All-pairs shortest path problem

Mărunțiș Andrei, 323CA

University POLITEHNICA of Bucharest, Faculty of Automatic Control

Abstract. This paper will succintly analyze various pathfinding algorithms that solve the all-pairs shortest path problem. The three algorithms will be compared based on their complexity and running time, and various optimisations will be proposed for each algorithm.

Keywords: Graphs · Shortest path · All-to-all

1 The problem

1.1 Description

Given a graph G with N nodes and E edges, where each edge has a cost C associated, find the shortest path between any two nodes.

1.2 Practical applications

The all-pairs shortest path problem allows us to find the shortest path between any two nodes in a graph (that can be a set of cities with roads connecting them, or computers connected to the internet via cables). This optimization problem allows us to save resources such as time or fuel by finding the most efficient path between two points. Practical applications for this problem can be transportation of goods between cities, or internet communication between computers.

This problem is used when we need to find to answer multiple queries of the type *"what is the shortest path from node A to node B?"*. The environment where this problem is applied needs to be static, meaning the costs of the vertices need to remain constant over time or at least change infrequently, so that a large number of interrogations can be executed. If this condition is not met (for example, a GPS application cannot use this type of preprocessing, since traffic changes all the time), then it is more efficient to answer each individual query by running a shortest path algorithm.

2 Graph features

The types of graphs where we need to solve the problem can influence the algorithms that we apply.

Undirected vs directed In an undirected graph, an edge connects two nodes both ways, while in a directed graph an edge only allows one-way travel. As such, an undirected graph can be a considered a directed graph with twice as many edges.

Cyclic vs acyclic A cycle in a graph is a set of edges that allows travel from one node back to itself. As such, an acyclic graph can have at most one possible path between two nodes, while a cyclic graph can have multiple paths that need to be checked and compared.

Dense vs sparse To define the notion of dense/sparse graph we need to define *graph density*[1]:

$$D = \frac{E}{N(N-1)} \tag{1}$$

However, the notion of sparse/dense graphs is not clearly defined. A sparse graph is a graph with density D almost equal to 0 (has very few edges compared to the number of nodes), and a dense graph is a graph with density D almost equal to 1 (has a large number of edges compared to the number of nodes).

Generally, it is more efficient to represent a sparse graph in memory as an **adjacency list**, while a dense graph is better represented as a **matrix of adjacency**.

Positive vs negative costs A graph with positive costs has all edges with positive costs, while a graph with negative costs has an least one edge with negative costs. However, a graph must not have a negative cycle (i.e. a cycle whose edge costs sum to a negative value) in order to be solvable (because if there is a negative cycle, then we can find paths between two nodes with infinitely negative costs, obtained by traveling multiple times around the negative cycle).

3 Algorithms

3.1 Dijkstra's Algorithm

Dijkstra's Algorithm *fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.*[2]

In order to solve the all-pairs shortest path problem with this algorithm, some modifications need to be made: we need to run Dijkstra's algorithm multiple times, once for each node except for the last one. This is because when the first $N-1$ computations are completed, we have found the shortest path from all $N-1$ nodes to the *Nth* node and no more processing is necessary.

Furthermore, with each iteration of the algorithm, the following iteration becomes faster, as it needs to analize fewer nodes (the *k-th* iteration needs to analize $N-k+1$ nodes, as the first $k-1$ nodes have already been processed).

3.2 Bellman-Ford Algorithm

The Bellman-Ford Algorithm is similar to Dijkstra's Algorithm; it is slower, but *more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.*[3] If there is a negative cycle in the graph that is analysed, the Bellman-Ford Algorithm can detect and report it. Similarly to Dijkstra's Algorithm, we need to apply the Bellman-Ford Algorithm multiple times to solve the all-pairs shortest path problem: once for each node, except for the last one.

One of the restrictions of this algorithm is that it can only be applied efficiently on directed graphs with or without negative edges. In an undirected graph, Dijkstra's algorithm is asymptotically faster than Bellman-Ford and it cannot run on graphs with negative edges, since any negative edge will define a negative cycle. Therefore, Dijkstra's Algorithm is always faster in such a case.

3.3 Floyd-Warshall Algorithm

The Floyd-Warshall Algorithm is an exhaustive algorithm that tests every possible combination of edges in a graph to calculate the shortest path between any 2 nodes. Unlike the other two algorithms mentioned, a single execution will find the lowest cost of the paths between any 2 nodes.

Similarly to the Bellman-Ford Algorithm, the Floyd-Warshall Algorithm can work on directed graphs with or without negative edges, but without negative cycles. It is also capable of detecting negative cycles.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{2}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal [2].

References

1. Wikipedia: Dense graphs (Last access: 24 nov 2022)
2. Wikipedia: Djikstra's Algorithm (Last access: 24 nov 2022)
3. Wikipedia: Bellman-Ford Algorithm (Last access: 24 nov 2022)
4. Wikipedia: Floys-Warshall Algorithm (Last access: 24 nov 2022)