CHEMICAL ENGINEERING 4G03

Module 07

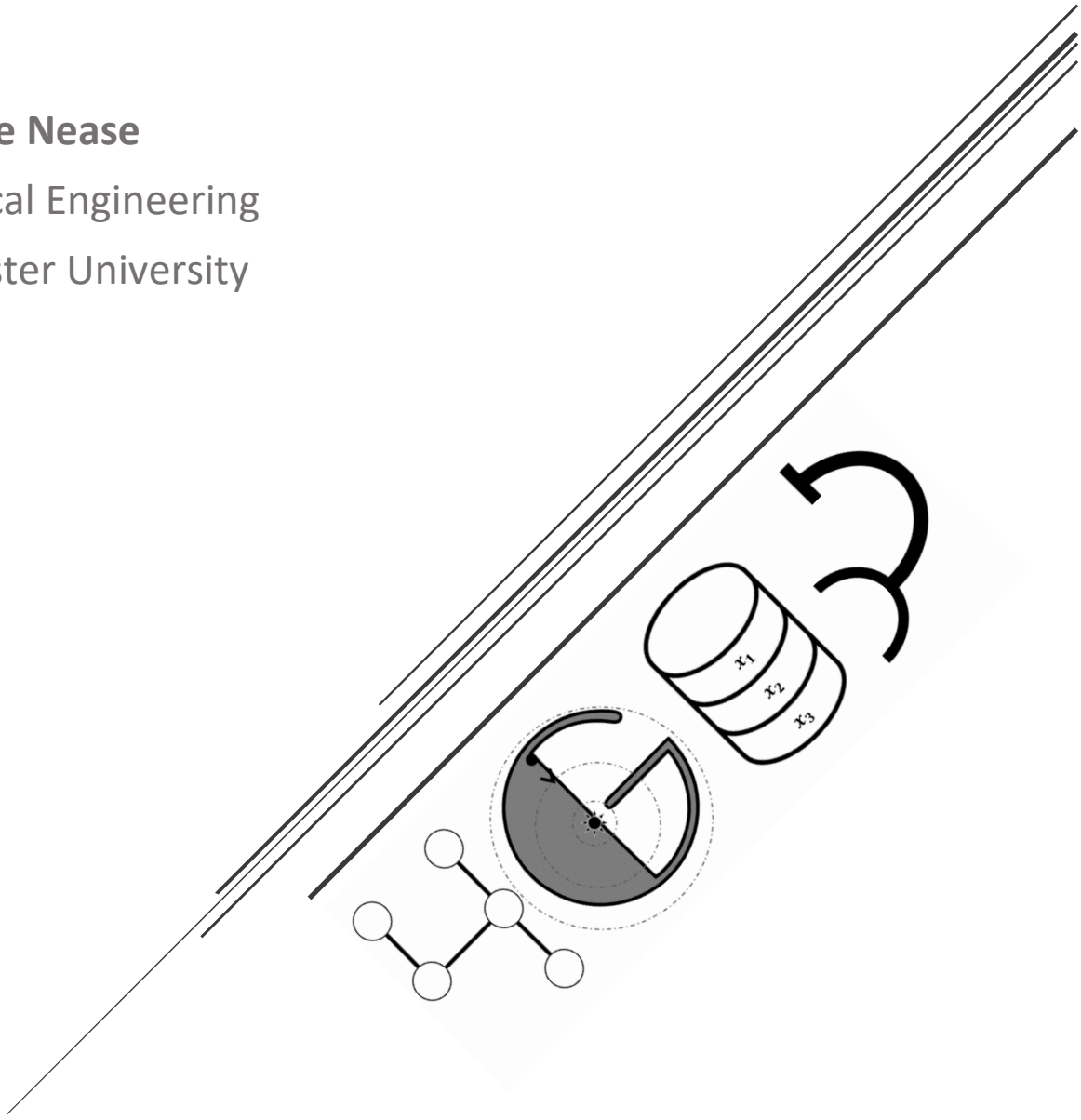
Mixed-Integer Linear Programs (II)

Branch-and-Bound

Dr. Jake Nease

Chemical Engineering

McMaster University



Updated March 14, 2023

Outline of Module

Suggested Readings

Rardin (1st edition): Chapter 12.4

Rardin (2nd edition): Chapter 12.4

Solving Discrete Optimization Problems

Let's kick things off by revisiting our general integer program formulation and discussing how we intend to solve it.

Mixed Integer Linear Program (MILP) Formulation

An optimization model is known as a **Mixed Integer Linear Program** if it obeys:

$$\min_{\boldsymbol{x},\boldsymbol{y}} \phi = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d}^T\boldsymbol{y} \quad \leftarrow \text{Objective Function}$$

$$s.t. \quad \leftarrow \text{"Subject to"}$$

$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{E}\boldsymbol{y} \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b \quad \leftarrow \text{CONTINUOUS and DISCRETE Constraints}$$

$$x_{lb} \leq x \leq x_{ub}, \quad y \in \{0,1\}^{n_y} \quad \leftarrow \text{Variable Bounds}$$

Generally, in this course we target the **integer variables** y to be **binary**. That is, the elements in y may only take on the values 0 or 1.

When it comes to our integer programs, we seek **rigorous** solution methods, but we suffer from a distinct lack of some of the conveniences afforded to us by using continuous variables only:

- The concepts of the **neighbourhood** and **improving search directions** no longer apply. This is because we are dealing with a discrete (and thus nonconvex) set, and thus *no improving directions are also feasible for any discrete point*.
- However, we **MAY** use the improving search methodology for the **continuous** variables, but **only if** we fix the discrete variables to fixed values (thus making them parameters).

Sufficed to say, we need a method that will combine the concepts of a rigorous search for continuous variables with the added complexity of dealing with the discrete variables separately.

Total Enumeration

Now, you might think to yourself: "wait, shouldn't discrete optimization problems be EASIER to solve than continuous? I mean, there are only a FINITE number of options the discrete variables can be!" **You are right**, of course, and you are thinking of the concept of **total enumeration**.

Total Enumeration

The optimization method that solves for the optimum based on the *continuous variables* for **ALL valid combinations of the discrete variables**, and keeps whichever solution is the **best**.

If there are no continuous variables, this method equates to what you might call "brute force" optimization. Basically, we just try all possible combinations of variables. IF there are continuous variables, we must solve the **LP or NLP** resulting from fixing the integer variables to a set of known values (in other words, we have to solve an optimization program for each "guess"). Let's try it out.

Class Workshop – Total Enumeration

Consider the following **integer program** and SOLVE IT using total enumeration:

$$\begin{aligned} \max_{y} \phi &= 7y_1 + 4y_2 + 23y_3 \\ \text{s.t.} & \\ y_1 + y_2 &\leq 2 \\ y_2 + y_3 &\leq 1 \\ y_i &\in \{0,1\} \qquad \forall\, i \end{aligned}$$

Workshop Solution – Total Enumeration

CASE	VARIABLES	OBJECTIVE	FEASIBLE?
1	$y = (0,0,0)$		
2	$y = (1,0,0)$		
3	$y = (1,1,0)$		
4	$y = (1,1,1)$		
5	$y = (0,1,0)$		
6	$y = (0,1,1)$		
7	$y = (0,0,1)$		
8	$y = (1,0,1)$		

Optimum:

WELL! That wasn't so bad, right? We only had to identify all of the combinations of variables and crunch the numbers. However, don't fall into this trap. As it turns out, for n binary variables, there are 2^n possible combinations.

- If $n = 10$, there are 1024 possible combinations.
- If $n = 20$, there are over 1 million.
- If $n = 30$, there are over **1 billion**.

If we are solving a relaxed LP that takes **1 second** to solve with the simplex search for every fixed combination of binaries, it would take 17 minutes to solve a problem with 10 binaries. 12 days to solve something with 20. Nearly **34 years** to solve something with 30. The total enumeration method is said to grow **exponentially** with the number of discrete variables. Now, think to yourself… How many binary variables are in your projects? How many in my typical assignment question? It should be instantly clear that total enumeration is not the best bet for general problems. In summary:

- IF there are only a *few* discrete variables, total enumeration might be the way to go.
- Any models with more than just a few (>10?) requires a more clever method that can exploit a comparative bounding method and **integer problem relaxations**.

Optimization Program Relaxations

OK, let's try a new idea where instead of trying all of the options, we develop a **sequence** of related, simpler (than the original) sub-problems, the solutions to which will converge (finitely) to the solution to the regular problem as the relaxations are removed.

There are lots of applications and methods to using relaxations that are not necessarily constrained to MILPs! In fact, relaxations are used all the time. We have already looked at one: **formulating scheduling deadlines as penalized objectives rather than hard-nosed constraints**. Some others include:

- Relaxing the feasible region to LPs and NLPs to form a best-case scenario (LP case) OR enhance the likelihood of finding a solution (NLP case).
- "Elevating" constraints in constrained NLPs to be penalized in the objective function (as Lagrange Multipliers) to turn the constrained NLP into a modified "unconstrained" NLP.
- Penalizing constraint violations for black-box optimizers (such as Particle Swarms) so they conform to the constraints but allow for an unconstrained search space.

When it comes to relaxing integer linear programs in particular, we get some very convenient properties:

Basis for Integer Program Relaxation

We may *relax* an integer linear program by expanding the feasible set $\mathcal{S}$. This can be done in a variety of ways, but the primary one is to consider subsets of the integer variables as *continuous*. THEN:

- The relaxed sub-problem(s) should be **easier to solve** than the original (why?).
- Each sub-problem provides a BOUND on the solution.
  - Relaxed minimization problems provide a LOWER bound on the original.
  - Relaxed maximization problems provide an UPPER bound on the original.
- Ideally, the number of sub-problems that must be solved to locate the original problem's optimum should be *much less* than total enumeration (but this is not guaranteed).

OK, so now that we know the "WHY" it is time to focus on the "HOW."

Constraint Relaxation

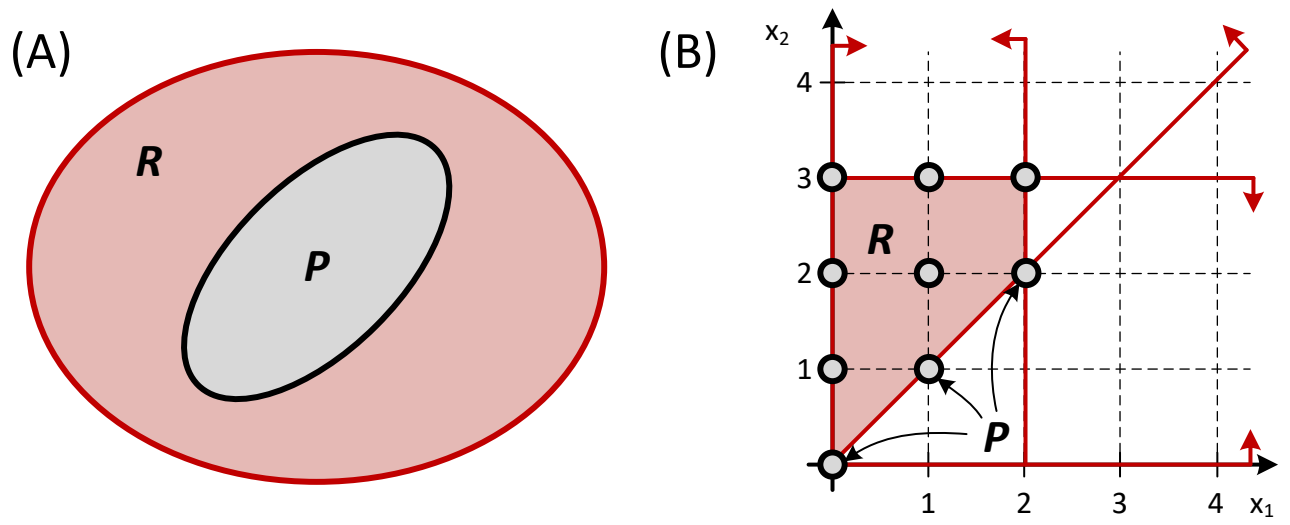
A **relaxed model $\mathcal{R}$** is said to be a *relaxation* of the **original (primal) model $\mathcal{P}$** if:

- EVERY feasible solution to $\mathcal{P}$ is also a feasible solution in $\mathcal{R}$. In other words, the feasible set for $\mathcal{P}$ ($\mathcal{S}_{(\mathcal{P})}$) is a subset of the feasible set of $\mathcal{R}$ ($\mathcal{S}_{(\mathcal{R})}$):

$$\mathcal{S}_{(\mathcal{P})} \subset \mathcal{S}_{(\mathcal{R})}$$

- $\mathcal{P}$ and $\mathcal{R}$ have the SAME objective function.

In plain language, the first bullet in the highlighted box above implies that the feasible region for my relaxed problem must *entirely contain* the feasible region for the original problem. This can be visualized using the figures below. In panel (A) you can see how a *continuous* relaxation of a set corresponding to $\mathcal{P}$ results in a larger region $\mathcal{R}$ that entirely contains the set for $\mathcal{P}$. In a scenario more related to IP relaxations, consider panel (B) in which we relax an integer program subject to constraints (dots) to be the entire continuous region contained by the constraints (red shading).



Methods of Relaxing Integer Programs

Consider the MILP model below, which we can relax in a variety of ways.

$$\begin{aligned} \min_{x,y} \phi &= 7x_1 + 2x_2 + 3y_1 - y_2 \\ s.t. & \\ x_1 + 10x_2 + y_1 - 2y_2 &\geq 100 \\ y_1 + y_2 &\leq 1 \\ y_j &\in \{0,1\} \quad \forall j \\ x_i &\geq 0 \quad \forall i \end{aligned}$$

Relaxation 1 – Drop the mutual exclusivity constraint. By removing the mutual exclusivity constraint, we increase the number of possible combinations of y_1 and y_2 that yield a feasible solution.

$$\begin{aligned} \min_{x,y} \phi &= 7x_1 + 2x_2 + 3y_1 - y_2 \\ s.t. & \\ x_1 + 10x_2 + y_1 - 2y_2 &\geq 100 \\ y_j &\in \{0,1\} \quad \forall j \\ x_i &\geq 0 \quad \forall i \end{aligned}$$

Relaxation 2 – Relax RHS of inequality constraint. By decreasing the RHS of the inequality constraint, we permit a greater number of combinations of x_1 and x_2 that may lead to a better objective function. Furthermore, relaxing this constraint *may permit* more combinations of the integer variables beyond those that are precluded due to the mutual exclusivity constraint.

$$\begin{aligned} \min_{x,y} \phi &= 7x_1 + 2x_2 + 3y_1 - y_2 \\ s.t. & \\ x_1 + 10x_2 + y_1 - 2y_2 &\geq \mathbf{50} \\ y_1 + y_2 &\leq 1 \\ y_j &\in \{0,1\} \quad \forall j \\ x_i &\geq 0 \quad \forall i \end{aligned}$$

Relaxation 3 – Remove integrality requirements for binaries. By allowing the integer variables to be continuous, we open up the (literally infinite) possible values on the range of 0-1. In our "fixed cost" integer scenario, for example, this is the case where we are permitting the optimization to apply a "partial" fixed cost to open up a partial capacity for its associated continuous variable x.

$$\begin{aligned} \min_{x,y} \phi \quad &= \quad 7x_1 + 2x_2 + 3y_1 - y_2 \\ s.t. & \\ x_1 + 10x_2 + y_1 - 2y_2 \quad &\geq \quad 100 \\ y_1 + y_2 \quad &\leq \quad 1 \\ y_j \quad &\in \quad (0,1) \text{ (continuous)} \qquad \forall j \\ x_i \quad &\geq \quad 0 \qquad \forall i \end{aligned}$$

Properties of LP Relaxations

Linear Program Relaxations

LP relaxations of a **MILP** are formed by treating all discrete variables as *continuous*, while retaining all other modeling constraints (corresponds to relaxation #3 above):

$$y_i \in \{0,1\} \xrightarrow{relax} 0 \leq y_i \leq 1$$

There are some very useful conveniences afforded to us by performing linear relaxations of discrete variables, namely:

- We can use our **existing LP solution methods (Simplex)** to solve the relaxed problems.
- The conclusions about **global optimality for LPs** carry over to the relaxed problems
- All LP relaxations of integer programs are GUARANTEED to contain the same optimum as the original MILP.

Furthermore, since we know for a fact that the MILP feasible region $S_{(\mathcal{P})}$ is a subset of the relaxed LP, we can make some very convenient conclusions about feasibility without actually testing *any* combinations of the integer variables.

Proving Infeasibility via Relaxation

IF an LP relaxation is shown to be *infeasible*, **so is the MILP model that it relaxes**.

NB – This does not prove anything with regards to *feasibility*. That is, if we prove the LP relaxation is feasible, we cannot conclude that the MILP it relaxes is feasible.

Make sense? OK, let's do a couple of examples!

Class Workshop – Showing Infeasibility of MILPs

Use LP relaxation to establish the infeasibility of the following integer programs.

Program 1

$$\begin{aligned} \min_{y} \phi &= 8y_1 + y_2 \\ \text{s.t.} & \\ y_1 - y_2 &\geq 2 \\ -y_1 + y_2 &\geq -1 \\ y_1, y_2 &\in \{0, 1, 2, \dots\} \end{aligned}$$

Program 2

$$\begin{aligned} \min_{y} \phi &= y_1 + 6y_2 \\ \text{s.t.} & \\ 2y_1 + y_2 &\leq 2 \\ 4y_1 + 2y_2 &\geq 6 \\ y_1, y_2 &\in \{0, 1, 2, \dots\} \end{aligned}$$

Workshop Solution – Showing Infeasibility of MILPs

Another very useful property of LP relaxations is that they will form an upper or lower bound on the true MILP optimum for the maximization and minimization cases, respectfully.

Solution Bounds from LP Relaxations

- The optimal value of an LP relaxation for a **maximization** will ALWAYS yield an **upper bound** on the optimal value of the original MILP model.
- The optimal value of an LP relaxation for a **minimization** will ALWAYS yield a **lower bound** on the optimal value of the original MILP model.

For those of you that are interested, this concept of forming upper and lower bounds comes in handy for us a little later. Moreover, NLP and MINLP solvers typically try to minimize what is known as a "Duality Gap," which is the distance between a KNOWN "optimistic" solution and the corresponding value of the original problem. For now, let's just apply the concept of relaxation bounds.

Class Workshop – Forming Bounds on an MILP

Consider the integer program below.

$$\begin{aligned} \max_{y} \phi &= y_1 + y_2 + y_3 \\ s.t. & \\ y_1 + y_2 &\leq 1 \\ y_1 + y_3 &\leq 1 \\ y_2 + y_3 &\leq 1 \\ y_1, y_2, y_3 &\in \{0, 1\} \end{aligned}$$

1. Formulate and Solve (by inspection) the LP relaxation of the above ILP model.
2. Solve the ILP model (again by inspection) and compare the optimal values.

Workshop Solution – Forming Bounds on an MILP

The FINAL useful property of LP relaxations is as follows:

Optimal Solutions via Relaxations

If the relaxed LP of an MILP model yields an optimal solution that is *feasible* in the original MILP, the solution to the relaxed LP and original MILP are **equivalent (location and optimal objective)**.

Class Workshop – Integer Solutions from LP Relaxations

Show that the LP relaxation yields the optimum to the following MILP with $y \in \{0,1\}$:

$$\begin{aligned} \max_{y} \phi &= 30y_1 + 5y_2 + y_3 \\ s.t. & \\ y_1 + y_2 + y_3 &\leq 1 \end{aligned}$$

Workshop Solution – Integer Solutions from LP Relaxations

We can now finally introduce the Branch and Bound Search!

Branch and Bound Search

Branch and Bound algorithms combine a hybrid of partial *enumeration* with *LP relaxations*. The primary idea is that I can solve a relaxed LP of the MILP, make some conclusions about that relaxed solution versus other (possibly integer) solutions I have encountered, and proceed in a fashion that will be guaranteed to solve them problem in *at most* the total number of iterations required for total enumeration (but in all likelihood much less than that). Basically:

- **Classes of solutions** conforming to certain combinations of variables are investigated to determine whether or not they can obtain candidate optimal solutions
- Each candidate solution is the product of at least one **relaxation**.
- Only **promising solutions** are investigated further (remember… If I know a relaxed problem is an optimistic bound and I have a better-known **integer** solution, do I bother constraining the relaxation?).

Partial Solutions

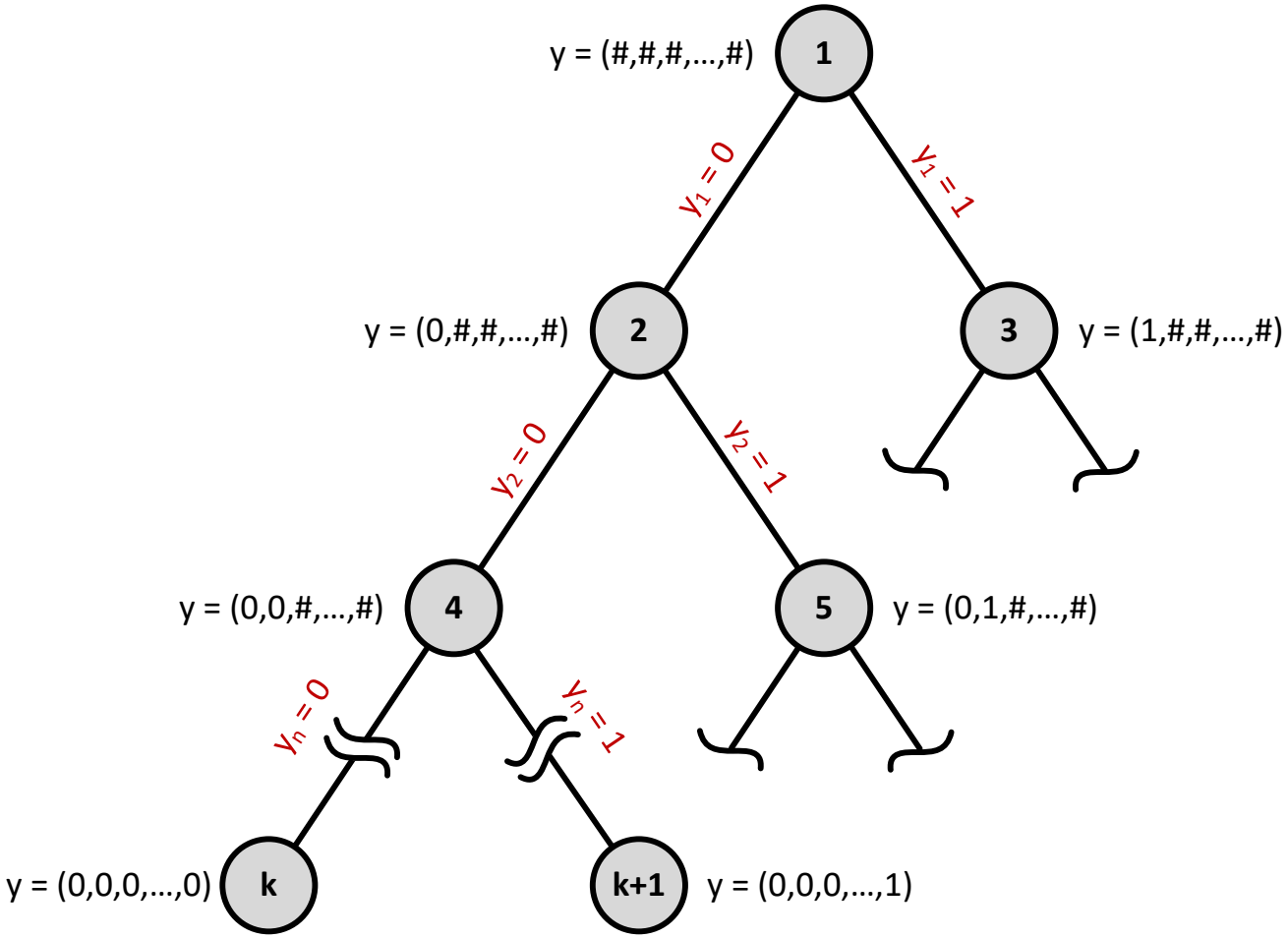
A **PARTIAL SOLUTION** has some discrete decision variables fixed (as in enumeration), while the remaining decision variables are left "free" as continuous variables (denoted here by a #).

A **COMPLETION** of a partial solution are the possible full solutions agreeing with the partial solution for the fixed variables.

For example, I might have the partial solution $y = (y_1, y_2, y_3, y_4) = (1, \#, 0, \#)$. The corresponding completions for this relaxation are $(1,0,0,0)$, $(1,1,0,0)$, $(1,1,0,1)$ and $(1,0,0,1)$.

When completing the Branch and Bound search using binary variables, it is typical to visualize it in the form of a **decision tree**. Each node of the decision tree represents a partial solution and has an objective

function value associated with it. Each node then splits into **two branches**, each representing **one** FREE variable from the partial solution being fixed to its permissible values of 0 or 1. For example, in the figure below, the first node (1) is the solution to the given optimization program with all variables free. Nodes (2) and (3) represent the solutions to the program when one binary variable (chosen to be y_1 in this case) is fixed to 0 (left branch) and 1 (right branch). **From that point on, the variable y_1 will always be fixed to 0 or 1**, depending on which side of the tree we are on. Subsequent nodes follow the same procedure: one variable is fixed to 0 or 1 while the remaining variables are left free.



In summary, you can think of the B&B tree as follows:

- **NODES** of the B&B tree represent partial solutions to the original optimization program.
- The numbers inside the nodes represent the order in which we have visited the nodes.
- The **edges (branches)** of the tree leading from one node to the next represent **fixing a free variable to an integer value**.

One other thing to note: the **TOTAL** number of nodes in the entire B&B tree is $1 + 2 + 2^2 + 2^3 + \ldots$ or:

$$N_{total} = \sum_{i=0}^{n_y} 2^i > 2^{n_y}$$

I am proposing a solution method that has **more possible solutions than total enumeration!** If you think this is a problem, you are not wrong! The good news, however, is that we have the **bound** part. We

can terminate extensive potions of the B&B tree early by exploiting what we know about *partial solutions having better objective function values than ANY integer solution in their feasible set*.

Incumbent Solutions

The **INCUMBENT SOLUTION** for the B&B search is the *best FEASIBLE SOLUTION* (in terms of the objective function) known so far.

- Incumbent solutions are usually discovered as the B&B tree progresses, or could have been an initial feasible solution known prior to optimization (why)?
- Since an incumbent solution is FEASIBLE in the integer space, we know if provides a *worst-case optimal solution*. In other words, we know the true optimum is the same or BETTER than the incumbent.
  - Incumbents provide an *upper bound* on the global MILP optimum for a minimization.
  - Incumbents provide a *lower bound* on the global MILP optimum for a maximization.

OK, so think about this… If I know that all solutions derived from a partial solution must have *at best* the same value of the objective, and that an incumbent solution provides a worst-case estimate of the global optimum, **I can safely truncate the B&B search at any node that has a worse partial solution objective than the current incumbent!** This allows me to terminate large portions of the B&B tree!

Beginning the Branch and Bound Search

The B&B search starts at the *root node*, which is the **first partial solution** $y^{(0)} \triangleq (\#, \#, \dots)$ with all binary variables "free." Free variables are obtained by performing a LP relaxation in which all binary variables are made continuous.

$$\min_{\boldsymbol{x},\boldsymbol{y}} \phi = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d}^T\boldsymbol{y} \qquad \min_{\boldsymbol{x},\boldsymbol{y}} \phi = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{d}^T\boldsymbol{y}$$

$$s.t. \qquad s.t.$$

$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{E}\boldsymbol{y} \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b \qquad \boldsymbol{LP} \Rightarrow \boldsymbol{RELAX} \qquad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{E}\boldsymbol{y} \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b$$

$$x_{lb} \leq x \leq x_{ub}, \quad y \in \{0,1\}^{n_y} \qquad x_{lb} \leq x \leq x_{ub}, \quad \mathbf{0 \leq y \leq 1}$$

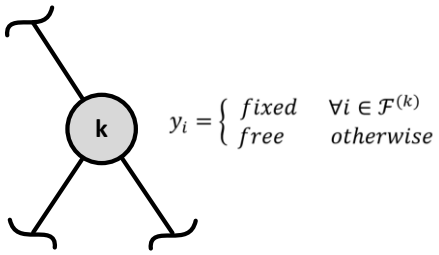
The solution at the root node provides the **best possible result** for the optimum of the MILP. This corresponds to a lower bound on the global minimum for a minimization (and vice-versa for maximization). After computing the objective at the root node, we have three possible results:

1. The solution is **infeasible** → **terminate by infeasibility**. The MILP is infeasible (why?).
2. The partial solution involves only **integer values** → **terminate by completion**. We have found the global optimum for the MILP problem on our first try (extremely lucky, but possible!).
3. The partial solution involves some binary variables with **fractional values** → **branch**. Select one binary variable with a value between 0 and 1 and create two nodes by fixing it to 0 and 1, resulting in **two new partial solutions** with one variable fixed.

It is important to note that all partial solutions are found using our usual linear programming methods, such as the Simplex Search. The B&B then proceeds to the two new intermediate nodes!

Handling Intermediate Nodes

Intermediate nodes are those that exist in the interior of the B&B tree (see the figure at right). The handling of the k^{th} intermediate node involves the solution of a **candidate relaxation** and a procedure based on the results of that candidate problem.



Candidate Problem

The **CANDIDATE PROBLEM** associated with a partial solution to an MILP model is the *restricted model* obtained by fixing the discrete variables as per the partial solution.

$$\min_{x,y} \phi = \boldsymbol{c^T x + d^T y} \qquad \min_{x,y} \phi = \boldsymbol{c^T x + d^T y}$$

$$s.t. \qquad s.t.$$

$$\boldsymbol{Ax + Ey} \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b \qquad \boldsymbol{Partial\ Sol^n} \Rightarrow \boldsymbol{\mathcal{F}^{(k)} \triangleq Fixed\ Set} \qquad \boldsymbol{Ax + Ey} \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b$$

$$x_{lb} \leq x \leq x_{ub} \qquad x_{lb} \leq x \leq x_{ub}$$

$$y \in \{0,1\}^{n_y} \qquad y_i = \begin{cases} fixed & \forall i \in \mathcal{F}^{(k)} \\ free & otherwise \end{cases}$$

Our objective at each interior node in the B&B search is to formulate the candidate problem, solve it, and compare it to our current incumbent. There are **four** possible results at each interior node (very similar to the root node):

1. The candidate problem has an **infeasible** LP relaxation → **terminate by infeasibility**. The relaxed problem at that node (and ALL possible integer problems contained by that node) are infeasible, hence there is no point searching further.
2. The candidate problem has a relaxed optimal solution no better than the current incumbent solution → **terminate by dominance**. NO feasible *completion* of the candidate problem can improve on the current incumbent (why?).
3. The candidate problem has an LP relaxation optimum that corresponds to an **integer solution**.
   → **Terminate by completion**. This is the optimum for the candidate problem.
   → **Update incumbent solution** if it is better than the current incumbent.
4. In **ANY OTHER CASE** → **branch** (same procedure as the root node).

This procedure continues until **termination**.

Terminating the Branch and Bound Search

The B&B terminates when *every* partial solution has either been branched or terminated. When this is complete, **the final incumbent solution is the global optimum to the MILP, OR the MILP is infeasible if no incumbent solution was found.** You may also terminate the B&B when an incumbent solution sufficiently close in objective value to the original relaxed problem is found within a user tolerance ϵ:

$$\frac{\phi_{rel} - \phi_{inc}^{(k)}}{0.5\left(\phi_{rel} + \phi_{inc}^{(k)}\right)} < \epsilon$$

Search Heuristics

When performing a **branch** and you have to decide which variable to branch on (fix to 0/1):

- Only branch on discrete variables having fractional values in the candidate problem.
- If there are multiple fractional discrete variables, choose the one closest to 0 or 1 to branch on.

When choosing the node pathway:

- Consider using the **Depth-First Heuristic**, which proceeds to the next possible active (as in not-terminated) partial solution with **the most variables fixed**. Can be visualized as always moving as far DOWN the B&B tree as possible before returning to upper levels.
- **Best-First** and **Depth-Forward Best-Track** are other options we will not worry too much about, but they can be found in the textbook (chapter 12.4).

Class Workshop – Branch and Bound Search

The following table shows ALL relaxed partial solutions and their corresponding combinations of fixed and free variables to a MILP with $x \geq 0$ and $y = (y_1, y_2, y_3) \in \{0.1\}$. **Solve the model** using the B&B method and the depth-first heuristic.

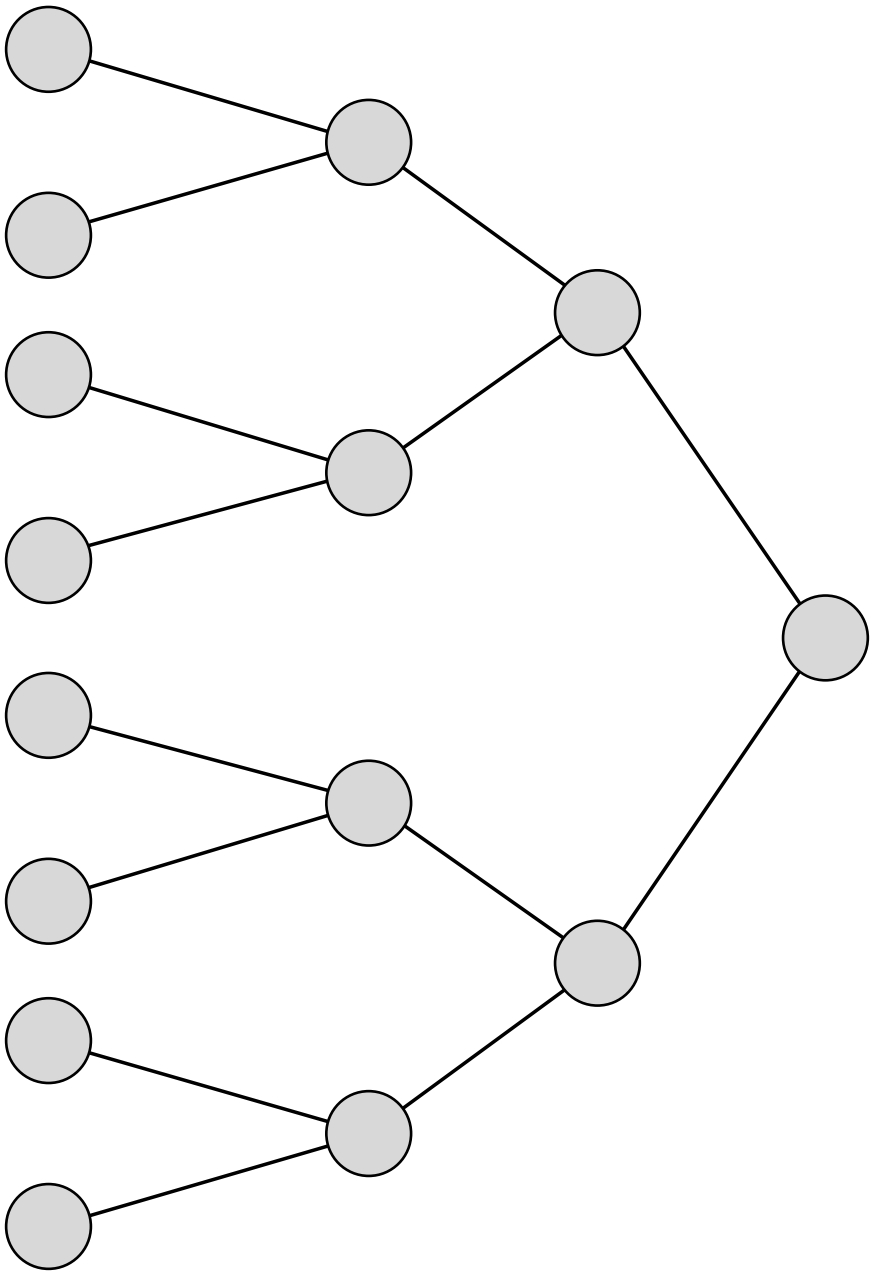
NOTE that in reality, every entry in this table comes from performing the Simplex Search (or another suitable LP solution method) and recording the optimum to that relaxed problem.

Partial	$\mathbf{y}_{\text{rel}}^{(k)}$	$x_{\text{rel}}^{(k)}$	$z_{\text{rel}}^{(k)}$	Partial	$\mathbf{y}_{\text{rel}}^{(k)}$	$x_{\text{rel}}^{(k)}$	$z_{\text{rel}}^{(k)}$
(#,#,#)	(0.2,1,0)	0	82.80	(0,0,1)	Infeasible		
(#,#,0)	(0.2,1,0)	0	82.80	(0,1,#)	(0,1,0.67)	0	80.67
(#,#,1)	(0,0.8,1)	0	79.40	(0,1,0)	(0,1,0)	2	28.00
(#,0,#)	(0.7,0,0)	0	81.80	(0,1,1)	(0,1,1)	0.5	77.00
(#,0,0)	(0.7,0,0)	0	81.80	(1,#,#)	(1,0,0)	0	74.00
(#,0,1)	(0.4,0,1)	0	78.60	(1,#,0)	(1,0,0)	0	74.00
(#,1,#)	(0.2,1,0)	0	82.80	(1,#,1)	(1,0,1)	0	63.00
(#,1,0)	(0.2,1,0)	0	82.80	(1,0,#)	(1,0,0)	0	74.00
(#,1,1)	(0,1,1)	0.5	77.00	(1,0,0)	(1,0,0)	0	74.00
(0,#,#)	(0,1,0.67)	0	80.67	(1,0,1)	(1,0,1)	0	63.00
(0,#,0)	(0,1,0)	2	28.00	(1,1,#)	(1,1,0)	0	62.00
(0,#,1)	(0,0.8,1)	0	79.40	(1,1,0)	(1,1,0)	0	62.00
(0,0,#)	Infeasible			(1,1,1)	(1,1,1)	0	51.00
(0,0,0)	Infeasible						

Then ask yourself… How many iterations would it take to solve this problem by total enumeration? How many did it take us to do it with the B&B?

THEN ask yourself… What if the candidate problems were nonlinear? What if it took us 20 minutes to solve each candidate problem? What if our solution to the *candidate problems* could not guarantee global optimality? How does that make us feel about our confidence that we can solve an MINLP to global optimality?

Workshop Solution – Branch and Bound Search



Conclusions

MILPs make up a HUGE portion of very applicable real-world optimization problems. Now, much like with LPs, we should have a much better appreciation for how to actually go about solving MILPs to global optimality. Furthermore, the fact that we needed to solve a candidate *optimization* problem at every node tells us that the computation time begins to take off on us. And, when we have no guarantee of global optimality for our candidate problems, it becomes very dicey when trying to solve MINLPs. Hopefully this gives you some perspective as to why they are so difficult to solve.

Either way, we have a new tool in the bag that we can use even if we don't have GAMS available! Imagine programming the Simplex Search in MATLAB and layering it with an over-arching B&B mechanism. How cool would that be?