybe the OUTPUT return value. You can use any Matlab functions that do not directly solve MILP problems.

In this homework, we are supposed to write a MATLAB function for mixed-integer linear programming. I used branch and bound algorithm since it is simple. The algorithm is similar to a binary tree. There are two common ways to code a tree, depth first or breadth first. In this problem, I will code in breadth-first order, so I use queue to record the order of nodes.

## Branching

To implement branch and bound algorithm, we need to figure out the branching conditions first. There are four possible branches in the problem.

### Branching 1

- 3 **linprog** converged to a solution X with poor constraint feasibility.
- 1 **linprog** converged to a solution X.
- 0 Maximum number of iterations reached.
- 2 No feasible point found.
- 3 Problem is unbounded.
- 4 NaN value encountered during execution of algorithm.
- 5 Both primal and dual problems are infeasible.
- 7 Magnitude of search direction became too small; no further progress can be made. The problem is ill-posed or badly conditioned.
- 9 **linprog** lost feasibility probably due to ill-conditioned matrix.

Branching 1 is when no solution available. In MATLAB linprog document, the only two conditions we can get a solution is exitflag = 1 or 3. So, the first base condition is when linprog does not give us any solutions. We can find it by checking exitflag of linprog. In this condition, we will just set the value of current node to inf. The program is unnecessary go deeper for this node since the child nodes will have more constraints than current and make the current problem even more unsolvable.

## Branching 2

Branching 2 is when we find a feasible solution and the solution is better (objective value is less than the best solution). So, we will record the current solution sets and update the overall bound. The program is unnecessary to go deeper for this node since child nodes will have more constraints and make the objective value bigger than the current.

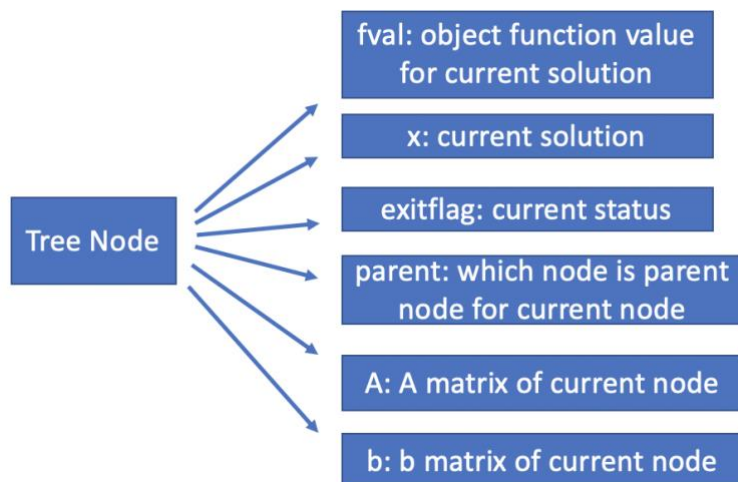
## Branching 3

Branching 3 is when we find a solution and the solution is worse than the best solution. Going deeper only makes the objective value bigger than the current, so we just set the value of current node to current and the program will not go deeper for this node.

## Branching 4

Branching 4 is when any integer constraints violations in our relaxation solution but potentially has better solution. In my program, it will choose the first violated variables and break each one into two subproblems. For example, if the violation variable is 3.2, the first subproblem will be added  $x \leq 3$  as a new condition and the second subproblem will be added  $x \geq 4$  as new condition of parent node and solve them respectively. We can create child nodes and add more constraints to child nodes in this branching condition. We also need add new node to queue here.

## Node Attributes



## Flowchart

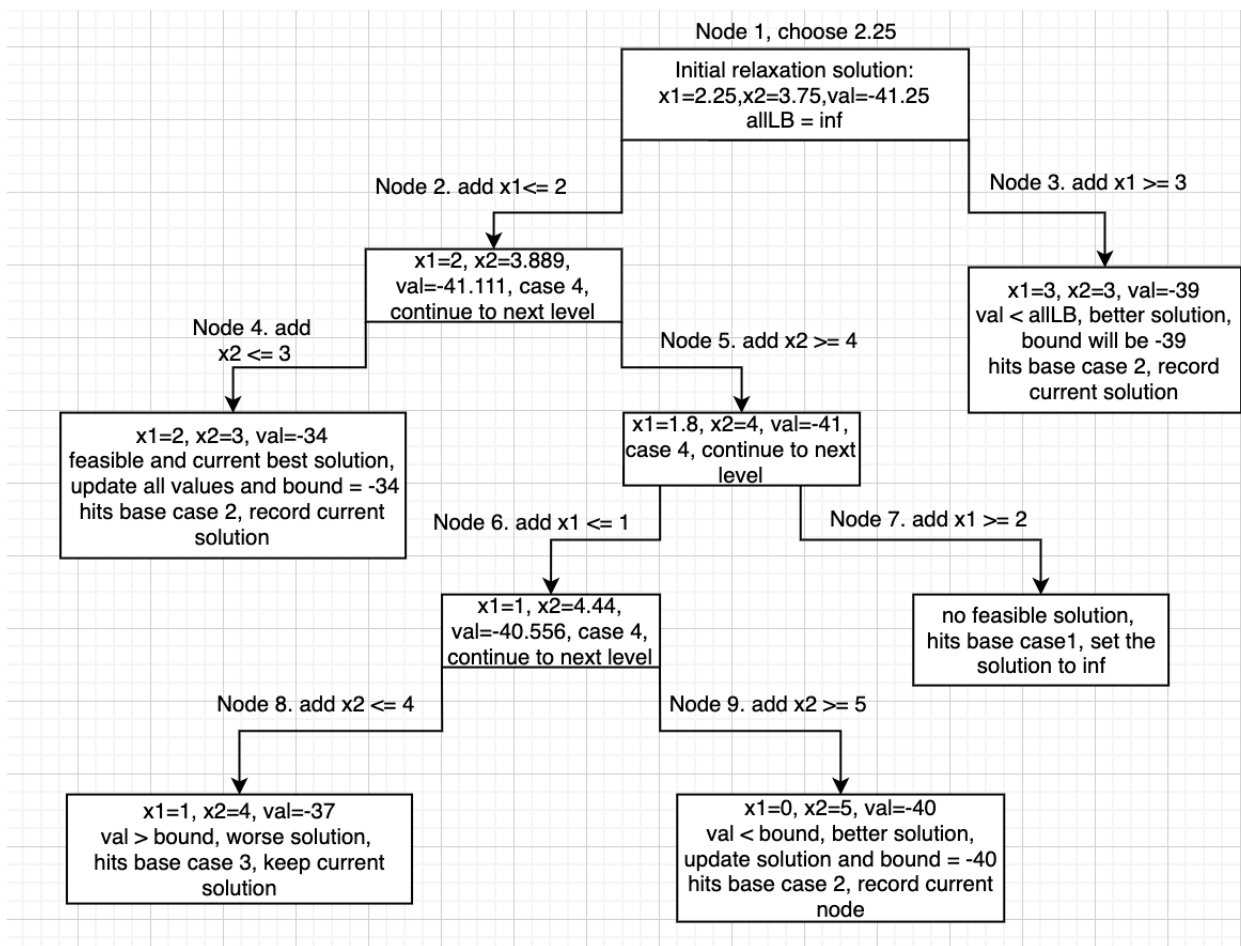
Next, I will use a real example to introduce how is my algorithm works.

$$\min -5x_1 - 8x_2$$

$$x_1 + x_2 = 6$$

$$5x_1 + 9x_2 = 45$$

$$x_1, x_2 \geq 0 \text{ Integer}$$

















## Function Testing

```
clear;
```

```
f=[-6 -2 -3 -5];  
A=[-3 5 -1 -6;2 1 1 -1;1 2 4 5];  
b=[-4 3 10]';  
Aeq = [1, 1, 3, -2];  
beq = [3];  
intcon=[1 2 3];  
LB=zeros(4,1);  
UB=[];
```

```
[x1,fval1,exitflag1] = milp(f,intcon,A,b,Aeq,beq,LB,UB);  
[x2,fval2,exitflag2]=intlinprog(f,intcon,A,b,Aeq,beq,LB,UB);
```

Workspace		Variables - tree
Name ▲	Value	
 A	3x4 double	
 Aeq	[1,1,3,-2]	
 b	[-4;3;10]	
 beq	3	
 exitflag1	1	
 exitflag2	1	
 f	[-6,-2,-3,-5]	
 fval1	-11.5000	
 fval2	-11.5000	
 intcon	[1,2,3]	
 LB	[0;0;0;0]	
 UB	[]	
 x1	[1;0;1;0.5000]	
 x2	[1;0;1;0.5000]	

My answer agree with MATLAB.