CIE 5015 Operations Research Lecture 7
Solving IPs with Branch and Bound

陳柏華 Albert Chen

National Taiwan University (NTU)

Department of Civil Engineering

November 1^{st}, 2019

Outline

Outline

The Lockbox Problem

The Lockbox Problem is another demonstration of the Integer Programming problem.

The Lockbox Problem will be used as an example when we discuss about variable selection for branching in the branch-and-bound method.

The Lockbox Problem

The Lockbox Problem is stated as follows:

A company receives credit card payments[1] from four regions of the USA country (West, Midwest, East, and South). The average daily value of payments mailed by customers from each region is as follows: the West, \$70,000; the Midwest, \$50,000; the East, \$60,000; the South, \$40,000.

The company must decide where customers should mail their payments. Because the company can earn 20% annual interest by investing these revenues, it would like to receive payments as quickly as possible.

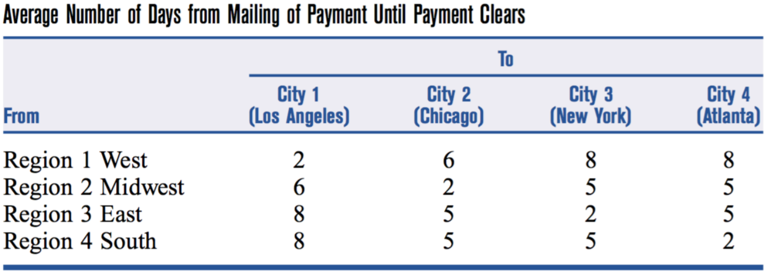
[1]payments are sent through checks (支票) in the old days...

The Lockbox Problem

The company is considering setting up operations to process payments (often referred to as lockboxes) in four different cities: Los $\overset{\text{West}}{\text{Angeles}}$, $\overset{\text{Midwest}}{\text{Chicago}}$, $\overset{\text{East}}{\text{New York}}$, and $\overset{\text{South}}{\text{Atlanta}}$. The cost of running a lockbox in any city is \$50,000 per day.

The Lockbox Problem

The average number of days (from the time payment is sent) until a check clears[2] so that the company can deposit the money, depends on the city to which the payment is mailed.



Average Number of Days from Mailing of Payment Until Payment Clears

	To			
From	City 1 (Los Angeles)	City 2 (Chicago)	City 3 (New York)	City 4 (Atlanta)
Region 1 West	2	6	8	8
Region 2 Midwest	6	2	5	5
Region 3 East	8	5	2	5
Region 4 South	8	5	5	2

[2] Clearing a check means to actually receive the money.

The Lockbox Problem

Assume that each region must send all its money to a single city and that there is no limit on the amount of money that each lockbox can handle.

Formulate an IP that the company can use to minimize the sum of costs due to lost interest and lockbox operations.

The Lockbox Problem

There are two types of decisions to be made:

- Establishment of lockboxes
- Which lockbox each region sends checks to

The Lockbox Problem

Establishment of lockboxes:

$$y_j = \begin{cases} 1 & \text{if a lockbox is operated in city } j \\ 0 & \text{otherwise} \end{cases}$$

For example, $y_2 = 1$ if a lockbox is operated in Chicago, and $y_3 = 0$ if no lockbox is operated in New York.

The Lockbox Problem

Which lockbox each region sends checks to

$$x_{ij} = \begin{cases} 1 & \text{if region } i \text{ sendspayments to city } j \\ 0 & \text{otherwise} \end{cases}$$

For example, $x_{12} = 1$ if the West sends payments to Chicago, and $x_{23} = 0$ if the Midwest does not send payments to New York.

The Lockbox Problem

The company wants to minimize total annual cost.

total annual cost =

Interest that could be earned

annual lost interest cost

+

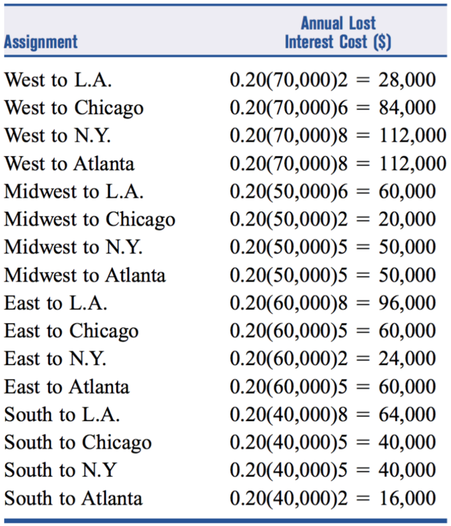
annual cost of operating lockboxes

The Lockbox Problem

For example, how much in annual interest would the company lose if customers from the West region sent payments to New York? On any given day, 8 days' worth, or 8(70,000) = \$560,000 of West payments will be in the mail and will not be earning interest.

Because the company can earn 20%, each day West funds will result in 0.20(560,000) = \$112,000 in lost interest.

The Lockbox Problem



Assignment	Annual Lost Interest Cost ($)
West to L.A.	0.20(70,000)2 = 28,000
West to Chicago	0.20(70,000)6 = 84,000
West to N.Y.	0.20(70,000)8 = 112,000
West to Atlanta	0.20(70,000)8 = 112,000
Midwest to L.A.	0.20(50,000)6 = 60,000
Midwest to Chicago	0.20(50,000)2 = 20,000
Midwest to N.Y.	0.20(50,000)5 = 50,000
Midwest to Atlanta	0.20(50,000)5 = 50,000
East to L.A.	0.20(60,000)8 = 96,000
East to Chicago	0.20(60,000)5 = 60,000
East to N.Y.	0.20(60,000)2 = 24,000
East to Atlanta	0.20(60,000)5 = 60,000
South to L.A.	0.20(40,000)8 = 64,000
South to Chicago	0.20(40,000)5 = 40,000
South to N.Y	0.20(40,000)5 = 40,000
South to Atlanta	0.20(40,000)2 = 16,000

The Lockbox Problem

$$
\begin{aligned}
\min z = {} & \overset{\overset{28{,}000}{\downarrow}}{28}x_{11} + 84x_{12} + 112x_{13} + 112x_{14} \\
& + 60x_{21} + 20x_{22} + 50x_{23} + 50x_{24} \\
& + 96x_{31} + 60x_{32} + 24x_{33} + 60x_{34} \\
& + 64x_{41} + 40x_{42} + 40x_{43} + 16x_{44} \\
& + 50y_1 + 50y_2 + 50y_3 + 50y_4 \\
& = \sum_i \sum_j c_{ij}x_{ij} + \sum_j d_j y_j
\end{aligned}
$$

The Lockbox Problem

Type 1 Constraint: Each region must send its payments to a single city.

Type 2 Constraint: If a region is assigned to send its payments to a city, that city must have a lockbox.

The Lockbox Problem

Type 1 Constraint: Each region must send its payments to a single city:

$x_{11} + x_{12} + x_{13} + x_{14} = 1$ (West region constraint)

$x_{21} + x_{22} + x_{23} + x_{24} = 1$ (Midwest region constraint)

$x_{31} + x_{32} + x_{33} + x_{34} = 1$ (East region constraint)

$x_{41} + x_{42} + x_{43} + x_{44} = 1$ (South region constraint)

$$\rightarrow \sum_{j} x_{ij} = 1, \forall i \in \{1, 2, 3, 4\}$$

The Lockbox Problem

Type 2 Constraint: If a region is assigned to send its payments to a city, that city must have a lockbox.

$x_{11} \leq y_1, x_{21} \leq y_1, x_{31} \leq y_1, x_{41} \leq y_1,$

$x_{12} \leq y_2, x_{22} \leq y_2, x_{32} \leq y_2, x_{42} \leq y_2,$

$x_{13} \leq y_3, x_{23} \leq y_3, x_{33} \leq y_3, x_{43} \leq y_3,$

$x_{14} \leq y_4, x_{24} \leq y_4, x_{34} \leq y_4, x_{44} \leq y_4$

$\rightarrow x_{ij} \leq y_j$, $\forall i \in \{1, 2, 3, 4\}$, and $\forall j \in \{1, 2, 3, 4\}$

The Lockbox Problem

$$\min \sum_i \sum_j c_{ij} x_{ij} + \sum_j d_j y_j$$

s. t.

$$\sum_j x_{ij} = 1,\ \forall i \in \{1,2,3,4\}$$

$$x_{ij} \le y_j,\ \forall i \in \{1,2,3,4\}, \text{ and } \forall j \in \{1,2,3,4\}$$

$$x_{ij} \text{ and } y_j \text{ binary},\ \forall i \in \{1,2,3,4\}, \text{ and } \forall j \in \{1,2,3,4\}$$

Outline

Integer Linear Program & It's Relaxed Problem

$\rightarrow$ have more x_1, x_2 to optimize obj.

Integer Linear Program

$\max 17x_1 + 12x_2$

s.t.

$10x_1 + 7x_2 \leq 40$

$x_1 + x_2 \leq 5$

$x_1, x_2 \geq 0$

$x_1, x_2 \in int$

in most case

$<$

Relaxed Linear Program

$\max 17x_1 + 12x_2$

s.t.

$10x_1 + 7x_2 \leq 40$

$x_1 + x_2 \leq 5$

$x_1, x_2 \geq 0$

~~$x_1, x_2 \in int$~~

Robert J. Vanderbei, ORF, Princeton

Facts on IP and Its Relaxed LP

- An integer linear program is a LP further constrained by the integrality restrictions.
- In a maximization problem, the value of the objective function, at the LP optimum, will always be an upper bound on the optimal IP objective. always true in max. problem.
- Any integer feasible point is always a lower bound on the optimal IP objective value.

(Bradley, Hax, and Magnanti, AMP, 1977)

Branch and Bound Example

$$\max 17x_1 + 12x_2$$

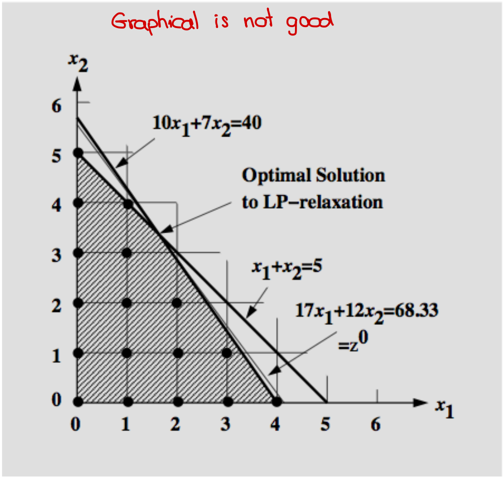
s.t.

$$10x_1 + 7x_2 \leq 40$$

$$x_1 + x_2 \leq 5$$

$$x_1, x_2 \geq 0$$

$$x_1, x_2 \in int$$



Robert J. Vanderbei, ORF, Princeton

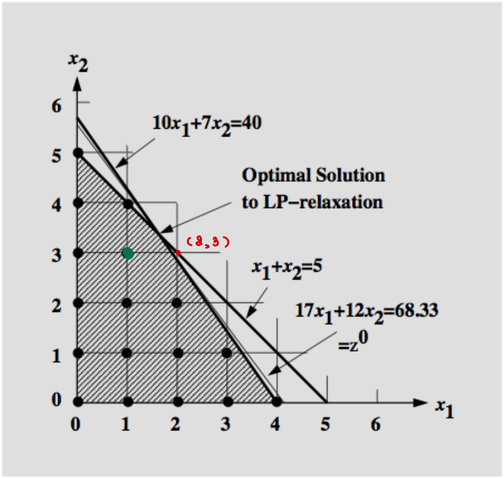
Branch and Bound Example

$\max 17x_1 + 12x_2$

Relaxed Solution
$= (x_1, x_2)$
$= (1.67, 3.33)$,
$z* = 68.33$,

$(2, 3)$ infeasible,

$(1, 3)$ suboptimal



Robert J. Vanderbei, ORF, Princeton

Branch and Bound Example

Branching on x_1

$x_1* = 1.67$

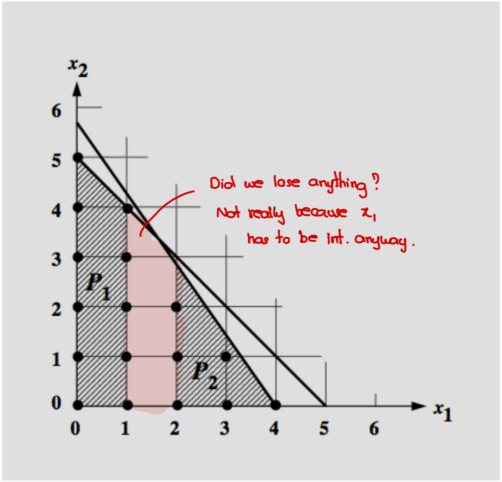
2 possibilities:

P_1 with $x_1 \leq 1$,

P_2 with $x_1 \geq 2$,

$P_1 : (1, 4)$ int,

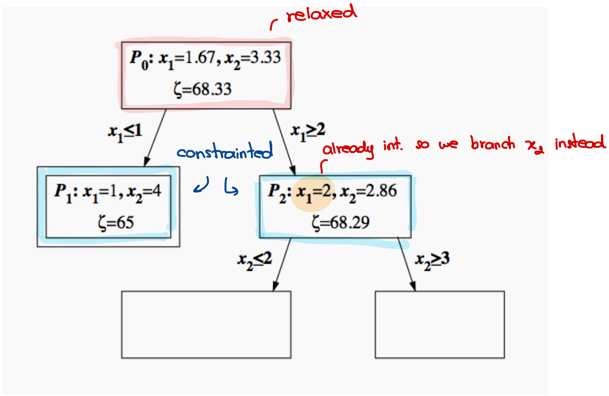
$P_2 : (2, 2.86)$



Robert J. Vanderbei, ORF, Princeton

Enumeration Tree

We can summarize the calculation progress/process in a tree-like graph.



Robert J. Vanderbei, ORF, Princeton

Repeatedly Splitting the Problem

You will keep solving relaxed Linear Programming problems, and further split them into smaller sub-problems (in solution space) with more constraints on the decision variables in hope you will find integer solutions.

But when can you stop branching or stop the entire process?

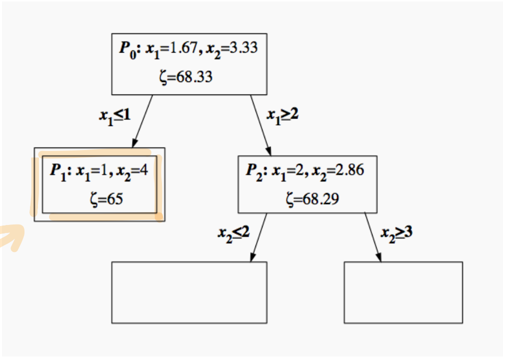
Enumeration Tree

zetha (?)

ζ is the current LP solution at a node, and ζ_{cur_lower} is the current know best lower bound (integer solution).

Fathom (stop branching at node):

1. int solution,
2. $\zeta \leq \zeta_{cur\_lower}$,
3. no feasible solution branching region is all fractional (no integer)



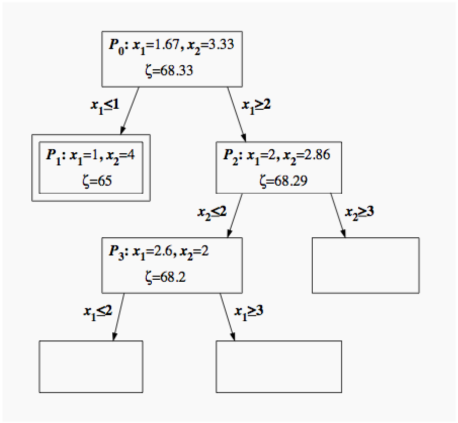
Robert J. Vanderbei, ORF, Princeton

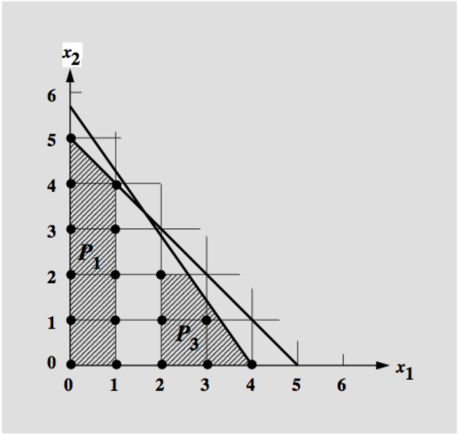
Enumeration Tree

Throughout the process, we keep track of facts:

1. The current tightest upper bound, which is the lowest objective value from a LP Relaxation.
2. The current best lower bound, which is the highest objective value from a feasible solution (int solution).

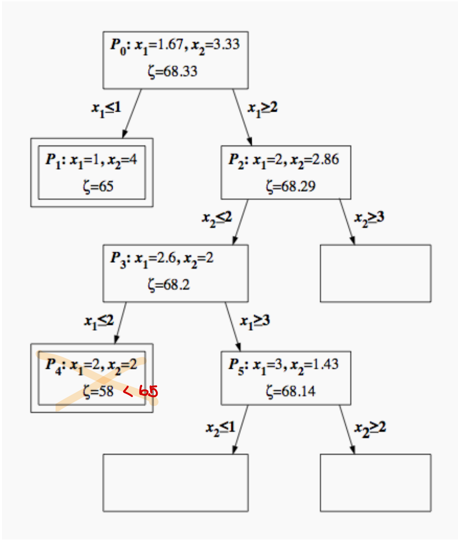
Further Steps

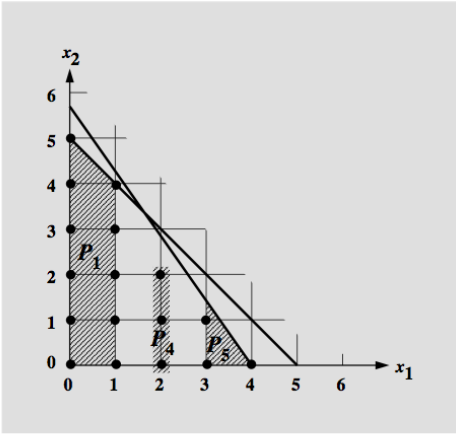




Robert J. Vanderbei, ORF, Princeton

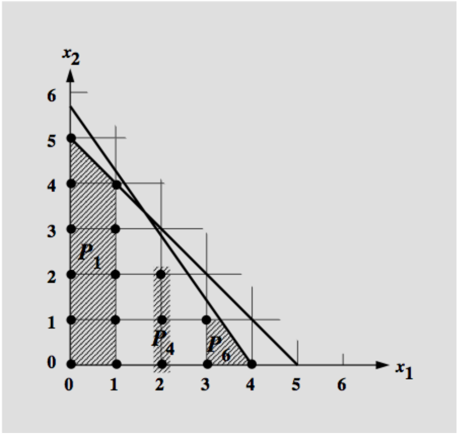
Further Steps

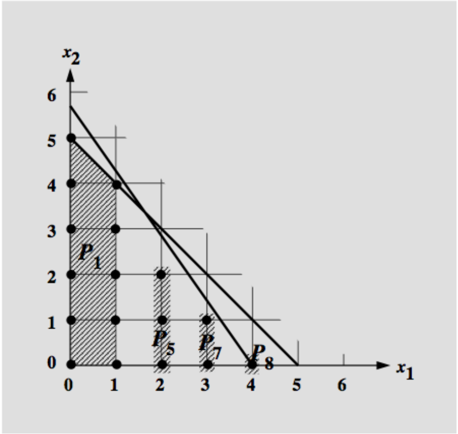




Robert J. Vanderbei, ORF, Princeton

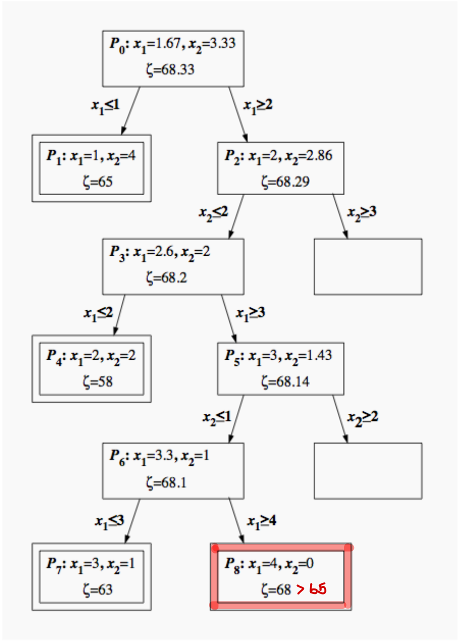
Further Steps



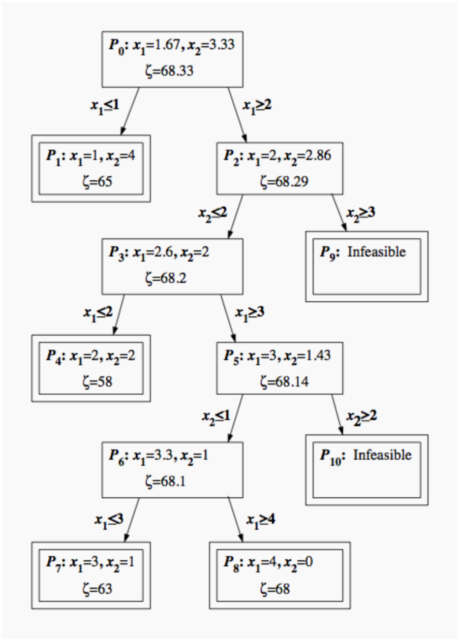


Robert J. Vanderbei, ORF, Princeton

Optimal: $(4, 0)$, $z = 68$



P_0: x_1=1.67, x_2=3.33 ζ=68.33
$x_1 \le 1$
$x_1 \ge 2$
P_1: x_1=1, x_2=4 ζ=65
P_2: x_1=2, x_2=2.86 ζ=68.29
$x_2 \le 2$
$x_2 \ge 3$
P_3: x_1=2.6, x_2=2 ζ=68.2
$x_1 \le 2$
$x_1 \ge 3$
P_4: x_1=2, x_2=2 ζ=58
P_5: x_1=3, x_2=1.43 ζ=68.14
$x_2 \le 1$
$x_2 \ge 2$
P_6: x_1=3.3, x_2=1 ζ=68.1
$x_1 \le 3$
$x_1 \ge 4$
P_7: x_1=3, x_2=1 ζ=63
P_8: x_1=4, x_2=0 ζ=68 > 65



P_0: x_1=1.67, x_2=3.33 ζ=68.33
$x_1 \le 1$
$x_1 \ge 2$
P_1: x_1=1, x_2=4 ζ=65
P_2: x_1=2, x_2=2.86 ζ=68.29
$x_2 \le 2$
$x_2 \ge 3$
P_3: x_1=2.6, x_2=2 ζ=68.2
P_9: Infeasible
$x_1 \le 2$
$x_1 \ge 3$
P_4: x_1=2, x_2=2 ζ=58
P_5: x_1=3, x_2=1.43 ζ=68.14
$x_2 \le 1$
$x_2 \ge 2$
P_6: x_1=3.3, x_2=1 ζ=68.1
P_{10}: Infeasible
$x_1 \le 3$
$x_1 \ge 4$
P_7: x_1=3, x_2=1 ζ=63
P_8: x_1=4, x_2=0 ζ=68

Branch and Bound

How do we choose which decision variable to branch on, and which subproblem to work on first?

Branch and Bound

Branching on the fractional-valued variable that has the greatest economic importance is often the best strategy.

In the lockbox example, suppose the optimal solution to a subproblem had y_1 and x_{12} fractional. Our rule would say to branch on y_1 because y_1 represents the decision to operate (or not operate) a lockbox in city 1, and this is presumably a more important decision than whether region 1 payments should be sent to city 2.

x_{12} is determined by $y_1 \Rightarrow y_1$ is more important

Branch and Bound

When more than one variable is fractional in a subproblem solution, many solvers will branch on the lowest-numbered fractional variable.

Thus, if an integer programming solver requires that variables be numbered, they should be numbered in order of their economic importance ($1 =$ most important, and in the lockbox case, to have the y_js set before the x_{ij}s).

Branch and Bound

Through experience, guidelines have been developed on how to make the necessary decisions.

Two general approaches are commonly used to determine which subproblems should be solved next.

- **Backtracking**
- **Jumptracking**.

Branch and Bound

Backtracking: feasible faster

The most widely used is the Last In First Out (LIFO) rule, which chooses to solve the most recently created subproblem. LIFO leads us down one side of the branch-and-bound tree and quickly finds a candidate (feasible) solution. Then we backtrack our way up to the top of the other side of the tree.

Branch and Bound

Jumptracking. optimal faster

When branching on a node, the jumptracking approach solves all the problems created by the branching. Then it branches again on the node with the best z-value. obj. value

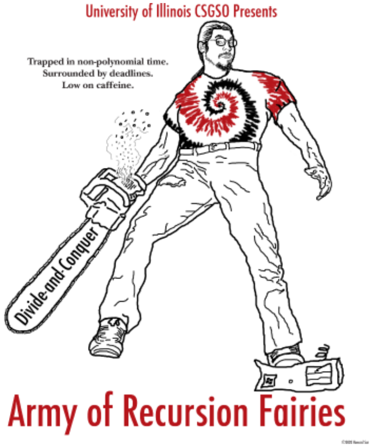
Jumptracking often jumps from one side of the tree to the other. It usually creates more subproblems and requires more computer storage than backtracking.

The idea behind jumptracking is that moving toward the subproblems with good z-values should lead us more quickly to the best z-value.

Branch and Bound

Branch-and-bound is essentially a strategy of "Divide-and-Conquer." The idea is to partition the feasible region into more manageable subdivisions and then, if required, to further partition the subdivisions.

(Bradley, Hax, and Magnanti, AMP, 1977)


University of Illinois CSGSO Presents
Trapped in non-polynomial time.
Surrounded by deadlines.
Low on caffeine.
Divide-and-Conquer
Army of Recursion Fairies

(Jeff Erickson, 2002)

Outline

MIP: Branch and Bound by Example

$\max 10 - 3x_1 - 2x_2$

s.t.

$x_1 - 2x_2 + x_3 = \frac{5}{2}$

$2x_1 + x_2 + x_4 = \frac{3}{2}$

$x_j \geq 0$, $j = 1, 2, 3, 4$

x_2 and x_3 are int

(Bradley, Hax, and Magnanti, AMP, 1977)

MIP: Branch and Bound by Example

$\max 10 - 3x_1 - 2x_2$

s.t.

$x_1 - 2x_2 + x_3 = \frac{5}{2}$

$2x_1 + x_2 + x_4 = \frac{3}{2}$

$x_j \geq 0,\ j = 1, 2, 3, 4$

x_2 and x_3 are int

(Bradley, Hax, and Magnanti, AMP, 1977)

Outline

Example of Gurobi in Python

```
from gurobipy import *

m = Model("IPExample")
```

c_i in obj.

```
x1 = m.addVar(vtype=GRB.INTEGER, name="x1", obj = 17)
x2 = m.addVar(vtype=GRB.INTEGER, name="x2", obj = 12)
```

c

$z = 17x_1 + 12x_2$

```
m.setAttr('ModelSense', GRB.MAXIMIZE)

# Variables and attributes are set in a lazy fashion
# in Gurobi. You need to update() to make it effective.
m.update()

m.addConstr(10 * x1 + 7 * x2 <= 40, "c0")
m.addConstr( 1 * x1 + 1 * x2 <=  5, "c1")
```

Example of Gurobi in Python

Continue from last page: "True" if debugging

```
m.setParam( 'OutputFlag', False )*

m.optimize()

print('Optimal from MIP solver:')
for v in m.getVars():
    print('%s: %f' % (v.varName, v.x))
print('Obj: %f' % m.objVal)
```

Output:

```
Optimal from MIP solver:
x1: 4.000000
x2: 0.000000
Obj: 68.000000
```

Gurobi’s Status

```
status = m.getAttr('Status')
print status
```

status =
1 LOADED
2 OPTIMAL
3 INFEASIBLE
4 INF_OR_UNBD
5 UNBOUNDED

http://www.gurobi.com/documentation/6.5/refman/optimization_status_codes.html

Classes in Python

Chapter 15, 16, and 17 of the book talks about Classes in Python.

Classes in Python - BBNode.py

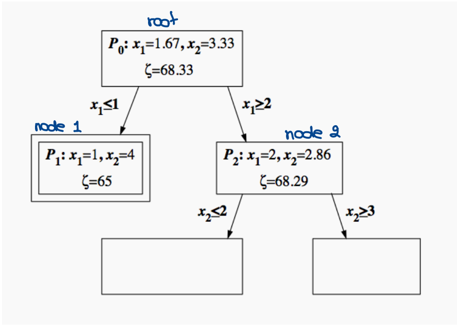
```
class BBNode(object): # save this in a file BBNode.py
    def __init__(self, parent, dv, boundSense, bound):
        # instance variables
        self.parent = parent
        self.dv = dv                 eg.  x_3 ≥ 3 = lower bound => "lb"
        self.boundSense = boundSense
        self.bound = bound eg. 3

    def set_obj(self, obj):
        self.obj = obj

    def set_dvSol(self, dvSol):
        self.dvSol = dvSol

    def set_status(self, status):
        self.status = status
```

Classes in Python - BBNode.py



```
import BBNode as bn
                  parent
root = bn.BBNode(None, '', '', 0)
node1 = bn.BBNode(root, 'x1', 'ub', 1)
node2 = bn.BBNode(root, 'x1', 'lb', 2)
node21 = bn.BBNode(node2, 'x2', 'ub', 2)
node22 = bn.BBNode(node2, 'x2', 'lb', 3)
```

Classes in Python - BBNode.py

```
dvNode = node22

print 'Showing bounds from node 22 to root'
while dvNode != root: to see all bound
    print dvNode.dv, dvNode.boundSense, dvNode.bound
    dvNode = dvNode.parent
```

Output:

```
Showing bounds from node 22 to root
x2 lb 3
x1 lb 2
```

Data Structure - Stack

The built-in list of Python has the capabilities to work as a *stack*, where the last element added is the first element retrieved (LIFO).

```
In  [1]: stack = [2,3,4]
In  [2]: stack
Out [2]: [2, 3, 4]
In  [3]: stack.append(5)
In  [4]: stack
Out [4]: [2, 3, 4, 5]
In  [5]: stack.pop()
Out [5]: 5
In  [6]: stack.pop()
Out [6]: 4
In  [7]: stack
Out [7]: [2, 3]
```

Homework 7-1

$$\max 8x_1 + 5x_2$$

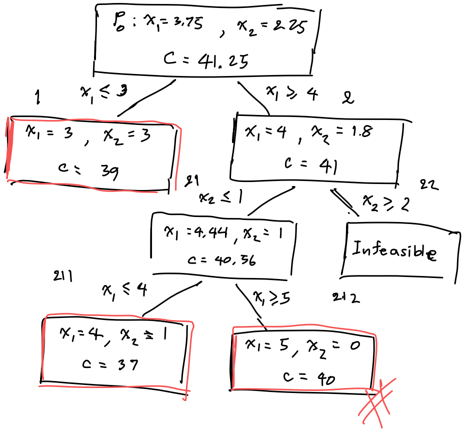
s.t.

$$x_1 + x_2 \leq 6$$

$$9x_1 + 5x_2 \leq 45$$

$$x_j \geq 0,\ j = 1, 2$$

x_1 and x_2 are int



Homework 7-1

For the previous problem, solve the problem using branch-and-bound.

Through the process, you can use Python and Gurobi to help you solve the Linear Programming sub-problems (not directly solving the Integer Program).

Please organize all your calculation (code) in a .py file, and output the answers for each subproblem.

HW is due the coming Thursday night.

Homework 7-Bonus

For the previous problem, please use Python and Gurobi to solve the problem using branch-and-bound, automatically.

Through the process, solve for the Linear Programming sub-problems (not directly solving the Integer Program).

Please organize all your calculation (code) in a .py file, and output the answers for each subproblem.

This bonus problem worths 2% of final grade.

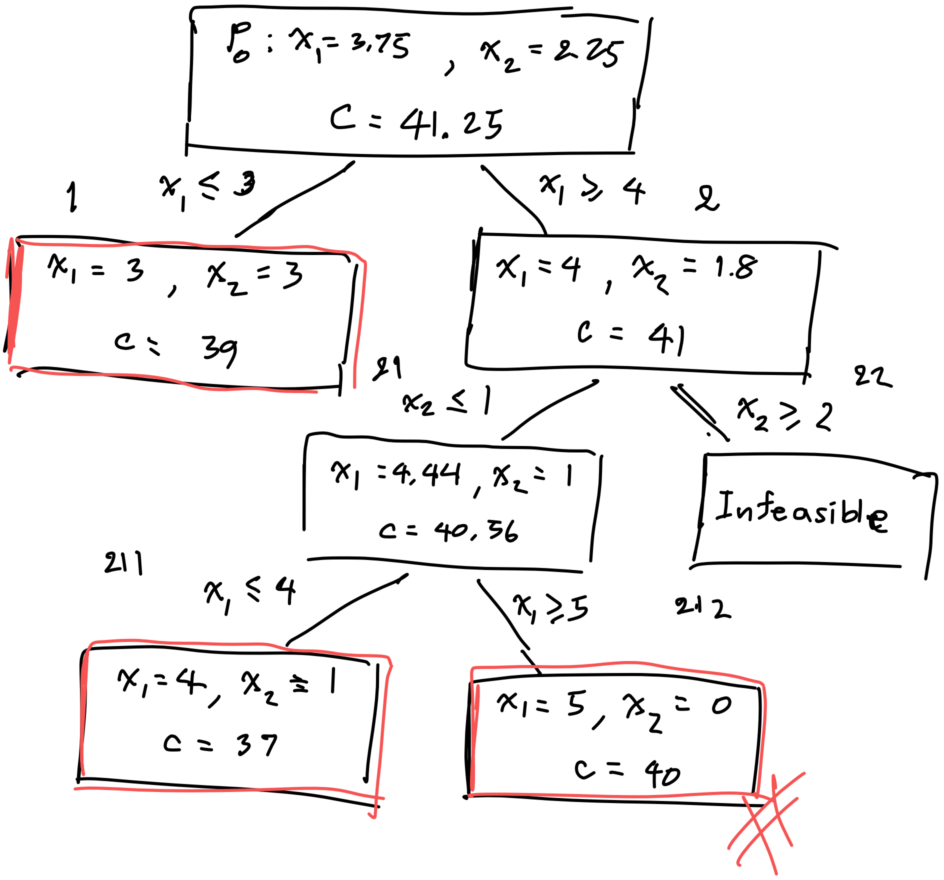
HW is due the coming Thursday night.

Next Week

Next week is for project discussion. There will be no class, and you are expected to ~~sepnd~~ spend the time to come up with a good project idea.

Next Lecture

1. Algorithms to solve Integer Programming Problems
2. *email: AlbertChen@ntu.edu.tw*


$P_0 : x_1 = 3.75 , x_2 = 2.25$
$C = 41.25$
1
$x_1 \leq 3$
$x_1 \geq 4$
2
$x_1 = 3 , x_2 = 3$
$c = 39$
$x_1 = 4 , x_2 = 1.8$
$C = 41$
21
$x_2 \leq 1$
22
$x_2 \geq 2$
$x_1 = 4.44 , x_2 = 1$
$c = 40.56$
Infeasible
211
$x_1 \leq 4$
$x_1 \geq 5$
212
$x_1 = 4, x_2 = 1$
$c = 37$
$x_1 = 5 , x_2 = 0$
$c = 40$