Module 8: Portfolio Project

Brady Chin

Colorado State University Global

CSC506-1: Design and Analysis of Algorithms

Dr. Dong Nguyen

January 12th, 2025

Portfolio Project

Shortest path algorithms, such as Dijkstra's and Bellman-Ford, are fundamental in graph theory and computer science for solving optimization problems. These algorithms determine the shortest path between nodes in weighted graphs, making them essential in areas like network routing, transportation, and logistics.

Dijkstra's algorithm, known for its efficiency, is suitable for graphs with non-negative weights, using a greedy approach to iteratively select the shortest path. Bellman-Ford, while slower, accommodates graphs with negative weights, leveraging dynamic programming to iteratively relax edges and detect negative cycles. Together, these algorithms provide versatile solutions for a wide range of real-world pathfinding problems (Sryheni, S., 2024, March 18).

Project Goals and Purpose

The purpose of this project is to compute the shortest path between two points in a weighted graph using the Dijkstra and Bellman-Ford algorithms, while comparing their efficiencies and complexities.

The goal is to deepen understanding of core concepts in graph theory and pathfinding. Additionally, the project provided valuable experience in implementing algorithms programmatically and exploring their practical applications in real-world scenarios.

Methodology

Dijkstra and Bellman-Ford's algorithm provide different approaches to find the shortest path between two points while taking efficiency and constraints into consideration. This makes them optimal for different use cases.

Dijkstra's algorithm uses a greedy approach to calculate the shortest path. In this project, the key concept that the algorithm relies on Is a priority queue to efficiently select the next node (Pizzo, B., 2024). It is important to note that Dijkstra's algorithm only works on non-negative edge weights. The steps for this algorithm are as follows:

1. **Initialization:** Start with a source node and assign it a distance of 0. All other nodes will have a distance of infinity and will be marked as unvisited.
2. **Relaxation:** Select an unvisited node with the shortest distance and update the distances of it's neighboring nodes.
3. **Greedy:** Mark the node as visited and repeat the process until all nodes are visited or the shortest path is determined.

Figure 1 show the code snippet of the priority queue concept. This has been implemented using a min-heap. The use of this priority queue ensures that the nodes with the smallest distance is always processed next, optimizing the algorithms efficiency.

Figure 1: *Priority queue in Dijkstra's algorithm.*

```
priority_queue = [(0, start)]

while priority_queue:
    current_distance, current_node = heapq.heappop(priority_queue)

    # Skip processing
    if current_distance > distances[current_node]:
        continue

    for neighbor, weight in graph[current_node]:
        new_distance = current_distance + weight
        if new_distance < distances[neighbor]:
            distances[neighbor] = new_distance
            parent[neighbor] = current_node
            heapq.heappush( *args: priority_queue, (new_distance, neighbor))
```

On the other hand, the Bellman-Ford algorithm uses dynamic programming to determine the shortest path. While this algorithm is less efficient than Dijkstra's, the Bellman-Ford is able to handle negative weights. The algorithm steps are as follows:

1. **Initialization:** Start with a. Source node and assign it a distance of 0. All other nodes will have a distance of infinity.
2. **Relaxation:** Update the distance to all nodes by traversing all edges. This will repeat for V - 1 iterations where V is the number of vertices in the weighted graph.
3. **Negative Cycle Detection:** If a distance can be updated, a negative-weight cycle exists.

Figure 2 shows the code snippet of the negative cycle detection. The negative cycle detection is an important element of the Bellman-Ford algorithm. This check is crucial because negative cycles allow for infinitely decreasing path lengths which will make it incorrect and impossible to find the true shortest path. The negative cycle detection checks if further relaxation is possible after the main iterations. If further relaxation is found, it signals a negative weight cycle.

Figure 2: *Negative weight cycle detection in Bellman-Ford's algorithm.*

```
# negative cycle check
for current in graph:
    for neighbor, weight in graph[current]:
        if distances[current] + weight < distances[neighbor]:
            print("Negative weight cycle detected.")
            return None, None  # Negative cycle found
```

Analysis

Table 1 shows the time and space complexities of Dijkstra (geeksforgeeks.org 2024, February 9) and Bellman-Ford's algorithm (geeksforgeeks.org 2024, February 9).
Dijkstra's algorithm was determined to have a better time complexity primarily due to the it's greedy nature and the use of a priority queue. Dijkstra's algorithm processes each node only once. The use of the priority queue optimizes the search for the next node which reduces time complexity for each selection and distance update.

On the other hand, the Bellman-Ford algorithm has a better space complexity. This is because the priority queue requires a more complex data structure.

Although the Bellman-Ford has a more efficient space complexity, it is outweighed by the more superior time efficiency of Dijkstra's algorithm, making it the less efficient algorithm overall.

Table 1: Time and space complexity comparison between Dijkstra and Bellman-Ford algorithms.

	Time Complexity	Space Complexity
Dijkstra	O((V + E) log V)	O(V + E)
Bellman-Ford	O(V * E)	O(V)

In the code that I have provided, both the Dijkstra and Bellman-Ford algorithms include methods to display the shortest distance and path from the start node to the target node. While this impacts the efficiency of the algorithms, it is an add-on and therefore, I did not include it in the calculations because I wanted to focus on the algorithms themselves.

Dijkstra's algorithm is highly efficient for graphs with non-negative edge weights, using a greedy approach to iteratively select the closest node and update its neighbors. This makes Dijkstra ideal for scenarios where performance is critical, such as network routing or GPS navigation.

On the other hand, Bellman-Ford's algorithm, although less efficient, is more versatile. It can handle graphs with negative edge weights and detect negative cycles, which Dijkstra

cannot do. This makes Bellman-Ford particularly useful in financial applications, such as detecting arbitrage opportunities, and in scenarios where negative weights are a factor.

Development Obstacles

Initially starting this project, I had very little knowledge about algorithm development and shortest path algorithms in general. In the beginning, it was difficult to understand the different methods that are used between these two algorithms.

After developing the algorithms, I was surprised that Dijkstra's algorithm was less efficient than the Bellman-Ford. I was confused at this time but soon realized that it was because I was not using a priority queue. It almost felt like I had to restart the algorithm to implement this new technique.

A concept that I had a difficult time understanding is the presence of the negative weight cycle. After developing the Bellman-Ford algorithm, I realized that I kept getting incorrect distances and paths (geeksforgeeks.org 2023, September 4). Once I understood that I needed a negative cycle detection, I easily implemented this and started getting correct outputs.

Project Outcomes and Skills Acquired

After working with and analyzing these algorithms, I have gained a much better understanding of algorithms as a whole and I am better able to identify their strengths and weaknesses. Is this case, I learned that although one algorithm may be more efficient, it is important to look at the use case to select the correct technique. A good example of this would be determining if the use case needs to consider negative weights or if efficient is the most important factor.

Conclusion

Dijkstra and Bellman-Ford algorithms are both essential tools for solving the shortest path problem in graph theory. Each algorithm has its own strengths and weaknesses, making them suitable for different types of graphs and applications. Understanding the specific use cases, graph characteristics, and performance requirements of an application is key to choosing the best algorithm. While Dijkstra's algorithm is typically the preferred choice for its speed and efficiency in non-negative graphs, Bellman-Ford provides greater flexibility when dealing with more complex graph structures that involve negative weights or cycle detection.

References

geeksforgeeks.org (2024, February 9) *Time and Space Complexity of Bellman-Ford's Algorithm.* https://www.geeksforgeeks.org/time-and-space-complexity-of-bellman-ford-algorithm/

geeksforgeeks.org (2024, February 9) *Time and Space Complexity of Dijkstra's Algorithm.* https://www.geeksforgeeks.org/time-and-space-complexity-of-dijkstras-algorithm/

geeksforgeeks.org (2023, September 4) *Detect a Negative Weight Cycle in a Graph | (Bellman Ford).* https://www.geeksforgeeks.org/detect-negative-cycle-graph-bellman-ford/

Pizzo, B., (2024) *CSC506: Design and Analysis of Algorithms. 5.11 Heaps.* zyBooks. https://learn.zybooks.com/zybook/CSC506-1_5/chapter/5/section/11

Sryheni, S., (2024, March 18), *Dijkstra's vs Bellman-Ford Algorithm.* Baeldung. https://www.baeldung.com/cs/dijkstra-vs-bellman-ford