Week 8

Chinese Postman Problem, Weighted Bipartite Matching, Transversal, Assignment Problem, Hungarian Algorithm*

8.1 Chinese Postman Problem

Chinese Postman Problem is discovered by the Chinese mathematician, Kwan Mei-Ko, in early 1960's. It is the problem that the Chinese Postman faces: he wishes to travel along every road in a city in order to deliver letters, with the least possible distance.

The problem is how to find a shortest closed walk of the graph in which each edge is traversed at least once, rather than exactly once. In graph theory, an **Euler circuit** in a connected, weighted graph is called the **Chinese Postman problem**.

Solution: An Euler circuit is possible if and only if every vertex is of even degree. If there exist some vertices of odd degrees, then we can construct a path between them (by pairing odd degree vertices) using existing edges in the graph.
If we can find the shortest path between odd vertices (paired) , we will have the edges that we will want to retrace in our tour, because we want the path that makes us cover edges more than once to be as short as possible.
After achieving the **Euler Circuit**, we will calculate the total weight of the tour.
This completes the solution of Chinese Postman Problem.

Algorithm

Algorithm to find shortest closed path or optimal Chinese postman route in a weighted graph that may not be Eulerian:

*Lecturer: Dr Anand Mishra. Scribe: Vandita Agarwal.

Step 1: If graph is Eulerian, return sum of all edge weights.Else do following steps.

Step 2: We find all the vertices with odd degree.

Step 3: List all possible pairings of odd vertices. For n odd vertices total number of pairings possible are, $(n-1)*(n-3)*(n-5)...*1$.

Step 4: For each set of pairings, find the shortest path connecting them.

Step 5: Find the pairing with minimum shortest path connecting pairs.

Step 6: Modify the graph by adding all the edges that have been found in step 5.

Step 7: Weight of Chinese Postman Tour is sum of all edges in the modified graph.

Step 8: Print Euler Circuit of the modified graph. This Euler Circuit is Chinese Postman Tour.

Let's take an example to show chinese postman problem.

Example 8.1. Consider the graph given below, we will find the optimal chinese postman tour for the following graph:

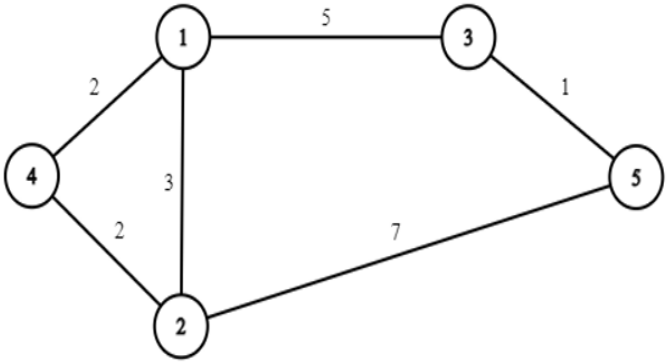

Figure 8.1

As we see above graph does not contain Eulerian circuit because it has odd degree vertices { 1, 2 }.
First we make all possible pairs of odd degree vertices. Since there are only two vertices of odd degrees, so there will be only one pair, which is { 12 }.
So pair with min sum of weight is given by red color edge in the modified graph 8.2. i.e., 12 = (3 units).
We add edge {12} to the original graph and create a modified graph:

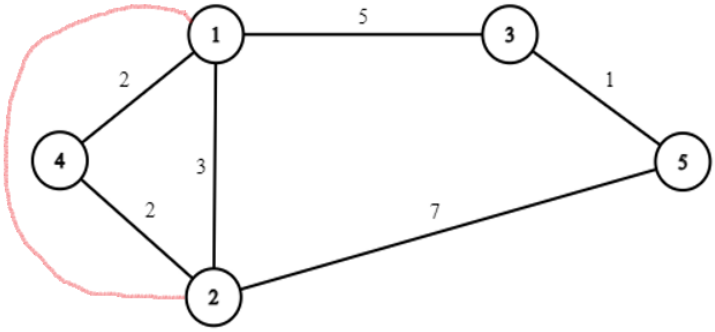

Figure 8.2

Optimal chinese postman route is of length : 5 + 1 + 7 + 3 + 2 + 2 + 3 = 23 [= sum of all edges of modified graph]

Chinese Postman Route : 1 - 3 - 5 - 2 - 1 - 4 - 2 - 1

This route is Euler Circuit of the modified graph.

8.2 Time Complexity of Chinese Postman Problem

Here, n=number of vertices in a graph

As the Chinese Postman Problem is undirected, so the undirected route inspection problem can be solved in polynomial time by an algorithm based on the concept of a T-join. Let T be a set of vertices in a graph. An edge set J is called a T-join if the collection of vertices that have an odd number of incident edges in J is exactly the set T. A T-join exists whenever every connected component of the graph contains an even number of vertices in T. The T-join problem is to find a T-join with the minimum possible number of edges or the minimum possible total weight.

For any T, a smallest T-join (when it exists) necessarily consists of $\frac{1}{2}|T|$ paths that join the vertices of T in pairs. The paths will be such that the total length or total weight of all of them is as small as possible. In an optimal solution, no two of these paths will share any edge, but they may have shared vertices. A minimum T-join can be obtained by constructing a complete graph on the vertices of T, with edges that represent shortest paths in the given input graph, and then finding a minimum weight perfect matching in this complete graph. The edges of this matching represent paths in the original graph, whose union forms the desired T-join. Both constructing the complete graph, and then finding a matching in it, can be done in $\mathbf{O}(n^3)$ computational steps. So, the time complexity for the worst case is $\mathbf{O}(n^3)$.

For the route inspection problem, T should be chosen as the set of all odd-degree vertices. By the assumptions of the problem, the whole graph is connected (otherwise no tour exists), and by the handshaking lemma it has an even number of odd vertices, so a T-join always exists. Doubling the edges of a T-join causes the given graph to become an Eulerian

multigraph (a connected graph in which every vertex has even degree), from which it follows that it has an Euler tour, a tour that visits each edge of the multigraph exactly once. This tour will be an optimal solution to the chinese postman problem.

8.3 Weighted Bipartite Matching

Definition 8.2. In a bipartite graph with partite sets $X = \{x1, ..., xn\}$ and $Y = \{y1, ..., yn\}$, placing weights on its edges gives us a *weighted bipartite graph.* Then selecting edges to form a matching that will minimize total weight is the ***weighted bipartite matching***.

Let's take two examples to understand weighted bipartite matching.

Example 8.3. Consider the matrix given below, where rows and columns are denoting cranes and construction sites respectively. The x_{ij} entry of the matrix is showing the distance(in kms) of crane i from the construction site j. Then the problem is where to send which crane so that the distance travelled is minimum.

$$\begin{array}{c} \begin{matrix} S_1 & S_2 & S_3 & S_4 \end{matrix} \\ \begin{pmatrix} 5 & 10 & 15 & 20 \\ 3 & 5 & 6 & 9 \\ 9 & 8 & 1 & 15 \\ 15 & 16 & 1 & 5 \end{pmatrix} \end{array} \begin{matrix} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{matrix}$$

We can take $\{S_1C_1, S_2C_2, S_3C_3, S_4C_4\}$ as one solution to minimize the cost, which is weighted bipartite matching.

Example 8.4. Consider the matrix given below, where rows and columns are denoting persons and tasks respectively. The x_{ij} entry of the matrix is showing the time period(in mins) in which the person i can do j task. Then the problem is to find suitable person for each task to complete it in minimum time period.

$$\begin{array}{c} \begin{matrix} T_1 & T_2 & T_3 \end{matrix} \\ \begin{pmatrix} 10 & 5 & 15 \\ 5 & 6 & 9 \\ 18 & 19 & 1 \end{pmatrix} \end{array} \begin{matrix} \\ P_1 \\ P_2 \\ P_3 \end{matrix}$$

We can take $\{T_2P_1, T_1P_2, T_3P_3\}$ as one solution to minimize the total time period, which is weighted bipartite matching.

8.4 Transversal

Definition 8.5. A **transversal** of an $n \times n$ matrix consists of n positions one in each row and one in each column.

In an $n \times n$ matrix, total number of tranversals is equal to $n!$.

Examples of transversals:

$$\begin{bmatrix} 24 & 47 & (75) \\ (54) & 29 & 85 \\ 71 & (34) & 25 \end{bmatrix}$$

$$\begin{bmatrix} 83 & (55) & 15 \\ 42 & 30 & (25) \\ (30) & 18 & 25 \end{bmatrix}$$

8.5 Assignment Problem as Optimization

Assignment Problem: Finding a transversal with minimum sum is **Assignment Problem**.

Then optimization problem is given as:

$$\begin{aligned} \min \quad & \sum_i \sum_j c_{ij} x_{ij} \\ \text{s.t.} \quad & (i) \sum_i x_{ij} = 1 \\ & (ii) \sum_j x_{ij} = 1 \\ & (iii) x_{ij} \in \{0, 1\}, x_{ij} \geq 0 \end{aligned}$$

where

$$C = \begin{bmatrix} c_{00} & c_{01} & \dots & c_{0n} \\ c_{10} & c_{11} & \dots & c_{1n} \\ \dots & \dots & \dots & \dots \\ c_{m0} & c_{m1} & \dots & c_{mn} \end{bmatrix}$$

and x_{ij} is the ij^{th} position of assignment.

8.6 Hungarian Algorothm

It is an algorithm to solve assignment problem as optimization.
The algorithm is given in the following steps:
In a given $n \times n$ matrix,

1. Find row minimum of each row.
2. Subtract row minimum from each element in its row.
3. Find column minimum from each column.

4. Then, same as step 2, subtract column minimum from each element in its column.

5. Cover all zeros in the matrix using minimum number of horizontal and vertical lines.

6. *Test for Optimality:* If the minimum number of covering lines is n, an optimal assignment is possible and we are finished. Else if lines are lesser than n, we haven't found the optimal assignment, and must proceed to step 7.

7. Determine the smallest entry not covered by any line. Subtract this entry from each element of the matrix uncovered by the lines of step 5, and then add it to each covered column. Return to step 5.

Let's take two examples to understand Hungarian Algorithm.

Example 8.6. Consider a cost matrix of any assignment problem given below:

$$\begin{bmatrix} 40 & 60 & 15 \\ 25 & 30 & 45 \\ 55 & 30 & 25 \end{bmatrix}$$

Step 1: The row minimum of each row is written along each row:

$$\begin{bmatrix} 40 & 60 & 15 \\ 25 & 30 & 45 \\ 55 & 30 & 25 \end{bmatrix} \begin{matrix} 15 \\ 25 \\ 25 \end{matrix}$$

Step 2: On subtracting the row minimum from each element in its row, we get the following matrix:

$$\begin{bmatrix} 25 & 45 & 0 \\ 0 & 5 & 20 \\ 30 & 5 & 0 \end{bmatrix}$$

Step 3: Now we write the column minimum of each column along each column:

$$\begin{bmatrix} 25 & 45 & 0 \\ 0 & 5 & 20 \\ 30 & 5 & 0 \end{bmatrix} \\ \begin{matrix} 0 & 5 & 0 \end{matrix}$$

Step 4: Subtracting the column minimum from each element in its column, we get the following matrix:

$$\begin{bmatrix} 25 & 40 & 0 \\ 0 & 0 & 20 \\ 30 & 0 & 0 \end{bmatrix}$$

Step 5: Covering all zeroes in the above matrix using minimum number of horizontal and vertical lines, we get the following matrix:

$$\begin{bmatrix} \cancel{25} & \cancel{40} & \cancel{0} \\ \cancel{0} & \cancel{0} & \cancel{20} \\ \cancel{30} & \cancel{0} & \cancel{0} \end{bmatrix}$$

Step 6: From the matrix in step 5, we came to know that the minimum number of covering lines is 3, which is same as the order of the matrix. So, we have achieved an optimal assignment, which is shown by the starred elements of the matrix given below:

$$\begin{bmatrix} 25 & 40 & 0^* \\ 0^* & 0 & 20 \\ 30 & 0^* & 0 \end{bmatrix}$$

which means

$$\begin{bmatrix} 40 & 60 & \textcircled{15} \\ \textcircled{25} & 30 & 45 \\ 55 & \textcircled{30} & 25 \end{bmatrix}$$

$\Rightarrow$ From the above matrix, **total cost of optimal assignment** $= 25 + 30 + 15 = 70$

Example 8.7. Consider another cost matrix of any assignment problem given below:

$$\begin{bmatrix} 30 & 25 & 10 \\ 15 & 10 & 20 \\ 25 & 20 & 15 \end{bmatrix}$$

Step 1: The row minimum of each row is written along each row:

$$\begin{bmatrix} 30 & 25 & 10 \\ 15 & 10 & 20 \\ 25 & 20 & 15 \end{bmatrix} \begin{matrix} 10 \\ 10 \\ 15 \end{matrix}$$

Step 2: On subtracting the row minimum from each element in its row, we get the following matrix:

$$\begin{bmatrix} 20 & 15 & 0 \\ 5 & 0 & 10 \\ 10 & 5 & 0 \end{bmatrix}$$

Step 3: Now we write the column minimum of each column along each column:

$$\begin{bmatrix} 20 & 15 & 0 \\ 5 & 0 & 10 \\ 10 & 5 & 0 \\ 5 & 0 & 0 \end{bmatrix}$$

Step 4: Subtracting the column minimum from each element in its column, we get the following matrix:

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

Step 5: Covering all zeroes in the above matrix using minimum number of horizontal and vertical lines, we get the following matrix:

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

Step 6: From the matrix in step 5, we came to know that the minimum number of covering lines is 2, which is lesser than the order of the matrix, 3. So, we have to go to step 7.

Step 7: The smallest entry not covered by any line is min{15,5}=5. On subtracting this entry from each element of the matrix uncovered by the lines of step 5, and then add it to each covered column, we get the following matrix:

$$\begin{bmatrix} 10 & 10 & 0 \\ 0 & 0 & 15 \\ 0 & 0 & 0 \end{bmatrix}$$

Then again following Step 5, we get the following matrix:

Here, the minimum number of covering lines is 3, which is same as the order of the matrix. So, we have achieved an optimal assignment, which is shown by the starred elements of the matrix given below:

$$\begin{bmatrix} 10 & 10 & 0^* \\ 0 & 0^* & 15 \\ 0^* & 0 & 0 \end{bmatrix}$$

which means

$$\begin{bmatrix} 30 & 25 & \textcircled{10} \\ 15 & \textcircled{10} & 20 \\ \textcircled{25} & 20 & 15 \end{bmatrix}$$

$\Rightarrow$ From the above matrix, **total cost of optimal assignment** $= 25 + 10 + 10 = 45$

8.7 Time Complexity of Hungarian Algorithm

Here, $|V|$= order/size of the given cost assignment matrix or cardinality of the bipartition set.

The Hungarian matching algorithm, also called the Kuhn-Munkres algorithm, is a $\mathbf{O}\big(|V|^3\big)$ algorithm that can be used to find minimum-weight matchings in bipartite graphs, which is sometimes called the assignment problem. A bipartite graph can easily be represented by an adjacency matrix, where the weights of edges are the entries. Thinking about the graph in terms of an adjacency matrix is useful for the Hungarian algorithm.

The Hungarian algorithm solves the following problem:

In a complete bipartite graph G, find the minimum-weight matching. (Recall that a maximum-weight matching is also a perfect matching.)

At each step, the algorithm adds one edge to the matching and this happens $\mathrm{O}\big(|V|\big)$ time.

It takes $\mathrm{O}\big(|V|\big)$ time to find the right vertex for the augmenting (if there is one at all), and $\mathrm{O}\big(|V|\big)$ time to flip the matching.

Improving the labeling takes $\mathrm{O}\big(|V|\big)$ time to find δ_l and to update the labelling accordingly. We might have to improve the labeling up to $\mathrm{O}\big(|V|\big)$ times if there is no augmenting path. This makes for a total of $\mathrm{O}\big(|V|^2\big)$ time.

In all, there are $\mathrm{O}\big(|V|\big)$ iterations each taking $\mathrm{O}\big(|V|\big)$ work, leading to a total running time of $\mathbf{O}\big(|V|^3\big)$. So, the time complexity for the worst case is $\mathbf{O}\big(|V|^3\big)$.