Shortest Path Problems

We can assign weights to the edges of graphs, for example to represent the distance between cities in a railway network:

![Graph representation of distances between cities](image)

Shortest Path Problems

Such weighted graphs can also be used to model computer networks with response times or costs as weights.

One of the most interesting questions that we can investigate with such graphs is:

What is the shortest path between two vertices in the graph, that is, the path with the minimal sum of weights along the way?

This corresponds to the shortest train connection or the fastest connection in a computer network.

Dijkstra’s Algorithm

Dijkstra’s algorithm is an iterative procedure that finds the shortest path between to vertices a and z in a weighted graph.

It proceeds by finding the length of the shortest path from a to successive vertices and adding these vertices to a distinguished set of vertices S.

The algorithm terminates once it reaches the vertex z.

Dijkstra’s Algorithm

procedure Dijkstra(G: weighted connected simple graph with vertices a = v₀, v₁, …, vn = z and positive weights w(vᵢ, vⱼ), where w(vᵢ, vⱼ) = ∞ if {vᵢ, vⱼ} is not an edge in G)

for i := 1 to n

L(vᵢ) := ∞

L(a) := 0

S := ∅

{the labels are now initialized so that the label of a is zero and all other labels are ∞, and the distinguished set of vertices S is empty}

while z ∉ S

begin

u := the vertex not in S with minimal L(u)

S := S ∪ {u}

for all vertices v not in S

if L(u) + w(u, v) < L(v) then

L(v) := L(u) + w(u, v)

{this adds a vertex to S with minimal label and updates the labels of vertices not in S}

end (L(z) = length of shortest path from a to z)
Dijkstra’s Algorithm

Example:

Step 1

Step 2

Step 3

Step 4

Step 5

Step 6
The Traveling Salesman Problem

The **traveling salesman problem** is one of the classical problems in computer science.

A traveling salesman wants to visit a number of cities and then return to his starting point. Of course he wants to save time and energy, so he wants to determine the **shortest path** for his trip.

We can represent the cities and the distances between them by a weighted, complete, undirected graph.

The problem then is to find the **circuit of minimum total weight** that visits **each vertex exactly once**.

---

**Example:** What path would the traveling salesman take to visit the following cities?

- Chicago
- Toronto
- New York
- Boston

**Solution:** The shortest path is Boston, New York, Chicago, Toronto, Boston (2,000 miles).

---

The Traveling Salesman Problem

**Question:** Given n vertices, how many different cycles \( C_n \) can we form by connecting these vertices with edges?

**Solution:** We first choose a starting point. Then we have \((n - 1)\) choices for the second vertex in the cycle, \((n - 2)\) for the third one, and so on, so there are \((n - 1)!\) choices for the whole cycle.

However, this number includes identical cycles that were constructed in **opposite directions**. Therefore, the actual number of different cycles \( C_n \) is \((n - 1)!/2\).

---

The Traveling Salesman Problem

Unfortunately, no algorithm solving the traveling salesman problem with polynomial worst-case time complexity has been devised yet.

This means that for large numbers of vertices, solving the traveling salesman problem is impractical.

In these cases, we can use **approximation algorithms** that determine a path whose length may be slightly larger than the traveling salesman’s path, but can be computed with polynomial time complexity.

For example, **artificial neural networks** can do such an efficient approximation task.