AOA
Johnson's Algorithm

Zain Ali

57113

BSCS 4-1

Instructor: Usman Shareef

Riphah International University

Johnson’s Algorithm for All-Pairs Shortest Path Problem

Introduction:

Johnson's Algorithm is a powerful method to compute the shortest paths between all pairs of nodes in a graph, even when some edges have negative weights (but no negative weight cycles). The algorithm handles this challenge by first using Bellman-Ford to check for negative cycles and re-weighting the graph, followed by Dijkstra's Algorithm for efficient calculation of shortest paths.

Core Concepts and Mechanisms:

- **Graph Representation:**
    - The graph is represented as a collection of nodes and edges, where each edge has a weight (positive or negative). A new node is added to the graph to aid in re-weighting and to help with the **Bellman-Ford** procedure.
- **Bellman-Ford Algorithm:**

    - Used to detect negative weight cycles and compute the shortest paths from the newly added node (a source node) to all other nodes.

- **Re-weighting Process:**
    - After running Bellman-Ford, the graph is re-weighted so that all edge weights become non-negative. The formula used is:
 new_weight(u,v)=weight(u,v)+h[u]−h[v]\text{new_weight}(u, v) = \text{weight}(u, v) + h[u] - h[v]new_weight(u,v)=weight(u,v)+h[u]−h[v]
 Where h[u]h[u]h[u] is the shortest path from the source node to uuu.

- **Dijkstra's Algorithm:**
    - Once the graph is re-weighted, **Dijkstra's algorithm** is applied from each node to calculate the shortest path from that node to every other node in the graph.

Algorithm Phases:

- **Graph Augmentation:**
    - A new node is added, connected to all other nodes with edges of weight zero.

- **Bellman-Ford Execution:**

- The **Bellman-Ford algorithm** is executed from the newly added node to compute the shortest path distances to all other nodes, ensuring no negative cycles.

- **Re-weighting of the Graph:**
    - Based on the results from Bellman-Ford, the graph edges are re-weighted to ensure all edge weights are non-negative.

- **Dijkstra's Execution:**
    - **Dijkstra's algorithm** is executed from each node in the graph to compute the shortest paths to every other node.

Convergence Properties:

- **Asymptotic Convergence:**
    - As the number of iterations approaches infinity, the probability of finding the global optimum solution approaches 1, provided the algorithm maintains a proper balance between exploration (via the somersault process) and exploitation (via climbing).

- **Exploration vs. Exploitation:**
    - The algorithm balances between **exploring** new areas of the search space and **exploiting** previously discovered good solutions to refine the search.

Time Complexity:

The time complexity of Johnson’s Algorithm can be analyzed as follows:

- **Initialization:** O(n×d+n×f)O(n \times d + n \times f)O(n×d+n×f)
- **Per iteration:**
    - Climb process: O(n×d×f)O(n \times d \times f)O(n×d×f)
    - Watch-jump process: O(n×f)O(n \times f)O(n×f)
    - Somersault process: O(n×d+n×f)O(n \times d + n \times f)O(n×d+n×f)

The overall complexity is:

O(n×d+n×f+t×(n×d×f+n×f+n×d+n×f))O(n \times d + n \times f + t \times (n \times d \times f + n \times f + n \times d + n \times f))O(n×d+n×f+t×(n×d×f+n×f+n×d+n×f))

Where:

- nnn is the population size (number of nodes)
- ddd is the dimension of the problem (number of nodes)
- fff is the cost of a single objective function evaluation
- ttt is the number of iterations

Search Space Characteristics:

Johnson's Algorithm performs well in a variety of situations based on the characteristics of the search space:

- **Modality:**
    - Works well for both **unimodal** and **multimodal** landscapes. In highly multimodal landscapes, a larger population size is beneficial for better exploration.
- **Separability:**
    - Effective for separable functions but may struggle with non-separable problems where variables interact.
- **Ruggedness:**
    - Johnson's Algorithm is robust against rugged functions with many local optima due to its combination of exploration and exploitation techniques.
- **Deceptiveness:**
    - The algorithm can handle deceptive problems (where local optima might mislead the search) due to its global exploration process.

Extensions and Variants:

- **Adaptive Parameter Control:**
  - Dynamically adjusts parameters (like climbing step and somersault range) to refine the search strategy over time.
- **Hybrid Approaches:**
  - Combines Johnson's Algorithm with other local search methods or gradient-based approaches for faster convergence.
- **Constraint Handling:**
  - Uses penalty functions for soft constraints or repair mechanisms for hard constraints.
- **Parallel Implementation:**
  - Utilizes a parallel approach where multiple populations explore different parts of the solution space concurrently, speeding up the process.
- **Multi-objective Extensions:**
  - Adapted to handle problems with multiple objectives, where the algorithm maintains a set of non-dominated solutions (Pareto front).

Applications:

Johnson’s Algorithm is widely used for solving **shortest path problems** in various fields such as:

- **Routing in Networks (Telecommunications)**: Optimizing communication paths between nodes in a network.
- **Logistics and Supply Chain**: Finding optimal transportation routes for goods between multiple points.
- **Urban Planning**: Identifying the shortest paths between different locations in a city.
- **Robotics**: Pathfinding for robot navigation in unknown environments with obstacles.

This summary of **Johnson's Algorithm** highlights the algorithm's core ideas, its efficiency in handling negative weights, and its application across a variety of optimization problems. You can adapt the algorithm to different scenarios where shortest path computations are needed, even in complex and large-scale networks.