# A Review And Evaluations Of Shortest Path Algorithms

Kairanbay Magzhan, Hajar Mat Jani

**Abstract**: Nowadays, in computer networks, the routing is based on the shortest path problem. This will help in minimizing the overall costs of setting up computer networks. New technologies such as map-related systems are also applying the shortest path problem. This paper's main objective is to evaluate the Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, and Genetic Algorithm (GA) in solving the shortest path problem. A short review is performed on the various types of shortest path algorithms. Further explanations and implementations of the algorithms are illustrated in graphical forms to show how each of the algorithms works. A framework of the GA for finding optimal solutions to the shortest path problem is presented. The results of evaluating the Dijkstra's, Floyd-Warshall and Bellman-Ford algorithms along with their time complexity conclude the paper.

**Index Terms**: Bellman-Ford Algorithm, Computer Networks, Dijkstra's Algorithm, Floyd-Warshall Algorithm, Genetic Algorithm (GA), Shortest Path Problem.

## 1 INTRODUCTION

THE shortest path problem is a problem of finding the shortest path or route from a starting point to a final destination. Generally, in order to represent the shortest path problem we use graphs. A graph is a mathematical abstract object, which contains sets of vertices and edges. Edges connect pairs of vertices. Along the edges of a graph it is possible to walk by moving from one vertex to other vertices. Depending on whether or not one can walk along the edges by both sides or by only one side determines if the graph is a directed graph or an undirected graph. In addition, lengths of edges are often called weights, and the weights are normally used for calculating the shortest path from one point to another point. In the real world it is possible to apply the graph theory to different types of scenarios. For example, in order to represent a map we can use a graph, where vertices represent cities and edges represent routes that connect the cities. If routes are one-way then the graph will be directed; otherwise, it will be undirected. There exist different types of algorithms that solve the shortest path problem. However, only several of the most popular conventional shortest path algorithms along with one that uses genetic algorithm are going to be discussed in this paper, and they are as follows:

1. Dijkstra's Algorithm
2. Floyd-Warshall Algorithm
3. Bellman-Ford Algorithm
4. Genetic Algorithm (GA)

Other than GA, nowadays, there are also many intelligent shortest path algorithms that have been introduced in several past research papers. For example, the authors in [1] used a heuristic method for computing the shortest path from one point to another point within traffic networks. They proposed a "new dynamic direction restricted algorithm obtained by extending the Dijkstra's algorithm [1]." In another paper [2], a heuristic GA was used for solving the single source shortest path (SSSP) problem. Its main goal was to investigate the SSSP problem within the Internet routing setting, particularly when considering the cost of transmitting messages/packets is significantly high, and the search space is normally very large. In a paper by Li, Qi, and Ruan [3], an efficient algorithm named Li-Qi (LQ) was proposed for the SSSP problem with the objective of finding a simple path of the smallest total weights from a specific initial or source vertex to every other vertex within the graph. The ideas of the queue and the relaxation form the basis of this newly introduced algorithm; the vertices may be queued several times, and furthermore, only the source vertex and relaxed vertices are being queued [3].

- Kairanbay Magzhan is currently pursuing bachelor's degree program in Computer Science and Software Engineering at International IT University, Kazakhstan. E-mail: magzhan.kairanbay@gmail.com
- Dr. Hajar Mat Jani is currently a Senior Lecturer at Universiti Tenaga Nasional, Malaysia. E-mails: hajar@uniten.edu.my, hajarmj@gmail.com.

## 2 RESEARCH OBJECTIVES

The following list gives the objectives of this research paper:

- To determine and identify the concepts of the shortest path problem.
- To determine the representation of graphs in computer in order to solve the shortest path problem, as well as to understand the different basic terms of a graph.
- To explain the general concepts and the implementations of Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, and Genetic Algorithm.
- To evaluate each algorithm, and presents the evaluations' results.

## 3 LITERATURE REVIEW

As mentioned earlier, a graph can be used to represent a map where the cities are represented by vertices and the routes or roads are represented by edges within the graph. In this section, a graph representation of a map is explained further, and brief descriptions and implementations of the four shortest path algorithms being studied are presented.

### 3.1 Representation of the Graph

In order to represent a graph in a computer we will use adjacency matrix *a*. The dimension of the matrix will be equal to *(n* x *n)*, where *n* is number of vertices in graph. The element of matrix *a[i][j]* is identified by an edge that connects the *i-th* and *j-th* vertices; the value here represents the weight of the corresponding edge. However, if there is no edge

between vertices *i* and *j*, the value in (*a[i][j]*) will be equal to *infinity*. An array of edges is another common representation of the graph. If *m* is the number of edges in a graph, then in order to represent the graph we have to use *m* x *3* two-dimensional arrays; in each row, the first vertex, the second vertex, and the edge that connects them are also stored. The benefit of using an array of edges in comparison to adjacency matrix is when there is more than one edge that connects two vertices we cannot use adjacency matrix in order to represent graph.

### 3.2 Dijkstra's Algorithm: Explanation and Implementation

For each vertex within a graph we assign a label that determines the minimal length from the starting point *s* to other vertices *v* of the graph. In a computer we can do it by declaring an array *d[]*. The algorithm works sequentially, and in each step it tries to decrease the value of the label of the vertices. The algorithm stops when all vertices have been visited. The label at the starting point *s* is equal to zero (*d[s]=0*); however, labels in other vertices *v* are equal to *infinity* (*d[v]=∞*), which means that the length from the starting point *s* to other vertices is unknown. In a computer we can just use a very big number in order to represent *infinity*. In addition, for each vertex *v* we have to identify whether it has been visited or not. In order to do that, we declare an array of *Boolean* type called *u[v]*, where initially, all vertices are assigned as unvisited (*u[v] = false*). The Dijkstra's algorithm consists of *n* iterations. If all vertices have been visited, then the algorithm finishes; otherwise, from the list of unvisited vertices we have to choose the vertex which has the minimum (smallest) value at its label (At the beginning, we will choose a starting point *s*). After that, we will consider all neighbors of this vertex (Neighbors of a vertex are those vertices that have common edges with the initial vertex). For each unvisited neighbor we will consider a new length, which is equal to the sum of the label's value at the initial vertex v (*d[v]*) and the length of edge *l* that connects them. If the resulting value is less than the value at the label, then we have to change the value in that label with the newly obtained value [4].

$$d\,[\,neighbors\,] = min\,(\,d\,[\,neighbors\,]\,,\;\; d[\,v\,]\;+\;l\,) \qquad (1)$$

After considering all of the neighbors, we will assign the initial vertex as visited (*u[v] = true*). After repeating this step *n* times, all vertices of the graph will be visited and the algorithm finishes or terminates. The vertices that are not connected with the starting point–will remain by being assigned to *infinity*. In order to restore the shortest path from the starting point to other vertices, we need to identify array *p []*, where for each vertex, where $v \neq s$, we will store the number of vertex *p[v]*, which penultimate vertices in the shortest path. In other words, a complete path from *s* to *v* is equal to the following statement [5]

$$P = (\,s\,,\,\ldots\,,\,p\,[\,p\,[\,p\,[\,v\,]\,]\,]\,,\,p\,[\,p\,[\,v\,]\,]\,,\,p\,[\,v\,]\,,\,v\,) \qquad (2)$$

Fig. 1 shows an excerpt of the Dijkstra's algorithm, which is written in Java.

```
for (int i = 0; i < n; i++) {
    // vertex
    int v = -1;
    // finding minimum label among unvisited vertices
    for (int j = 0; j < n; j++) {
        if (u[j] == false && (v == -1 || d[j] < d[v])) {
            v = j;
        }
    }
    // set as visited.
    u[v] = true;

    for (int j = 0; j < n; j++) {
        // if there is exists path between v and j vertices
        if (a[v][j] > 0) {
            if (d[v] + a[v][j] < d[j]) {
                d[j] = d[v] + a[v][j];
            }
        }
    }
}
```

Fig. 1. Implementation in Java: An Excerpt

### 3.3 Floyd-Warshall Algorithm: Explanation and Implementation

Consider the graph *G*, where vertices were numbered from *1* to *n*. The notation $d_{ijk}$ means the shortest path from *i* to *j*, which also passes through vertex *k*. Obviously if there is exists edge between vertices *i* and *j* it will be equal to $d_{ij0}$, otherwise it can assigned as *infinity*. However, for other values of $d_{ijk}$ there can be two choices: (1) If the shortest path from *i* to *j* does not pass through the vertex *k* then value of $d_{ijk}$ will be equal to $d_{ijk-1}$. (2) If the shortest path from *i* to *j* passes through the vertex *k* then first it goes from *i* to *k*, after that goes from *k* to *j*. In this case the value of $d_{ijk}$ will be equal to $d_{ikk-1} + d_{kjk-1}$. And in order to determine the shortest path we just need to find the minimum of these two statements [6]:

$$d_{ij0} = \textit{the length of edge between vertices i and j} \qquad (3)$$

$$d_{ijk} = min\,(d_{ijk-1},\, d_{ikk-1} + d_{kjk-1}) \qquad (4)$$

Fig. 2 shows an excerpt of the Floyd-Warshall algorithm, which is written in Java.

```
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (d[i][j] > d[i][k] + d[k][j]) {

                /*
                * d[i][j] -  is equal to
                * the shortest path from i-th to j-th vertices.
                */
                d[i][j] = d[i][k] + d[k][j];
            }
        }
    }
}
```

Fig. 2. Implementation in Java: An Excerpt

### 3.4 Bellman-Ford Algorithm: Explanation and Implementation

In comparison to Dijkstra's algorithm, the Bellman-Ford algorithm admits or acknowledges the edges with negative weights. That is why, a graph can contain cycles of negative weights, which will generate numerous number of paths from

the starting point to the final destination, where each cycle will minimize the length of the shortest path. Taking into consideration this fact let's assume that our graph does not contain cycles with negative weights. The array *d[]* will store the minimal length from the starting point *s* to other vertices. The algorithm consists of several phases, where in each phase it needs to minimize the value of all edges by replacing *d[b]* to following statement *d[a]* + *c*; *a* and *b* are vertices of the graph, and *c* is the corresponding edge that connects them. And in order to calculate the length of all shortest paths in a graph it requires *n – 1* phases, but for those vertices of a graph that are unreachable, the value of elements of the array will remain by being assigned to *infinity* [**7**]. Fig. 3 shows an excerpt of the Bellman-Ford algorithm, which is written in Java.

```
// the distance from start point to inself
d[0] = 0;

for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < m; j++) {
        // if less that infinity (30000)
        if (d[a[j][0] - 1] < 30000) {

            d[a[j][1] - 1] =
            Math.min(d[a[j][1] - 1], d[a[j][0] - 1] + a[j][2]);
        }
    }
}
```

Fig. 3. Implementation in Java: An Excerpt

### 3.5 Genetic Algorithm (GA)

Intelligent algorithms have been introduced in finding optimal shortest paths in many situations that require the systems to search through a very large search space within limited time frame and also in accommodating an ever-changing environment. One of these algorithms is GA. By definition, genetic algorithms are a class or group of "stochastic search algorithms" that are based on biological evolution [8]. GA is mostly used for optimization problems. It uses several genetic operations such as selection, crossover, and mutation in order to generate a new generation of population, which represents a set of solutions (chromosomes) to the current problem. In addition, on average, this new generation is supposed to be better in terms of their overall fitness value as compared to the previous population. Each individual or chromosome within the population will be assigned a fitness value, which is calculated based on a pre-determined fitness function that measures how optimal its solution is in solving the current problem. In order to solve the shortest path problem using the GA [9], we need to generate a number of solutions, and then choose the most optimal one among the provided set of possible solutions. In order to solve the problem, an initial population that forms the first set of chromosomes to be used in the GA is randomly created. Each chromosome represents one possible solution to the current problem at hand. After that, they (chromosomes) are estimated using certain fitness function, which determines how well the solutions are. Taking into account the fitness value of each solution or chromosome, some chromosomes or individuals will be selected (selection operation), and the basic genetic operations such as crossover and mutation are applied on these chromosomes. Then, the fitness value of each chromosome is re-calculated, and the best solutions are selected to be considered for the next generation. This process continues until the criteria of the given problem will not be achieved. Thus we can identify the following stages of a GA:

- ***Step 1:*** Determine the fitness function; in our case we need to maximize the following function $f(Ch_k) = (\sum edge)-1$, where $Ch_k$ is *k-th* chromosome and $\sum edge$ is the sum of edges from starting point to final destination.
- ***Step 2:*** Create initial population – a population that contains *n* individuals. At this stage we do not need to create fittest individuals, because it is probable that GA will transfer them into viable population. In order to create chromosomes for initial population, we will produce random paths from the starting point to final destination.
- ***Step 3:*** Selection – the stage of GA that is used to select two chromosomes for genetic operations such as crossover and mutation. There are different types of selection methods; however, the *Roulette Wheel* selection method is chosen in order to solve the shortest path problem.
- ***Step 4:*** Crossover – the process of reproduction where descendants are inherit traits of both parents mixing them in some way. Individuals for reproduction will be chosen from whole population (not from the survivors in the first iteration), because we need to keep diversity of individuals, otherwise entire population will be hammered with single copies of one individual. There exist different types of crossover methods; however, for our problem we will use the simplest method, which is called single point crossover.
- ***Step 5:*** Mutation – the act of changing the value of some gene. Mutation keeps the genetic diversity of the population by changing genes of selected chromosome.

If the fittest chromosome does not change after a specific number of iterations, which was described above, then the algorithm will terminate; the most optimal solution is automatically the fittest chromosome among the whole population. Fig. 4 illustrates the flowchart of the shortest path problem's solution using GA. This diagram is a framework that will be used for the implementation of GA in finding the shortest path or route in a given map of a city named Almaty in Kazakhstan, which is part of our current and future works. The GA used here is quite general in nature except for the part where loops might be introduced while executing GA in finding the most optimal solution; all loops must be removed because loops must not exist in a path.

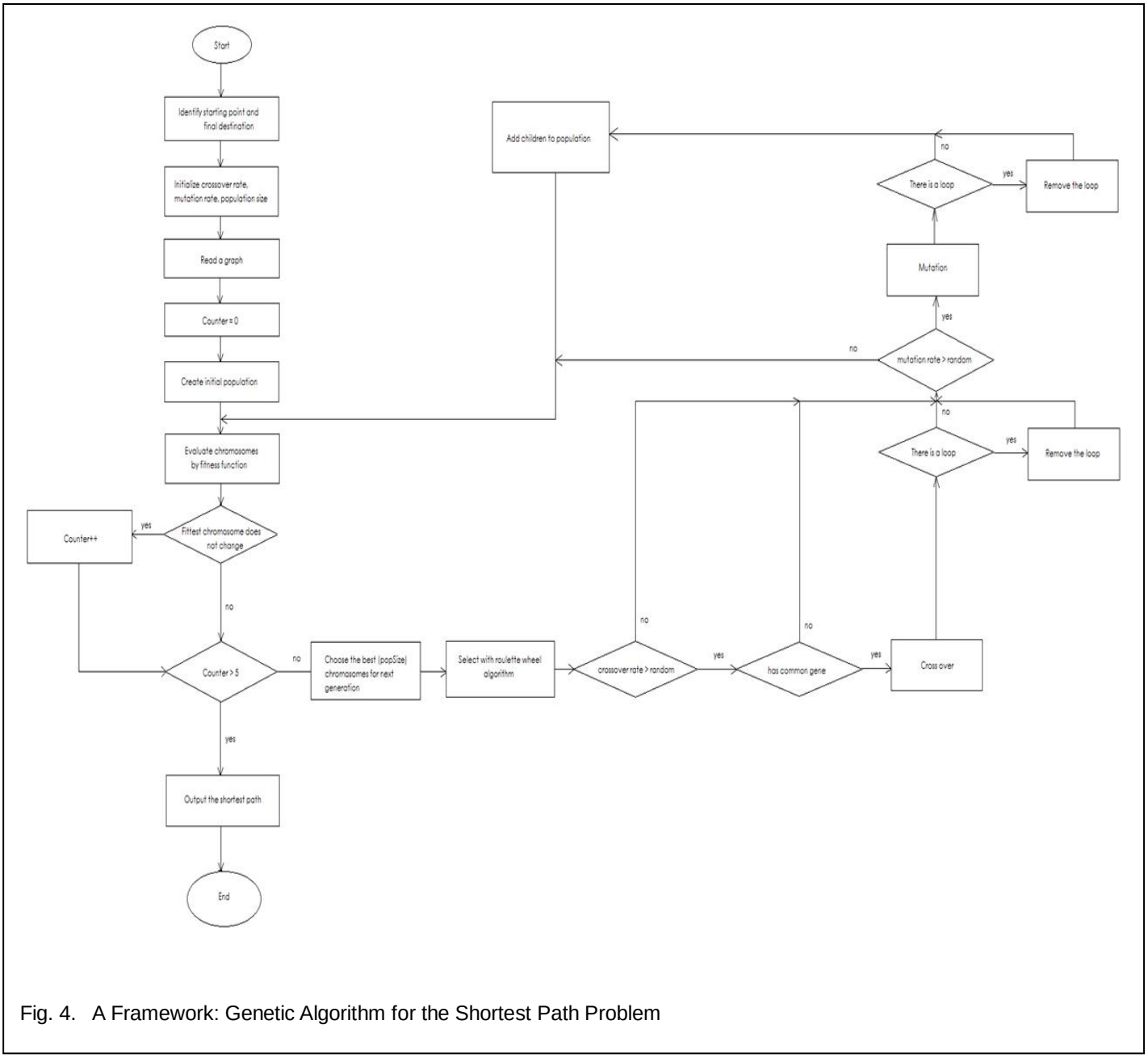


Fig. 4. A Framework: Genetic Algorithm for the Shortest Path Problem

## 4 RESULTS

### 4.1 Test Results

Dijkstra's, Floyd-Warshall and Bellman-Ford algorithms were tested using pre-defined test cases and automated checking system available in the websites [10][11][12]. In the following figures (Fig. 5, Fig. 6, Fig. 7) some information such as the number of test cases, results, author, date of submission, and programming language used for each algorithm are provided.

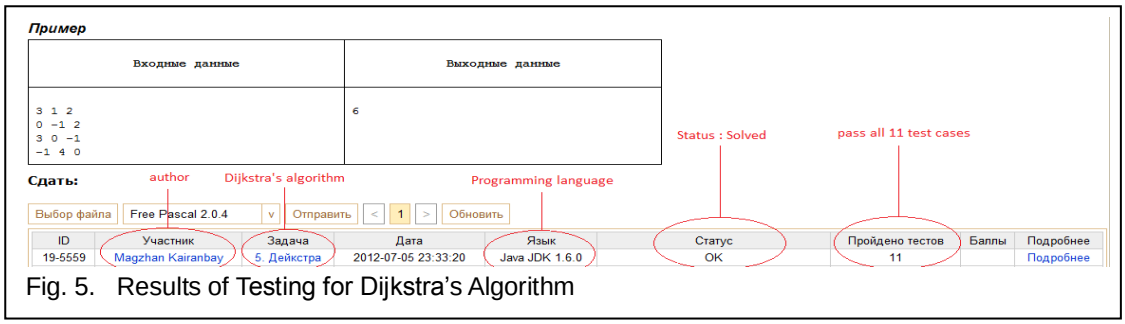


Fig. 5. Results of Testing for Dijkstra's Algorithm

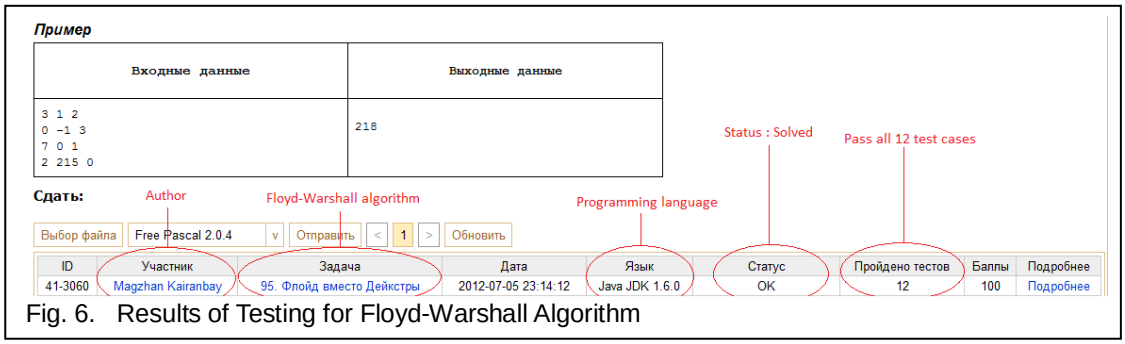


Fig. 6. Results of Testing for Floyd-Warshall Algorithm

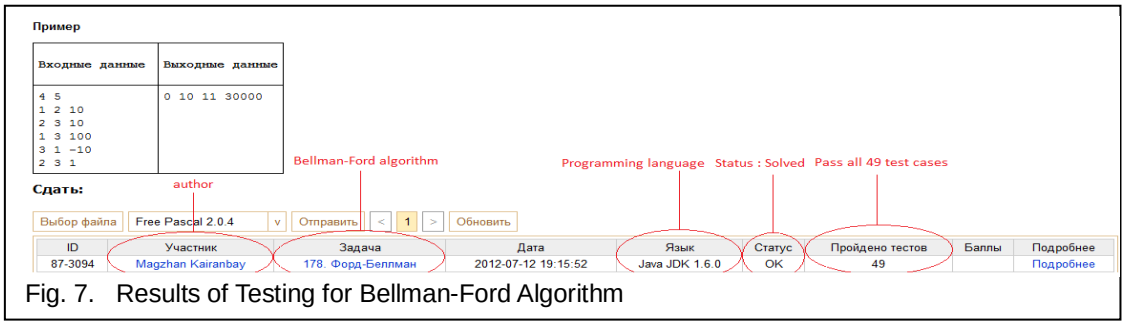


Fig. 7. Results of Testing for Bellman-Ford Algorithm

### 4.2 The Time Complexity

The time complexity for each algorithm is illustrated in Table I; *n* represents the total number of vertices, and *m* is the total number of edges.

**TABLE 1**
TIME COMPLEXITY

| Algorithm | Time complexity |
|---|---|
| Dijkstra | $n^2 + m$ |
| Bellman-Ford | $n^3$ |
| Floyd-Warshall | nm |

## 5 Conclusion and Future Work

The computed time complexity for each of the Dijkstra's, Floyd-Warshall and Bellman-Ford algorithms show that these algorithms are acceptable in terms of their overall performance in solving the shortest path problem. All of these algorithms produce only one solution. However, the main advantage of GA over these algorithms is that it may produce a number of different optimal solutions since the result can differ every time the GA is executed. In the future, the proposed GA framework will be extended and improved in finding the shortest path or distance between two places in a map that represents any types of networks. In addition, other artificial intelligence techniques such as fuzzy logic and neural networks can also be implemented in improving existing shortest path algorithms in order to make them more intelligent and more efficient.

## Acknowledgments

This research study was partially funded by the Fundamental Research Grant Scheme (FRGS), Ministry of Higher Education (MOHE), Malaysia. This paper would not have been possible without the assistance, support and patience of my supervisor, Dr. Hajar Mat Jani. I would like to thank Dr. Hajar for her invaluable advice and unsurpassed knowledge. Finally, I would like to thank my parents for giving birth to me and supporting me throughout my life.

## References

[1] C. Xi, F. Qi, and L. Wei, "A New Shortest Path Algorithm based on Heuristic Strategy," Proc. of the 6th World Congress on Intelligent Control and Automation, Vol. 1, pp. 2531–2536, 2006.

[2] B.S. Hasan, M.A. Khamees, and A.S.H. Mahmoud, "A Heuristic Genetic Algorithm for the Single Source Shortest Path Problem, " Proc. of International Conference on Computer Systems and Applications,

pp. 187-194, 2007.

[3] T. Li, L. Qi, and D. Ruan, “An Efficient Algorithm for the Single-Source Shortest Path Problem in Graph Theory”, Proc. of 3rd International Conference on Intelligent System and Knowledge Engineering, Vol. 1, pp. 152-157, 2008.

[4] M. Jordan, “Notes 7 for CS170”, UC Berkeley, 2005.

[5] J. Chamero, “Dijkstra’s Algorithm” Discrete Structures & Algorithms, 2006.

[6] S. Skiena, A. Revilla, “Programming Challenges, The Programming Contest Training Manual” pp. 248 – 250.

[7] S. Hougardy, The Floyd-Warshall, “Algorithm on Graphs with Negative Cycles”, University of Bonn, 2010.

[8] M. Negnevitsky, Artificial Intelligence: A Guide to Intelligent Systems, Third Edition, Addison-Wesley, 2011.

[9] I. Rakip, U. Atila, “A Genetic Algorithm Approach for Finding the Shortest Driving Time on Mobile Devices”, Scientific Research and Essays, Dept. of Computer Engineering, 2011.

[10] Dijkstra’s Algorithm, Available at http://informatics.mccme.ru/moodle/mod/statements/view.php?id=193#1. 2012.

[11] Floyd-Warshall Algorithm, Available at http://informatics.mccme.ru/moodle/mod/statements/view.php?id=218#1. 2012.

[12] Bellman-Ford Algorithm, Available at http://informatics.mccme.ru/moodle/mod/statements/view.php?id=260#1. 2012.