1 Other Algorithms of Linear Programming

1.1 Reminders from Duality

Proof of bounding condition of primal and dual, that is $c^T x \leq b^T y$

\[
\begin{align*}
Ax & \leq b \\
x & \geq 0 \\
A^T y & \geq c \\
y & \geq 0 \\
[Ax]_i & \leq b_i \text{ for all } i \\
[A^T y]_j & \geq c_j \text{ for all } j \\
x_j & \geq 0 \\
y_i & \geq 0 \\
(b_i - Ax_i)y_i & \geq 0 \\
(A^T y_j - c_j)x_j & \geq 0 \\
(b - Ax)^T y & \geq 0 \\
(A^T y - c)x & \geq 0 \\
b^T y - y^T Ax & \geq 0 \\
y^T Ax - c^T x & \geq 0 \\
b^T y & \geq y^T Ax \geq c^T x
\end{align*}
\]

1.2 Dual Simplex

The dual simplex method makes use of the duality between the primal problem and it’s 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.

1.3 **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 it’s 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.

1.4 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.

It offers an LP that include this upper bound idea, namely,

$$0 \leq x_j \leq u_j.$$  

For example, we could add this as a line to the A matrix, e.g.,

$$x_1 + 2x_2 \leq 5$$  

$$x_1 - 3x_2 \leq 6$$  

$$x_1 \leq 10(U_1)$$  

$$x_2 \leq 20(U_2)$$

Using the simplex method, the number of constraints makes it harder to solve for this case. We note that $0 \leq x_1$ and $x_1 \leq U_1$ can not both be the binding condition. Therefore, we can use another variable,

$$y_j = U_j - x_j.$$
That is, we either worry about \( y_j \geq 0 \) or \( x_j \geq 0 \). Hence, we begin with \( x_j \geq 0 \) (constraint).
Whenever \( x_j = 0 \) we use \( x_j \), whenever \( x_j = U_j \) we use \( y_j \) instead. Hence we are swapping between \( x_j \) and \( y_j \) whenever the bound is reached. i.e.

\[
0 \leq x_1 \leq 5 \\
u_1 = 5 - x_1 \\
0 \leq u_1 \leq 5
\]

This rule is followed when deciding on the leaning basic variable. That is, increasing the entering (none basic) variable until either,

- \( x_j \searrow 0 \)
- \( x_j \nearrow U_j \)

That is we are maximizing

\[
Z = 2x_1 + x_2 + 2x_3.
\]

Hence,

\[
4x_1 + x_2 = 12 \\
-2x_1 + x_3 = 4. \\
0 \leq x_1 \leq 4 \\
0 \leq x_2 \leq 15 \\
0 \leq x_3
\]

We then rearrange the objective function in terms of the non basic variables, i.e.,

\[
Z - 2x_1 = 20(0) \\
4x_1 + x_2 = 12(1) \\
-2x_1 + x_3 = 4.(2)
\]

We notice that \( Z \) increases as \( x_1 \) increases. While \( x_1 \) is the entering variable.

- (0) increase \( x_1 \) until \( x_1 = 4 \)
- (1) increase until \( x_2 = 0 \)
- We can increase \( x_1 \) until \( x_2 \) hits upper bound, that is \( x_1 = 1 \)

Hence we first increase \( x_1 \) to 1, then \( x_3 \) becomes the upper bound. And we change

\[
y_3 = U_3 - x_3
\]
Now the new set of equations is now

\[
\begin{align*}
Z + y_3 &= 22 \\
x_2 - 2y_3 &= 8 \\
x_1 + \frac{1}{2}y_3 &= 1
\end{align*}
\]

Alternatively, we have maximizing \(2x_1 + x_2 + 2x_3\), s.t.,

\[
\begin{align*}
4x_1 + x_2 &= 12 \\
-2x_1 + x_3 &= 4 \\
x_1 + x_4 &= 4 \\
x_2 + x_5 &= 12 \\
x_3 + x_6 &= 6
\end{align*}
\]

and \(x_i \geq 0\).

### 1.5 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. This provides an improvement over the worst case performance of the simplex. 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. The idea is to first compute an optimizing search direction based on a first order term (predictor). The step size that can be taken in this direction is used to evaluate how much centrality correction is needed. Second, a corrector term is computed: this contains both a centrality term and a second order term. Therefore, the search direction is the sum of the predictor direction and the corrector direction.

A type of this algorithm is the barrier method. It only searches inside the feasibility region. A large amount of computation is required per step, but perhaps only a very small number of steps is required. This also leads the first provably linear algorithm for interior points. The steps of the algorithms are as follows,

- Find the initial solution
- Move in the direction that increase the objective function at greatest rate
- Transform the feasible region to keep the point near the center of the feasible region
Here is an example, we are maximizing \( x_1 + 2x_2 + 2 \), such that, \( x_1 + x_2 \leq 8, x_1, x_2 \geq 0 \). We start at \( x(2, 2) \), we compute the increase with the fastest ratio, that is the gradient of the objective function, e.g., \((1, 2)\). The question is now how far should we go?

\[(2, 2) + (1, 1) = (3, 4)\]  

(1)

The constraint is now \( x_1 + x_2 + x_3 = 8 \) with \( x_3 \geq 0 \). We check that,

\[(2, 2, 4) + (1, 1, 0) = (3, 4, 4)\]  

(2)

Is in the feasible region. Hence the solution works out how far we can move or shift back from outside the feasible region. That is,

\[(3, 4, 4) + (1, 1, 1) = (2, 3, 3)\]  

(3)

2 Transportation and Network Problems

There is a special subset of linear programming problems that involve moving goods from on 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.

A network flow is an assignment of values to edges of a weighted directed graph (called a flow network in this case), such that:

- The value along an edge never exceeds the weight of that edge (here known as the capacity).

- The total incoming flow of some node (i.e. the sum of the values of all edges coming into the node) exactly equal the total outgoing flow, with the exception of two special nodes s and t. The node s can have any outgoing flow but no incoming flow, while the node t can have any incoming flow, and no outgoing flow.

In such a situation, s is considered a source and t is considered a sink.

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. 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.

### 2.1 Minimum cost network flow

In the minimum cost network flow, we are minimizing

\[
Z = c^T x
\]

With,

\[
Ax = b \\
L \leq x \leq u
\]

In the figure shown., the x are labeled with \( x_{ij} \) representing the flow \( i \rightarrow j \). The idea is that we find the minimum total cost given all the demands.

- \( x_{ij} = \) flow of \( i \rightarrow j \).
- \( b_i = (\text{flow out node } i) - (\text{flow into node } i) \)
- \( C_{ij} = \) cost per unit to send the flow \( i \)
- Supply=demand.

\[
\sum_{i\mid b_i > 0} b_i = - \sum_{i\mid b_i < 0} b_i
\]
If this doesn’t hold, we can add dummies variables, eg., minimizing \(12x_{12} + 15x_{13} + 7x_{34} + \ldots\), w.r.t.,
\[
\begin{align*}
x_{13} - x_{34} - x_{35} &= 0 \quad (4) \\
x_{12} + x_{13} &= 50 \quad (5)
\end{align*}
\]

2.2 Transportation problem
This algorithm minimizes the cost of transporting goods from \(m\) origins to \(n\) destinations along \(m \times n\) direct routes from origin to destination. The algorithm proceeds with every node is a source or sink, i.e.,
\[
\begin{align*}
\sum_j x_{ij} &= b_i \quad (6) \\
\sum_i x_{ij} &= b_j \quad (7)
\end{align*}
\]

2.3 Assignment
There are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the assignment. It is required to perform all tasks by assigning exactly one agent to each task in such a way that the total cost of the assignment is minimized. In this case, the numbers of agents and tasks are equal and the total cost of the assignment for all tasks is equal to the sum of the costs for each agent.
- Same number of source and sink
- Different cost per sink.
- Constraints ensure 1 link per node
- Total flows into jobs = 1
- Total flows into agents = 1

2.4 Shortest Path Problem
Shortest path problem is the problem of finding a path between two vertices such that the sum of the weights of its constituent edges is minimized. We can easily formulate it as:
- In the shortest path problem we find the path from one source to one sink.
- \(C_{ij}\) is the length of the path.
- Each flow is '1' unit from the source to the sink.
2.5 Maximum Flow Problem

Intuitively, it is the net total amount of stuff we are pumping from $s$ to $t$. We are interested in flows with maximal amount. That is what is the maximum total we can send from the source to the destination? We can formulate it as an LP as following.

maximizing $z = x_{m1}$, st.

$$\sum_i x_{ij} - \sum_k x_{ik} = 0 \quad \text{(8)}$$

$$0 \leq x_{ij} \leq U_{ij} \quad \text{(9)}$$

Now let us consider the "Dual" of the max flow problem, that is:

minimizing $\sum_{ij} U_{ij} V_{ij}$ such that

$$y_m - y_1 = 1 \quad \text{(10)}$$

$$y_i - y_j + V_{ij} \geq 0 \quad \text{(11)}$$

$$V_{ij} \geq 0 \quad \text{(12)}$$

• Magically the dual corresponds to the minimum cut problem!!! **Primal = Dual**

• A cut $C$ on $G$ between source and sink is a set of edges such that every directed path from source to sink passes through at least one edge in $C$.

• The capacity of the cut $C$ is the sum of all its edge capacities.

• Using Primal-Dual we have that **The maximal amount of a flow is equal to the capacity of a minimal cut.**
Let us interpret the dual Values:

\begin{align*}
    y_i &= 0 \text{ if node } i \text{ in } N_1 \quad (13) \\
    y_i &= 1 \text{ if node } i \text{ in } N_2 \quad (14)
\end{align*}

and

\begin{align*}
    V_{ij} &= 1 \text{ if } (i,j) \text{ connects } N_1 \text{ and } N_2 \quad (15) \\
    V_{ij} &= 0 \text{ otherwise} \quad (16)
\end{align*}

Where we have \( N_1 \) containing the source, and \( N_2 \) containing the sink.
Let us look at the matrix \( A \), which is all in 0, -1, 1.

\[
A = \begin{pmatrix}
    1 & -1 & 0 & 0 & 0 & 1 \\
    0 & -1 & 1 & 1 & 1 & 0 \\
    0 & 0 & 0 & 1 & 0
\end{pmatrix}
\]

If \( A \) is like that then we have integer parameters in \( b \), and \( C \) as well. Implying that we have an integer set of solutions to the problem.

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 5. Note, all figures taken from [?].
Figure 3: Network representation of transportation problem.
Figure 4: Parameter table for transportation.

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Cost per Unit Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$c_{11}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$c_{12}$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>$c_{1n}$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$c_{21}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$c_{22}$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>$c_{2n}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>$c_{m1}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$c_{m2}$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>$c_{mn}$</td>
</tr>
<tr>
<td></td>
<td>Demand</td>
<td>$d_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$d_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$d_m$</td>
</tr>
</tbody>
</table>
Figure 5: Constraint coefficients for the transportation problem.