Column Generation

The idea of the column generation can be motivated by the trim-loss problem: We receive an order to cut 150 pieces of 1.5-meter (pipe) segments, 250 pieces of 2-meter segments, and 200 pieces of 4-meter segments from some 7-meter steel pipes of a given diameter. Assume that the cutting operation does not consume any length of a steel pipe, and any cut pipes and segments left after satisfying the demands are regarded as loss. Determine how to cut the steel pipes so as to minimize the loss.

Let \( X \) be the total number of steel pipes used to cut the segments. Then the trim loss = \( 7X - (150)(1.5) - (250)(2) - (200)(4) = 7X - 1525 \). Thus, the minimal solution is given by minimizing \( X \), the number of steel pipes used.

There are different formulations. For ease of reference, let \((l, k, m)\) be a cut pattern that produce \( l \) pieces of 1.5-meter segments, \( k \) pieces of 2-meter segments, and \( m \) pieces of 4-meter segments.

Formulation 1. The order can be satisfied by 255 steel pipes: 200 steel pipes with the \((0, 1, 1)\) cut pattern, 38 steel pipes with the \((4, 0, 0)\) cut pattern, and 17 steel pieces with the \((0, 3, 0)\) cut pattern. Thus, the minimum number should be no more than 255. Let

\[
y_i = \begin{cases} 1, & \text{if the } i\text{th pipe is used}, \\ 0, & \text{o.w.} \end{cases} \quad i = 1, \ldots, 255;
\]

\[
x_{ij} = \text{the number of the } j\text{th type of segments cut from the } i\text{th pipe, where } j = 1 \text{ for the 1.5-meter segments, } j = 2 \text{ for the 2-meter segments, and } j = 3 \text{ for the 4-meter segments}; \text{ e.g., the cut pattern } x_i = (2, 2, 0) \text{ gives two 1.5-meter and two 2-meter segments.}
\]

Let \( M \) be a large positive number, and \( \mathbb{Z}^+ \) be the set of non-negative integers.

\[
\min \sum_{i=1}^{255} y_i,
\]

s.t.

\[
\sum_{i=1}^{255} x_{i1} \geq 150;
\]

\[
\sum_{i=1}^{255} x_{i2} \geq 250;
\]

\[
\sum_{i=1}^{255} x_{i3} \geq 200;
\]

\[
1.5x_{i1} + 2x_{i2} + 4x_{i3} \leq 7y_i, \quad i = 1, \ldots, 255;
\]

\[
x_{ij} \in \mathbb{Z}^+ \cup \{0\}, \quad y_i \in \{0, 1\} \quad \forall \ i, \ j.
\]
The formulation is a correct one (where 255 can be replaced by other upper bounds, possibly tighter, of the minimum number of steel pipes). However, the formulation is hard to solve in the sense that it contains many alternate minimum; e.g., for any minimum solution, one get an alternate minimum by swapping the values of \( x_{i_1}^*, x_{i_2}^*, x_{i_3}^* \) with \( x_{i_1}^*, x_{i_2}^*, x_{i_3}^* \) (i.e., by swapping the cut patterns of the \( i_1 \)th and the \( i_2 \)th steel pipe.) Finding the solution by branch and bound will face an enormous search tree.

**Formulation 2.** A more efficient formulation is to define variables on the cut patterns. Let \( x_i \) be the number of steel pipes cut in the \( i \)th pattern, where the cut patterns are given by the following table. Let \( a_{ij} \) be the number of \( j \) type segments produced by the \( i \)th cut pattern. Thus, the \( i \)th cut pattern is defined by \( (a_{i1}, a_{i2}, a_{i3})^T \). As an example, \( a_1 = (0, 0, 1) \). Further let \( t_i \) be the trim loss of the \( i \)th cut pattern; e.g., \( t_1 = 3 \).

<table>
<thead>
<tr>
<th>Cut Patterns</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
<th>( x_7 )</th>
<th>( x_8 )</th>
<th>( x_9 )</th>
<th>( x_{10} )</th>
<th>( x_{11} )</th>
<th>( x_{12} )</th>
<th>( x_{13} )</th>
<th>( x_{14} )</th>
<th>( x_{15} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5-meter segment</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2-meter segment</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4-meter segment</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trim loss (meter)</td>
<td>3</td>
<td>1.5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.5</td>
<td>3</td>
<td>2</td>
<td>3.5</td>
<td>5</td>
<td>1</td>
<td>2.5</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\min & \quad \sum_{i=1}^{15} t_i x_i, \\
\text{s.t.} & \quad \sum_{i=1}^{15} a_{i1} x_i = 150; \\
& \quad \sum_{i=1}^{15} a_{i2} x_i = 250; \\
& \quad \sum_{i=1}^{15} a_{i3} x_i = 200; \\
& \quad x_i \in \mathbb{Z}^+ \cup \{0\}, \quad \forall \ i.
\end{align*}
\]

Here is a question on Formulation 2: Is it necessary to include inefficient cut patterns such as \( x_1 \) in which a 3-meter segment is wasted?

**Example.** It is necessary to include the inefficient cut patterns. Otherwise, there may not be a feasible solution for the above formulation. Consider cutting 3-meter and 7-meter segments from 10-meter steel pipes. If we only consider the efficient segments, Formulation 2 gives no feasible solution for an order of 3 pieces of 3-meter segments and 1 piece of 7-meter segment.

Is there any way to skip the inefficient cut patterns? If so, we can reduce the number of variables.
**Formulation 3.** This formulation ignores inefficient cutting patterns.

<table>
<thead>
<tr>
<th>Cut Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>1.5-meter segment</td>
</tr>
<tr>
<td>2-meter segment</td>
</tr>
<tr>
<td>4-meter segment</td>
</tr>
<tr>
<td>trim loss (meter)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\min & \quad \sum_{i=1}^{5} x_i, \\
\text{s.t.} & \quad \sum_{i=1}^{5} a_{i1} x_i \geq 150; \\
& \quad \sum_{i=1}^{5} a_{i2} x_i \geq 250; \\
& \quad \sum_{i=1}^{5} a_{i3} x_i \geq 200; \\
& \quad x_i \in \mathbb{Z}^+ \cup \{0\} \quad \forall \ i.
\end{align*}
\]

Formulation 3 is the simplest one for the trim-loss problem. It is an integer program, which is hard to solve. The formulation has another drawback: the number of variables (i.e., cut patterns) increases rapidly with the length of steel pipes and the number of pipe segments. Try to, for example, find out the number of efficient cut patterns for steel pipes of 10 meters long, or for the 6-meter steel pipes with an additional demand of 0.8-meter pipe segments.

In the following we introduce the idea of column generation to tackle the explosive number of variables. It is basically an elegant revised simplex method that at each iteration identifies the best entering variable by solving an optimization problem. Note that in the introduction of the idea, we work with continuous rather than discrete variables; the techniques to handle integer variables is beyond the scope of this course.

Observe that the formulation is in the form of:

\[
\begin{align*}
\min & \quad \sum_i x_i, \\
\text{s.t.} & \quad \sum_i a_{ij} x_i \geq b_j, j = 1, 2, 3, \\
& \quad x_i \geq 0, \quad i = 1, 2, 3. 
\end{align*}
\]

(Recall we ignore the integral constraints.)

The reduced cost of the $i$th variable is given by $c_i = c_i - \mathbf{c}_b^T \mathbf{B}^{-1} \mathbf{a}_i = c_i - \mathbf{\pi}^T \mathbf{a}_i$, where $\mathbf{\pi}^T$ is the vector of dual variables. Note that all $c_i = 1$. Thus, the most negative reduced cost is given by $\min_i 1 - \mathbf{\pi}^T \mathbf{a}_i$, where $\mathbf{a}_i = (a_{ij})$ be a desirable cut pattern. Let $l_j$ be the length of the $j$th type segment required; $L$ be the length of a steel pipe. Then the most negative reduced cost can be found by solving:

**Subproblem**

\[
\begin{align*}
\max & \quad \sum_j \pi_j a_j, \\
\text{s.t.} & \quad \sum_j l_j a_j \leq L;
\end{align*}
\]
This is the standard knapsack problem, which is NP-hard. Fortunately, for a given L the problem can be solved by algorithms polynomial in L; e.g., dynamic programming. From this knapsack problem, we generate a new column to enter the problem. The problem with the columns selected so far is called the master problem.

**Generic Column Generation Algorithm** (for the trim loss problem):

1° Select columns for the initial basis.
2° Solve the master problem (by revised simplex) and find the dual variables $\pi$.
3° Solve the knapsack subproblem $\max z_{\text{sub}} = \sum_j \pi_j a_j$ s.t. $\sum_j l_j a_j \leq L$. Stop if the maximal value $\leq 1$; else go to 2°.

To use column generation, we may start with inefficient cut patterns. For example, consider $a_1 = (1, 0, 0)^T$, $a_2 = (0, 1, 0)^T$, and $a_3 = (0, 0, 1)^T$.

**Iteration 1. Master Program**

$$\begin{align*}
\min & \quad x_1 + x_2 + x_3, \\
\text{s.t.} & \quad x_1 \geq 150; \\
& \quad x_2 \geq 250; \\
& \quad x_3 \geq 200; \\
& \quad x_1, x_2, x_3 \geq 0.
\end{align*}$$

Obviously, $x_B = (x_1, x_2, x_3)^T$; the minimum solution is $x_1^* = 150$, $x_2^* = 250$, and $x_3^* = 200$, with $z^* = 600$; $B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = B^{-1}$. With $c_B^T = (1, 1, 1)$, $c_B^T B^{-1} = (1, 1, 1)$.

**Sub-problem:**

$$\begin{align*}
\max & \quad a_1 + a_2 + a_3, \\
\text{s.t.} & \quad 1.5a_1 + 2a_2 + 4a_3 \leq 7, \\
& \quad a_1, a_2, a_3 \in \mathbb{Z}^+ \cup \{0\}.
\end{align*}$$

The maximum of this knapsack problem, $(a_1, a_2, a_3)^* = (3, 1, 0)$, with $z^*_{\text{sub}} = 4 > 1$. Thus, it is good to include the cut pattern $a_4 = (3, 1, 0)^T$ into the master program.

**Iteration 2. Master Program**

$$\begin{align*}
\min & \quad x_1 + x_2 + x_3 + x_4, \\
\text{s.t.} & \quad x_1 + 3x_4 \geq 150; \\
& \quad x_2 + x_4 \geq 250; \\
& \quad x_3 \geq 200; \\
& \quad x_1, x_2, x_3, x_4 \geq 0.
\end{align*}$$
Clearly $x_3$ remains in the optimal basis. Similarly, by comparing $x_1$ and $x_2$, $x_1$ should be the leaving variable. $x_B = (x_2, x_3, x_4)^T$; The minimum solution is $x_2^* = 200$, $x_3^* = 200$, and $x_4^* = 50$, with $z^* = 450$; $B = \begin{pmatrix} 0 & 0 & 3 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ and $B^{-1} = \begin{pmatrix} -1/3 & 1 & 0 \\ 0 & 0 & 1 \\ 1/3 & 0 & 0 \end{pmatrix}$. $c_B^T B^{-1} = (0, 1, 1)$.

Remark. The new basis inverse can be found from $(B_{\text{new}})^{-1} = E (B_{\text{old}})^{-1}$ with the basic variables arranged in the order of the equations as in the revised simplex method. Here, to avoid being sidetracked by such procedure, we simply arrange basic variables in increasing indices and assume that $B^{-1}$ can be found as $B$ is known.

**Sub-problem:**

$$\max a_2 + a_3,$$

$s.t.\ 1.5a_1 + 2a_2 + 4a_3 \leq 7,$

$a_1, a_2, a_3 \in \mathbb{Z}^+ \cup \{0\}$.

The maximum of this knapsack problem, $(a_1, a_2, a_3)^* = (0, 3, 0)$, with $z_{\text{sub}}^* = 3 > 1$. Let us include $a_5 = (0, 3, 0)^T$ into the problem.

**Iteration 3. Master Program**

$$\min x_1 + x_2 + x_3 + x_4 + x_5,$$

$s.t.\ x_1 + 3x_4 \geq 150;$

$x_2 + x_4 + 3x_5 \geq 250;$

$x_3 \geq 200;$

$x_1, x_2, x_3, x_4, x_5 \geq 0.$

$x_3$ remains in the basis. The entering of $x_5$ replaces $x_2$ (agree?). $x_B = (x_3, x_4, x_5)^T$; the minimum solution is $x_3^* = 200$, $x_4^* = 50$, and $x_5^* = \frac{200}{3}$, with $z^* = 316 \frac{2}{3}$; $B = \begin{pmatrix} 0 & 3 & 0 \\ 0 & 1 & 3 \\ 1 & 0 & 0 \end{pmatrix}$ and $B^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1/3 & 0 & 0 \\ -1/9 & 1/3 & 0 \end{pmatrix}$. $c_B^T B^{-1} = (\frac{2}{9}, \frac{1}{3}, 1)$.

**Sub-problem:**

$$\max \frac{2}{9}a_1 + \frac{1}{3}a_2 + a_3,$$

$s.t.\ 1.5a_1 + 2a_2 + 4a_3 \leq 7,$

$a_1, a_2, a_3 \in \mathbb{Z}^+ \cup \{0\}$.

The maximum of this knapsack problem, $(a_1, a_2, a_3)^* = (2, 0, 1)$, with $z_{\text{sub}}^* = \frac{13}{9} > 1$. Let us include $a_6 = (2, 0, 1)^T$ into the problem.
Iteration 4. Master Program

\[
\min \quad x_1 + x_2 + x_3 + x_4 + x_5 + x_6,
\]
\[
s.t. \quad \begin{align*}
  & x_1 + 3x_4 + 2x_6 \geq 150; \\
  & x_2 + x_4 + 3x_5 \geq 250; \\
  & x_3 + x_6 \geq 200;
\end{align*}
\]
\[
x_1, x_2, x_3, x_4, x_5, x_6 \geq 0.
\]

\[\mathbf{x}_B = (x_3, x_5, x_6)^T; \text{ the minimum solution is } x_3^* = 125, x_5^* = \frac{250}{3}, \text{ and } x_6^* = 75, \text{ with } z^* = 283\frac{1}{3};\]

\[
\mathbf{B} = \begin{pmatrix} 0 & 0 & 2 \\ 0 & 3 & 0 \\ 1 & 0 & 1 \end{pmatrix} \text{ and } \mathbf{B}^{-1} = \begin{pmatrix} -1/2 & 0 & 1 \\ 0 & 1/3 & 0 \\ 1/2 & 0 & 0 \end{pmatrix}, \quad \mathbf{c}_B^T \mathbf{B}^{-1} = \left(0, \frac{1}{3}, 1\right).\]

Sub-problem:

\[
\text{Max } \frac{1}{3} a_2 + a_3,
\]
\[
s.t. \quad 1.5a_1 + 2a_2 + 4a_3 \leq 7,
\]
\[
a_1, a_2, a_3 \in \mathbb{Z}^+ \cup \{0\}.
\]

The maximum of this knapsack problem, \((a_1, a_2, a_3)^* = (0, 1, 1)\), with \(z_{\text{sub}}^* = \frac{4}{3} > 1\). Let us include \(a_7 = (0, 1, 1)^T\) into the problem.

Iteration 5. Master Program

\[
\min \quad x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7,
\]
\[
s.t. \quad \begin{align*}
  & x_1 + 3x_4 + 2x_6 \geq 150; \\
  & x_2 + x_4 + 3x_5 + x_7 \geq 250; \\
  & x_3 + x_6 + x_7 \geq 200;
\end{align*}
\]
\[
x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0.
\]

\[\mathbf{x}_B = (x_5, x_6, x_7)^T; \text{ the minimum solution is } x_5^* = \frac{125}{3}, \text{ } x_6^* = 75, \text{ and } x_7^* = 125, \text{ with } z^* = 241\frac{2}{3};\]

\[
\mathbf{B} = \begin{pmatrix} 0 & 2 & 0 \\ 3 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \text{ and } \mathbf{B}^{-1} = \begin{pmatrix} 1/6 & 1/3 & -1/3 \\ 1/2 & 0 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}, \quad \mathbf{c}_B^T \mathbf{B}^{-1} = \left(\frac{1}{6}, \frac{1}{3}, \frac{2}{3}\right).\]

Sub-problem:

\[
\text{Max } \frac{1}{6} a_1 + \frac{1}{3} a_2 + \frac{2}{3} a_3,
\]
\[
s.t. \quad 1.5a_1 + 2a_2 + 4a_3 \leq 7,
\]
\[
a_1, a_2, a_3 \in \mathbb{Z}^+ \cup \{0\}.
\]

- 6 -
The maximum of this knapsack problem cannot be larger than 1 (why?), and the minimum of the master problem has been reached. The optimal cutting of the pipes are: cut pattern (0, 3, 0): \( \frac{125}{3} \) pieces; cut pattern (2, 0, 1): 75 pieces; cut pattern (0, 1, 1): 125 pieces.

2. A shortest-Path Sub-problem in Column Generation

In solving network-based optimization problems by column generation, we frequently encounter a shortest-path sub-problem in looking for new columns. In this section we give an example to provide the intuition. Our example is from [1], which is in fact originated from our textbook.

Figure 1 shows a network \( G \) where the pair \((c_{ij}, t_{ij})\) beside arc \((i, j)\) is the cost spent and time taken in passing through the arc. The objective is to find a shortest path such that the total time should be no longer than 14 units. Let \( A \) be the set of arcs in \( G \); \( x_{ij} = 1 \) if the arc is taken, and \( x_{ij} = 0 \) otherwise; \((i, j) \in A\).

![Diagram of network](image)

**Figure 1. Time-Constrained Network**

**Problem P1.**

\[
\begin{align*}
\text{min} & \quad \sum_{(i,j) \in A} c_{ij}x_{ij}, \quad (1) \\
\text{s.t.} & \quad \sum_{(i,j) \in A} x_{ij} = 1, \quad (2) \\
& \quad \sum_{(j,(i,j) \in A)} x_{ij} - \sum_{(j,(i,j) \in A)} x_{ji} = 0, \quad (3) \\
& \quad \sum_{(i,j) \in A} x_{ij} = 1, \quad (4) \\
& \quad \sum_{(i,j) \in A} t_{ij}x_{ij} \leq 14, \quad (5) \\
& \quad x_{ij} \in \{0, 1\}, \quad (i,j) \in A. \quad (6)
\end{align*}
\]

This is a constrained shortest-path problem, which is \( NP \)-hard. We will solve this problem by column generation. In doing so, we will use the path formulation of the problem.

**Table 1. The set of Paths in the Problem**

<table>
<thead>
<tr>
<th>path</th>
<th>cost</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2-4-6</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>1-2-4-5-6</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>1-2-5-6</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>
There are 9 paths from node 1 to node 6. Only three of them are feasible and the cheapest one is path 1-3-2-4-6. For general problems, there are so many paths that it is impossible to list out all the paths. We only use this small example to illustrate the idea of running into a short-path subproblem in column generation.

Observe that \( P1 \) is easy to solve if there is no constraint (5). Moreover, the optimal solution of (1)-(4) and (6) is a path from node 1 to node 6. We will manipulate this fact. To do so, we first introduce the path formulation of the problem.

Let \( a_{ij} = \begin{cases} 1, & \text{if arc } (i, j) \text{ is in path } p, \\ 0, & \text{a.w.} \end{cases} \)

For example, for path 1-2-4-6 to be the first path, \( a_{112} = 1, a_{124} = 1, a_{146} = 1, \) and other \( a_{ij} = 0. \) Note that \( a_{ij} \) are parameters, not variables.

Define variable \( \lambda_p \) to be the “probability” that path \( p \) is taken. Let \( P \) be the set of all paths.

\[
x_{ij} = \sum_{p \in P} a_{ij} \lambda_p, \quad (i, j) \in A, \tag{7}
\]

\[
\sum_{p \in P} \lambda_p = 1, \tag{8}
\]

\[
\lambda_p \geq 0, \quad p \in P. \tag{9}
\]

We can substitute \( x_{ij} \) by \( \lambda_p \) and \( a_{ij} \) to form a new optimization problem.

\[
\sum_{(i, j) \in A} c_{ij} x_{ij} = \sum_{(i, j) \in A} c_{ij} \sum_{p \in P} a_{ij} \lambda_p = \sum_{p \in P} \left( \sum_{(i, j) \in A} c_{ij} a_{ij} \right) \lambda_p;
\]

\[
\sum_{(i, j) \in A} t_{ij} x_{ij} = \sum_{(i, j) \in A} t_{ij} \sum_{p \in P} a_{ij} \lambda_p = \sum_{p \in P} \left( \sum_{(i, j) \in A} t_{ij} a_{ij} \right) \lambda_p.
\]

**Problem P2.**

\[
\begin{align*}
\min \quad & \sum_{(i, j) \in A} c_{ij} x_{ij} = \sum_{(i, j) \in A} c_{ij} \sum_{p \in P} a_{ij} \lambda_p = \sum_{p \in P} \left( \sum_{(i, j) \in A} c_{ij} a_{ij} \right) \lambda_p, \\
\text{s.t.} \quad & \sum_{p \in P} \left( \sum_{(i, j) \in A} t_{ij} a_{ij} \right) \lambda_p \leq 14, \\
& \lambda_p \geq 0, \quad p \in P, \\
& \lambda_p \geq 0, \quad p \in P, \\
& x_{ij} = \sum_{p \in P} a_{ij} \lambda_p, \quad (i, j) \in A, \\
& x_{ij} \in \{0, 1\}, \quad (i, j) \in A.
\end{align*}
\]
We solve $P_2$ to find $\lambda_p$, where the paths $p$ are found from the optimal problem with (1)-(4) and (6), and we need to ensure that $x_{ij}$ being binary.

$P_2$ is a hard integer program. To solve the problem, our first step is to relax the integer constraints (15). Note that we can also drop (14), because the relationship has been used to transform decision from the arc variables $x_{ij}$ to the path variables $\lambda_p$. The resulted optimization problem (10)-(13) is called the master problem.

Here the main idea of column generation: We start with a restricted master problem with a small set of paths and incorporate a path in the set for consideration only if it is beneficial to do so. Let $\pi_1$ and $\pi_0$ be the dual variable of constraint (11) and (12), respectively. Our steps are:

(a). Solve the restricted master problem with the current set of paths to obtain $\pi_1$, $\pi_0$, and the minimum value.
(b). Search for another path that reduces objective function value.

In step (b), we look for variable $\lambda_p$ with negative reduced cost, i.e.,
\[
\bar{\tau}_p = \sum_{(i,j) \in A} c_{ij} a_{pij} - \pi_1 \left( \sum_{(i,j) \in A} t_{ij} a_{pij} \right) - \pi_0 < 0.
\]

To do so, we solve the following shortest-path problem $P_3$:

\[
\begin{align*}
\min & \quad \sum_{(i,j) \in A} \left( c_{ij} - \pi_1 t_{ij} \right) x_{ij} - \pi_0, \\
\text{s.t.} & \quad (2)-(4), (6).
\end{align*}
\] (16)

If the minimum value of $P_3$ is non-negative, the current set of paths is already optimal; otherwise, the new path is added into $P_2$ to solve for $\pi_1$, $\pi_0$, and minimum value.

Remarks.

(1). The procedure discussed only solves the LP relaxation but not the original $P_1$. To really solve the problem, we need to adopt branch and bound search in case there is any fractional $x_{ij}$.

(2). Note the tricks of our solution method. There is no simple efficient algorithm for an NP-problem. Thus, we consider the LP relaxation of the problem, which gives a network problem plus some hard constraints. Then we use the path-based formulation to express the problem in terms of the path variables. This path-based optimization problem is solved by column generation. The upper level is a master problem that contains paths considered up to the current iteration, and the lower-level sub-problem is to select a new path for the next iteration of the master problem. As usual, the objective function of the sub-problem is the reduced cost of the master program, which requires the dual variables of the master problem. Since the constraints of the sub-problem usually form a network optimization problem, the minimal reduced cost is often found from a shortest-path problem.
Reference