

15.053

Tuesday, May 1

- **Branch and Bound**

Quotes of the Day

The time to relax is when you don't have time for it.

**-- Attributed to Jim Goodwin and
Sydney J. Harris**

There is more to life than increasing its speed.

-- Mohandas K. Gandhi

Overview

- **Branch and bound is a clever way of enumerating all possible solutions.**
- **We first start by understanding “bounding.” In particular, how can solving LPs provide useful information for solving IPs?**

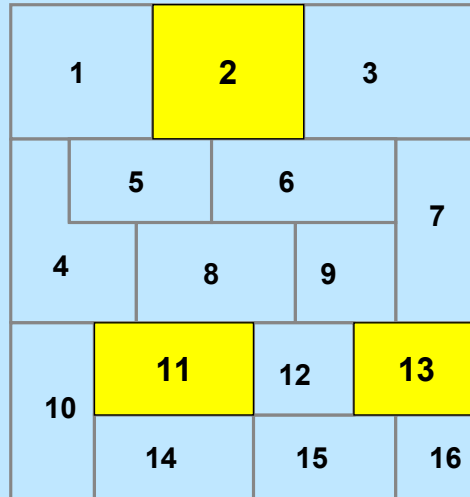
3

You will recall that when we are maximizing, any solution to the dual of an LP gives an upper bound on an LP. But here, we want upper bounds on an integer program when we are maximizing.

It turns out that there is a straightforward way to get an upper bound on an integer program in which we are maximizing. We will see it soon.

The 053 Chocolate Store Problem

Locate a minimum number of stores so that each district has one or is adjacent to one.



$$z_{IP} = \text{Min} = x_1 + \dots + x_{16}$$

$$x_1 + x_2 + x_4 + x_5 \geq 1$$

$$x_1 + x_2 + x_3 + x_5 + x_6 \geq 1$$

...

$$x_{13} + x_{15} + x_{16} \geq 1$$

$$x_j \in \{0,1\} \text{ for all } j$$

4

The upper bound is obtained by dropping the integrality constraints, and thereby giving the decision maker many more options. That is, the feasible region will be enlarged, and so the optimum solution will be better (or at least it will be no worse).

We call the problem obtained by dropping the integrality constraints the “linear programming relaxation.”

When we relax binary constraints on x_j , we replace them by the linear constraints $0 \leq x_j \leq 1$. It is important to remember to include these constraints as part of any linear programming relaxation.

Using LPs solution to get bounds

The ***LP relaxation*** of an integer program is what you get if you remove the integrality constraints from an integer program.

$$z_{LP} = \text{Min } x_1 + \dots + x_{16}$$

Note: we replace
 $x_j \in \{0,1\}$ with $0 \leq x_j \leq 1$.

$$x_1 + x_2 + x_4 + x_5 \geq 1$$

$$x_1 + x_2 + x_3 + x_5 + x_6 \geq 1$$

...

$$x_{13} + x_{15} + x_{16} \geq 1$$

$$0 \leq x_j \leq 1 \text{ for all } j$$

Theorem: For a minimization problem $Z_{IP} \geq Z_{LP}$.

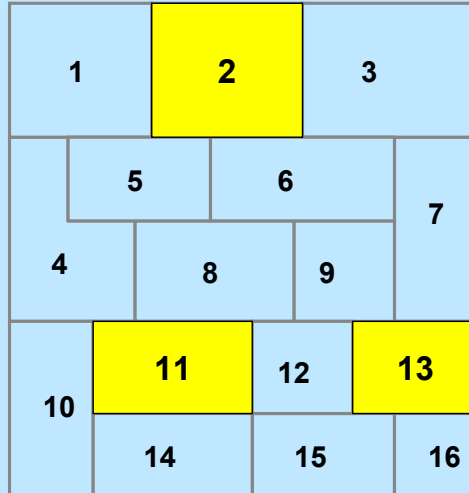
(Why?)

5

Since the covering problem is a minimization problem, the LP relaxation will give a lower bound on the optimum objective value. That is, by eliminating constraints and making the feasible region larger, the objective will improve or stay the same; that is, it will get lower or stay the same.

We can write this as $Z_{LP} \leq Z_{IP}$, as was done on this slide, where Z_{IP} is the optimum solution value for the original integer program, and Z_{LP} is the optimum solution value for the linear programming relaxation.

Using the LP bounds to help solve the IP



The **LP solution** to this problem was integral.
 $x_2 = 1, x_{11} = 1, x_{13} = 1$
all other variables were 0. **The solution value** is 3.

Theorem: If the optimal solution for an LP relaxation is feasible for the IP, then it is also optimal for the IP.

(Why?)

6

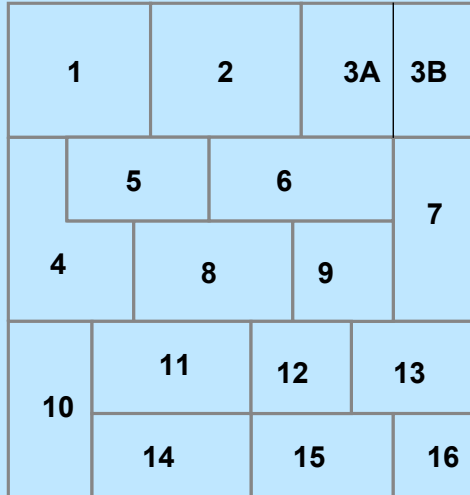
When I solved the linear programming relaxation using Excel Solver, I found out that $Z_{LP} = 3$. This implies that $Z_{IP} \geq 3$. You may recall in class that we found a solution for the integer program with objective value 3. Since no solution could be better, this solution is now provably optimal.

Note, we could have proved it optimal by considering all possible solutions in which only two districts were selected (that is, 15.053 Chocolates were located in only two districts.) We would have seen that none of these solutions was feasible.

But instead, we solved a single linear program, and proved that we needed at least three stores to be located. In general, the bounds obtained by solving linear programs can be extremely useful in solving the integer programs.

A Modified Version of the Chocolate Problem

This version differs from the last one in that District 3 was replaced by two different districts.



$$z_{IP} = \text{Min} = x_1 + \dots + x_{16}$$

$$x_1 + x_2 + x_4 + x_5 \geq 1$$

$$x_1 + x_2 + x_{3A} + x_5 + x_6 \geq 1$$

...

$$x_{13} + x_{15} + x_{16} \geq 1$$

$$x_j \in \{0,1\} \text{ for all } j$$

7

Here we consider a very similar integer program. We split district 3 into two districts, labeled 3A and 3B. The previous integer programming solution that located 15.053 stores in 3 districts is no longer feasible because district 3B is not covered. You can try, but you will not be able to find any feasible solution with only three districts.

The Solution for the LP Relaxation

The LP relaxation allows fractional stores to be opened. The number of stores (sum of fractions) adjacent to a district is at least 1.

1	.25	.5	3B
.75	5	.25	.5
	8	9	
.25	11	12	.25
	14	.75	16

z_{LP} = The total number of fractional stores is 3.5.

$$z_{IP} \geq z_{LP} = 3.5$$

Can we conclude that $z_{IP} \geq 4$?

8

When we solve the LP relaxation, we get this fractional solution. At first glance, it looks very odd. It corresponds to locating fractional stores. But if you look at it closely, it starts to make sense. Of course, it doesn't tell us how to locate stores since one cannot build a fractional store.

But if we were permitted to locate fractional stores, we could satisfy all of the constraints with only 3.5 stores. For example, look at district 1. There is .25 stores located in district 2 and .75 stores located in district 4. So the total number of stores located in or next to district 1 is 1. In other words, the constraint for district one is satisfied.

If we look at district 6, we will notice that there is .25 stores located in district 2, .5 stores located district 3A, .25 stores located in district 6, and .5 stores located in district 10, for a total coverage of 1.5 stores. So, the constraint for district 6 is also satisfied.

In fact, every district has the sum of the fractional stores adjacent to it (or in it) summing to at least 1. So, this solution is feasible for the linear programming relaxation.

So, we conclude that $z_{IP} \geq z_{LP} = 3.5$. But we also know that z_{IP} is integer valued. So, we further conclude that $z_{IP} \geq 4$.



Theorem: For a minimization problem, if all cost coefficients of a pure integer program are integers, then $Z_{IP} \geq \lceil Z_{LP} \rceil$.

Proof. $Z_{IP} \geq Z_{LP}$ and Z_{IP} is integer valued.

Conclusion: we need at least 4 stores to cover the districts.

Here is a solution with four stores. Since we know that $z_{IP} \geq 4$, we know that we can not find a solution that is any better. So, this solution must be optimal.

Summary of useful facts.

Let z_{IP} be the optimal objective value for a minimization integer program.

Let z_{LP} be optimal for the linear relaxation.

1. $z_{IP} \geq z_{LP}$.
2. If the solution for the linear relaxation is feasible for the IP, then it is optimal for the IP.
3. If all costs are integral, then $z_{IP} \geq \lceil z_{LP} \rceil$.
4. If the LP is infeasible, then so is the IP.

10

There are several techniques for finding lower bounds for minimization problems, or analogously finding upper bounds for maximization problems. In this lecture, we will focus on a single technique. We will drop the integrality constraints and solve an LP relaxation.

For a minimization problem, $z_{IP} \geq z_{LP}$.

If all costs are integral, then so is z_{IP} , and we conclude that $z_{IP} \geq \lceil z_{LP} \rceil$.

For a maximization problem $z_{IP} \leq z_{LP}$.

If all costs are integral, then so is z_{IP} , and we conclude that $z_{IP} \leq \lfloor z_{LP} \rfloor$.

Overview of this lecture

- **Explain and illustrate branch and bound.**
 - **It is the starting point for all solution techniques for integer programming**
 - **Lots of research has been carried out over the past 40 years to make it more and more efficient**
 - **But, it is an art form to make it efficient. (We shall get a sense why.)**
 - **Integer programming is intrinsically difficult.**

11

So far, we have seen two examples in which we could prove that an integer programming solution is optimal by solving a single linear programming relaxation. We will not be so fortunate in general. But nevertheless, the solving of linear programming relaxations to get upper bounds for maximization problems (or lower bounds for minimization problems) is a fundamental tool and subroutine for solving integer programs.

Trading for Profit Game

Prize	iPod	server	brass rat	gift certificate	6.041	15.053
	1	2	3	4	5	6
Points	5	7	4	3	4	6
Utility	16	22	12	8	11	19

Budget: 14 IHTFP points.

$$\text{maximize } 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6$$

$$\text{subject to } 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14$$

$$x_j \text{ binary for } j = 1 \text{ to } 6$$

12

We are going to illustrate how to solve integer programs on the IHTFP problem.

This problem is actually rather tricky to solve. The optimum solution is as follows:

$$x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 1. \quad z = 43.$$

We can find this by solving the problem using Excel Solver. However, today we will discuss what Excel Solver needs to do to solve this problem. It uses a technique called "Branch and Bound."

Complete Enumeration

- **Systematically considers all possible values of the decision variables.**
 - If there are n binary variables, there are 2^n different ways.
- **Usual idea: iteratively break the problem in two. At the first iteration, we consider separately the case that $x_1 = 0$ and $x_1 = 1$.**

13

Before we get to Branch and Bound, we will illustrate a very simple approach called “complete enumeration.” Given that there are only 6 binary variables, the number of ways of assigning values to these binary variables is $2^6 = 64$. Complete enumeration, considers all 64 different solutions, and then chooses the best one. This is an incredibly efficient way of solving a problem with 6 binary variables. It is even very efficient if there are 20 binary variables. But we shall soon see that it doesn’t scale up at all well.

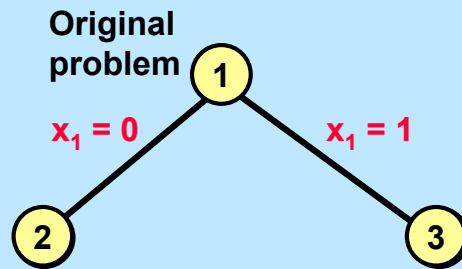
An Enumeration Tree

Original
problem ①

14

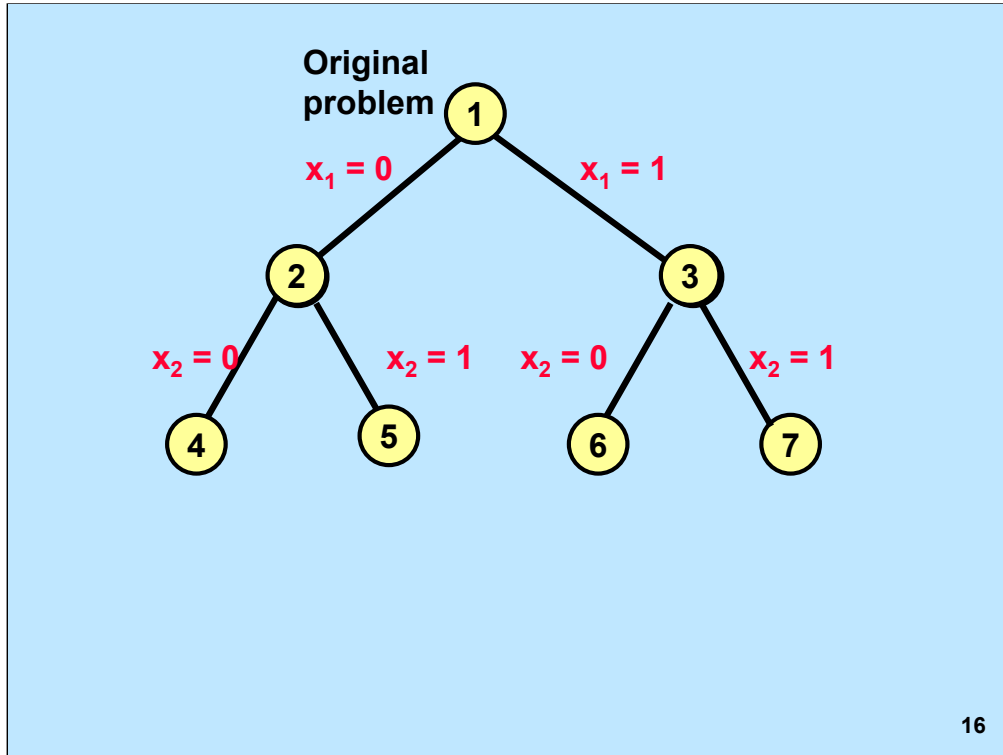
In enumerating all solutions, we will do it in a systematic manner. We write a node of the tree representing that no variables have been fixed in value.

An Enumeration Tree



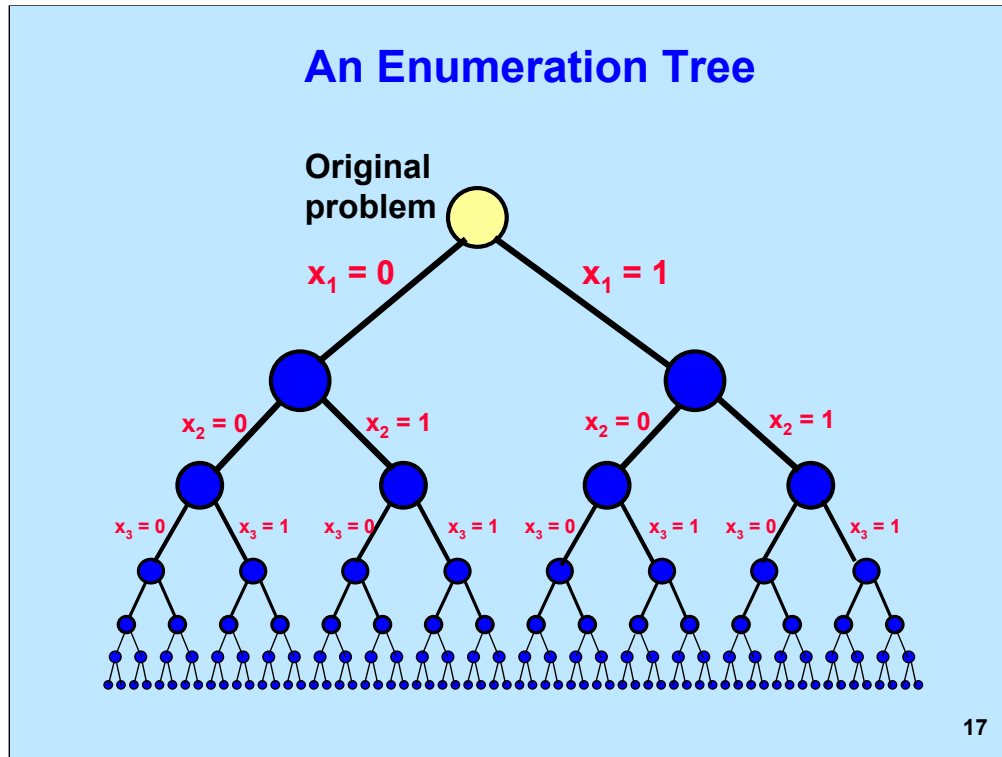
15

We then consider two possibilities. Either $x_1 = 0$ or $x_1 = 1$. We make two branches off of node 1, creating nodes 2 and 3 of the tree.



We then consider two possibilities for the second prize, the server. Either $x_2 = 0$ or $x_2 = 1$. We then make two branches off of node 2 and two branches off of node 3.

An Enumeration Tree



17

We then consider two possibilities for x_3 . Either $x_3 = 0$ or $x_3 = 1$. We and make two branches off of nodes 4, 5, 6 and 7 of the tree.

We then consider two possibilities for x_4 and make branches.

And we consider two possibilities for x_5 and make branches.

Finally, we consider two possibilities for x_6 and make branches.

At the end, we have 64 leaves of the tree (the nodes at the bottom), each corresponding to a complete assignment of variables.

We can then evaluate all of these solutions, and choose the best one.

On complete enumeration

- Suppose that we could evaluate 1 billion solutions per second.
- Let n = number of binary variables
- Solutions times
 - $n = 30$, 1 second
 - $n = 40$, 17 minutes
 - $n = 50$ 11.6 days
 - $n = 60$ 31 years
 - $n = 70$ 31,000 years

18

For those not used to exponential growth, evaluating all solutions sounds like a viable approach. After all, computers are very fast. So, suppose we could evaluate 1 billion solutions per second. Then we could solve a problem with 30 variables in a second. This sounds pretty good.

But as we move to 40, 50, 60 and 70 variables, the solution times degrade horribly. Enumerating all solutions for a problem with 70 binary variables would take 31,000 years.

On complete enumeration

- Suppose that we could evaluate 1 trillion solutions per second, and instantaneously eliminate 99.9999999% of all solutions as not worth considering
- Let n = number of binary variables
- Solutions times
 - $n = 70$, 1 second
 - $n = 80$, 17 minutes
 - $n = 90$ 11.6 days
 - $n = 100$ 31 years
 - $n = 110$ 31,000 years

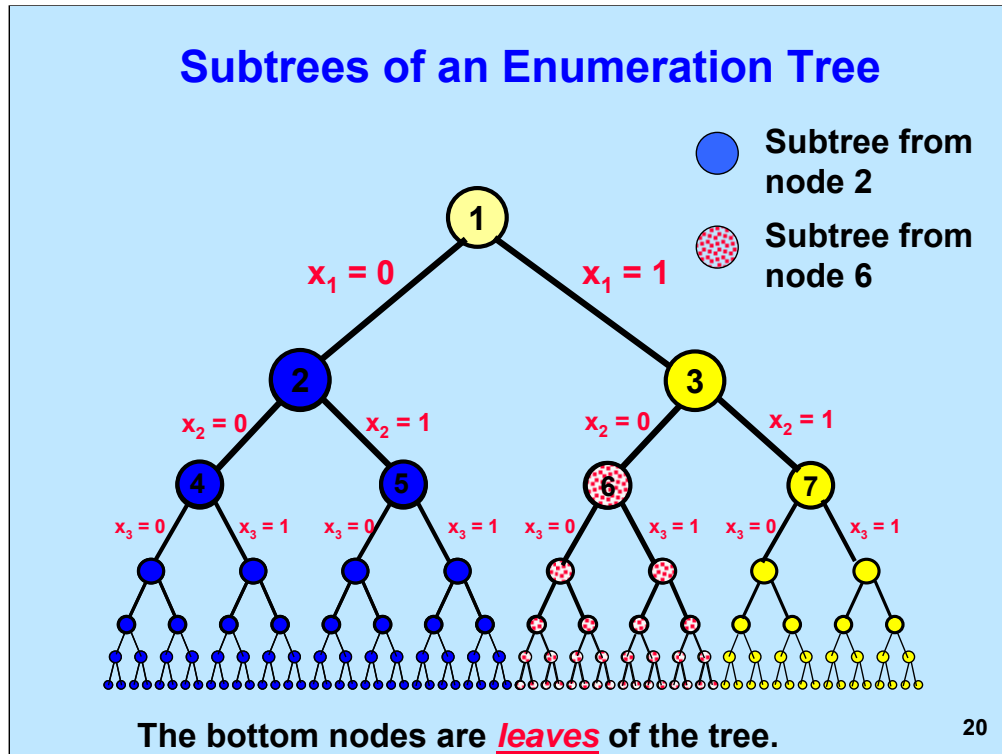
19

Often students are not ready to give up on complete enumeration at this point. They argue that computers will get even faster, and that we can often eliminate the fast majority of solutions immediately because they clearly won't be optimal.

So, in this case, we assume that we evaluate 1 trillion solutions per second. And we assume that we need only evaluate 1 out of 10^{10} different solutions.

At first glance, this seems to have solved the problem. What used to take 31,000 years can now be solved in 1 second. But as we try to scale up, we run into the exact same problem. By the time we try to solve a problem with 110 binary variables, the running time is 31,000 years.

This seems very discouraging, and it may lead one to conclude falsely that enumeration based techniques have no chance of working. Actually, branch and bound can be remarkably effective in solving problems with several hundred binary decision variables. We shall soon explain branch and bound, and give a sense for why it is so useful in practice.



We will consider the enumeration tree from before.

The key idea in Branch and Bound is to stop branching from a node as soon as possible.

Consider for example, all of the blue nodes. They are all the nodes obtained from node 2 by subsequent branching. We will refer to this as the subtree from node 2. We refer to the blue nodes as the “descendants” of node 2.

Suppose that we look at node 2 and conclude that none of its descendants can be optimal. (Don’t worry yet how we will make that conclusion). If we can do this, then we will have eliminated 32 leaves of the tree, or equivalently, we would have eliminated half of the tree at once. This sounds pretty good for a 6 variable problem. For a 50 variable problem, it would eliminate half a quadrillion solutions immediately, which would be amazingly good.

The essence of Branch and Bound is to create an enumeration tree one branch at a time, and to try to eliminate subtrees as soon as possible.

Something needed for Branch and Bound: The incumbent.

We need a feasible solution to the integer program.
We call this the *incumbent*.

Suppose that x_i is the incumbent.

Let z_i be its objective value.

Important question:
how does one find
an incumbent?

We'll deal with that
later. Let us just
assume we have one.

Starting incumbent (which I found by inspection.)

$x_1 = 1; x_2 = 1; x_3 = x_4 = x_5 = x_6 = 0; z_i = 38;$

21

When running the algorithm, we will typically keep track of the best solution so far, and we will call this solution the incumbent. Most branch and bound algorithms have a special subroutine run at the beginning that tries to get a good feasible solution. The details of these subroutines are quite complex.

Let it suffice to say that having a good solution on hand is very important to the performance of the algorithm in practice, as we shall soon see.

The Essence of Branch and Bound

- **Select nodes of the “enumeration tree” one at a time. But branch from a node if none of its descendants can be a better solution than that of the incumbent.**

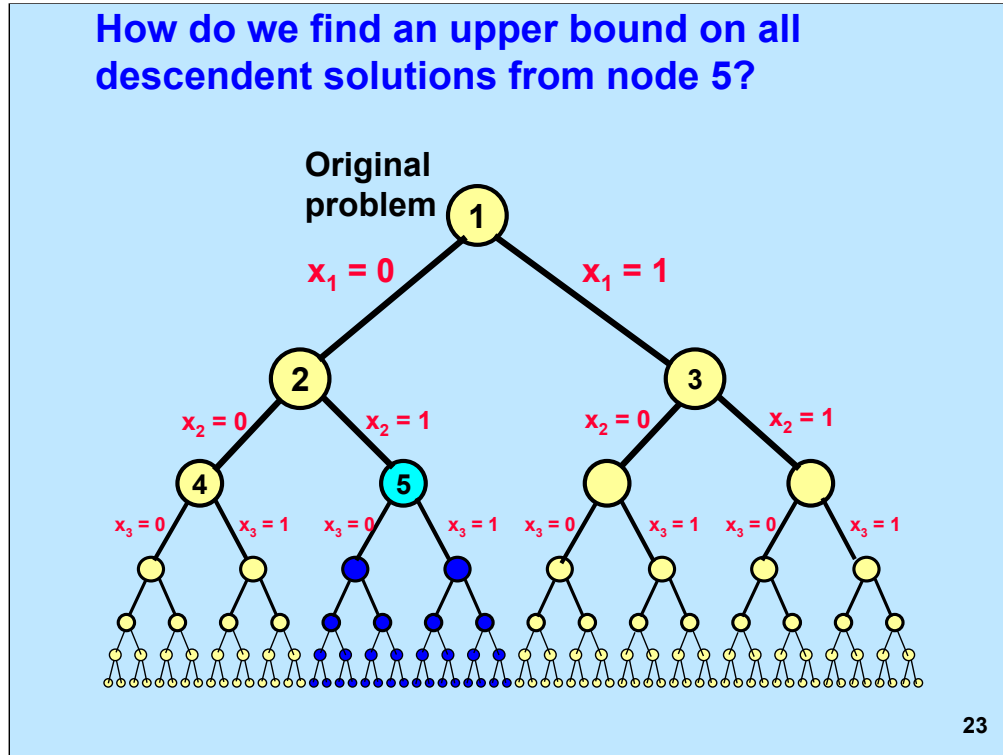
22

To eliminate a subtree, say one starting at node 2, we need three things:

1. We need a current feasible solution called the incumbent. Suppose that the solution value for this incumbent is z_I .
2. We need a bound at node 2. This bound for a maximization problem will be an upper bound on the objective value for all descendants for the node 2. We will obtain this bound by solving a linear programming relaxation (more on this later). For now, let us say that the value of the bound is $z_{LP}(2)$, which means that every descendent solution of node 2 has an objective value that is at most $z_{LP}(2)$.
3. We need that $z_{LP}(2) \leq z_I$.

Note, if we have all three conditions, then we can conclude that every descendent solution of node 2 in the enumeration tree has an objective value that is at most $z_{LP}(2)$ and thus is at most z_I . Thus, none of the descendants of node 2 can improve upon the objective value of the current incumbent solution. So, we will not branch any further from node 2 of the tree.

How do we find an upper bound on all descendent solutions from node 5?



At some point in the Branch and Bound algorithm we will look at node 5. In principle, we will create two children nodes, one found by setting x_3 to 0 and one found by setting x_3 to 1. But it would be far better if we could conclude a priori that no descendent of node 5 can be a better solution than the incumbent. So, we need to find an upper bound, which we discuss next.

Finding an upper bound for descendants of node 5 (or any other node)

Original problem

To find the optimum descendent of node 5, we can solve the following IP called Subproblem 5.

Subproblem (5)

$$\begin{aligned} \max \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & x_1 = 0, x_2 = 1 \quad x_j \text{ binary for } j = 3 \text{ to } 6 \end{aligned}$$

The LP relaxation:

$$\begin{aligned} \max \quad & z_{LP}(5) = 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & x_1 = 0, x_2 = 1 \quad 0 \leq x_j \leq 1 \text{ for } j = 3 \text{ to } 6. \end{aligned}$$

$Z_{LP}(5) = 44$. Found by solving the LP.

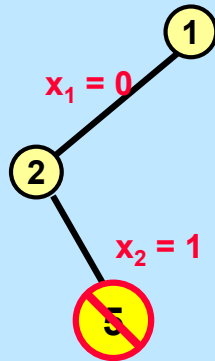
24

Recall that the leaves that correspond to complete solutions, that is, every variable is specified. We want an upper bound on all of the solutions that are descendants of node 5.

The maximum value descendent of node 5 can be found by solving Subproblem 5. This is the original integer program except that x_1 is required to be 0, and x_2 is required to be 1. (All descendants of node 5 have this property.) We could in principle solve Subproblem 5, but this requires solving an integer program at an intermediate step. That would be too much work.

So instead, we solve the LP relaxation of Subproblem 5. This also gets us an upper bound on all solutions that are descendants of node 5. But it is far easier to solve an LP. We call the optimum objective value $z_{LP}(5)$. In this case, $z_{LP}(5) = 44$.

Can we eliminate node 5?



The incumbent solution has value $z_i = 38$

$$z_{LP}(5) = 44.$$

Possibly, some descendent of node 5 has a better solution value than 38.

Conclusion: we cannot stop enumerating solutions from node 5. We need to branch from node 5.

There would be no further branching from node 5 if $z_i = 45$.

But suppose that we had an incumbent with $z_i = 45$.

Then no descendent of node 5 can be better than z_i . We can *fathom* node 5.

25

At this point, we are disappointed because we have spent time solving an LP, and we still need to keep branching from node 5. We learned that the max profit descendent of node 5 has an objective value of at most 44. But our current best solution has objective 38. So, it is possible that a solution that is a descendent of node 5 is better than our current incumbent.

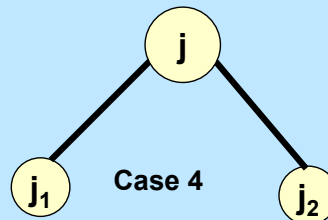
Suppose instead that the value of the incumbent were 45. Then we would be much happier (assuming that solving integer programs faster makes you happy). No solution that is a descendent of node 5 can have an objective value better than 44. Thus no solution that is a descendent of node 5 is better than the incumbent. There is no reason to branch on node 5 of the tree since we cannot find a better solution. This greatly speeds up the time to search the tree.

Branch and Bound overview

- Branch and bound creates the enumeration tree, one node at a time, and one branch at a time.
- Before branching on a node j , it solves $LP(j)$. Depending on the solution to $LP(j)$, Branch and Bound either fathoms node j or it branches on node j and creates two children.



Cases 1, 2, 3



26

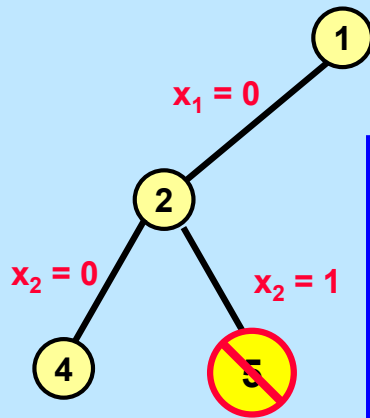
So, when we look at any node j of the branch and bound tree, we will solve a linear programming relaxation, obtaining $z_{LP}(j)$ as well as the solution that gives that objective value. We will look at four possible outcomes of solving that LP. In three of the cases, we establish that no solution that is a descendent of node j can be better than the incumbent. In these cases, we “fathom” node j , meaning that we do not create any children of node j .

(I suppose that if we liked the use of the word “children” we could use the expression “neuter” node j . But, I do not believe that term is ever used.)

In the fourth case, we cannot fathom node j , and we create two children of node j by branching. The left child corresponds to setting some variable to 0, and the right child corresponds to setting that variable to 1.

Unfortunately for us, case 4 seems to be much more common in practice than the other three cases.

Branch and Bound: Case 1.



The incumbent solution has value $z_i = 38$

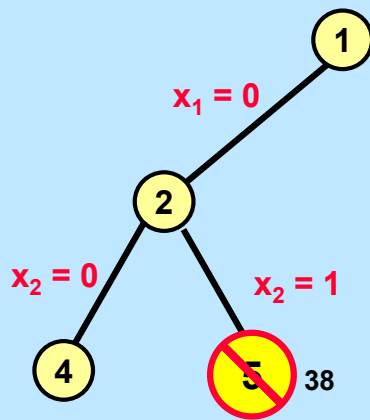
CASE 1. $z_{LP}(j) = -\infty$ That is, the LP for Subproblem j is infeasible.

Fathom Node j .

Do not search any of the subtree hanging from node j because none of these subproblems has a feasible solution.

In case 1, there is no feasible solution to the LP relaxation at node 5. This implies that no descendent of node 5 is a feasible solution to subproblem 5, and so we can fathom node 5. (Actually, we showed the stronger result that there is no feasible solution to the linear relaxation of subproblem 5.)

Branch and Bound: Case 2.



$$z_i = 38$$

Assume all feasible integer solutions have integer objective value.

CASE 2. $-\infty < \lfloor z_{LP}(j) \rfloor \leq z_i$
 Then $z_{IP}(j) \leq \lfloor z_{LP}(j) \rfloor \leq z_i$
 Fathom node j

No subproblem in the subtree hanging from node 5 has a solution that is better than the incumbent.

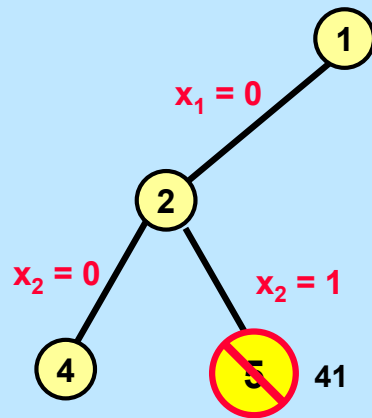
$$\text{Bound}(5) = 38$$

e.g. Suppose $z_{LP}(5) = 38.7$

28

In case 2, we found that that $z_{LP}(5) = 38.8$. But then the max profit feasible solution that is a descendent of node 5 has objective value at most 38 since all solutions have integer profits. Since every solution that is a descendent of node 5 has objective at most 38, none can be better than the incumbent. Note that it is possible that there are alternative solutions with objective value 38. We are assuming that we will be satisfied with one optimal solution. If we wanted all optimal solutions, we could not fathom node 5 at this time.

Branch and Bound: Case 3.



$z_i = 38$

CASE 3. $z_{LP}(j) > z_i$ and the optimal solution for LP(j) is feasible for the IP.

In this case, we first replace the incumbent by the integral for LP(j), which is feasible for the IP. No descendent of node 5 can be better. So we can fathom node j.

e.g. Suppose the opt solution for LP(5) was

$x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 1, x_6 = 0, z = 41$

$z_i = 41$

29

Case 3 is an odd case; it merits very close inspection.

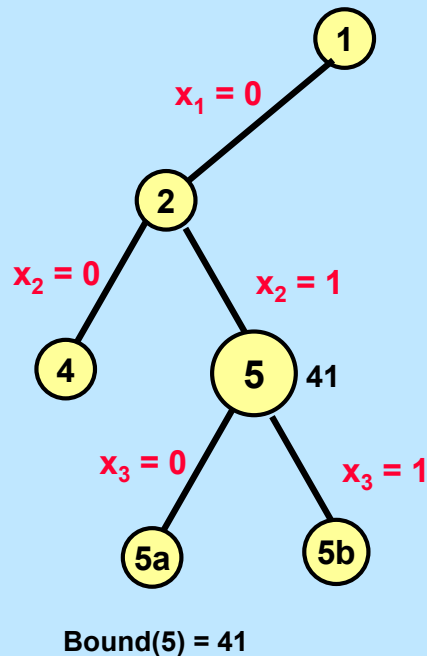
Suppose we solved the LP, and obtained $z_{LP}(5) = 41$. Suppose further that the solution we obtained to the LP relaxation was feasible for the original problem. Then we have a new solution that is better than our current incumbent. At this point, we can swap our incumbent for the solution with objective value 41.

Once we swap the incumbent for this better one, there is no need to search node 5 any further. We have already found the best solution that is a descendent of node 5, and we cannot do better than the objective value of 41. So we fathom node 5.

Note that it is not enough that the solution value of $z_{LP}(5)$ be integral. We need that the optimal solution for the linear programming relaxation was itself a feasible integer solution.

Note also that when we determine $z_{LP}(5)$, we solve a linear program. So, we know what solution we get and we know whether it is feasible for the original IP.

Branch and Bound: Case 4.



CASE 4. $\lfloor z_{LP}(j) \rfloor > z_i$ and the optimal solution for LP(j) is not feasible for the IP.

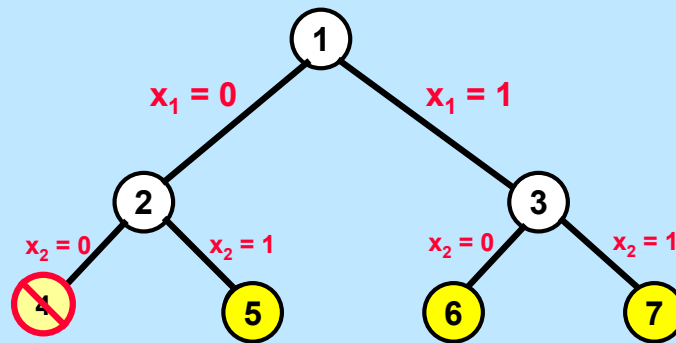
In this case, we cannot fathom node 5. Instead we add its two "children" to our (growing) tree and continue the branch and bound algorithm.

e.g. Suppose $z_{LP}(5) = 41$, but the solution for the relaxation is not feasible for the IP.

30

Case 4 is the most common one. We solve the LP at node 5, and $z_{LP}(5) > \lceil z_i \rceil$. Moreover, the solution we get at node 5 is not feasible for the original IP because it has at least one fractional component. So, we cannot fathom node 5. In this case, we create two children of node 5. Then we look at some other node and find an upper bound for it.

Active nodes



A node is called **active** if it has no children and it has not yet been fathomed. The active nodes are 5, 6, 7.

Initially, the only active node is node 1. The algorithm ends when there are no active nodes.

31

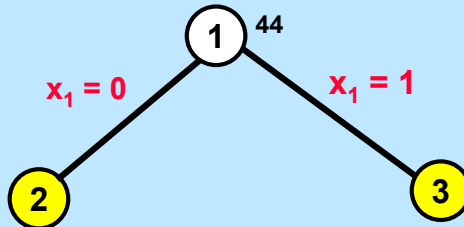
We have basically carried out all of the subroutines that we need for Branch and Bound. But you may notice that we create new nodes of the tree faster than we can scan them. So, when we create new nodes of the tree, we call these nodes active. And we maintain a list of active nodes.

If we fathom a node or if we give a node two children, we label the node as inactive, and we remove it from the list of active nodes.

Initially, the only active node is node 1. The algorithm will end when there are no active nodes.

Branch and Bound for 0-1 Integer Programs

$$z_i = 38$$



LP(1)

$$\begin{aligned} \text{maximize} \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{subject to} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & 0 \leq x_j \leq 1 \text{ for } j = 1 \text{ to } 6. \end{aligned}$$

$$\begin{aligned} z_{LP}(1) &= 44 \frac{3}{7}. \\ \text{Bound}(1) &= 44. \end{aligned}$$

This is Case 4. We add the two children of node 1.

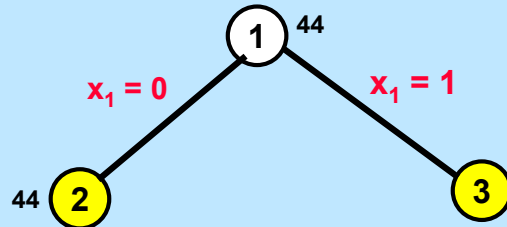
32

So, we now carry out Branch and Bound.

We find a bound for node 1 by solving the LP relaxation of Subproblem 1. We call this LP(1).

Subproblem 2

$$z_i = 38$$



LP(2)

$$\begin{aligned} \text{maximize} \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{subject to} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & x_1 = 0, \quad 0 \leq x_j \leq 1 \text{ for } j = 2 \text{ to } 6. \end{aligned}$$

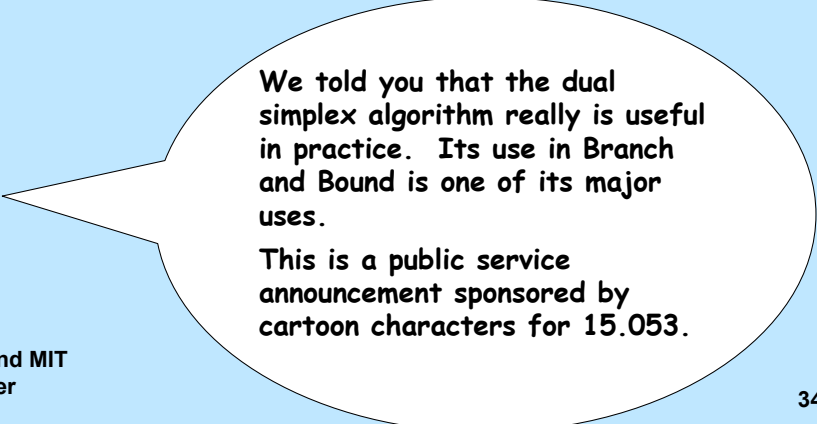
$$z_{LP}(2) = 44. \text{ Opt solution: } x_1 = 0, x_2 = 1, x_3 = 1/4, x_4 = x_5 = 0, x_6 = 1$$

Bound(2) = 44. This is case 4. We add the two children for node 2.

We then solve LP(2) and find $z_{LP}(2)$. This gives us a bound of 44 for node 2. Recall that this means that all solutions that are descendants of node 2 have an objective value of at most 44.

The Dual Simplex Algorithm (a digression)

Subproblem 2 can be solved starting from the solution for Subproblem 1, using the dual simplex algorithm. All subproblems can be solved much faster using the dual simplex algorithm.

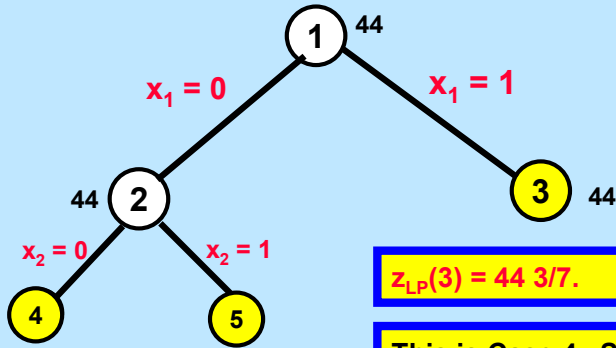


We told you that the dual simplex algorithm really is useful in practice. Its use in Branch and Bound is one of its major uses.

This is a public service announcement sponsored by cartoon characters for 15.053.

Subproblem 3.

$$z_i = 38$$



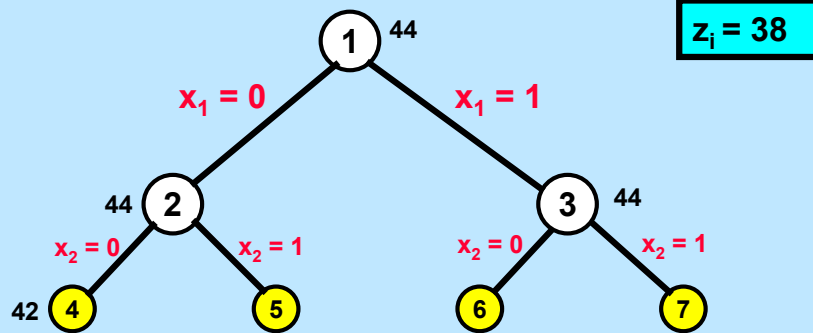
$$z_{LP}(3) = 44 \frac{3}{7}. \quad \text{Bound}(3) = 44.$$

This is Case 4. So we add the two children of node 3.

Active nodes are colored yellow. Other nodes are white. Fathomed nodes have a “no nodes permitted” label.

We now solve LP(3), obtaining $z_{LP}(3) = 44 \frac{3}{7}$. So, $\text{Bound}(3) = 44$.

Node 4.



$z_{LP}(4) = 42$. The solution for LP(4) is

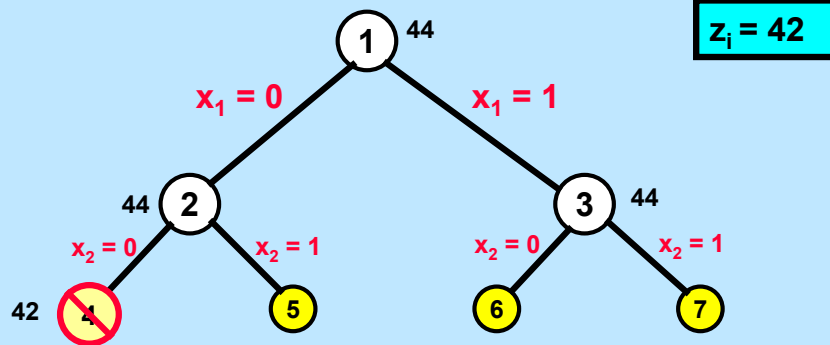
$$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 1.$$

It is a feasible solution for the IP that is better than the incumbent. This is Case 3.

36

We now solve LP(4). We get $z_{LP}(4) = 42$. But we get much more. When the linear programming solver solved LP(4), its optimal solution was the integer solution on this slide, which means we have a new integer solution that is better than our incumbent. This is case 3.

Result of Subproblem 4.



Replace the incumbent by

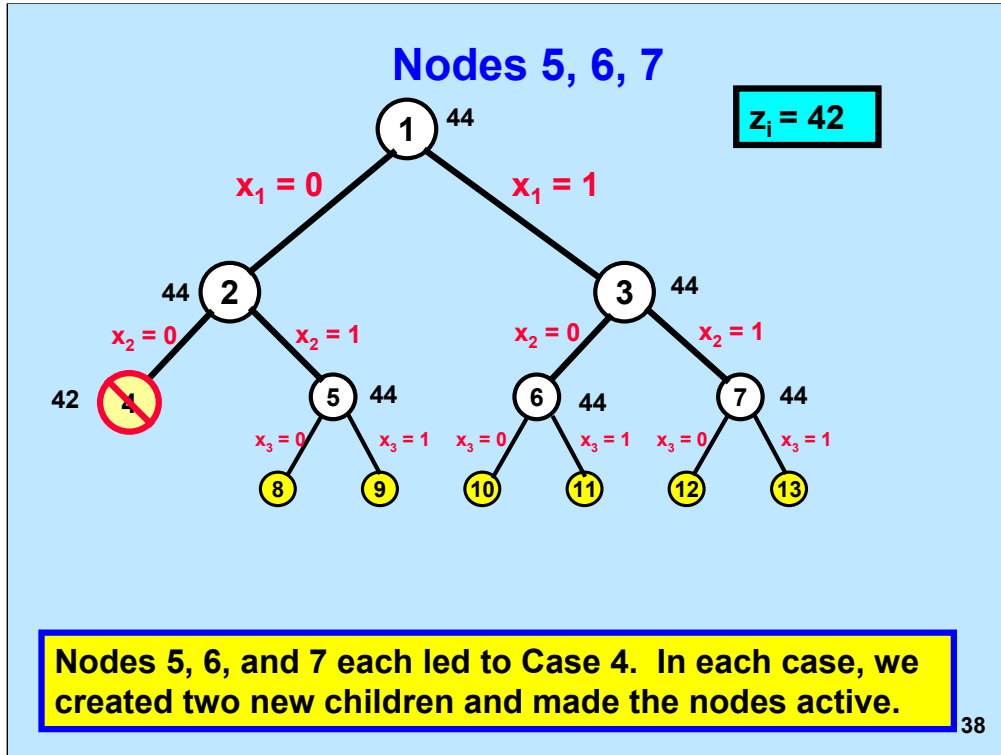
$$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 1.$$

$$z_i := 42.$$

Fathom node 4.

37

Since we have a better incumbent, we replace the incumbent and record that $z_1 = 42$. Once we do that, we realize that we can fathom node 4. The best solution that is a descendent of node 4 has objective value 42, and so we cannot do better than the current incumbent.



Nodes 5, 6, and 7 were disappointments. We obtained bounds of 44, 44, and 44. None of these bounds were good enough to fathom the node, and no new feasible solutions were found for the integer program.

Mental Break

- **Sherlock**
- **Hunters**

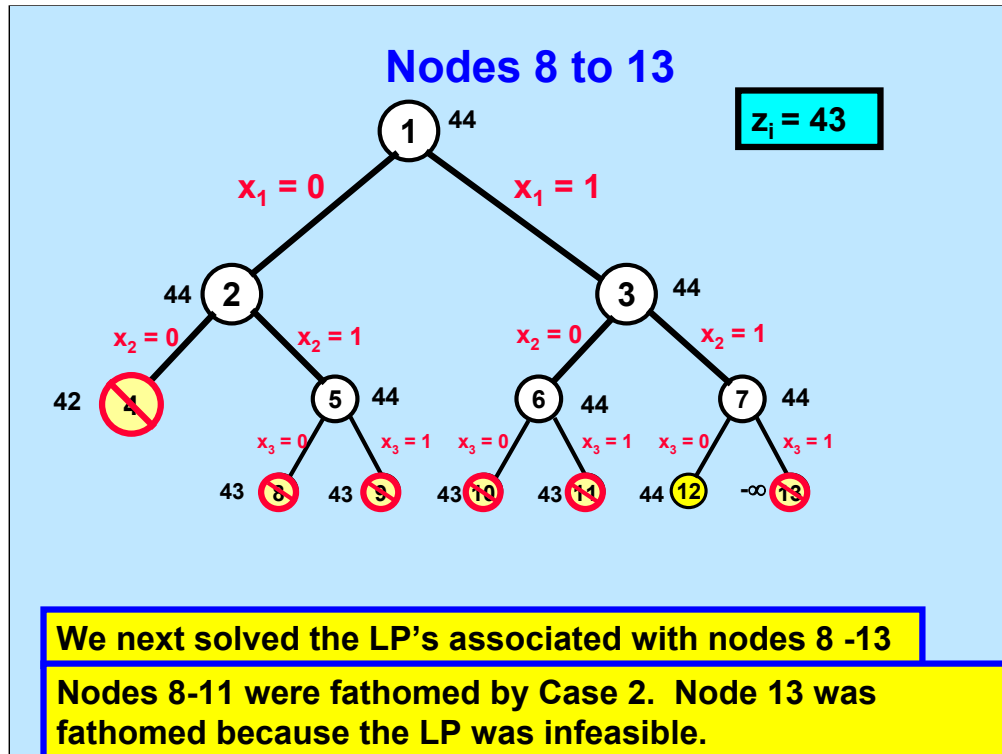
Getting to the end a little quicker

- This algorithm if continued in its current way would explore almost all of the nodes.
- So, we're going to get to the end quicker by supposing that we became aware of the following solution.

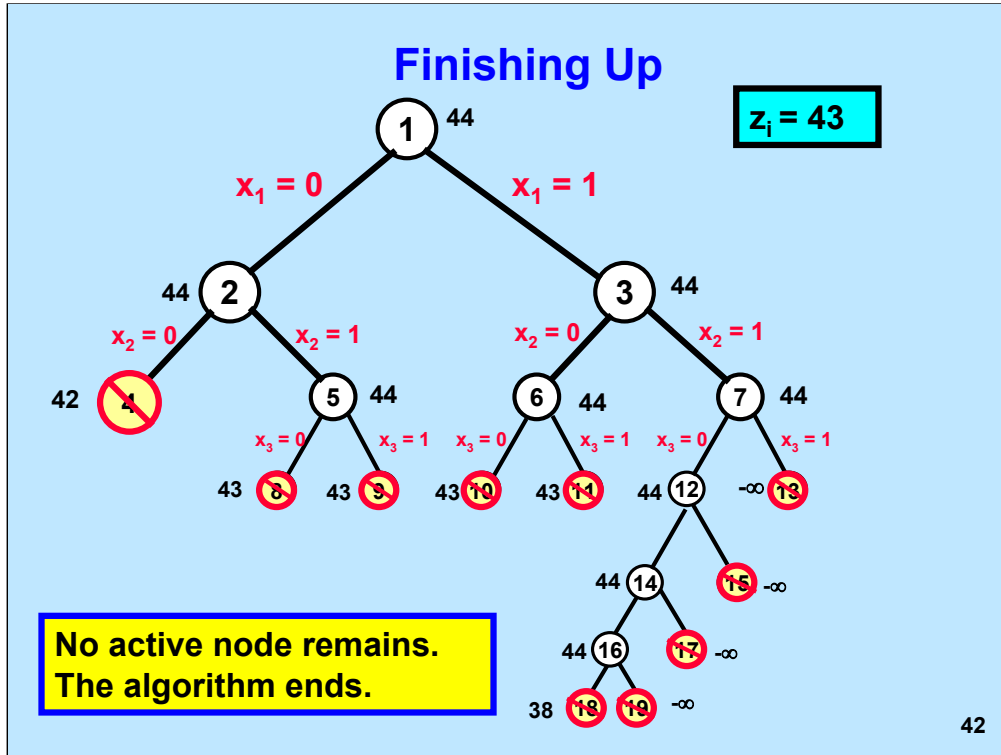
iPod	Gift certificate	15.053	This solution has 43 utils.
5 points 16 utils	3 points 8 utils	6 points 19 utils	

40

I will admit that branch and bound can be a bit wearisome to watch. So, I'll speed it up but making an assumption. The solution on this slide is actually the optimum solution. I'm going to assume that it was revealed to us so that we can replace the incumbent by this better solution. This will speed up fathoming.



The incumbent really did speed up fathoming. We obtained bounds of 43 for nodes 8, 9, 10, and 11. Because $z_1 = 43$, we know that no solution that is a descendent of one of these nodes can be better than the incumbent. And so we can fathom nodes 8 to 11.



At that point, only node 12 was active. We ended up having to solve a lot more LPs in order to get down to no active nodes. In particular, we had to solve LP(12), LP(14), LP(15), LP(16), LP(17), LP(18), and LP(19).

Lessons Learned

- **Branch and Bound can speed up the search**
 - Only 19 nodes (linear programs) were evaluated
 - Other nodes were fathomed. (It would have taken longer if we had not assumed we had the incumbent with value 43.)
- **Each linear program can be solved quicker than usual because it can use the dual simplex algorithm.**
- **The speed of branch and bound depends on how deep we need to go in the tree before we fathom nodes. This is very hard to predict.**
- **The improvements over enumeration are critical when we have 30 or more variables.**

Branch and Bound Algorithm

- **INITIALIZE** Active = {1} -- node 1 is the original problem
Incumbent: = \emptyset (or some heuristic finds an incumbent)
- **SELECT:**
 - If Active = \emptyset , then the Incumbent is optimal if it exists, and the problem is infeasible if no incumbent exists;
 - else, let j be a node from Active. Remove j from Active.

CASE 1. $z_{LP}(j) = -\infty$. Then fathom node j .

CASE 2. $-\infty < \lfloor z_{LP}(j) \rfloor \leq z_i$. Then fathom node j .

CASE 3. $z_{LP}(j) > z_i$ and the optimal solution for LP(j) is feasible for the IP. Then fathom node j , and replace the incumbent with this new solution.

CASE 4. $\lfloor z_{LP}(j) \rfloor > z_i$ and the optimal solution for LP(j) is not feasible for the IP. Then create two children for node j .

Remarks

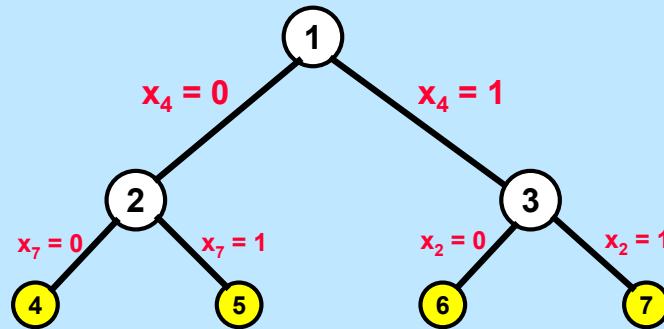
- **We could have selected any active node.**
- **In practice, it is not good to select the node with lowest index. It leads to too many active nodes at a time, and too much storage**
- **The rule in branching is this: any feasible solution for a subproblem must be feasible for one of its children. (We do not want to lose any feasible solutions when we branch).**

45

In our algorithm, we always selected the active node with the lowest index. But in reality, any active node can be selected next.

Actually, selecting the active node with the lowest index is not widely used as it sometimes leads to the algorithm failing because of excessive memory requirements.

Other Branching Rules



We don't need to branch on any specific variables in any order. Each branch should divide the "population of solutions" into two parts

Commercial algorithms use good branching rules that will lead to faster run times.

46

In writing the enumeration tree, it looked like we had to branch on x_1 first, and then x_2 and then x_3 , etc. We could have branched on the variables in any order.

In fact, we don't even need to have "symmetry." We can branch on variable x_7 at node 2 and branch on variable x_2 at node 3. The only aspect that must be enforced is every feasible solution with $x_4 = 1$ must be a descendent of node 3, and therefore must be a descendent of node 6 or node 7.

B & B for Pure Integer Programs

maximize $8x_1 + 5x_2$

subject to $x_1 + x_2 \leq 6$

$9x_1 + 5x_2 \leq 45$

$x_1, x_2 \geq 0, x_1, x_2$ integer

47

This is an example with general integer variables. We need to give a little more thought to branching rules and bounding to make this work.

The first node of the B & B tree

41 (1)

There is no initial incumbent.

$$z_i = -\infty$$

maximize $8x_1 + 5x_2$

subject to $x_1 + x_2 \leq 6$

$9x_1 + 5x_2 \leq 45$

$x_1, x_2 \geq 0,$

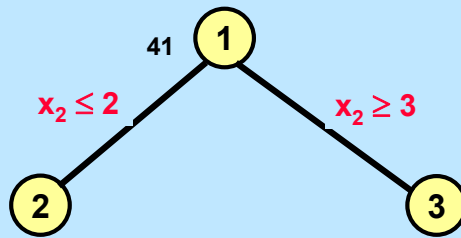
LP(1)

Optimal solution to LP(1) is
 $x_1 = 3.75, x_2 = 2.25, z_{LP(1)} = 41.25.$

This is Case 4 of B&B. Create two children.

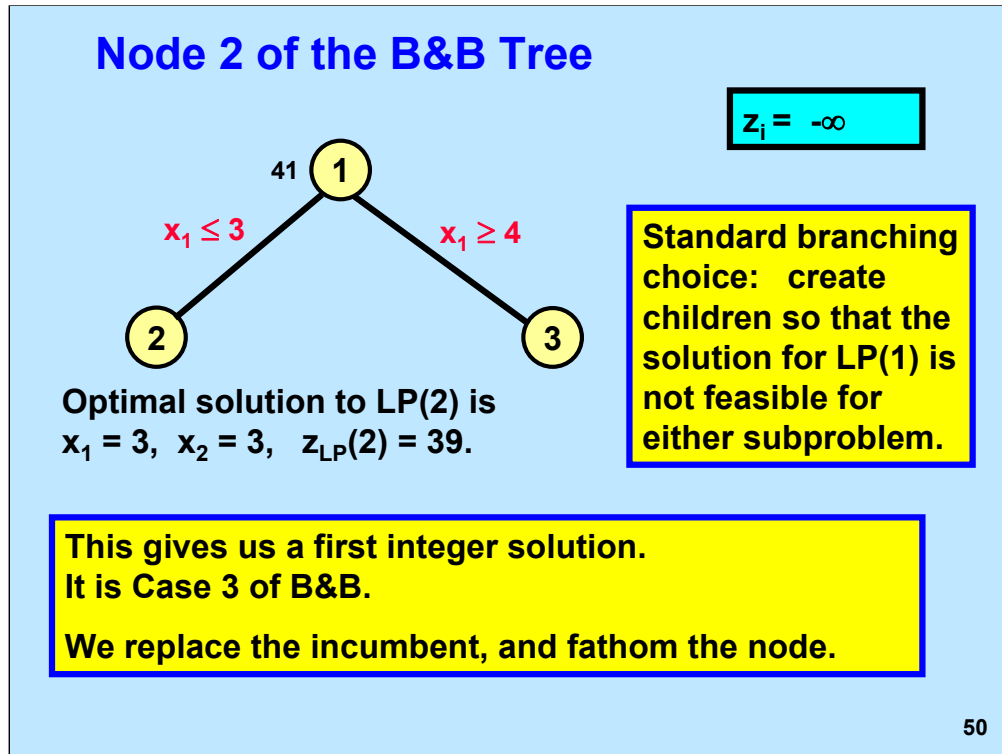
48

LP(1) consists of solving the LP relaxation of the original problem.



Note. can create subproblems any way that we want, so long as eventually every solution would be enumerated if we did not fathom.

That is, no feasible solution to the integer program ever gets eliminated by branching. It will be feasible for one of the branches.



We want to ensure that every feasible solution to the original problem is descendent of node 1 and thus a descendent of node 2 or node 3.

Notice that the solution to LP(1) had $x_1 = 3.75$, which was fractional.

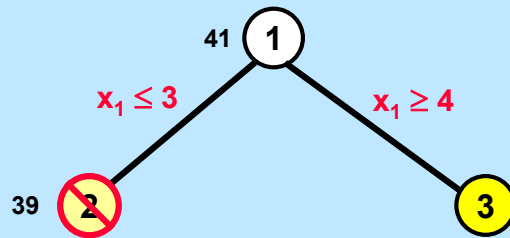
We can say that every feasible solution has $x_1 \leq 3$ or $x_1 \geq 4$. This is obvious. But when we perform branching in this manner, we get two good things.

1. Every feasible solution to the original problem will be a solution to subproblem 2 or subproblem 3;
2. We are guaranteed to eliminate the LP solution from LP(1) as this solution is not feasible for LP(2) or LP(3).

We then solved LP(3) and the optimum solution was integer valued. It was $x_1 = 3$ and $x_2 = 3$. This gives us the first incumbent. In addition, any feasible solution to the original problem that is a descendent of node 2 (that is, any feasible solution to the original problem with $x_1 \leq 3$) will have an objective value of at most 39, and so cannot be better than $x_1 = 3$ and $x_2 = 3$. So we can fathom node 2.

Node 3 of the B & B tree

$$z_i = 39$$

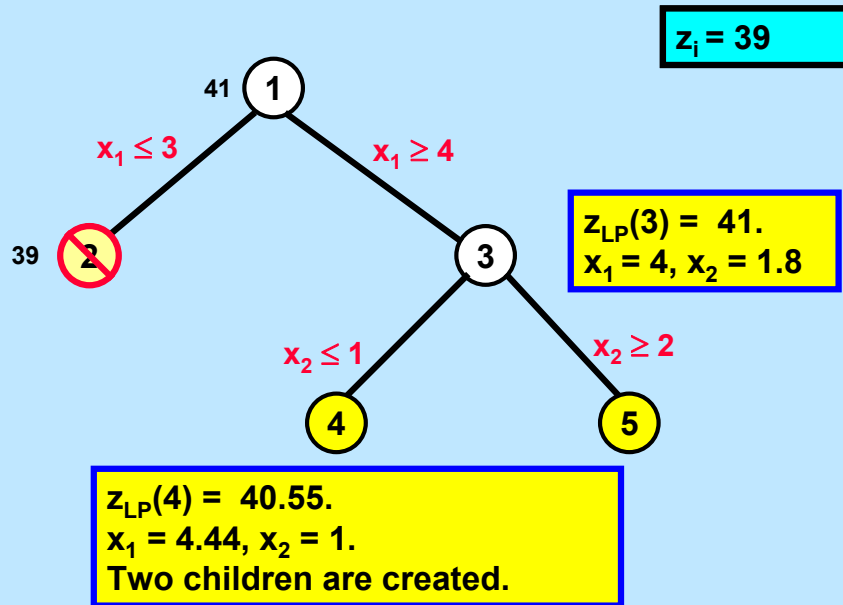


$$z_{LP}(3) = 41. \quad x_1 = 4, \quad x_2 = 1.8$$

This is case 4 of B&B. We create two children.

We now solve LP(3), obtaining a fractional solution with $z_{LP}(3) = 41$. So $\text{Bound}(3) = 41$, and we create two children. Since its LP solution had $x_2 = 1.8$, our usual branching rule is to add the constraint $x_2 \leq 1$ to the left branch, and to add the constraint $x_2 \geq 2$ to the right branch. Any feasible solution to subproblem 3 must satisfy one of these two constraints. Recall that subproblem 3 is the original problem plus the constraint $x_1 \geq 4$.

Node 4 of the B & B tree

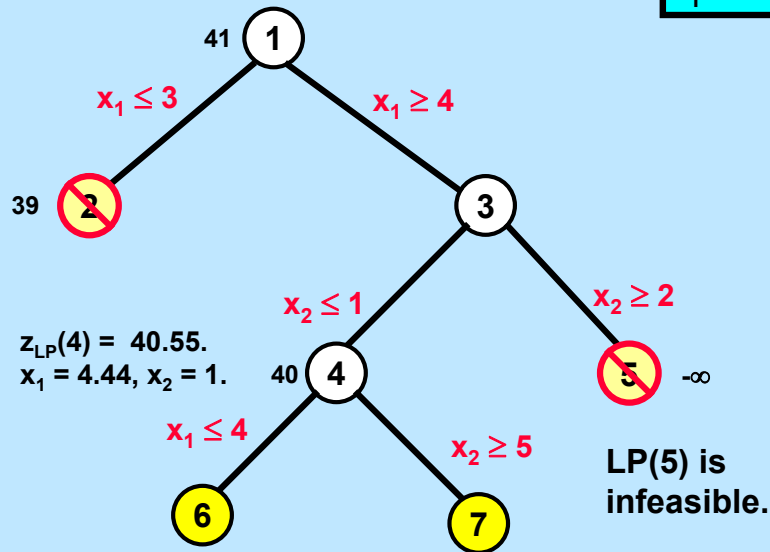


52

We continue by solving LP(4) finding a bound of 40 and creating two new children.

Node 5 of the B & B tree

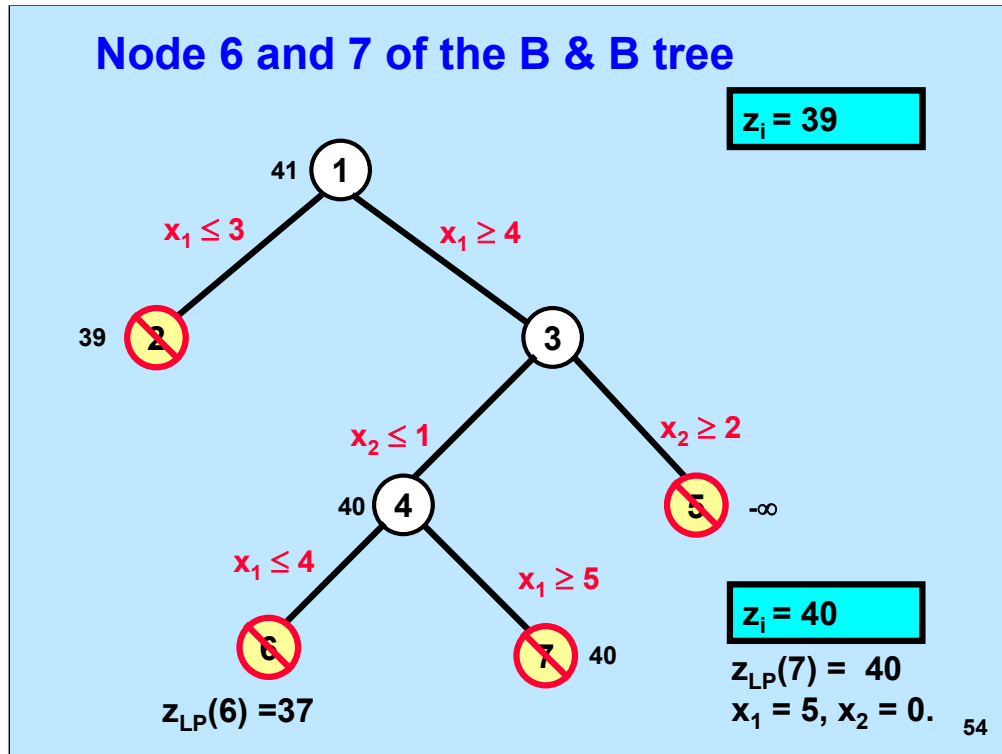
$$z_i = 39$$



53

The solution to LP(4) had $x_1 = 4.44$. So we branch on node 4 by requiring that $x_1 \leq 4$ or $x_1 \geq 5$.

We then solve LP(5), which was infeasible. So we fathom node 5.

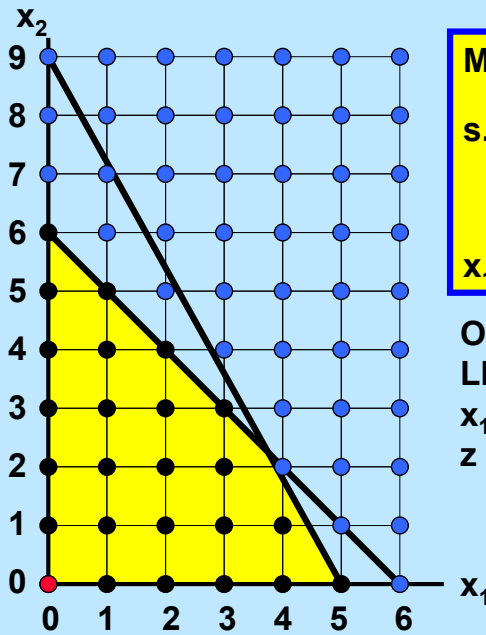


Node 6 could be fathomed because $z_{LP}(6) = 37$.

LP(7) had an integer optimal solution with $x_1 = 5$ and $x_2 = 0$. So, we obtained a new incumbent, with $zI = 40$. This also permits us to fathom node 7 since no descendent of node 7 will have objective value better than 40.

At this point, there are no active nodes, and so the Branch and Bound algorithm ends.

The Graphical Representation



$$\text{Max } 8x_1 + 5x_2$$

$$\text{s.t } x_1 + x_2 \leq 6$$

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

$$x_1, x_2 \geq 0, \quad x_1, x_2 \text{ integer}$$

Optimal solution to the
LP relaxation is

$$x_1 = 3.75, \quad x_2 = 2.25$$

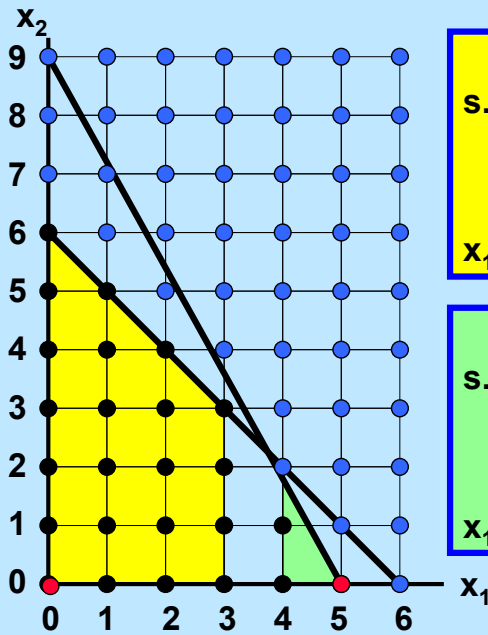
$$z = 41.25$$

55

You can follow the solution geometrically.

Here we solve LP(1) and obtain the solution $x_1 = 3.75$ and $x_2 = 2.25$.

Subproblems 1 and 2



$$\begin{aligned} \text{Max } & 8x_1 + 5x_2 \\ \text{s.t } & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1 \leq 3 \\ & x_1, x_2 \geq 0, \quad x_1, x_2 \text{ integer} \end{aligned}$$

$$\begin{aligned} \text{Max } & 8x_1 + 5x_2 \\ \text{s.t } & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1 \geq 4 \\ & x_1, x_2 \geq 0, \quad x_1, x_2 \text{ integer} \end{aligned}$$

56

The yellow feasible region is for LP(2).

The green feasible region is for LP(3).

Branch and Bound in General

Notation:

- ***Incumbent***: the best solution on hand
 - Its value is z_i
- $z_{IP}(j)$ the optimum value for subproblem j
- $z_{LP}(j)$: value of the LP relaxation of node j
- **Bound(j)**. In cases in which we are maximizing an d in which integer solutions have integer objective values, we let $\text{Bound}(j) = \lfloor z_{LP}(j) \rfloor$
- ***Children of a node***: the two problems created for a node that such that every solution that is a descendent of the original node is also a descendent of one of its children.
- **Active**: the collection of ***active*** (not fathomed) subproblems.

On Branch and Bound

- **Branch and Bound is a standard way of solving integer programs.**
- **Lots of art and engineering can go into making it work effectively**
- **Next: a couple of issues that arise.**

Different Branching and Bounding Rules are Possible

- **The more accurate the bound, the quicker the fathoming.**
 - **The better the bounding technique, the more accurate the bound and the quicker the fathoming.**
 - **The quicker that the important decisions are resolved, the quicker the fathoming**
- **The better the incumbent, the quicker the fathoming.**

59

It turns out that accurate bounding is incredibly important, and will be one of the focal points of the next lecture.

Summary

- **Branch and Bound is the standard way of solving IPs to optimality.**
- **There is art to making it work well in practice.**
- **Much of the art is built into state-of-the-art solvers such as CPLEX from ILOG.**

People often use terms to describe slightly different algorithms, such as “branch and price”. But the essence of all integer programming techniques is “branch and bound.”

The Solution for the LP Relaxation for node 1

maximize $16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6$
subject to $5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14$
 $0 \leq x_j \leq 1$ for $j = 1$ to 6

x_1	x_2	x_3	x_4	x_5	x_6
1	0.429	0	0	0	1
Objective value			44.43		

LP solution is obtained
from Excel Solver.

You can ignore further slides from here. They are Excel output for the solutions to the LP relaxations.

The Solution for the LP Relaxation for node 2

maximize $16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6$
subject to $5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14$
 $0 \leq x_j \leq 1$ for $j = 1$ to 6 , and $x_1 = 0$

x_1	x_2	x_3	x_4	x_5	x_6
0	1	0.25	0	0	1
Objective value:			44		

The Solution for the LP Relaxation for node 3

maximize $16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6$
subject to $5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14$
 $0 \leq x_j \leq 1$ for $j = 1$ to 6 , and $x_1 = 1$

x_1	x_2	x_3	x_4	x_5	x_6
1	0.429	0	0	0	1
Objective value			44.43		

The Solution for the LP Relaxation for node 4

maximize $16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6$
subject to $5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14$
 $0 \leq x_j \leq 1$ for $j = 1$ to 6 , and $x_1 = x_2 = 0$.

x_1	x_2	x_3	x_4	x_5	x_6
0	0	1	0	1	1
Objective value:			42		

The Solution for the LP Relaxation for node 5

x_1	x_2	x_3	x_4	x_5	x_6
0	1	0.25	0	0	1
Objective value:			44		

The Solution for the LP Relaxation for node 6

x_1	x_2	x_3	x_4	x_5	x_6
1	0	0.75	0	0	1
Objective value:			44		

The Solution for the LP Relaxation for node 7

x_1	x_2	x_3	x_4	x_5	x_6
1	1	0	0	0	0.333
Objective value:			44.33		

The Solution for the LP Relaxation for node 8

x_1	x_2	x_3	x_4	x_5	x_6
0	1	0	0	0.25	1
Objective value:			43.75		

The Solution for the LP Relaxation for node 9

x_1	x_2	x_3	x_4	x_5	x_6
0	1	1	0	0	0.5
Objective value:			43.5		

The Solution for the LP Relaxation for node 10

x_1	x_2	x_3	x_4	x_5	x_6
1	0	0	0	0.75	1
Objective value:			43.25		

**The Solution for the LP Relaxation
for node 11**

x₁	x₂	x₃	x₄	x₅	x₆
1	0	1	0	0	0.833
Objective value:			43.83		

The Solution for the LP Relaxation for node 12

x_1	x_2	x_3	x_4	x_5	x_6
1	1	0	0	0	0.333
Objective value:			44.33		

The Solution for the LP Relaxation for node 13

**This problem has no feasible
solution.**