20. Logic constraints, integer variables

- If-then constraints
- Generalized assignment problems
- Logic constraints
- Modeling a restricted set of values
- Sudoku!
If-then constraints

A single simple trick (with suitable adjustments) can help us model a great variety of if-then constraints.

The trick

- We’d like to model the constraint: if $z = 0$ then $a^T x \leq b$.
- Let $M$ be an upper bound for $a^T x - b$.
- Write: $a^T x - b \leq M z$
- If $z = 0$, then $a^T x - b \leq 0$ as required. Otherwise, we get $a^T x - b \leq M$, which is always true.
If-then constraints

Slight change: if \( z = 1 \) then \( a^T x \leq b \)

- Again, let \( M \) be an upper bound for \( a^T x - b \)
- Write: \( a^T x - b \leq M(1 - z) \)

Reversed inequality: if \( z = 0 \) then \( a^T x \geq b \)

- Write constraint as \( -a^T x + b \leq 0 \)
- Let \( m \) be an upper bound on \( -a^T x + b \)
- Write: \( -a^T x + b \leq mz. \) Same as: \( a^T x - b \geq -mz \)
- Note: \( -m \) is a lower bound on \( a^T x - b. \)
If-then constraints

The converse: if \( a^T x \leq b \) then \( z = 1 \)

- Equivalent to: if \( z = 0 \) then \( a^T x > b \) (contrapositive).
- The strict inequality is not really enforceable. Instead, write: if \( z = 0 \) then \( a^T x \geq b + \varepsilon \) where \( \varepsilon \) is small.
- Let \( m \) be a lower bound for \( a^T x - b \) and we obtain the equivalent constraint: \( a^T x - b \geq mz + \varepsilon (1 - z) \)
- If \( z = 0 \), we get \( a^T x \geq b + \varepsilon \), as required. Otherwise, we get: \( a^T x - b \geq m \), which is always true.

- **Note:** If \( a, x, b \) are integer-valued, we may set \( \varepsilon = 1 \).
### If-then constraints (summary)

<table>
<thead>
<tr>
<th>Logic statement</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $z = 0$ then $a^T x \leq b$</td>
<td>$a^T x - b \leq Mz$</td>
</tr>
<tr>
<td>if $z = 0$ then $a^T x \geq b$</td>
<td>$a^T x - b \geq mz$</td>
</tr>
<tr>
<td>if $z = 1$ then $a^T x \leq b$</td>
<td>$a^T x - b \leq M(1 - z)$</td>
</tr>
<tr>
<td>if $z = 1$ then $a^T x \geq b$</td>
<td>$a^T x - b \geq m(1 - z)$</td>
</tr>
<tr>
<td>if $a^T x \leq b$ then $z = 1$</td>
<td>$a^T x - b \geq mz + \varepsilon(1 - z)$</td>
</tr>
<tr>
<td>if $a^T x \geq b$ then $z = 1$</td>
<td>$a^T x - b \leq Mz - \varepsilon(1 - z)$</td>
</tr>
<tr>
<td>if $a^T x \leq b$ then $z = 0$</td>
<td>$a^T x - b \geq m(1 - z) + \varepsilon z$</td>
</tr>
<tr>
<td>if $a^T x \geq b$ then $z = 0$</td>
<td>$a^T x - b \leq M(1 - z) - \varepsilon z$</td>
</tr>
</tbody>
</table>

Where $M$ and $m$ are upper and lower bounds on $a^T x - b$. 

---

20-5
Return to fixed costs and lower bounds

- Modeling a fixed cost: if $x > 0$ then $z = 1$.
  - Use the contrapositive: if $z = 0$ then $x \leq 0$.
  - Apply the 1\textsuperscript{st} rule on Slide 20-5.

- Modeling a lower bound: either $x = 0$ or $x \geq m$.
  - Equivalent to: if $x > 0$ then $x \geq m$.
  - Equivalent to the following two logical constraints:
    - if $x > 0$ then $z = 1$, and if $z = 1$ then $x \geq m$.
    - The first one is a fixed cost (see above)
    - The second one is the 4\textsuperscript{th} rule on Slide 20-5.
Generalized assignment problems (GAP)

- Set of machines: $\mathcal{M} = \{1, 2, \ldots, m\}$ that can perform jobs. (Think of these as the facilities in the facility problem)

- Machine $i$ has a fixed cost of $h_i$ if we use it at all.

- Machine $i$ has a capacity of $b_i$ units of work (this is new!)

- Set of jobs: $\mathcal{N} = \{1, 2, \ldots, n\}$ that must be performed. (Think of these as the customers in the facility problem)

- Job $j$ requires $a_{ij}$ units of work to be completed if it is completed on machine $i$.

- Job $j$ will cost $c_{ij}$ if it is completed on machine $i$.

- Each job must be assigned to exactly one machine.
GAP model

minimize \( x, z \sum_{i \in M} h_i z_i + \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} \) (fixed cost + assignment cost)

subject to:

\[ \sum_{i \in M} x_{ij} = 1 \quad \forall j \in N \] (one machine per job)

\[ \sum_{j \in N} a_{ij} x_{ij} \leq b_i \quad \forall i \in M \] (work budget)

\[ x_{ij} \leq z_i \quad \forall i \in M, j \in N \] (if \( x_{ij} > 0 \) then \( z_i = 1 \))

\[ x_{ij}, z_i \in \{0, 1\} \quad \forall i \in M, j \in N \] (all binary!)

- \( z_i = 1 \) if machine \( i \) is used, and
- \( x_{ij} = 1 \) if job \( j \) is performed by machine \( i \).
- **Note:** many choices possible for \( M_i \) and aggregations.
New constraints

Let’s make GAP more interesting...

1. If you use $k$ or more machines, you must pay a penalty of $\lambda$.
2. If you operate either machine 1 or machine 2, you may not operate both machines 3 and 4 at the same time.
3. If you operate both machines 1 and 2, then machine 3 must be operated at 40% of its capacity.
4. Each job $j \in \mathcal{N}$ has a duration $d_j$. Minimize the time we have to wait before all jobs are completed. (this is called the makespan).
If you use $k$ or more machines, you must pay a penalty of $\lambda$.

- Using $k$ or more machines is equivalent to saying that
  
  \[ z_1 + z_2 + \cdots + z_m \geq k \]

- Let $\delta_1 = 1$ if we incur the penalty. We now have the if-then constraint: if $\sum_{i \in \mathcal{M}} z_i \geq k$ then $\delta_1 = 1$.

- Use the 6th rule on Slide 20-5 and obtain:
  
  \[ \sum_{i \in \mathcal{M}} z_i \leq m\delta_1 + (k - 1)(1 - \delta_1) \]

- add $\lambda \delta_1$ to the cost function.
If you operate either machine 1 or machine 2, you may not operate both machines 3 and 4 at the same time.

- Operating machine 1 or machine 2: \( z_1 + z_2 \geq 1 \).
- Not operating machines 3 and 4: \( z_3 + z_4 \leq 1 \).
- We must model \( z_1 + z_2 \geq 1 \implies z_3 + z_4 \leq 1 \)
  - Same trick as before: model this in two steps:
    \( z_1 + z_2 \geq 1 \implies \delta_2 = 1 \) and \( \delta_2 = 1 \implies z_3 + z_4 \leq 1 \)
  - First follows from 6\(^{th}\) rule on Slide 20-5
  - Second follows from 3\(^{rd}\) rule on Slide 20-5
- Result: \( z_1 + z_2 \leq 2\delta_2 \) and \( z_3 + z_4 + \delta_2 \leq 2 \).
GAP 2 (cont’d)

If you operate either machine 1 or machine 2, you may not operate both machines 3 and 4 at the same time.

We didn’t do anything to ensure that when $z_i = 1$, the machines are actually operating! (we didn’t explicitly disallow paying the fixed cost without using the machine).

- To force the converse as well, include the constraint:
  
  if $z_i = 1$ then $\sum_{j \in \mathcal{N}} x_{ij} \geq 1$

- Use the 4$^{th}$ rule on Slide 20-5.

- Result: $\sum_{j \in \mathcal{N}} x_{ij} \geq z_i$ (for $i = 1, 2, 3, 4$)
If you operate both machines 1 and 2, then machine 3 must be operated at 40% of its capacity.

- Operate both machines 1 and 2: \( z_1 + z_2 \geq 2 \)

- Capacity of machine 3 drops: \( b_3 \) becomes 0.4\( b_3 \).

- Two parts to the implementation:
  - \( z_1 + z_2 \geq 2 \quad \implies \quad \delta_3 = 1. \) (6th rule on Slide 20-5)
  - \( \delta_3 = 1 \quad \implies \quad \sum_{j \in \mathcal{N}} a_{3j} x_{3j} \leq 0.4 b_3. \) (3rd rule on Slide 20-5)

- Equivalently, just replace \( b_3 \) by: \( b_3(1 - \delta_3) + 0.4 b_3 \delta_3. \)
Each job $j \in \mathcal{N}$ has a duration $d_j$. Minimize the time we have to wait before all jobs are completed. (the makespan)

- Machine $i$ completes all its jobs in time: $\sum_{j \in \mathcal{N}} x_{ij} d_j$
- Minimax problem (no integer variables needed!)
- Let $t$ be the makespan; $t = \max_{i \in \mathcal{M}} \left( \sum_{j \in \mathcal{N}} x_{ij} d_j \right)$
- Model: minimize $t$ subject to:
  \[ t \geq \sum_{j \in \mathcal{N}} x_{ij} d_j \quad \text{for all } i \in \mathcal{M} \]
Logic constraints

- A **proposition** is a statement that evaluates to true or false. One example we’ve seen: a linear constraint $a^T x \leq b$.

- We’ll use binary variables $\delta_i$ to represent propositions $P_i$:
  $$\delta_i = \begin{cases} 1 & \text{if proposition } P_i \text{ is true} \\ 0 & \text{if proposition } P_i \text{ is false} \end{cases}$$

  The term for this is that $\delta_i$ is an **indicator variable**.

How can we turn logical statements about the $P_i$’s into algebraic statements involving the $\delta_i$’s?

Some standard notation:

- $\lor$ means “or”
- $\land$ means “and”
- $\neg$ means “not”
- $\implies$ means “implies”
- $\iff$ means “if and only if”
- $\oplus$ means “exclusive or”
Boolean algebra

Basic definitions:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P \land Q$</th>
<th>$P \lor Q$</th>
<th>$P \oplus Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Useful relationships:

- $\neg(P_1 \land \cdots \land P_k) = \neg P_1 \lor \cdots \lor \neg P_k$
- $\neg(P_1 \lor \cdots \lor P_k) = \neg P_1 \land \cdots \land \neg P_k$
- $P \land (Q \lor R) = (P \land Q) \lor (P \land R)$
- $P \lor (Q \land R) = (P \lor Q) \land (P \lor R)$
- $P \oplus Q = (P \land \neg Q) \lor (\neg P \land Q)$
## Logic to algebra

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬(P_1)</td>
<td>(\delta_1 = 0)</td>
</tr>
<tr>
<td>(P_1 \lor P_2)</td>
<td>(\delta_1 + \delta_2 \geq 1)</td>
</tr>
<tr>
<td>(P_1 \oplus P_2)</td>
<td>(\delta_1 + \delta_2 = 1)</td>
</tr>
<tr>
<td>(P_1 \land P_2)</td>
<td>(\delta_1 = 1, \delta_2 = 1)</td>
</tr>
<tr>
<td>¬((P_1 \lor P_2))</td>
<td>(\delta_1 = 0, \delta_2 = 0)</td>
</tr>
<tr>
<td>(P_1 \implies P_2)</td>
<td>(\delta_1 \leq \delta_2) (equivalent to: (\neg P_1 \lor P_2))</td>
</tr>
<tr>
<td>(P_1 \implies (\neg P_2))</td>
<td>(\delta_1 + \delta_2 \leq 1) (equivalent to: (\neg(P_1 \land P_2)))</td>
</tr>
<tr>
<td>(P_1 \iff P_2)</td>
<td>(\delta_1 = \delta_2)</td>
</tr>
<tr>
<td>(P_1 \implies (P_2 \land P_3))</td>
<td>(\delta_1 \leq \delta_2, \delta_1 \leq \delta_3)</td>
</tr>
<tr>
<td>(P_1 \implies (P_2 \lor P_3))</td>
<td>(\delta_1 \leq \delta_2 + \delta_3)</td>
</tr>
<tr>
<td>((P_1 \land P_2) \implies P_3)</td>
<td>(\delta_1 + \delta_2 \leq 1 + \delta_3)</td>
</tr>
<tr>
<td>((P_1 \lor P_2) \implies P_3)</td>
<td>(\delta_1 \leq \delta_3, \delta_2 \leq \delta_3)</td>
</tr>
<tr>
<td>(P_1 \land (P_2 \lor P_3))</td>
<td>(\delta_1 = 1, \delta_2 + \delta_3 \geq 1)</td>
</tr>
<tr>
<td>(P_1 \lor (P_2 \land P_3))</td>
<td>(\delta_1 + \delta_2 \geq 1, \delta_1 + \delta_3 \geq 1)</td>
</tr>
</tbody>
</table>
## More logic to algebra

<table>
<thead>
<tr>
<th>Statement</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 \lor P_2 \lor \cdots \lor P_k )</td>
<td>( \sum_{i=1}^{k} \delta_i \geq 1 )</td>
</tr>
<tr>
<td>((P_1 \land \cdots \land P_k) \implies (P_{k+1} \lor \cdots \lor P_n))</td>
<td>( \sum_{i=1}^{k} (1 - \delta_i) + \sum_{i=k+1}^{n} \delta_i \geq 1 )</td>
</tr>
<tr>
<td>at least ( k ) out of ( n ) are true</td>
<td>( \sum_{i=1}^{n} \delta_i \geq k )</td>
</tr>
<tr>
<td>exactly ( k ) out of ( n ) are true</td>
<td>( \sum_{i=1}^{n} \delta_i = k )</td>
</tr>
<tr>
<td>at most ( k ) out of ( n ) are true</td>
<td>( \sum_{i=1}^{n} \delta_i \leq k )</td>
</tr>
<tr>
<td>( P_n \iff (P_1 \lor \cdots \lor P_k) )</td>
<td>( \sum_{i=1}^{k} \delta_i \geq \delta_n, \delta_n \geq \delta_j, j = 1, \ldots, k )</td>
</tr>
<tr>
<td>( P_n \iff (P_1 \land \cdots \land P_k) )</td>
<td>( \delta_n + k \geq 1 + \sum_{i=1}^{k} \delta_i, \delta_j \geq \delta_n, j = 1, \ldots, k )</td>
</tr>
</tbody>
</table>
Modeling a restricted set of values

- We may want variable $x$ to only take on values in the set \{a_1, \ldots, a_m\}.

- We introduce binary variables $y_1, \ldots, y_m$ and the constraints

$$x = \sum_{j=1}^{m} a_j y_j, \quad \sum_{j=1}^{m} y_j = 1, \quad y_j \in \{0, 1\}$$

- $y_i$ serves to select which $a_i$ will be selected.

- The set of variables \{y_1, y_2, \ldots, y_m\} is called a **special ordered set** (SOS) of variables.
Example: building a warehouse

- Suppose we are modeling a facility location problem in which we must decide on the size of a warehouse to build.
- The choices of sizes and associated cost are shown below:

<table>
<thead>
<tr>
<th>Size</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>180</td>
</tr>
<tr>
<td>40</td>
<td>320</td>
</tr>
<tr>
<td>60</td>
<td>450</td>
</tr>
<tr>
<td>80</td>
<td>600</td>
</tr>
</tbody>
</table>

Warehouse sizes and costs
Example: building a warehouse

- Using binary decision variables $x_1, x_2, \ldots, x_5$, we can model the cost of building the warehouse as
  \[
  \text{cost} = 100x_1 + 180x_2 + 320x_3 + 450x_4 + 600x_5.
  \]

- The warehouse will have size
  \[
  \text{size} = 10x_1 + 20x_2 + 40x_3 + 60x_4 + 80x_5,
  \]

- and we have the SOS constraint
  \[
  x_1 + x_2 + x_3 + x_4 + x_5 = 1.
  \]
What about integers?

- What if $x$ is an integer, i.e. $x \in \{1, 2, \ldots, 10\}$
- First option: use 10 separate variables:

  $$x = \sum_{k=1}^{10} k y_k, \quad \sum_{k=1}^{10} y_k = 1, \quad y_k \in \{0, 1\}$$

- Another option: use 4 binary variables (less symmetry):

  $$x = y_1 + 2y_2 + 4y_3 + 8y_4, \quad 1 \leq x \leq 10, \quad y_k \in \{0, 1\}$$

Performance is solver-dependent. If the solver allows integer constraints directly, that’s often the right choice.
Example: Sudoku

- fill grid with numbers \( \{1, 2, \ldots, 9\} \)
- each row and each column contains distinct numbers
- each \( 3 \times 3 \) cluster contains distinct numbers
Example: Sudoku

- Decision variables: $X \in \{0, 1\}^{9 \times 9 \times 9}$ (729 binary variables)

  $X_{ijk} = \begin{cases} 
  1 & \text{if } (i, j) \text{ entry is a } k \\
  0 & \text{otherwise}
\end{cases}$

  Can fill in known entries right away.

- Basic constraints: (324 in total)
  - $\sum_{k=1}^{9} X_{ijk} = 1 \quad \forall i, j$ (SOS constraint)
  - $\sum_{i=1}^{9} X_{ijk} = 1 \quad \forall j, k$ (column $j$ contains exactly one $k$)
  - $\sum_{j=1}^{9} X_{ijk} = 1 \quad \forall i, k$ (row $i$ contains exactly one $k$)
  - $\sum_{(i,j) \in C} X_{ijk} = 1 \quad \forall C, k$ (cluster $C$ contains exactly one $k$)

- Much trickier to model using other integer representations!

- Julia code: Sudoku.ipynb