Path Problems and Complexity Classes

Henry Z. Lo

July 19, 2014

1 Path Problems

1.1 Euler paths

1.1.1 Description

Problem 1. *Find a walk which crosses each of the seven bridges in figure 1 exactly once. The land masses can only be reached by crossing these bridges.*

The relevant pieces of this problem are the land masses and the bridges. This allows the problem to be represented in an undirected graph, as shown in figure 2. A path which traverses each edge exactly once is called an *Euler path*.

Euler found that there is no such walk. Furthermore, in any undirected graph, there can only be an Euler walk if there are exactly 0 or two odd-degree vertices. The degree of a vertex is the number of edges involving that vertex.

Intuitively, this is because in order for the walk to include an odd-degree vertex, it must go in and out of that vertex, which uses up two edges. At some point, there will only be one edge left for that vertex, and if the path leads to the vertex, it must end there. This is not a problem if we visit the odd-degree vertex last, or if we start at it. However, when we have more than 2 odd-degree vertices, this is not possible.

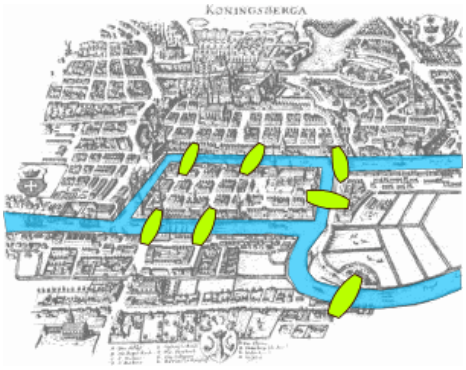

Figure 1: The seven bridges of Konigsberg.

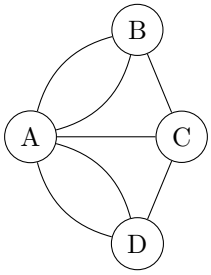

Figure 2: Graph representing the seven bridges of Konigsberg. Edges are bridges, and nodes are land masses.

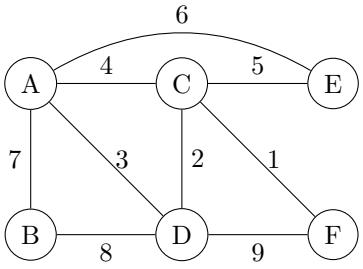

Figure 3: Example graph. The edges of an Euler path are labeled in the order in which they were added.

1.1.2 Algorithm

If an Euler path exists, however, we can find it in polynomial time. The key idea is to remove edges as they are added to the path, and avoid using bridges, which are the only edge connecting two nodes.

As an example, suppose we start at node F in the graph in figure 3. Once we move to C and D, DF becomes a bridge. Thus, we cannot traverse this edge unless we have no other choice. The rest of the algorithm is a simple DFS.

This algorithm works whether we have zero or two odd-degree vertices. In the latter case, we must start at one and end at the other. Otherwise, we can start anywhere.

```
function fleury(g):
  g2 = copy of graph g
  q = empty queue
  if number of odd degree vertices = 2
    n = random odd degree vertex
  else if = 0
    n = random vertex
  else
    return null (no Euler path)
```

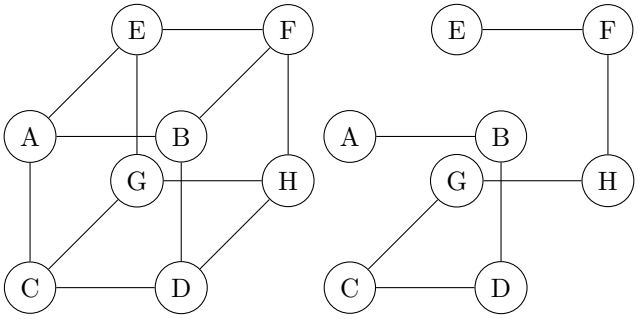

Figure 4: Example graph and one of its Hamiltonian paths.

```
q.add(n)
while all nodes not visited
  do (pick neighbor n2 of n)
    while (n->n2 is a bridge)
  remove n->n2 from g2
return q
```

We can detect a bridge from $n \to n_2$ by performing a BFS from n, looking for n_2 after removing the edge. If n_2 is not found, then that edge is a bridge. However, if there is only one edge out of n, we should take it regardless.

The bridge-finding algorithm is $O(E)$, since in the worst case it will look through every edge in the graph (E here is larger or equal to V if we are to have an Euler path). This bridge checking is done for every iteration of the outer loop, which is also $O(E)$, and thus the entire algorithm is $O(E^2)$.

1.2 Hamiltonian paths

Problem 2. *Given a graph, find a path which visits every vertex exactly once.*

See figure 4 for a visual example.

This problem is conceptually similar to the Euler paths, but unlike the previous problem, there is no known polynomial time algorithm for determining the existence of Hamiltonian paths.

As mentioned in the previous set of notes, it is possible to enumerate all possible paths, but this is a factorial time algorithm.

2 P and NP

2.1 NP

However, suppose we were able to try every possible path in parallel. Then, we would be able to solve the Hamiltonian path problem in polynomial time (in fact, it would be linear).

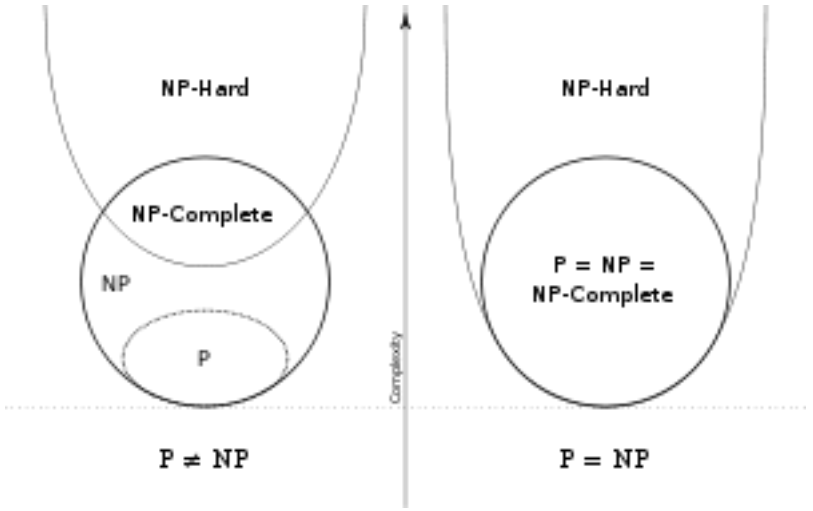

Figure 5: Visualization of the problem classes P, NP, NP-complete, and NP-hard, if P≠NP and if P=NP.

This property (solvability in polynomial time with parallelism) places the Hamiltonian path problem in the class of problems known as NP, or non-deterministic polynomial time.

Most of the problems we have studied are in P, the class of polynomial-time solvable problems. P algorithms are also NP, because anything solvable in polynomial time without parallelism is also solvable in polynomial time with it.

It is not known if NP problems are actually P. If so, this would imply that P=NP. This P=NP problem is a very high-profile unsolved problem in computer science / mathematics, with a prize of one million dollars. Many believe P=NP to be false, but this has not been shown; if P=NP is in fact true, this would have profound implications for computer science as a whole, as many problems previously believed to be difficult will then be tractable.

The problem of determining if a Hamiltonian path exists is in fact an NP-complete problem, which means that any NP problem can be solved if Hamiltonian paths can be solved. We will not show this to be true, but it has been proven.

3 NP-Hard Problems

NP-hard problems are those which are at least as hard as an NP-complete problem. We show that two well-known problems are in fact NP-hard by *reducing* the Hamiltonian path problem to these problems. This reduction argument is constructive, showing that if we can solve these problems, we can easily construct a solution to the Hamiltonian path problem. Since Hamiltonian path is known to be as hard as every problem in NP, these problems are at least as

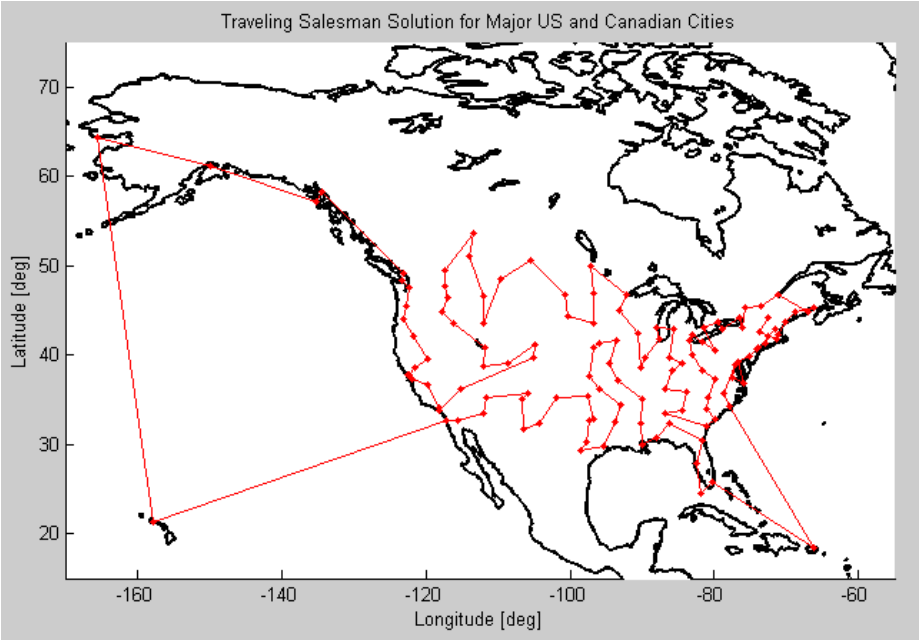

Figure 6: Graphical.

difficult as all problems in NP.

3.1 Travelling salesman problem

Problem 3. *Given a weighted graph, find the path which travels to all vertices exactly once with the lowest total weight.*

This is known as the travelling salesman problem because of its application to route planning. This is an NP problem, for the same reason as Hamiltonian paths (if we can try every path in parallel, we can find the answer quickly).

The travelling salesman problem is perhaps the best known example of an NP-hard problem, with applications in many different fields. Typically when encountering this type of problem, one would use a heuristic such as greedy to get an approximate solution quickly. The problem is not known to be solvable in polynomial time, though dynamic programming approaches have reduced it from $O(n!)$ to $O(n^2 2^n)$.

To show that TSP is NP-hard, notice that any path which solves TSP is necessarily a Hamiltonian path. Thus, if we have an algorithm which solves TSP, and a graph for which we want to find a Hamiltonian path, we can do so by using the TSP-solver on the graph, where the edge weights are 1 if the nodes are connected, and infinite otherwise.

Because we can solve Hamiltonian paths if we can solve TSP, TSP is said to be at least as hard as Hamiltonian paths (and by extension, at least as hard as

every problem in NP).

3.2 Longest path in a graph

Problem 4. *Given a graph, find a simple path with the maximum total weight.*

We have discussed the solvability of this problem for DAGs in previous notes, but it is not known to be solvable in polynomial time for graphs in general. Using the same argument as TSP, we can see that longest path is an NP problem.

To show its NP-hardness, we use the longest-path solver to solve Hamiltonian paths. Note that if we want to determine whether or not a Hamiltonian path exists for an unweighted graph, we can give it edge weights of 1. Then, we just use the longest-path solver to find the longest path in this graph, and if the total path length is the number of vertices minus one, then the graph must have a Hamiltonian path.