Shortest Path Algorithm and its implementation

M. MUTHULAKSHMI[#1], M.M. SHANMUGAPRIYA*[2]

[#1]*Research Scholar, Karpagam University, Coimbatore, Tamil Nadu, India*

[*2]*Assistant professor, Department of mathematics, Karpagam University, Coimbatore-641021*

Abstract: *Shortest Path problems are among the most studied network flow optimization problems, with the interesting applications in the range of fields. The shortest path algorithms are applied automatically to find the directions between the physical locations, In the driving directions on their web mapping websites like MapQuest or Google Maps. In a networking or telecommunications mindset, the shortest path problem is sometimes called as min-delay path problem and usually tied with a widest path problem. For example, the algorithm seeks the shortest (min-delay) widest path, or widest shortest (min-delay) path. Many problems are framed as a form of the shortest path for some suitably substituted notions of addition along a path and taking the minimum. The general approach to these is to consider as the two operations to be those of a semiring. Semiring multiplication is done along the path, and the addition is between the paths. This general framework is known as the algebraic path problem. Most of the classic shortest-path algorithms (and new ones) are formulated as the solving linear systems over the algebraic structures.*

Shortest path problems form the foundation of an entire class of optimization problems that are solved by a technique called column generation. Examples include vehicle routing problem, survivable network design problem, amongst others. In such problems there is a master problem (usually a linear program) in which each column represents a path (think of a path in a vehicle routing problem as one candidate route that a vehicle takes, think of a path in a network design problem as a possible route over which a commodity can be is sent between a source and a destination). The master problem is repeatedly solved. Each time, using the metrics of the solution, a separate problem (called the column-generation sub problem) is also solved. This problem turns out to be a shortest path problem (usually with side constraints or negative arc lengths rendering the problem NP-Hard). If a new useful path is obtained, it is added to the original master problem which is now re-solved over a larger subset of paths leading to better (lower cost, usually) solutions.

I. INTRODUCTION

The shortest path problem is a problem for finding the shortest path or route from a starting point to a final destination. Generally, in order to represent the shortest path problem graphs are used. A graph is a mathematical abstract object, which contains sets of vertices and edges. Edges connect the 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 the graph is a directed graph or an undirected graph. In addition, lengths of the 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 for the different types of scenarios. For example, in order to represent a map use a graph, where vertices represent cities and edges represent routes that connect the cities. If routes are one-way then the graph is a directed; otherwise, it is an undirected. There exist different types of algorithms that solve the shortest path problem. The most popular conventional shortest path algorithms along with one that uses genetic algorithm are going to be discussed in the paper, and they are as follows:

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

Nowadays, there are many intelligent shortest path algorithms that are introduced in several past research papers. For example many of the researches use heuristic method for computing the shortest path from one point to another point within the traffic networks.

II. RESEARCH OBJECTIVES

The following list gives the objectives of the research paper:

- To determine and identify the concepts of the shortest path problem.
- To explain the general concepts and the implementations of Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, Genetic Algorithm and Heuristic Algorithm.

III. LITERATURE REVIEW

As mentioned earlier, a graph is used to represent a map where the cities are represented by vertices and the routes or roads are represented by the edges within the graph. In this section, a graph is a representation of a map which is explained further, and the brief descriptions and implementations of the four shortest path algorithms being studied and are presented. Between vertices *i* and *j*, the value in (*a[i][j]*) is equal to the *infinity*. An array of the 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 use the *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 the edges in comparison to the adjacency matrix is when there is more than one edge that connects the two vertices of the adjacency matrix in order to represent graph cannot be used.

Shortest path

The shortest-route problem determines a route of minimum weight connecting two Specified vertices, source and destination, in a weight graph (digraph) in a Transportation network. Other situations are represented by the same model like VLSI design, equipment replacement and others, there are different types of shortest path algorithm to find the shortest path of any graph. Most frequently encountered are the following:

• Shortest path between the two specified vertices
• Shortest path between all pairs of vertices
• Shortest path from a specified vertex to all others

The most efficient algorithm used to find the shortest path between two nodes. Vertices are the Dijkstra's algorithm. Some improvements on the Dijkstra algorithm are
done in terms of efficient implementation and cost matrix .In the paper, It is
Proposed some improvement in order to reduce the number of iterations and to find
easily and quickly the shortest path.

A) Dijkstra's Algorithm:

For each vertex within a graph assign a label that determines the minimal length from the starting point *s* to other vertices *v* of the graph. In a computer 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 the vertices have been visited. The label at the starting point *s* is equal to the zero ($d[s]=0$); however, labels in the other vertices *v* are equal to the *infinity* ($d[v]=\infty$), which means that the length from the starting point *s* to other vertices is unknown. In a computer just use a very big number in order to represent the infinity. In addition, for the each vertex *v* whether it is visited or must be identified or not In order to do that, the researcher declare an array of *Boolean* type called *u[v]*, where initially, all the vertices are assigned as unvisited ($u[v] = false$).
The Dijkstra's algorithm consists of *n* iterations. If all the vertices are visited, then the algorithm finishes; otherwise, from the list of unvisited vertices choose the vertex which has the minimum (smallest) value at its label (At the beginning, choose a starting point *s*).

After that, consider all the neighbors of the vertex (Neighbors of a vertex are those vertices that have common edges with the initial vertex). For the each unvisited neighbor 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 change the value in that label with the newly obtained value.

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

After considering all the neighbors, assign the initial vertex as visited ($u[v] = true$). After repeating this step *n* times, all the vertices of the graph visit and the algorithm finishes or terminates. The vertices that are not connected with the starting point remains by being assigned the *infinity*. In order to restore the shortest path from the starting point to other vertices, identify array *p []*, where for each vertex, where $v \neq s$, Stores 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
P = (s , ... , p [p [p [v]]] , p [p [v]] , p [v] , v) (2)

a)Modified Dijkstra's Algorithm

It is presented a modified version of Dijkstra's Algorithm in order to reduce the total number of iteration by optimizing the situation of many shortest paths:
a. Every vertex j that is not yet permanently labeled gets a new temporary label
whose value is given by min[old label of j, (old label of i + dij)], where I is
the latest vertex permanently labeled, in the previous iteration, and dij is the
direct distance between the vertices i and j. If i and j are not joined by an edge,then dij=infinity .
b. The smallest value among all the temporary labels is found, and it becomes the permanent label of the corresponding vertex. In case of more
than one shortest path, select all of them for permanent labeling.
c. Steps (a) and (b) are repeated alternately maximum n-1 times

B) Floyd-Warshall Algorithm:

Consider the graph *G*, where vertices are numbered from *1* to *n*. The notation *dij k* means the shortest path from *i* to *j*, which also passes through the vertex *k*. obviously if there exists an edge between the vertices *i* and *j is* equal to *dij0*, otherwise it is assigned as *infinity*. However, for other values of *dijk* there can be two choices: (1) if the shortest path from *i* to *j* does not pass through the vertex *k* then value of *dijk* will be equal to *dijk-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 *dijk* will be equal to *dikk-1* + *dkjk-1*. And in order to determine the shortest path we just need to find the minimum of these two statements:
dij0 = the length of edge between vertices i and j (3)
dijk = min (dijk-1, dikk-1 + dkjk-1) (4)

C) Genetic Algorithm (GA)

Intelligent algorithms are introduced in finding the optimal shortest paths in many situations that require the systems to search through a very large search space within the 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 the biological evolution .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, generate a number of solutions, and then choose the most optimal one among the provided set of the 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 the 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 which is to be considered for the next generation. The process continues until the criteria of the given problem will not be achieved. Thus the following stages of a GA can be identify :

- ***Step 1:*** Determine the fitness function.
- ***Step 2:*** Create initial population – a population that contains *n* individuals. At this stage we need not create the fittest individuals, because it is probable that GA will transfer them into a viable population. In order to create chromosomes for initial population, produce random paths from the starting point to the final destination.
- ***Step 3:*** Selection – the stage of GA that is used to the select two chromosomes for the 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 is chosen from the whole population (not from the survivors in the first iteration), because to keep diversity of individuals is needed , otherwise entire population is hammered with single copies of one individual. There exist different types of crossover methods; however, for the problem as the 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 the genes of selected chromosome.

D) Bellman-Ford Algorithm:

In comparison to Dijkstra's algorithm, the Bellman-Ford algorithm admits or acknowledges the edges with the negative weights. That is why, a graph contains the cycles of negative weights, which generates numerous number of paths from the starting point to the final destination, where each cycle minimize the length of the shortest path. Taking into consideration this fact let's assume that the graph does not contain cycles with negative weights. The array *d[]* stores 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 the shortest paths in a graph it requires *n – 1* phase, but for those vertices of a graph that are unreachable, the value of elements of the array will remain by being assigned to *infinity*

E) Heuristic algorithms

A heuristic algorithm is one that is designed to solve a problem in a faster and more efficient fashion than the traditional methods by sacrificing optimality, accuracy, precision, or completeness for the speed. Heuristic algorithms often used to solve NP-complete problems, a class of decision problems. In these problems, there is no known efficient way to find a solution quickly and accurately although solutions are verified when given. Heuristics produces a solution individually used to provide a good baseline and are supplemented with optimization algorithms. Heuristic algorithms are most often employed when approximate solutions are sufficient and exact solutions are necessarily computationally expensive. For sufficiently great inputs heuristics are developed. Branch-and-bound technique and dynamic programming are quite effective but their time-complexity often is too high and unacceptable for NP-complete tasks. Hill-climbing algorithm is effective, but it has a significant drawback called pre-mature convergence. Since it is"gresedy", it always finds the nearest local optima of low quality. The goal of modern heuristics is to overcome this disadvantage

IV. CONCLUSIONS 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 the 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 the existing shortest path algorithms in order to make them more intelligent and more efficient.

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

[3] J. Chamero, ―Dijkstra's Algorithm‖ Discrete Structures & Algorithms, 2006.

[4] E. W. Dijkstra, A note on two problems in connexion with graphs.
Numerische Mathematik 1: 269–271, 1959.

[5] Floyd, R. W. (1962). Algorithm 97: Shortest path. Communications of the ACM,Volume 5 No 6.

[6] W. K. Chen. Theory of Nets: Flows in Networks. John Wiley Sons, 1990.
&
[7] Ddition. Prentice Hall, 1992.

[8] Matthias Hentschel, Daniel Lecking, Bernardo Wagner,

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

[10]J. Chamero, ―Dijkstra’s Algorithm‖ Discrete Structures & Algorithms, 2006.

[11]Kroger, Martin (2005). "Shortest multiple disconnected path for the analysis of entanglements in two- and three-dimensional polymeric systems". Computer Physics Communications 168 (168): 209–232. doi:10.1016/j.cpc.2005.01.020.
.

[12] Ravindra K. Ahuja, Thomas L. Magnanti, *and* James B. Orlin *(1993). Network Flows: Theory, Algorithms and Applications. Prentice Hall.* ISBN 0-13-617549-X