Week 8

Graph Theory[*]

8.1 Chinese Postman problem

The Chinese postman problem, postman tour or route inspection problem is to find a shortest closed path or circuit that visits every edge of an (connected) undirected graph. When the graph has an Eulerian circuit (a closed walk that covers every edge once), that circuit is an optimal solution. Otherwise, the problem reduces to finding the smallest number of graph edges to duplicate (or the subset of edges with the minimum possible total weight) so that the resulting multi-graph does have an Eulerian circuit.

8.1.1 Introduction

Problem statement Given a directed or undirected graph, The problem is to find a minimum length closed walk that traverses each edge at least once. This problem is also known as Route Inspection or Arc Routing problem.

For example, take the following graph G_1 given in the Figure 8.1.

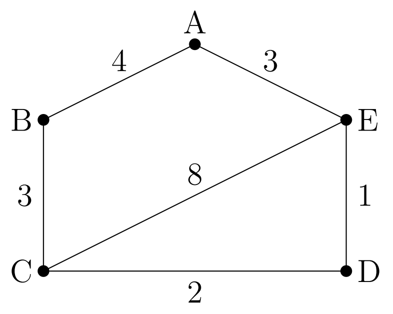

Figure 8.1: Graph G_1

Consider two circuits given by the sequences

$$S_1 = ((A, B), (B, C), (C, E), (E, C), (C, D), (D, E), (E, A))$$

$$S_2 = ((A, B), (B, C), (C, D), (D, E), (E, C), (C, D), (D, E), (E, A))$$

[*]Lecturer: Anand Mishra. Scribe: Ashish Jacob Sam(M21CS003).

Both S_1 and S_2 start from the vertex A, traverse all edges and return back to A. In that sense, both these paths fulfill the initial condition for the problem, which is to traverse every edge in the graph at least once. However, S_2 is the circuit with the minimum cost. Thus, only S_2 is a valid solution to the Chinese Postman Problem for this graph among the two sequences. Note that although both the circuits had some edges repeated, S_2 is the solution since the edges repeated in S_2 $((C, D), (D, E))$ had the least sum of weights of the repeated edges. Needless to say, there can be multiple such circuits that start at vertex A with the same cost, and all of these would be a valid solution to the problem.

8.1.2 Solving the Chinese Postman Problem

It is trivial to note that some edges might get repeated for a graph. Since the minimum cost circuit is desired, it is required to ensure that the edges which are getting repeated are picked such that their weight is minimum of all possible selections. Consider a connected graph $G = (V, E)$. Then there are exactly zero repetitions of edges required if the graph has an Eulerian circuit. The necessary condition for a graph to have an Eulerian circuit is

$$\forall v \in V, \quad \text{degree of v is even}$$

This size of this set of vertices having odd degrees will always be even. Therefore we can always pair these vertices having odd degrees, and repeat some existing edges on the graph such that the repeated edges have the minimum possible weights. With this, all edges are now of even degree and thus, an Eulerian circuit is now possible in the graph.

Example Consider graph G_1. The vertices having odd degree are $\{C, E\}$. Therefore we need to repeat edges along the said vertices. E can be reached from C by any one of $((C, E))$, or $((C, D), (D, E))$. Since the path $((C, D), (D, E))$ has the least cost of all the choices, these edges are selected to be repeated. Thus, with repeated edges, all the edges present in the graph are now shown, with the repeated edges denoted with '*'

$$E' = \{(A, B), (B, C), (C, D), (C, E), (D, E), (E, A), (C, D)^*, (D, E)^*\}$$

The sum of all the weights of these edges is 24. Since the graph is now Eulerian, a circuit can exist which uses all these edges, starting from any vertex in the graph.

8.1.3 Algorithm

We can formally devise an algorithm to solve the Chinese Postman Problem as follows:

```
Solve_CPP(Graph g)
    1. Evaluate O, Set of vertices with odd degrees. Let k = |O|
    2. Compute shortest paths between all pairs of odd-nodes
    3. Select pairs in increasing order, pick k/2 pairs.
    4. Repeat the edges between these k/2 pairs
    5. Find the weight of the circuit by summing weights of all edges.
    6. Compute the Eulerian tour of the graph.
```

We can analyse this algorithm as follows:

- Step 1 is bounded by $O(|V|)$, since at most all vertices of the graph can be odd.
- Step 2 is bounded by $O(|V|^3)$, since at most all vertices of the graph can be odd, and the all-pairs shortest-pair algorithm, also known as the Floyd-Warshall's Algorithm is bounded by $O(|V|^3)$
- Step 3 is bounded by $O(|V|^2)$, since the k vertices have to be selected in increasing order, which can be done by maintaining a heap by adding all pairs in the heap and removing them one by one and picking the required $\frac{k}{2}$ pairs.
- The time complexity of Step 4 depends on the implementation of the graph, but it is always lower or bounded by the step 5, which $O(|E|)$
- Finally, step 6 can be done using Fleury's Algorithm, which is bounded by $O((|V|+|E|)^2)$.

Thus the time complexity of this algorithm is bounded by $O(|V|^3+(|V|+|E|)^2)$. For sparse graphs, this complexity bound becomes $O(|V|^3)$ itself.

Note : The following is to be noted for this algorithm:

- Some references only limit the problem to find the minimum cost (hence excluding step 6). In such a case, the algorithm is always bounded within $O(|V|^3)$
- For a directed graph the above algorithm works, although a solution exists if and only if the graph is strongly connected. Also, O would be set of vertices whose in-degree is not equal to their out-degree. The step 5 would select edges such that all vertices end up having their in-degree equal to their out-degree.
- There are other approaches for solving the Chinese Postman Problem, like using a generalization of the T-join problem. It bounds the solution to an undirected graph under $O(|V|^3)$, and directed graph under $O(|V|^2|E|)$ time complexity.

Example using the above algorithm, the Chinese postman problem on the graph given in Figure 8.2 can be solved as follows:

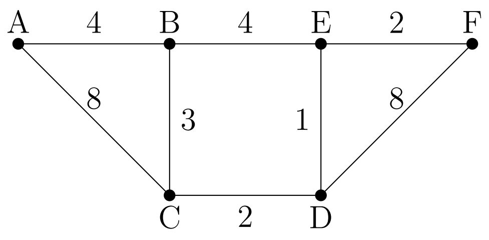

Figure 8.2: Graph G_2

All the odd-degree vertices of the graph as found to be:

$$O = \{B, C, D, E\}$$

Now, edges are to be repeated in this graph such that the graph becomes an Eulerian graph and the repeated edges have the minimum weight possible. Thus the edge between pairs (E, D) and (B, C) are selected. These repeated edges are drawn as curve in the figure 8.3

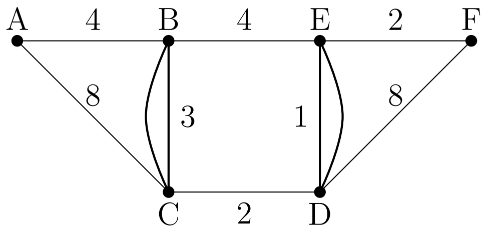

Figure 8.3: Graph G_2 with repeated edges

Now the sum of weight of all edges (including repeated edges) is 36. Thus, the cost of any eulerian circuit in this graph should be at most 36.

We can see such a circuit can be started from A as shown:

$$P = ((A, B), (B, C), (C, B), (B, E), (E, F), (F, D), (D, E), (E, D), (D, C), (C, A))$$

The cost of P is $Cost((A, B)) + Cost((B, C)) + ... = 36$

8.2 Weighted Bipartite matching

Given a weighted bipartite graph, the Weighted Bipartite matching problem is to find the matching in which the sum of the weights is minimized. The terms are defined in the following subsection.

8.2.1 Introduction

The problem is focused on a bipartite graph denoted as $G = (X, Y, E)$, with E as the set of edges, and X, Y are the disjoint set of vertices.

A matching assigns every vertex in X to exactly one neighbour in Y. A matching $M \subseteq E$ is called a perfect matching if every vertex in G is adjacent to some edge in M.

A matching M is called the maximum weight matching if the sum of the weights in the matching are the highest possible of all possible matchings.

The sum of weights of the matching M is denoted by $W(M)$, which is given by

$$W(M) = \sum_{\forall e \in M} w(e)$$

Therefore, a minimum-weighted matching M is such that for every other matching M' in the graph,

$$\forall M' \subseteq E, \qquad W(M) \leq W(M')$$

8.2.2 The Assignment problem

For a given complete bipartite graph $K_{n,n}$, the assignment problem is to find the perfect matching such that the matching is a minimum-weighted matching.

The problem instance has 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 agent-task assignment. It is required to perform as many tasks as possible by assigning at most one agent to each task and at most one task to each agent, in such a way that the total cost of the assignment is minimized. The graph weights can also be adjusted to compute the maximum instead of minimum weighted matching.

More formally, the assignment problem can be described in terms of an optimization problem as

$$\begin{aligned} \text{minimize} \sum_i \sum_j C_{ij} X_{ij} \qquad & \text{s.t} \\ X_{ij} \in \{0,1\}&, \\ \sum_i X_{ij} = 1&, \\ \sum_j X_{ij} = 1& \end{aligned}$$

Example Let there be 3 items X,Y,Z which you want to buy. You get offers from various shops A,B,C to buy the item at their offered prices. You wish to obtain all the items at the minimum cost. You realize that this is an application of the assignment problem and work on the details to solve it.

The costs can be given by the following table as shown:

Shop	X	Y	Z
A	30	25	10
B	15	10	20
C	25	20	15

There can be multiple matching possible, such as

$$M_1 = \{(A,X),(B,Y),(C,Z)\}, \qquad W(M_1) = 30 + 10 + 15 = 55$$

$$M_2 = \{(A,Y),(B,X),(C,Z)\}, \qquad W(M_2) = 25 + 15 + 15 = 55$$

$$M_3 = \{(A,Z),(B,Y),(C,X)\}, \qquad W(M_3) = 10 + 10 + 25 = 45$$

For an bipartite graph $K_{n,n}$, there will be $n!$ ways of selections. While it is possible to compute for all permutations for $K_{3,3}$, it is trivial to see that this becomes exponentially difficult to compute as n increases.

8.2.3 The Hungarian Method

The Hungarian Method is an algorithm to solve the assignment problem for a given complete bipartite graph $K_{n,n}$, based on its $n \times n$ matrix form. The algorithm is given as follows

```
Step 1: Find row_{min} for each row, and subtract it from every row
Step 2: For each row, row = row-row_min
Step 3: Find col_min for each column,
Step 4: for each column, col = col - col_min
Step 5: find the minimum number of lines needed to cover all 0s.
    If min_num = n,
        then stop,
    else,
        subtract every element with the non-zero min of matrix, goto step 5
Optimal matching is given by the n lines for each row/column.
```

Analysing the algorithm, we see that

- Step 1-4 can be done in $O(n^2)$, which is finding the minimum of n different n-sized vectors, and doing subtraction on them.
- Step 5 features finding a zero, and covering/deleting the row and column from the matrix. Thus this is an $O(n^2)$ operation. Since it takes n lines before terminating, this step is bounded within $O(n^3)$

Thus the algorithm is bounded within $O(n^3)$

This algorithm is used below, beginning by detailing the graph edges as a matrix as shown for the problem mentioned earlier as:

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

In step 1, the minimum elements of each rows are evaluated as $(10, 10, 15)$. In step 2, each row is subtracted with its minimum to obtain

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

In step 3, the minimum elements of each column are evaluated as $(5, 0, 0)$ In step 4, each column is subtracted with its minimum to obtain

$$\begin{pmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{pmatrix},$$

In step 5, it is checked if all 0s can be selected by 3 lines. It is false. Thus the least non-zero element of the entire matrix is found, which is 5. Thus all non-zero elements are subtracted with 5 and checked again. The matrix now becomes

$$\begin{pmatrix} 10 & 10 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 0 \end{pmatrix},$$

Thus, for row 1, the 3 column is selected, while column 2,3 can select any of 1,2.

Thus a valid matching for the assignment problem is $\{(1,3),(2,1),(3,2)\}$, and the total cost is $10 + 15 + 20 = 45$. Alternatively the matching $\{(1,3),(2,2),(3,1)\}$ will also have the same cost of 45.
