21. More models, QAP and SOS

- Quadratic assignment problems
- SOS1 constraints
- SOS2 constraints
- General piecewise linear functions
Quadratic assignment problems (QAP)

Two big classes of problems seen so far:

- **GAP**: Assign jobs to machines. Each assignment has a price, each machine has a fixed cost and a max capacity.

- **TSP**: Find the optimal sequence of items (or cities). Each possible transition has an associated cost. (probably the most famous integer program)

One more important class of problems:

- **QAP**: Assign facilities to fixed locations. The facilities have known communication requirements and incurred costs depend on the distances between the facilities. (second most famous integer program – harder than TSP)
A small shopping mall has four shop locations. The walking distance, in feet, between all pairs of locations are shown below. Four shops, designated A, B, C, D, are to be assigned to the four locations in such a way that customers traveling between pairs of shops will not walk too far. We have data on the number of customers per week that travel between the shops, shown below.

<table>
<thead>
<tr>
<th>Distance</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>80</td>
<td>150</td>
<td>170</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>130</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

- let $x_{ij} = 1$ if shop $i$ is in location $j$. 
QAP example: shopping mall

• Each shop can only be in one location:

\[ \sum_{j=1}^{L} x_{ij} = 1 \quad \text{for } i = 1, \ldots, S \]

• Each location can only contain one shop:

\[ \sum_{i=1}^{S} x_{ij} = 1 \quad \text{for } j = 1, \ldots, L \]

• Cost depends on pairs of facilities! If shop \( i \) is in location \( j \) and shop \( k \) is in location \( \ell \), then between them we have a flow of \( f_{ik} \) and a distance of \( d_{j\ell} \). The total cost is:

\[
\frac{1}{2} \sum_{i=1}^{S} \sum_{j=1}^{L} \sum_{k=1}^{S} \sum_{\ell=1}^{L} f_{ik} d_{j\ell} x_{ij} x_{k\ell}
\]
QAP example: shopping mall

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \sum_{i=1}^{S} \sum_{j=1}^{L} \sum_{k=1}^{S} \sum_{\ell=1}^{L} f_{ik} d_{j\ell} x_{ij} x_{k\ell} \\
\text{subject to:} & \quad \sum_{j=1}^{L} x_{ij} = 1 \quad \text{for } i = 1, \ldots, S \\
& \quad \sum_{i=1}^{S} x_{ij} = 1 \quad \text{for } j = 1, \ldots, L \\
& \quad x_{ij} \in \{0, 1\}
\end{align*}
\]

• cost is \textbf{quadratic} in the variables! Can we linearize?
• define new binary variable: \( z_{ijk\ell} = x_{ij} x_{k\ell} \)
QAP example: shopping mall

minimize \[ \frac{1}{2} \sum_{i=1}^{S} \sum_{j=1}^{L} \sum_{k=1}^{S} \sum_{\ell=1}^{L} f_{ik} d_{j\ell} z_{ijk\ell} \]

subject to:
- \[ \sum_{j=1}^{L} x_{ij} = 1 \quad \text{for } i = 1, \ldots, S \]
- \[ \sum_{i=1}^{S} x_{ij} = 1 \quad \text{for } j = 1, \ldots, L \]
- \( x_{ij} \in \{0, 1\} \)
- \( z_{ijk\ell} = x_{ij} x_{k\ell} \)

• Equivalent to: \( (z_{ijk\ell} = 1) \iff (x_{ij} = 1) \land (x_{k\ell} = 1) \)
QAP example: shopping mall

How do we model: \((z_{ijkl} = 1) \iff (x_{ij} = 1) \land (x_{k\ell} = 1)\)?

- we saw this when we discussed logic constraints!
  - \((\implies\): \(x_{ij} \geq z_{ijkl}\) and \(x_{k\ell} \geq z_{ijkl}\)
  - \((\impliedby\): \(x_{ij} + x_{k\ell} \leq z_{ijkl} + 1\)
QAP example: shopping mall

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \sum_{i=1}^{S} \sum_{j=1}^{L} \sum_{k=1}^{S} \sum_{\ell=1}^{L} f_{ik} d_{j\ell} z_{ijk\ell} \\
\text{subject to:} & \quad \sum_{j=1}^{L} x_{ij} = 1 \quad \forall i \\
\quad & \quad \sum_{i=1}^{S} x_{ij} = 1 \quad \forall j \\
\quad & \quad x_{ij} \geq z_{ijk\ell} \quad \text{and} \quad x_{k\ell} \geq z_{ijk\ell} \quad \forall i, j, k, \ell \\
\quad & \quad x_{ij} + x_{k\ell} \leq z_{ijk\ell} + 1 \quad \forall i, j, k, \ell \\
& \quad x_{ij}, z_{ijk\ell} \in \{0, 1\} \quad \forall i, j, k, \ell
\end{align*}
\]

- Julia code: QAP.ipynb
QAP example: circuit layout

- Place $n$ electronic modules in $n$ predetermined positions on a backplate.
- We are given a wiring specification that tells us how the various modules must be connected.
- Identical to the facility location problem:
  - $f_{ij}$ is the number of wires between module $i$ and module $j$
  - $d_{ij}$ distance between positions $i$ and $j$ on the backplate.
  - Minimize total length of wire used.
Special ordered sets

- Another type of constraint that is standard among many solvers is called the special ordered set (SOS).

- We saw one type of SOS constraint in the context of a variable belonging to a discrete set of values. There are other types of SOS constraints (we will see them next!)

- All SOS constraints can be implemented using logic tricks but some solvers also allow you to specify them explicitly. Only two reasons you would ever want to do this:
  - It makes your code run faster
  - It makes your code easier to understand

- Some solvers can automatically detect SOS constraints.
SOS1 (type 1) constraint

**SOS1 constraint:** The variables \( \{x_1, \ldots, x_m\} \) satisfy an SOS1 constraint if *at most one of them is nonzero.*

- An SOS1 constraint represents a **multiple-choice** notion.
- Standard use: representing a **discrete-valued function**
SOS1 (type 1) constraint

- Let \((x_i, a_i), i = 1, \ldots, m\) be the admissible \((x, y)\) pairs.
- Write: \(x = \sum_{i=1}^{m} x_i \lambda_i\) and \(y = \sum_{i=1}^{m} a_i \lambda_i\).
- Constraint: exactly one of the \(\lambda_i\) is equal to 1, the rest are equal to zero.
SOS1 constraint

Using SOS1 constraint

\[ y = \sum_{i=1}^{m} a_i \lambda_i \]

\[ \sum_{i=1}^{m} \lambda_i = 1, \quad \lambda_i \geq 0 \]

\{\lambda_1, \ldots, \lambda_m\} \text{ is SOS1}

Using algebraic formulation

\[ y = \sum_{i=1}^{m} a_i \lambda_i \]

\[ \sum_{i=1}^{m} \lambda_i = 1 \]

\[ \lambda_i \in \{0, 1\} \]

- The \( \lambda_i \) are real (not binary) in the SOS1 formulation
- Solver will still use binary variables internally, of course
- IJulia example: SOS.ipynb
SOS2 (type 2) constraint

**SOS2 constraint:** The variables \( \{x_1, \ldots, x_m\} \) satisfy an SOS2 constraint if *at most two of them are nonzero*. Also, nonzero elements must be consecutive.

- SOS2 constraints are typically used to represent *piecewise-linear functions*.
Let \((x_i, a_i), \ i = 1, \ldots, m\) be the transition points.

\[
x = \sum_{i=1}^{m} x_i \lambda_i \quad \text{and} \quad y = \sum_{i=1}^{m} a_i \lambda_i \quad \text{with} \quad \sum_{i=1}^{m} \lambda_i = 1.
\]

If \(x_i < x < x_{i+1}\) then \(\lambda_i + \lambda_{i+1} = 1\) and the other \(\lambda_j\) are zero. Then, \(x = \lambda_i x_i + \lambda_{i+1} x_{i+1}\) and \(y = \lambda_i a_i + \lambda_{i+1} a_{i+1}\)
How do we represent the constraint that at most two of the variables \( \{\lambda_1, \ldots, \lambda_m\} \) are nonzero, and nonzero variables must be consecutive?

- Let \( \{z_1, \ldots, z_{m-1}\} \) be binary with \( \sum_{i=1}^{m-1} z_i = 1 \).
- If \( z_i = 1 \), then \( \lambda_i \) and \( \lambda_{i+1} \) can be nonzero.
- In other words: if \( z_{i-1} \) and \( z_i \) are zero, then \( \lambda_i = 0 \).
**SOS2 constraint**

\[
y = \sum_{i=1}^{m} a_i \lambda_i
\]

\[
\sum_{i=1}^{m} \lambda_i = 1, \quad \lambda_i \geq 0
\]

\{\lambda_1, \ldots, \lambda_m\} \text{ is SOS2}

- Extra binary variables needed in algebraic formulation.
- Solver still uses binary variables for SOS2.

**Using algebraic formulation**

\[
y = \sum_{i=1}^{m} a_i \lambda_i
\]

\[
\sum_{i=1}^{m} \lambda_i = 1, \quad \lambda_i \geq 0
\]

\[
\lambda_1 \leq z_1
\]

\[
\lambda_i \leq z_{i-1} + z_i, \quad i = 2, \ldots, m - 1
\]

\[
\lambda_m \leq z_{m-1}
\]

\[
\sum_{i=1}^{m-1} z_i = 1, \quad z_i \in \{0, 1\}
\]
General piecewise linear functions

We may have:

- segments overlapping
- semi-infinite or infinite segments
- holes in the space of $x$ coordinates

For each segment, record:

- an endpoint $(x, f)$
- the $x$-length $\ell$
- the slope $g$

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$f$</th>
<th>$\ell$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>$-\infty$</td>
<td>$-1$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>2</td>
<td>$\infty$</td>
<td>1</td>
</tr>
</tbody>
</table>
General piecewise linear functions

- The segment $i$ has two variables associated with it:
  - $z_i$, a binary variable that selects whether segment $i$ is active or not. Note that the $\{z_1, \ldots, z_m\}$ are SOS1.
  - $\lambda_i$, a nonnegative real variable that chooses how far along segment $i$ we are situated. Note that $0 \leq \lambda_i \leq |\ell_i|$.

- The general point $(x, f)$ on the function is given by:
  - $x = \sum_{i=1}^{m} (z_i x_i + \text{sign}(\ell_i) \lambda_i)$,
  - $f = \sum_{i=1}^{m} (z_i f_i + \text{sign}(\ell_i) g_i \lambda_i)$

- We also require some constraints:
  - $z_i \in \{0, 1\}$ and $\sum_{i=1}^{m} z_i = 1$
  - $0 \leq \lambda_i \leq |\ell_i|$.
  - if $z_i = 0$ then $\lambda_i = 0$. (this is tricky!)
General piecewise linear functions

How do we impose the constraint that if \( z_i = 0 \) then \( \lambda_i = 0 \)?

- If \( \ell_i \) is finite, then we have the bound \( 0 \leq \lambda_i \leq |\ell_i| \) so we can use the standard fixed cost trick:

\[
\lambda_i \leq |\ell_i| z_i
\]

- If \( \ell_i = \pm \infty \) then this won’t work!

- For the infinite case, we can use the constraint:

\[
\{\lambda_i, 1 - z_i\} \text{ is SOS1}
\]

There is no upper bound for \( \lambda_i \) (and so we can’t model algebraically) but solvers can nonetheless implement this constraint efficiently.