The Chinese Postman Problem

Write-up by Armando H. Taucer
The Problem

- First proposed by Kwan Mei-ko in 1962
- On a weighted graph, minimum cost cycle that passes through all edges.
- Not the traveling salesman problem, which has to pass through every vertex once.
- Can actually be solved in reasonable time.
Euler Tours

- You know of these by now
- Cycle passing through each edge exactly once.
- On an Eulerian graph, an Euler tour is also an optimal postman tour, as it passes through all edges, and any cycle with less cost will not contain all edges
Non-Eulerian Graphs

• What Kwan figured out is that on even on non-Eulerian graphs, it reduces to an Euler tour.

• Step 1- find all the odd-degree nodes in the graph. $O(|V|)$ to search through all nodes.
Non-Eulerian Graphs

• Step 2- calculate shortest paths between all pairs of odd-degree vertexes. $O(|V|^3)$.

• Step 3- create a new graph consisting of the odd-degree vertexes, with an edge between every pair of vertexes with a cost of the shortest path between them. $O(|V|^2)$
Non-Eulerian Graphs

• Step 4- Calculate a minimum cost maximum matching on the new graph. Various ways to do this, several of which have better than $O(|V|^3)$ time.

• This will always be a perfect matching. Why?
Non-Eulerian Graphs

- **Property**- That the number of odd-degree nodes in any graph be even.
- **Proof**- The sum of the degrees of all the nodes on a graph is equal to the sum of the number of ends of edges in that graph. Edges have two ends.
Non-Eulerian Graphs

- The shortest paths that these edges in the matching represent are the paths that will have to be travelled on multiple times in the postman tour, as they are the most efficient paths between odd-degree nodes, and as such, the most efficient ways for a tour to leave an odd-degree node through an edge that it has already been through, and then enter another through an edge that it has already been through (or will be through again).
Non-Eulerian Graphs

• Step 5- Create a graph that is a copy of the original graph, except that every edge gains one parallel edge for each of the shortest paths corresponding to the edges in the matching from step 4 that they appear in.

• This new graph will be Eulerian, as each of the odd degree nodes will have 1 edge added to them from the shortest path that they are one end of (making them even-degree) and any node appearing in a matching path but not at the end of it will gain 2 to its degree-one for the edge going in, and another for the edge going out.
Non-Eulerian Graphs

• Final step- calculate an Euler tour on this graph. This can be done in linear time.
Linear-time Euler tour algorithm

- Euler never determined any sort of formal algorithm to fine an Euler tour. The techniques needed to do so didn’t exist at the time. Carl Hierholzer figured out an efficient algorithm to do this 140 years later.
Linear-time Euler tour algorithm

- Maintain a cycle (initially, chose an arbitrary node and no edges). Add subcycles to it as follows.
- For an arbitrary node on the cycle with at least one edge not in the cycle adjacent to it, chose an arbitrary edge not in the cycle adjacent to it and the vertex on the other end of it. Add these to the cycle after it. Then repeat this on the other vertex, until you get back to the one you started at.
Linear-time Euler tour algorithm

• Repeat this until every edge is in the cycle.
• This works since we know that the graph is Eulerian, and thus has even degree on every node. At any time in its execution, there are only two edges with an odd number of nodes adjacent to it— the one currently being examined and the one at the start of the current sub-cycle. There will always be another valid edge to add if there are any unclaimed edges at all.
Directed graphs.

• Requires a strongly-connected graph.
• You’d think that these are far, far harder to find, but in reality they are far simpler.
• The same general idea applies, adding hypothetical edges to the graph until it is Eulerian.
Directed graphs.

• Why is it easier? Because you know whether the path will be entering or leaving a node along an edge.

• A directed Eulerian graph is one with equal in-degree and out-degree.

• Technique discovered by Edmonds and Johnson in 1973 (I think)
Directed graphs

• For every node in the graph calculate the difference between its out-degree and its in-degree. $O(|V|)$ time.

• This is a network flow problem, albeit of a slightly different type than those seen in this class thus far.

• All edges have infinite capacity.
Directed Postman as Network Flow

• In this case, since all edges have infinite capacity, and the sum of the flow produced and consumed is equal, All flow will reach its destination.
• Runs in $O(|V|^2|E|)$. Can be modified to find min cost max flow.
• (I was expecting all this material to be covered in the course under max flow).
Directed Postman as Network Flow

• The amount of flow along an edge is the number of parallel edges to insert into the graph to make it Eulerian.

• This works because the sources were nodes with more edges going in than coming out, while sinks were the opposite. As such, nodes with too few out-edges will gain out-edges in the form of the flow from them, nodes with too few in-edges will gain in-edges in the form of flow into them, and all other nodes will gain an equal number of in-edges and out edges.
Directed Euler Tours

• Then, all that has to be done is to add an Euler tour.

• Can be done using basically the same algorithm as in the undirected case. Still linear time.
• Overall time complexity - $O(|V|^2|E|)$. 
Sources

• Matching, Euler tours and the Chinese postman - Jack Edmonds - Ellis L. Johnson - Mathematical Programming - 1973
