

We continue our study of graph theory in this chapter, eventually defining and solving the traveling-salesman problem via several methods.

In Chapter 5 we solved the routing problem in which we traveled every **edge** of a graph exactly once (or as few additional times as possible). The problem applies to, for example, a mail carrier minimizing the amount of travel to cover every sidewalk in a neighborhood.

In Chapter 6 we will be interested in traveling to every **vertex** of a graph exactly once. The problem applies to, for example, a traveling salesman trying to visit a collection of cities exactly once. A twist to the problem will be that the cost of getting to the different cities varies; the salesman is interested in the *cheapest* route to all the cities.

- **Hamilton circuit:** Circuit that passes through *every vertex* of a connected graph exactly once (except the starting/ending vertex)
- **Complete graph:** Graph with N vertices in which *every* pair of vertices is joined by exactly one edge. A complete graph is denoted K_N .

The number of edges in K_N is $\frac{N \times (N-1)}{2}$.

How many Hamilton circuits in a complete graph K_N begin and end at a given vertex? (We call these the *distinct Hamilton circuits* for K_N .)

- **Weighted graph:** Graph with numbers associated with the edges

If two vertices represent cities, the number (**weight**) on the edge between them could represent time, distance, or cost of travel from one city to the other.

- **Traveling-salesman problem (TSP):** Find the “optimal” Hamilton circuit—that is, a Hamilton circuit such that if we add the weights of the edges traveled, we get the smallest possible number.
- **Brute-Force algorithm**
 1. Make a list of all distinct Hamilton circuits without mirror images.
 2. Calculate the total weight for each.
 3. Select the circuit with the lowest weight.

(OVER)

- **Optimal? Efficient?** The Brute-Force algorithm finds the *optimal* Hamilton circuit but is too computationally-intensive to be practical for large problems. People have been trying for 50 years to find an *efficient* algorithm.

In the meantime, we have to use *efficient* methods that only *approximate* the optimal Hamilton circuit. In other words, the methods find a good route, but not necessarily the best. We'll study three such methods:

- **Nearest-Neighbor algorithm**

1. Pick a vertex as a starting point (usually home).
2. Travel the edge with the lowest weight. That edge leads to the “nearest-neighbor” vertex.
3. Continue building the circuit, one vertex at a time, traveling to the “nearest-neighbor” (of your current location), choosing from among vertices that have not yet been visited.
4. From the last vertex, return to the starting point.

- **Repetitive nearest-neighbor algorithm**

1. Pick a vertex X as a starting point. Apply the nearest-neighbor algorithm.
2. Repeat the process using each of the other vertices as the starting point.
3. Of the Hamilton circuits obtained, keep the best one. If there is a designated starting vertex, rewrite the circuit with that vertex as the starting point.

- **Cheapest-link algorithm**

1. Pick the link (edge) with the smallest weight first. (Break ties randomly.)
2. Pick the next-cheapest link, and mark the corresponding edge.
3. Continue picking the cheapest link and marking the corresponding edge *except* when
 - (a) it closes a circuit or
 - (b) it results in three edges coming out of a single vertex.
4. When there are no more vertices to link, close the marked circuit.