VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

HO CHI MINH UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Discrete Structure (CO1007)

Assignment Report

BELLMAN-FORD ALGORITHM

Mentor:	Nguyen An Khuong
	Mai Xuan Toan
	Nguyen Tien Thinh
Student:	Trần Hà Tuấn Kiệt – 2011493

Ho Chi Minh City, 6/2024

Table of Contents

List of Figures

List of Tables

1 Introduction

Graph algorithms are essential in solving problems related to networks, routing, and resource allocation. The Bellman-Ford algorithm provides a robust method for finding shortest paths in graphs that may have negative weights. The Traveling Salesman Problem (TSP), on the other hand, poses a challenge in combinatorial optimization, seeking the shortest possible route visiting each city and returning to the origin city.

2 Bellman-Ford Algorithm

2.1 Description

The Bellman-Ford algorithm computes shortest paths from a single source vertex to all other vertices in a weighted digraph. It is capable of handling graphs with negative weight edges, unlike Dijkstra's algorithm.

2.2 Complexity Analysis

The time complexity of Bellman-Ford is $O(VE)$, where V is the number of vertices and E is the number of edges. Its space complexity is $O(V)$, as it maintains a distance array.

2.3 Bellman-Ford Path Calculation

Path reconstruction is carried out using the predecessor array filled during the execution of the Bellman-Ford algorithm. By tracking predecessors from the destination node back to the source, one can determine the complete path traversed.

3 Traveling Salesman Problem

3.1 Introduction

The Traveling Salesman Problem (TSP) presents a classic dilemma in combinatorial optimization and operations research, asking for the shortest possible route that visits each city exactly once and returns to the origin city. The problem is classified as NP-hard, which implies that no efficient algorithm exists that can solve all instances of TSP within polynomial time. This makes it an ideal benchmark for many optimization techniques, both exact and heuristic, in fields ranging from logistics to DNA sequencing, where the goal is often to minimize travel distance or cost.

3.2 Theoretical Significance

Understanding TSP helps illuminate the complexity inherent in many real-world optimization problems. It also serves as a gateway to discussing other computational problems in NP, such as the Hamiltonian cycle problem, and provides insight into the theory of computational complexity itself. Its study has helped develop foundational concepts in algorithm design and complexity theory, including the introduction of approximation algorithms for NP-hard problems.

3.3 Genetic Algorithm for TSP

Genetic Algorithms (GAs) are inspired by Darwinian principles of natural selection and genetics, and they have proven to be particularly adept at finding good solutions to TSP with relatively low computational costs compared to exhaustive search methods[2]. They simulate the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

3.4 Implementation

GAs for TSP start with a population of possible solutions (tours) represented as sequences of city visits. These solutions undergo processes analogous to natural genetic variation[1]:

- **Selection**: Tours are selected for reproduction based on their fitness, which is typically inversely proportional to their travel cost.
- **Crossover**: Selected tours are recombined to produce new offspring tours, hoping to combine advantageous traits from each parent.

– **Mutation**: To maintain genetic diversity within the population and to explore a wider solution space, tours are randomly altered.

Iterative application of these processes results in the evolution of tours that tend to be more fit over time.

3.5 Pseudocode

Provide detailed pseudocode for a Genetic Algorithm applied to TSP. The following pseudocode outlines the process of evolving a population of tours to find an approximate solution to the TSP:

```
function GeneticAlgorithm()
    Initialize population with random tours
    Evaluate fitness of each tour
    Repeat
        Selection: Choose the fitter tours for reproduction
        Crossover: Combine parts of two tours to create new tour
        Mutation: Randomly alter a tour's path
        Evaluate new fitness: Assess the quality of new tours
    Until termination condition met
    (e.g., number of generations, time limit)
    return best tour found
end function
```

Each step of the Genetic Algorithm is detailed below:

Selection

The selection process preferentially chooses individuals from the current population based on their fitness scores.

```
Function Selection(Population, FitnessScores)
    Selected = []
    For each tour in Population
```

```
        Probability = FitnessScore of tour / Sum of all FitnessScores
        If random number < Probability
            Add tour to Selected
        End If
    End For
    Return Selected
End Function
```

Crossover

Crossover is a genetic operator used to combine the genetic information of two parents to generate new offspring.

```
Function Crossover(Parent1, Parent2)
    Child = New Tour()
    SplitPoint = Random number between 1 and length of Tour - 1
    Copy first SplitPoint genes from Parent1 to Child
    Fill remaining genes in Child with genes from Parent2
not already in Child maintaining order
    Return Child
End Function
```

Mutation

Mutation introduces small random changes in the offspring, providing genetic diversity.

```
Function Mutation(Tour, MutationRate)
    For each gene in Tour
        If random number < MutationRate
            Swap this gene with another random gene in the Tour
        End If
    End For
    Return Tour
End Function
```

Evaluate New Fitness

After mutation and crossover, the fitness of new solutions must be evaluated.

```
Function EvaluateFitness(Tour)
    TotalDistance = 0
    For i from 1 to length of Tour - 1
        TotalDistance += Distance between Tour[i] and Tour[i+1]
    End For
    TotalDistance += Distance between Tour[end] and Tour[begin]
    Fitness = 1 / TotalDistance
    Return Fitness
End Function
```

3.6 Results Discussion

The performance of Genetic Algorithms in solving the TSP can vary significantly with changes in parameters such as population size, mutation rate, and selection method. While GAs do not guarantee an optimal solution, they are highly effective at producing very good solutions within reasonable time limits, particularly for very large instances where other methods become impractical. Their effectiveness also hinges on the proper tuning of parameters and adaptation of the algorithm to specific characteristics of the problem instance.

4 Evaluation

4.1 How to Compile and Run the Program

To compile and run the program, follow the steps outlined below. These instructions assume you have a Makefile configured with the necessary commands to compile the program.

Compiling the Program

To compile the program, open a terminal in the directory containing your source files and Makefile, and execute the following command:

```
make -s
```

This command calls ‘make‘ with the ‘-s‘ (silent) option to suppress the display of commands being executed, making the output cleaner.

Running the Program

Once the compilation is successful, you can run the compiled program by entering the following command in the terminal:

```
./main
```

This command executes the ‘main‘ executable that was generated by the compilation process. Ensure that you have the appropriate permissions to execute the file, and that the current working directory is where the ‘main‘ executable is located.

4.2 Test Results

0	17	40	100	32	11	29	77	73	53	52	72
48	0	23	42	81	5	12	37	31	55	18	66
48	97	0	74	21	43	8	37	48	13	55	88
79	53	66	0	8	97	38	61	48	76	75	71
18	56	42	97	0	59	40	51	77	5	65	39
45	86	82	20	23	0	96	100	44	84	45	97
15	53	93	53	80	7	0	29	21	78	13	43
87	9	1	26	26	44	98	0	92	49	9	77
30	29	66	93	28	9	76	73	0	73	91	92
31	9	38	38	38	25	83	17	68	0	69	93
35	61	19	45	24	10	94	33	53	90	0	62
19	82	57	95	57	96	67	13	54	65	90	0

TSP

Parameters: Population size = 100, Mutation Rate = 0.05, Generations = 2000

Results:

```
Running TSP Solver:
TSP Path: IFDEJABKLHCGI
TSP Path Cost: 213
Accuracy: 95.5882%
```

Parameters: Population size = 200, Mutation Rate = 0.05, Generations = 2000

Results:

```
Running TSP Solver:
TSP Path: GIAFDEJBKLHCG
TSP Path Cost: 206
Accuracy: 99.0196%
```

Parameters: Population size = 200, Mutation Rate = 0.05, Generations = 10000

Results:

```
Running TSP Solver:
TSP Path: HKCJBGIAFDELH
TSP Path Cost: 204
Accuracy: 100%
```

Parameters: Population size = 200, Mutation Rate = 0.07, Generations = 2000

Results:

```
Running TSP Solver:
TSP Path: GIAFDELHKCJBG
TSP Path Cost: 204
Accuracy: 100%
```

4.3 Limitations and Potential Improvements

Despite their advantages, GAs can suffer from premature convergence to suboptimal solutions and can be sensitive to the initial population and parameter settings. Ongoing research aims to improve the robustness and efficiency of GAs by integrating advanced techniques such as hybrid algorithms that combine GAs with local search methods, adaptive parameter tuning, and machine learning strategies to predict and adjust evolutionary dynamics dynamically.

5 Conclusion

The Bellman-Ford algorithm's ability to handle negative weights makes it a powerful tool for graph analysis. Meanwhile, GAs offer a robust heuristic method for tackling NP-hard problems like the TSP, especially in large datasets where exact methods are computationally impractical.

References

[1] Ra'ed M. Al-Khatib, Mohammed Al-Betar, Mohammed Awadallah, Khalid Nahar, Mohammed Abu Shquier, Ahmad Manasrah, and Ahmad Doumi. Mga-tsp: Modernized genetic algorithm for the traveling salesman problem. *International Journal of Reasoning-based Intelligent Systems*, 11:1, 01 2019. doi: 10.1504/IJRIS.2019.10019776.

[2] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384, 2019. doi: 10.1109/COMITCon.2019.8862255.