CENG 319 - Algorithm Analysis

Semester Project Report

Project Title: Shortest Path Finding with Constraints in US Backbone Topology

Project Group:

- Ahmed Albarazanchi – 210212309
- Khaled Al-sayyad - 210212338
- Khaled Hussein - 200212314

Project Delivery Deadline: January 11, 2024 at 23:59

1. Introduction

The project aims to develop and assess alternative methods for determining the shortest path in the US backbone architecture, considering limitations such as bandwidth, delay, and dependability. The technique, implementation specifics, and algorithm evaluation are presented in this study.

Numerous strategies for resolving network optimization issues are implemented in Python code. These techniques are used on a graph-representation of a network, where nodes stand for network elements and edges for connections with bandwidth allocated to them. The goal is to find the shortest path between a source and a destination node while considering constraints such as bandwidth, delay, and reliability.

2. Problem Description

- **Input Data:** The adjacency, bandwidth, latency, and reliability matrices are included in the US backbone topology input file. Within designated ranges, link lengths, bandwidth, latency, and reliability metrics are consistently distributed.
- **Request Format:** The source node ID, destination node ID, and bandwidth demand are all included in a request.

4. Solution Approach

- **Objective Function:** Minimize ��×∑���×�������*bw*×∑*δij*×*distij* where ���*δij* is 1 if edge (i, j) is used, 0 otherwise, and ������*distij* is the edge distance.
- **Algorithms:** Utilized Dijkstra Algorithm, Bellman-Ford Algorithm, A* Algorithm,

5. Implementation

- **Code Structures:** The report shares relevant code structures that were used in the coding process rather than including all code.

6. Results

Code Overview:

1. **Input Reading:**
    - The code reads input from a file, constructing an adjacency matrix for the graph.
    - Bandwidth, delay, and reliability matrices are generated with random values.
2. **Graph Construction:**
    - A NetworkX graph is created from the adjacency matrix, with edge attributes representing bandwidth.
3. **Shortest Path Algorithms:**
    - Dijkstra's, Bellman-Ford, A*, and Floyd-Warshall algorithms are implemented.
    - A* heuristic function considers node degree and random components.
4. **Metaheuristic Algorithms:**
    - Simulated Annealing, Tabu Search, Ant Colony, Bee Colony, and Firefly algorithms are implemented.
    - These algorithms aim to find optimal or near-optimal solutions through iterative exploration.
5. **Solution Construction and Update:**
    - Functions for constructing and updating solutions in metaheuristic algorithms are defined.
6. **Main Function:**
    - Reads input, constructs the graph, and applies various algorithms.
    - Prints the shortest path obtained by each algorithm.

Algorithm Examples:

- **Simulated Annealing:**
    - Utilizes a temperature-based approach for exploring solution space.
    - Adjusts the solution based on a probability function.
- **Tabu Search:**

    - Employs a tabu list to avoid revisiting previous solutions.
    - Iteratively explores the solution space, updating based on neighborhood solutions.
- **Ant Colony Optimization:**
    - Models the algorithm on ant foraging behavior.
    - Utilizes pheromone trails to guide exploration, updating based on solution quality.
- **Bee Colony Optimization:**
    - Employs scout bees and best bees to explore the solution space.
    - Iteratively updates the best solution based on neighboring solutions.
- **Firefly Algorithm:**
    - Models the algorithm on the flashing behavior of fireflies.
    - Fireflies move toward brighter ones, representing better solutions.

Code Structure Details:

1. **Input Reading:**
    - The **read_input** function reads the network information from a file, generating matrices for adjacency, bandwidth, delay, and reliability.
2. **Graph Construction:**
    - NetworkX is utilized for graph representation (**my_graph**).
    - Nodes are added to the graph, and edges are created based on the adjacency matrix.
    - Bandwidth information is assigned to edges as attributes.
3. **Shortest Path Algorithms:**
    - Dijkstra's, Bellman-Ford, A*, and Floyd-Warshall algorithms are implemented for finding the shortest path.
    - Heuristic functions and constraints are considered in A* and Bellman-Ford.
4. **Metaheuristic Algorithms:**
    - Simulated Annealing, Tabu Search, Ant Colony, Bee Colony, and Firefly algorithms are implemented.
    - Solution construction and update functions are defined (**get_neighbor_solution**, **update_pheromones**, **move_firefly**).

5. **Cost Calculation:**
    - The **calculate_cost** function computes the total cost of a path, considering bandwidth constraints.
6. **Main Function:**
    - The **main** function orchestrates the execution.
    - The input is read, the graph is constructed, and various algorithms are applied, with results printed.
7. **Algorithm Examples:**
    - Each algorithm is implemented as a separate function (**simulated_annealing_algorithm**, **tabu_search_algorithm**, etc.).
    - The solutions obtained from these algorithms are printed in the main function.
8. **Random Solution Modification:**
    - **get_neighbor_solution** function randomly modifies the current solution for metaheuristic algorithms.
9. **Heuristic Function:**
    - **heuristic_function** calculates the heuristic value for A* based on node degree and a random component.
10. **Solution Cost Calculation:**
    - **calculate_cost** computes the total cost of a solution based on the bandwidth attribute of the edges.
11. **Temperature and Cooling Rate in Simulated Annealing:**
    - Temperature and cooling rate parameters control the exploration strategy in simulated annealing.
12. **Tabu Search Parameters:**
    - Iterations, tabu size, and solution exploration strategies are defined in the tabu search function.
13. **Ant Colony Optimization Parameters:**
    - Number of ants, pheromone decay, and weights for pheromone and visibility are specified in ant colony optimization.
14. **Bee Colony Optimization Parameters:**
    - Number of scout bees, best bees, best sites, selected sites, elite sites, and maximum epochs are set in bee colony optimization.

15. **Firefly Algorithm Parameters:**

    - Number of fireflies, maximum iterations, attractiveness base, beta, alpha, and gamma parameters are defined for firefly algorithm.

Conclusion: An extensive collection of algorithms for resolving network optimization issues is provided by the code. Every algorithm has a different strategy and set of benefits. The network's unique requirements and the intended trade-offs between computing efficiency and optimality determine which algorithm is best. Researchers and practitioners working on network optimization projects can benefit greatly from the code.

The orderly and modular structure of the code makes it easier to comprehend and maintain. Dedicated functions handle key functions such reading input, building graphs, implementing algorithms, and updating solutions. Because of the code's clear structure, developers can more easily comprehend and change particular parts to suit their needs.