1 Other Algorithms of Linear Programming

1.1 Dual Simplex

The dual simplex method makes use of the duality between the primal problem and its dual in order to find an optimal solution. This method works well for sensitivity analysis because it typically takes fewer iterations than the simplex method while going through a very similar process. The dual simplex method is made up of three main steps:

1. Initialize
   
   (a) Confirm that all constraints are in \( \leq \) form.
   
   (b) Find a basic solution such that the basic variable coefficients are zero.

2. Feasibility Test
   
   (a) Check if all basic variable coefficients are > zero.
   
   (b) If so, the solution is optimal.

3. Iteration
   
   (a) *Leaving basic variable*: Select the most negative basic variable from Eq. 0.
   
   (b) *Entering basic variable*: Select the non-basic variable in Eq. 0 whose coefficient reaches zero first by adding a multiple of the equations with the basic leaving variable.
   
   (c) *New basic solution*: Solve for basic variables in terms of non-basic variables by Gaussian Elimination.
   
   (d) Return to Feasibility Test.
1.2 Parametric Linear Programming

Parametric linear programming is used in sensitivity analysis and can handle both changes in the objective function and the functional constraints. Changes in the objective of the form:

\[
Z = \sum_{j=1}^{n} c_j x_j \rightarrow Z(\theta) = \sum_{j=1}^{n} (c_j \alpha_j \theta) x_j,
\]

Changes in the functional constraints of the form:

\[
b_i \rightarrow b_i + \alpha_i \theta,
\]

The \( \alpha \) factor allows for different growth rates with respect to \( \theta \). Regardless of what is changing, the first two steps are the same, while steps 3 and 4 depend on what is being changed:

1. Solve problem using \( \theta = 0 \) with the Simplex method.
2. Add \( \alpha_j \theta \) or \( \alpha_i \theta \) to the problem where a change is desired (as shown above).
3. Increase \( \theta \) and one of the following:
   - If changing the objective, stop increasing \( \theta \) once a non-basic variable becomes negative, or once \( \theta \) has been increased as much as desired.
   - If changing the functional constraints, stop increasing \( \theta \) once a basic variable has its RHS value go negative, or once \( \theta \) has been increased as much as desired.
4. Perform an iteration of the Simplex method with one of the following in mind:
   - If changing the objective, use the non-basic variable that became negative in step 3 as the entering basic variable.
   - If changing the functional constraints, use the basic variable whose RHS value went negative in step 3 as the leaving basic variable.

1.3 Upper Bound Technique

Often, an upper bound is put on the solution to a problem,

\[
x_j \leq u_j.
\]

The simplex method’s computational complexity is affected by the number of functional constraints, while the number of nonnegativity constraints has little effect on the run time. The upper bound technique uses this fact and deals with the upper bound constraints separately, like the nonnegativity constraints. The upper bound technique works like the Simplex method but turns the upper bound into a nonnegativity bound if \( x_j \) gets past the bound \( u_j \). Two basic rules when using the upper bound technique:
1. If \( x_j = 0 \), use \( x_j \), where \( 0 \leq x_j \leq u_j \). Start with this one.

2. If \( x_j = u_j \), replace \( x_j \) with \( u_j - y_j \), where \( 0 \leq y_j \leq u_j \).

Anywhere an upper bound \( u_j \) could be a problem, it is switched into a nonnegativity constraint.

### 1.4 Interior Point Methods

Rather than move along the edge of a feasible region in order to find an optimal solution like is done with Simplex, interior point methods shoot straight through the feasible region toward the optimal solution. The technique makes use of gradient ascent to move in the direction which improves the objective as fast as possible. After a trial feasible point is found, the feasible region is moved such that the trial point is in the center of the feasible region. This lends to giving the greatest improvement in the objective on the next gradient projection. This cycle continues until an optimal solution is found.

### 2 Transportation and Network Problems

There is a special subset of linear programming problems that involve moving goods from one location to another called transportation problems. A different but very similar problem is the assignment problem, which deals with assigning people to particular tasks. Both these problems are a special case of the minimum cost flow problem.

The typically large number of constraints contained in these kinds of problems eliminates the practicality of using the Simplex method to solve them, so other methods must be used. This section will involve setting up the problem to use these methods.

The format of the transportation, and similar problems, is as follows: there is a given commodity \( x \) with some source location \( i \) that must meet a demand at a destination location \( j \). The cost to take \( x \) from \( i \rightarrow j \) is denoted as \( x_{ij} \). Common representations of this problem are in a table or a network diagram, see figures 1 and 2. One assumption placed on the requirements is the source has a fixed supply and the destination has a fixed demand. In order for a feasible solution to exist, the sum of all the sources’ supply must equal the sum of all the destinations’ demand,

\[
\sum_{i=1}^{m} s_i = \sum_{j=1}^{n} d_j.
\]

where \( s_i \) denotes the supply at a given source \( i \) and \( d_j \) denotes the demand at a destination \( j \). An assumption placed on the cost of moving a commodity from \( i \rightarrow j \) is that the cost must be proportional to the number of units being shipped. A problem can be classified as this kind of problem if the requirement and cost assumptions hold and the parameters can be put in a table as shown in figure 2.

When the supply and demand constraints are in matrix form, like that of the \( A \) matrix of previous chapters, it will take the form shown in figure 3. Note, all figures taken from [1].
Figure 1: Network representation of transportation problem.

<table>
<thead>
<tr>
<th>Cost per Unit Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Destination</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Source</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$m$</td>
</tr>
<tr>
<td><strong>Demand</strong></td>
</tr>
<tr>
<td>$d_1$</td>
</tr>
<tr>
<td>$d_2$</td>
</tr>
<tr>
<td>$d_3$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$d_n$</td>
</tr>
</tbody>
</table>

Figure 2: Parameter table for transportation.
Figure 3: Constraint coefficients for the transportation problem.

References