CHEMICAL ENGINEERING 4G03

Module 06

Mixed-Integer Linear Programs (I)

Formulation

Dr. Jake Nease
Chemical Engineering
McMaster University

Updated March 1, 2023

Outline of Module

Suggested Readings

Rardin (1st edition): Chapter 11

Rardin (2nd edition): Chapter 11

What Makes an Integer Program

There is always a debate in this course as to whether or not we proceed from LPs to integer programming or unconstrained nonlinear programming. I am thinking we go the route of integer programming because I think the solution methodology for mixed-integer programs follows the LP section nicely. However, I would love your feedback on this at the end of the term!

Integer programs are a little different than what we have seen in class so far. However, they are *very important* and applicable to a wide range of real-world problems. Let's start with a definition:

Integer Programs (IPs)

An optimization program is known as an **integer program** if *ANY* of its decision variables are *discrete (integer)*. There are two sub-classes of integer programs:

- If ALL variables are discrete, the model is a **pure integer program**.
- Otherwise, the model is a **mixed-integer program**.

NB – All integer programs, by definition, are **non-convex** (why!?)

Integer variables occur all the time. In fact, they have *already* occurred in a lot of the problems we have faced, we are just not really considering them yet. As we mentioned in one of the very first modules, some integer variables can be **treated as continuous** and then **rounded**:

- Number of trays in a distillation column (34.6 → 35).
- Number of employees in a large company (1298.2 →1298 or 1299).
- Number of barrels of reformate to use in blend U87 (3254.1 → 3254).

However, some variables **cannot be treated as continuous without SIGNIFICANT loss of model accuracy:**

- Whether or not to operate batch reactor #2 on Mondays.
- The number of distillation columns used for a separation.
- Whether or not to invest in a fixed capital project.
- How many tests are given when teaching a course.

Class Workshop – Integer Variables

In the space below, give some examples of **integer decisions** in the field of Chemical Engineering.

Workshop Solution – Integer Variables

Fluid Mechanics	Heat Transfer	Mass Transfer	Reactor Design

Mixed Integer Linear Programs

Of course, when we introduce integer variables to an optimization program, we can end up with an integer linear program or an integer nonlinear program depending on the interactions of the variables.

Linear Versus Nonlinear Integer Programs

Consider any optimization program with integer variables:

- The optimization model is known as an **integer linear program (ILP)** if its (only) objective function and *all constraints* are linear.
- Otherwise, the model is an **integer nonlinear program (INLP)**.

Our final addition to this concept is if we combine the above two definitions to result in either a **mixed integer linear program** (**MILP**) or a **mixed integer nonlinear program** (**MINLP**). For now, we are going to focus on MILP problems because they allow us to continue the use of our LP methods (Simplex and formulations)! MINLP models are notoriously difficult to solve. As a matter of fact, *global* MINLP solvers are a continuing topic of significant academic and industrial research. GAMS has access to several MINLP solvers that are pretty good (BARON, DICOPR, ANTIGONE, KNITRO), but even they are not guaranteed to solve every MINLP problem.

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 speaking**, 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.

Types of MILPs

Formulating MILPs takes another level of ingenuity than regular LPs. Usually, we have to derive nifty methods to get around decisions through the inclusion of binaries. These binaries might also affect the variable bounds for our continuous variables (we will see how). In this course we will look at four commonly used “types” of MILPs… Although it is unlikely you will face these exact problems, formulating their *constraints* will help you formulate any integer program. These might be especially useful for your **projects**.

- “Lumpy” linear models.
- Knapsack Models.
- Assignment and Matching Models.
- Scheduling Models.

MILP Formulation Techniques

Lumpy Linear Programs

One broad case of MILPs occurs when we have *either/or* decisions added to our regular linear programming formulation. If this is the case, we can formulate the problem as a **standard LP**, but replace "either/or" or "all/nothing" decisions with binary variables.

Expressing All-or-Nothing Decisions

All-or-nothing variables requirements of the form:

$$x_j = 0 \vee x_j = u_j$$

Can be reformulated by making the substitution:

$$x_j = y_j u_j \quad s.t. \quad y \in \{0,1\}$$

- Note that $\vee$ in the above expression is the logical representation of "or."

For example, consider the following blending model in which we must **use all or none** of a particular ingredient:

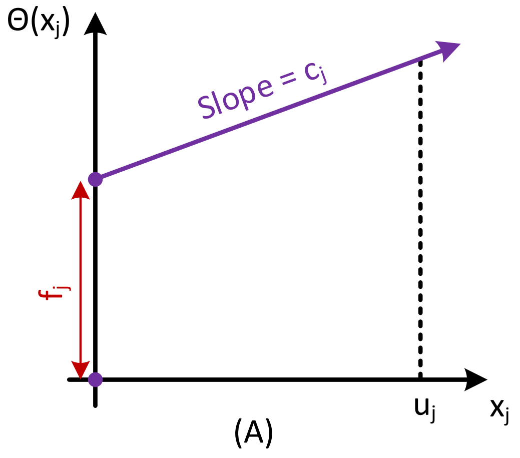
$$\begin{aligned} \min_{x} \boldsymbol{\phi} &= \mathbf{20}\boldsymbol{x_1} + \mathbf{10}\boldsymbol{x_2} + \mathbf{9}\boldsymbol{x_3} \\ \textit{Subject to} & \\ \mathbf{2}\boldsymbol{x_1} + \mathbf{3}\boldsymbol{x_2} + \mathbf{5}\boldsymbol{x_3} &\leq \mathbf{150} \\ \boldsymbol{x_1} &\leq \mathbf{60} \\ \boldsymbol{x_2} &\leq \mathbf{30} \\ \boldsymbol{x_i} &\geq \mathbf{0} \qquad \forall\, \boldsymbol{i} = \mathbf{1,2} \\ \boldsymbol{x_3} &= \mathbf{0} \vee \mathbf{20} \end{aligned}$$

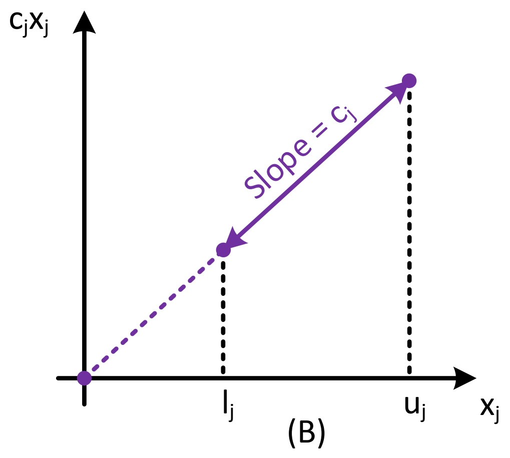
We may re-formulate the problem above as follows:

$$\begin{aligned} \min_{x} \boldsymbol{\phi} &= \mathbf{20}\boldsymbol{x_1} + \mathbf{10}\boldsymbol{x_2} + \mathbf{9}(\mathbf{20}\boldsymbol{y_3}) \\ \textit{Subject to} & \\ \mathbf{2}\boldsymbol{x_1} + \mathbf{3}\boldsymbol{x_2} + \mathbf{5}(\mathbf{20}\boldsymbol{y_3}) &\leq \mathbf{150} \\ \boldsymbol{x_1} &\leq \mathbf{60} \\ \boldsymbol{x_2} &\leq \mathbf{30} \\ \boldsymbol{x_i} &\geq \mathbf{0} \qquad \forall\, \boldsymbol{i} = \mathbf{1,2} \\ \boldsymbol{y_3} &\in \{\mathbf{0,1}\} \end{aligned}$$

We have effectively replaced the variable x_3 with a new variable y that can only take the values of 0 or 1, and we *weighted* the objective function and constraints as if that binary variable actually is either 0 or 20.

Another very common type of lumped variable is the inclusion of a **fixed charge**, also sometimes referred to as a **start-up charge** or in some cases a **switching constraint**. Fixed charges occur when there is an initial cost of accessing a resource (say, purchasing a supply contract) followed by a variable (operating) cost per unit of that variable you use. Consider panel (A) in the figure below:





Expressing Fixed Charges

Fixed charge requirements for variable x_j of the form:

$$\theta(x_j) \triangleq \begin{cases} f_j + c_j x_j \ if \ x_j > 0 \\ 0, \qquad otherwise \end{cases}$$

With some non-negative fixed cost f_j and continuous variable bounded above by u_j can be modeled by making the substitution:

$$\theta(x_j) = f_j y_j + c_j x_j \quad s.t. \quad x_j \le u_j y_j, \qquad y \in \{0,1\}$$

In other words, you claim that x_j must be **0** unless $y_j = 1$, at which point x_j cannot exceed u_j and the total cost for having $x_j > 0$ is *at least* f_j.

Class Workshop – Fixed Charges

Consider the fixed-charge objective function:

$$\min_{x_1,x_2} \phi = \theta_1(x_1) + \theta_2(x_2)$$

Where the fixed-cost constraints are:

$$\theta_1(x_1) \triangleq \begin{cases} 200 + 3x_1 \ if \ x_1 > 0 \\ 0, \qquad otherwise \end{cases}$$

$$\theta_2(x_2) \triangleq \begin{cases} 125 + 6x_2 \ if \ x_2 > 0 \\ 0, \qquad otherwise \end{cases}$$

Subject to the constraints:

$$\begin{aligned} \boldsymbol{x_1 + x_2} &\ge \boldsymbol{12} \\ \boldsymbol{x_1} &\le \boldsymbol{6} \\ \boldsymbol{x_2} &\le \boldsymbol{10} \\ \boldsymbol{x_i} &\ge \boldsymbol{0} \qquad \forall\, \boldsymbol{i} \end{aligned}$$

Formulate an appropriate MILP model.

Workshop Solution – Fixed Charges

As a final wrinkle, we can also introduce the concept of a fixed charge for a continuous variable that also has a *minimum allowable value*. Mathematically speaking, this means that our variable is **semi-continuous** (only exists between certain ranges). Note that this is different from setting upper and lower bounds for a continuous variable because we are permitting it to be **zero** as well as between some lower and upper bound. See panel (B) in the figure above.

Expressing Semi-Continuous Requirements

Semi-continuous requirements with lower and upper bounds $l_j > 0$ and $u_j > 0$ of the form:

$$x_j = 0 \ \vee \ \ l_j \le x_j \le u_j$$

Can be modeled through the addition of **two new constraints** that use a single binary variable $y_j \in \{0,1\}$ to guarantee that x_j takes on 0 or a value between l_j and u_j (think why):

$$x_j \ge l_j y_j$$

$$x_j \le u_j y_j$$

If $y_j = 1$, we end up with $l_j \le x_j \le u_j$. If $y_j = 0$, we end up with $0 \le x_j \le 0$.

As a final perspective piece on fixed cost and semicontinuous variables – there is a good chance you will encounter these throughout your careers (should you have the chance to use optimization). Things like fixed costs of pieces of equipment (followed by operating costs thereafter) are classic decisions that have to be made by chemical engineers.

Class Workshop – Semicontinuous Variables

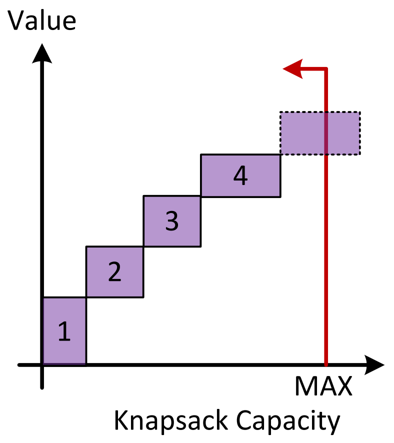
Consider the optimization program below and re-formulate it as a MILP:

$$\begin{aligned} \min_{x} \phi &= 20x_1 + 10x_2 + 9x_3 \\ &\textit{Subject to} \\ 2x_1 + 3x_2 + 5x_3 &\leq 150 \\ x_1 &\leq 60 \\ x_2 &\leq 30 \\ x_i &\geq 0 \qquad \forall\, i = 1, 2 \\ x_3 &= 0 \vee 10 \leq x_3 \leq 30 \end{aligned}$$

Workshop Solution – Semicontinuous Variables

Knapsack Models

The whole idea of a knapsack model is that you either have the presence or absence of certain objects (variables) that contribute to some known objective. The variables are like items you can fit in your knapsack (say, food for camping) that have fixed volumes. Each food item gets you a certain number of days-worth of food (**maximize** the food you can pack), and you have only a certain maximum volume available (**constrained** combination of items you may take). See the figure at right: we can't fit more than items 1 → 4 in the knapsack. Adding more items would increase the value of the knapsack, but will violate our capacity constraint (note we also may not take a portion of an item).



More non-trivial example of knapsack constraints may include the decision to invest in certain fixed projects subject to a maximum available investment budget (sound familiar…?).

A very easy scenario that is actually a knapsack problem is returning the minimum number of 5, 10, 25 100 and 200 cent coins in order to give exact change to a customer after a cash purchase (consider formulating this as an exercise).

Knapsack Constraints

Knapsack problems are the purest form of integer programming models that attempt to select the presence or absence of a collection of options in order to attain the **highest possible value** (or lowest possible cost) subject to a limitation on resources that are consumed.

- Each element is either **all-in** or **all-out** of the decision: $y_j \triangleq \begin{cases} 1, & selected \\ 0, & otherwise \end{cases}$
- **Dependence** of choice j on choice i can be enforced: $y_j \leq y_i$
- **Mutual exclusivity** of choice j from a set of choices J: $\sum_{j \in J} y_j \leq 1$
- **At most p** of a selection of J choices is similar: $\sum_{j \in J} y_j \leq p$
- Using modifications of the above constraint can ensure that **at least** p or **exactly** p can be selected as well.

Class Workshop – Knapsack Modeling

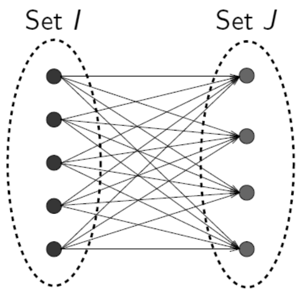
Your company is looking to invest in some optimization solvers to be able to solve a variety of industry problems. You are required to be able to solve LP, IP, NLP and MILP problems and have the following solvers available for purchase. A "Y" in the table indicates that the given solver can solve that type of problem, whereas a blank indicates it cannot.

Problem	Solver Package (j)				
Type	1	2	3	4	5
LP	Y		Y	Y	Y
IP			Y	Y	Y
NLP		Y			Y
MILP				Y	Y
COST	2	3	5	7	14

1. Formulate the MILP that ensures you can solve all types of problems at the lowest cost.
2. Formulate the MILP that ensures you have precisely *one* solver that can solve each type of problem.
3. The software developers for packages 2 and 4 are rivals and will not grant you concurrent licenses. Formulate the MINLP that ensures we can solve all types of problems without using packages 2 and 4 concurrently.

Workshop Solution – Knapsack Modeling

Assignment Models



Assignment models match objects of two distinct types (sets $i \in I$ and $j \in J$) via some sort of objective such as minimizing cost, maximizing interaction, or otherwise. For example, we might match a salesperson to a sales region, a manufacturing job to a certain machine, or a utility header to a certain collection of process units (figure courtesy of Dr. Chachuat).

Assignment Models

Assignment models are given the decision variables:

$$y_{i,j} \triangleq \begin{cases} 1, & \text{if object } i \text{ is matched with object } j \\ 0, & \text{otherwise} \end{cases}$$

Constraints **forcing** the matching of every i and/or every j to be assigned are of the form:

$$\sum_j y_{i,j} = 1 \ \forall\, i \in I \qquad\qquad \sum_i y_{i,j} = 1 \ \forall\, j \in J$$

Space constraints that dictate how the assignment $y_{i,j}$ impacts the availability of some maximum available resource b_j according to the variable $s_{i,j}$ are written as:

$$\sum_i s_{i,j} y_{i,j} \le b_j \qquad \forall\, j \in J$$

Linear assignment costs determine the cost of associating object i with object j via an assignment cost $c_{i,j}$ and have the objective function of the form:

$$\phi = \sum_i \sum_j c_{i,j} y_{i,j}$$

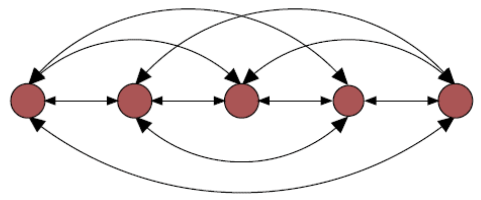
Class Workshop – Assignment Modeling

Items $i = 1 \ldots 1000$ sold at a local brick-and-mortar store, each with a volume c_i, are being stored at a local warehouse. The warehouse has $j = 1 \ldots 50$ designated storage locations a distance d_j from the (only) shipping terminal. Each storage location has (equal) capacity b. Formulate the MILP that determines the storage location for each item with the smallest total travel distance. Then, report any simplifying assumptions you made in your model.

Workshop Solution – Assignment Modeling

Matching Models

Matching models are a subtype of assignment models in which the discrimination between sets i and j are eliminated, thus resulting in the matching of certain elements in a set i to other elements in the same set, i' (figure courtesy of Dr. Chachuat).



Matching Models

Matching models are given the decision variables:

$$y_{i,i'} \triangleq \begin{cases} 1, & \text{if object } i \text{ is paired with object } i' \\ 0, & \text{otherwise} \end{cases}$$

Where we have the additional constraint that $i > i'$ to avoid double-counting (rail cars?). **Matching constraints** that force the pairing of object i with and ONLY with another object i' are:

$$\sum_{i'<i} y_{i',i} + \sum_{i'>i} y_{i,i'} = 1$$

Class Workshop – Matching Modeling

Professors in chemical engineering for a course has been given anonymous ranked preferences $p_{s,s'}$ of having a student s work on a class project with student s'. Formulate a MILP model that maximizes the Pareto-optimal preference of matching students together for the project.

Workshop Solution – Matching Modeling

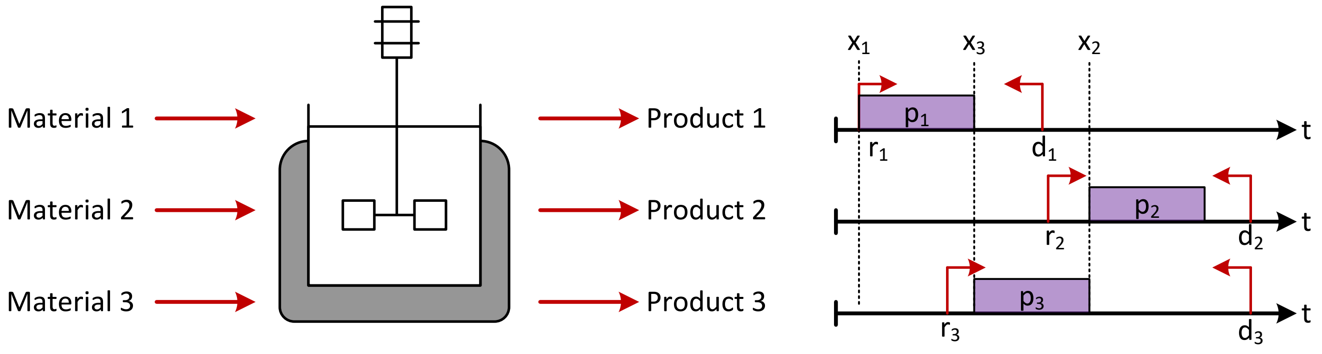
Scheduling Optimization

Scheduling optimization is perhaps the most sought-after area of expertise in the operations planning industry today. Patrick Carter, our friend from Dofasco who visited 4N04 last term, did his entire master's degree on the scheduling optimization of the steel production lines at Dofasco. Why were they willing to pay a student for two years for this? Because it is important and it a HUGE waste of money to have your process being run at a sub-optimal schedule. Some definitions:

- **Scheduling** is the allocation of resources over time.
- **Single-process scheduling** attempts to sequence a series of desired processes that must be completed one at a time and can only be completed by a **single process**.

See the figure below, for example, in which we have a batch reactor that we must use to convert three intermediate feeds to three finished products. We are given the following typical data and variables:

- $p_j \triangleq$ the estimated **p**rocess time for task (or product) j.
- $r_j \triangleq$ the time at which the intermediates for task j are **r**eleased (and we may begin process j).
- $d_j \triangleq$ the **d**ue date for product or process j (meet demand, go downstream, *etc*.).
- $x_j \triangleq$ the time at which we **begin processing product *j*** (*DECISION* variables).



It is important to note that when it comes to **deadlines** we typically *avoid* formulating the problem such that all tasks must be completed before their associated deadlines, although these constraints can be used for very critical deadlines:

$$x_j + p_j \leq d_j \;\forall\, j$$

When dealing with deadlines it is typically more suitable to establish **goals** that can be **optimized** subject to resource availability constraints. Formulating **goals** allow us to have an objective that is not just "do this as fast as possible" in the event that *it might be impossible to meet all deadlines within the time frame given*. Consider formulating it as follows:

Scheduling Programs – Objectives and Resource Availabilities

Consider a scheduling program in which the process start times x_j for $j \in J$ products are to be selected, each with processing times p_j, desired deadlines d_j and release times r_j. The constraint that indicates **our intermediate must be released before it may be processed** is simply:

$$x_j \geq r_j \;\forall\, j$$

Possible **objective functions we are trying to minimize** may take any of the following forms:

Maximum Completion Time	→	$\max\{x_j + p_j : j = 1 \dots n\}$
Mean Completion Time	→	$\frac{1}{n}\sum_{j=1}^{n}(x_j + p_j)$
Maximum Lateness	→	$\max\{x_j + p_j - d_j : j = 1 \dots n\}$
Mean Lateness	→	$\frac{1}{n}\sum_{j=1}^{n}(x_j + p_j - d_j)$
Maximum Tardiness	→	$\max\{0, max\{x_j + p_j - d_j : j = 1 \dots n\}\}$
Mean Tardiness	→	$\frac{1}{n}\sum_{j=1}^{n}\max\{0, x_j + p_j - d_j\}$

Class Workshop – Scheduling Objectives

The following table shows the process times, release times, due dates and scheduled starts for three jobs to be performed on a single process unit. Compute the values for maximum and mean completion, lateness and tardiness as defined above.

	Job 1	Job 2	Job 3
Process Time	15	6	9
Release Time	5	10	0
Due Date	20	25	36
Scheduled Start	9	24	0

Workshop Solution – Scheduling Objectives

Our final requirement is that we ensure that only *one* task is completed at any given time. This means that no task is allowed to begin before the end of the current task. This is what is known as a **conflict constraint** between processes j and j':

$$x_j + p_j \leq x_{j'} \vee x_{j'} + p_{j'} \leq x_j$$

In plain language, this translates to "I can't start process j' until process j is complete" **OR** "I can't start process j until process j' is complete." They may seem the same, but they are in fact saying different things. NOW, how am I going to formulate these constraints? Well, you might recall that when we encounter "OR" scenarios we use a binary variable. In this case, we will introduce what is known as a **scheduling disjunctive variable $y_{j,j'}$**. We may then formulate a **Big-M** method to account for the sequencing of the tasks as shown below.

Scheduling Programs – Disjunctive Variables and Constraint Pairs

A set of (discrete) **disjunctive variables** in a process scheduling optimization determine the sequence of jobs on a single processing unit by specifying whether job j is performed before job j' or vice-versa. They are defined as:

$$y_{j,j'} \triangleq \begin{cases} 1, & \text{if task } j \text{ is scheduled before } j' \\ 0, & \text{if task } j' \text{is scheduled before } j \end{cases}$$

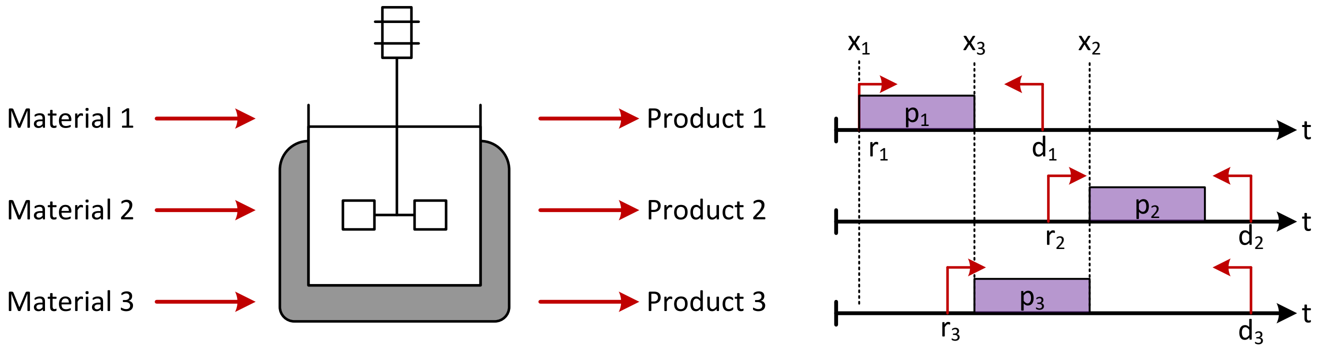
As in the matching case, we only consider $j' > j$ (one half of the scenarios) since we do not want to double count and the other half are symmetrical. We may now formulate the **disjunctive constraints** via a **Big-M methodology** as:

$$x_j + p_j \le x_{j'} + M(1 - y_{j,j'})$$

$$x_{j'} + p_{j'} \le x_j + My_{j,j'}$$

NB – the selection of M is up to the user, but good heuristics may include the maximum possible makespan (why?) or some other **large positive** constant that must be guaranteed to be larger than the makespan.

For example, consider our schedule from the figure above, repeated here for reference:



Consider M to be some large number. We know from the above sequencing diagram that $y_{1,2} = 1$, $y_{1,3} = 1$ and $y_{2,3} = 0$. Are the disjunctive constraints for this problem satisfied?

Consider the $y_{j,j'} = y_{1,2}$ case. We have:

$$x_1 + p_1 \le x_3 + 0$$

$$x_3 + p_3 \le x_1 + M$$

Both constraints are satisfied since M is a large number and thus $x_3 + p_3$ cannot *ever* happen before x_1. Now consider the $y_{j,j'} = y_{2,3}$ case. We have:

$$x_2 + p_2 \le x_3 + M$$

$$x_3 + p_3 \le x_2 + 0$$

In this case, the first constraint is satisfied but inactive, and the second constraint tells us that we must process x_3 before we may begin processing x_2. The system works!

Class Workshop – Conjunctive Constraints

Formulate integer linear programming constraints that ensure a feasible schedule for tasks having processing times of 4, 8, and 12 hours, respectively.

Workshop Solution – Conjunctive Constraints

Conclusions

OK! You are now equipped to formulate a variety of types of integer programs. However, it does not end here! Hopefully should you encounter other types of integer situations you will be able to adapt the mathematical ideas in this module to formulate those situations into mathematical programs. With a little ingenuity, you should be able to apply binary decisions to a whole variety of optimization problems!

~~ END OF MODULE ~~

\|T|/