**Network for Research and Development in Africa**
International Journal of Information, Technology and Innovations in Africa
ISSN: 2360-9772. Volume 13, Number 2,
Pages 1-23 (April, 2018)
www.arcnjournals.org

# A Study of Intelligent Route Guidance System Using Dijkstra's Heuristic Shortest Path Algorithm

[1]Ukwosah Ernest Chukwuka , [2]Oladunjoye, John Abiodun and [3]Siman Emmanuel
Department of Computer Science, Federal University Wukari, Taraba State, Nigeria
[1]eukwosah@gmail.com, ukwosah@fuwukari.edu.ng
[2]Oladunjoye.abbey@yahoo.com

***Abstract:*** *In this research paper, we propose TransRoute: a web-based, intelligent route planning application that will leverage advanced data structures, heuristic shortest path algorithms (SPAs), graph and network optimization models in routing vehicles for emergency response during natural disasters, fire outbreaks, health crisis and courier package for cost minimization in logistics and transport sector of the economy so as to save lives, increase the bottom line of logistics firms and improve efficiency in service delivery. The importance of transportation and logistics in this twenty-first century globalized economy cannot be overemphasized. Due to its widespread applications, graph and network optimization is an important sub-field within the broader field of optimization. Shortest-path problems are among the fundamental graph and network optimization problems.*

***Key words:*** *Dijkstra, guidance system, heuristic Shortest path, intelligent route*

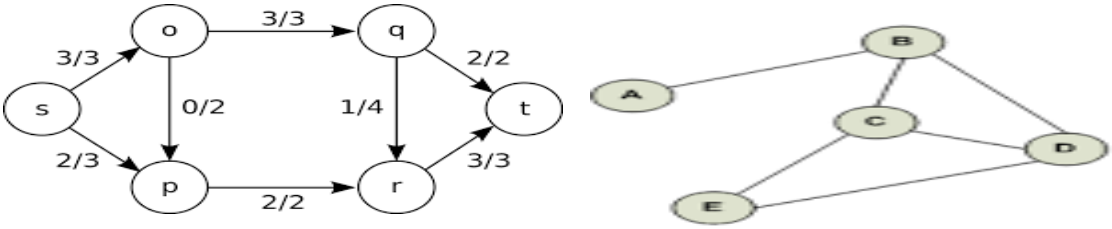
## 1. Introduction

In daily life, everyone is always confronted with the problem of choosing shortest-path from one location to another location. The routing of vehicles or personnel in complex logistics systems is a task that needs to be solved in numerous applications, e.g., detailed models of transport networks or order picking areas [2]. The number of relevant nodes in such networks can easily exceed 10,000 nodes [2]. Often, a basic task is finding the shortest-path from one node (source or starting) node to another (sink or destination) node [2]. Graphs and networks are all-pervasive. Electrical networks and power networks bring lighting and entertainment into our homes [1]. Telephone networks permit us to communicate with each other almost effortlessly within our local communities and across regional and international borders [1]. National highway systems, rail networks, and airline service networks provide us with the means to cross great geographical distances to accomplish our work, to see our loved ones, and to visit new places and enjoy new experiences [1]. Manufacturing networks and distribution networks give us access to life's essential foods and to consume products [1]. In each of these settings, one wishes to send some goods (electricity, messages, goods, services and water) from one point to another, typically as efficiently as possible. The field of study concerning the optimal flow of goods on graphs and networks is known as graph and network optimization [1]. Graphs provide the ultimate in data structure flexibility [3]. Graphs can model both real-world

systems and abstract problems; hence find wide applications [3-9]. The SPA is an extremely well-studied problem in both operations research and theoretical computer science communities because of its applicability in a wide range of domains [10]. Many interesting route planning problems can be modeled and solved by computing shortest-paths in a suitably modeled, weighted graph representing a transportation network. For large networks, the classical Dijkstra's algorithm [11] to compute shortest path in robot path-planning [12]; logistics distribution lines [13]; link-state routing (LSR) protocols [14]; open-shortest path first (OSPF) [15] and intermediate system to intermediate system (IS-IS) routing algorithms in computer and telecommunication networks, [16] respectively are used. One of the basic problems of network modeling [17] is to find the shortest path from an origin to a destination. In 1955, [18] presented a conference paper that included the first formulation of the shortest path problem. His paper was subsequently published in Operations Research in 1957. Based upon that paper, [19] suggested a format for solving the shortest path problem using a network represented as a web of strings and knots, and [20] developed an algorithm to solve the shortest path problem in the presence of some negative arc lengths. [11] followed the works of [19-20] with a new algorithm that appeared to be considerably more efficient and required less data storage [11]. The algorithm of Dijkstra remains to this day one of the best approaches for optimally solving the simple shortest path problem where all arcs have non-negative lengths. In another early contribution, [21] developed a search strategy, called A*-Search, to solve for minimum cost paths.

## 2. Background

Shortest-path problems according to [46] are among the best known problems in graph theory and arise in a wide variety of applications [29, 47] with several important variants. The *s-t* shortest path problem requires finding a single shortest-path between given vertices s and t; it has obvious practical applications like driving directions and has received a great deal of attention. It is also relatively easy, solutions in typical graphs like road networks visit a tiny fraction of vertices, with Lumsdaine *et .al,* 2007 [48] observing visits to 80,000 vertices out of 32 million in one example [46]. A third variant, all-pairs shortest paths, is impractical for large graphs because of its $O(|V|^2)$ storage requirements. The origin of graph theory according to Singh & Vandana, 2014 [4], started with the problem of Königsberg Bridge in 1735, [49, 50, 51, 52, 53, 54]. This problem led to the concept of Eulerian graph. A graph as shown in figure 2.1 (a-b) below, is a pair **G = (V, E)**, [3, 7]; where ***V*** is the set of all vertices and **E** the set of all edges; and the elements of **E** are subsets of V containing exactly 2 elements is called a labelled graph if each edge ***e = UV*** is given the value ***f (UV) = f (u)*f (v),*** where ***** is a binary operation. In literature one can find ***** to be either *addition, multiplication, modulo addition* or *absolute difference, modulo subtraction or symmetric difference*. The number of vertices is written as $|\mathbf{V}|$, and the number of edges is written $|\mathbf{E}|$. $|\mathbf{E}|$ can range from zero to a maximum of $|\mathbf{V}|^2$-$|\mathbf{V}|$.




**(a)** **(b)**

**Figure 1:** Examples of graphs: (a). Directed graph (digraph), (b). Undirected graph

Several speed-up techniques for SPAs have been proposed e.g., [55] and [56] respectively. The approach of highway hierarchies is of special importance to path finding in road networks. It is based on the fact that logistics systems often contain clusters of densely connected stations while the clusters are interrelated by only a few main edges - i.e., there are cities and highways between the cities [2]. The shortest path problem is defined on a directed, weighted graph, where the weights may be thought of as distances [1, 57]. The objective is to find a path from a source node, s, to a sink node, t that minimizes the sum of weights along the path. To formulate it as a network flow problem, let $x_{i,j}$ be 1 if the arc (i, j) is in the path and 0 otherwise. Place a supply of one unit at s and a demand of one unit at t. Given such a graph, a typical problem is to find the total length of the shortest path between two specified vertices. This is not a trivial problem, because the shortest path may not be along the edge (if any) connecting two vertices, but rather may be along a path involving one or more intermediate vertices [3]. The linear programming (LP) formulation [18] for the shortest path problem (SPP) is:

$$\textbf{Min} \quad \sum_{(i,j)\in A} d_{i,j}x_{i,j}$$

$$\textbf{subject to} \quad \sum_{(i,k)\in A} x_{i,k} - \sum_{(j,i)\in A} x_{j,i} = 0 \qquad \forall i \neq s, t$$

$$\sum_{(s,i)\in A} x_{s,i} = 1$$

$$\sum_{(j,t)\in A} x_{j,t} = 1$$

$$0 \leq x_{i,j} \leq 1$$

## 3. Statement of problem

In contemporary societies, a plethora of problems in emergency response, logistics and transportation sector of the economy exists. High traffic congestion, inefficient route planning, bad roads and vehicular air pollution respectively. Despite heightened research interest in transportation related decision analysis within a GIS environment e.g., Ralston *et al*. [86]; Zhan *et al*. [31, 63, 75] and Erkut [101], the need for efficient, effective and timely response to emergencies, facility location, network flows, vehicle routing and delivery of logistics package by emergency response agencies like hospitals, fire service, courier service firms from source to their respective destinations at minimal cost has been a major challenge faced by these agencies and firms around the world as it is similar to the Travelling Salesman Problem (TSP)

[103], hence NP-Complete. We consider a directed, weighted graph, *G = (V, E)*, as shown in figure 3.1 below, with a vertex set *V* with n vertices, where each vertex refers to a city and an edge set E with m edges, where edges connect one city to another city. The weights on the edges indicate the distance between two connected cities, the shortest-paths problem (SPP) consist of finding a directed path of minimum cost (length) from a specified source node or vertex $v \in V$ to all other vertices in *V* in a directed network in which each arc (i,j) has an associated cost (or length) $c_{ij}$. The SPP is a special case of the minimum-cost flow problem. This formulation assumes that there are no negative costs directed cycles, called negative cycles in the network, Ahuja and Orlin [1].

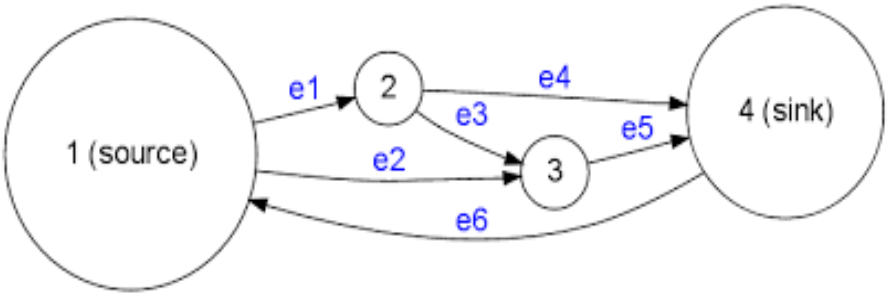



**Figure 2:** Showing *source* and *sink* nodes for a directed, weighted graph

## 4. Motivation

This work is motivated by real world problems that arise in emergency response, logistics, transportation problems, computer networks and telecommunication respectively that seeks to determine the optimum path, hence requires the adoption of heuristic shortest path algorithms like Dijkstra's SPA implemented with a dynamic data structure known as double bucket.

## 5. Aim of the Research

This research work aims at leveraging existing fastest shortest-path algorithms (SPAs) like - *Dijkstra's* algorithm implemented with double buckets data structure to implement a robust, scalable web-based, GIS-enabled route planning system for emergency response, travel planning, logistics and transportation application that will fundamentally address the problem of delays, difficulties and hitches experienced in moving people, goods and services from one location to another.

## 6. Objectives of the Research

This research is primarily focused on achieving the following objectives:

(a) Evaluate existing shortest-paths algorithms with a view to determine the algorithm that offers optimum results, hence suitability in obtaining optimal path and dealing with routing problems in emergency response, travel planning, logistics and transportation networks.

(b) Develop a robust, scalable and efficient web-based, GIS-enabled, route/travel planning, logistics/courier transport application with geographic information

system (GIS) capabilities using either- *Google Maps, MapQuest, FromToMap* or *Microsoft Bing* maps, so as to accurately determine optimal and alternative routes from source to destination locations respectively.

(c) Simulate and calculate distances and timestamp of source and destination locations in real-time for ease of decision-making for vehicle dispatch and scheduling.

## 7. Significance of Study

By leveraging on the proposed system it would aid in saving lives during emergencies, fast track the delivery of time-bound package and movement of goods and services from one location to another, hence minimizing delivery cost and delay in the long run, boost the efficiency and service delivery of courier service firms in Nigeria in terms of delivering time-sensitive package within and outside the country, hence maximizing profit in the process, as most indigenous logistics and courier firms lack an automated routing or dispatch planning system to guide them on the shortest or fastest route to deliver package from their source to destinations. *TransRoute* will reduce cost of monitoring or tracking fleets or goods and services, improve the speed of decision-making by providing decision-makers with real-time data and information; centralized repository that provides a single source of common information, hence facilitating standardization, faster retrieval and selective modification of information, Montgomery (1993) and the ability to produce reports based on user specified parameters respectively.

## 8. Scope and Limitations of Study

This research takes into account the study of the routing network of some courier firms within the country. It also considers the simulation of only the road routes in which packages could be delivered to their various offices scattered all over the country. Some of the limitations of this project work include:

(a) The proposed application only takes into cognizance the shortest-parts to the intended destination(s) from the source without considering some extraneous factors that can hinder the movement of the vehicle such as traffic snarls, bad roads, obstruction and bad weather.

(b) The proposed application also did not factor other means of transportation such as railways and water ways *et cetera*.

## 9. Methodology and Application Implementation Framework

A web-based, GIS-enabled model is selected as the main implementation approach, as we envisioned the realized system would be accessible online by any web user from any location within the geographical test area . The proposed system will be accessible via three (3) different mediums viz: web interface, mobile access through mobile web and mobile access through Android smart phones. A three-tier architectural framework will be adopted in realizing the proposed system. Hence, the system will have - user interface, application logic, data and server layers respectively as shown in figures 9.1 and 9.2 below respectively. PHP is widely used as a general-purpose scripting language that is especially suited for web development (PHP.net, 2004). It can be embedded in (X)HTML or XML. It serves as a tool for creating dynamic data-driven web pages. Alternatively, Microsoft .NET Framework Visual Studio 7.0 integrated

development environment/ASP .NET will be used to realize the proposed system. Mapping services like - Google Maps, MapQuest, FromToMap and Microsoft Bing maps alongside UMN MapServer will be used to render spatial data into map data for route visualization. Apache MySQL or Microsoft SQL RDBMS will be leveraged for the data repository implementation. Topology building, network creation and data cleaning and correction will be handled using Quantum GIS 1.6.

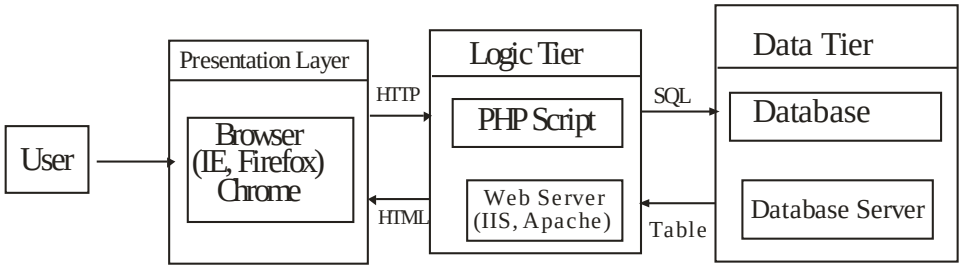



**Figure 3:** Three-tier architectural framework of TransRoute application

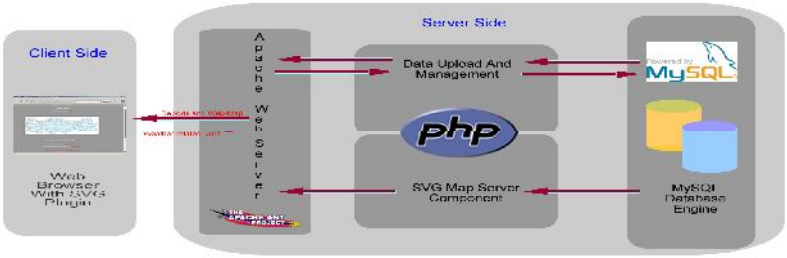



**Figure 4:** Architectural layout of the proposed system

This software architectural pattern separates the presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the view and the Model. Figure 5 below illustrates the M-V-C component design pattern concept.

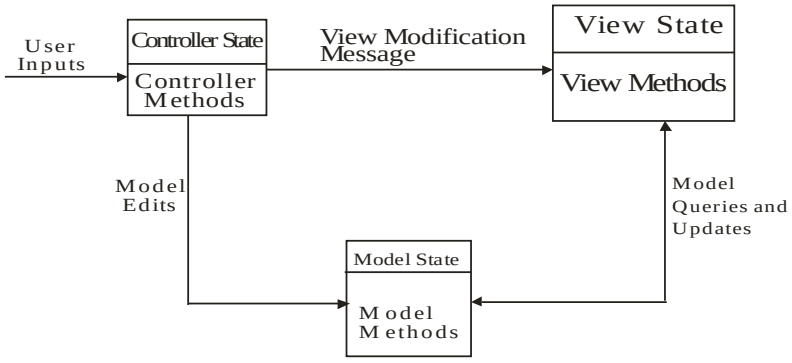



**Figure 5:** Model-View-Controller (MVC) application framework

There are some programmers who, upon hearing that an application has been split into 3 areas of responsibility, automatically assume that they have the same responsibilities as the Model-View-Controller (MVC) Design Pattern. This is not the case. While there are similarities there are also some important differences: The View and Controller both fit into the Presentation layer. Although the Model and Business layers seem to be identical the MVC pattern does not have a separate component which is dedicated to data access. The overlaps and differences are shown in Figure 6 below:

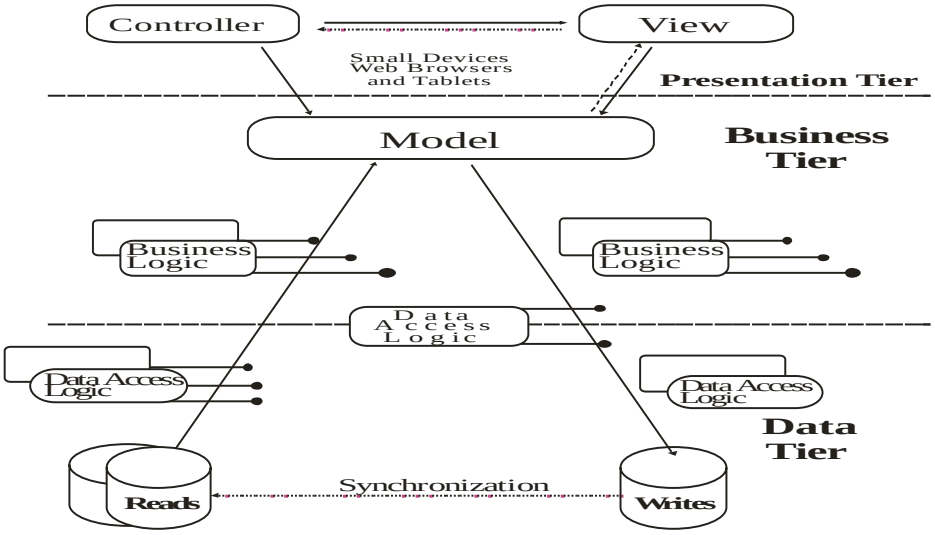



**Figure 6:** The model-view-controller and the three-tier model combined

The study area covers the six geo-political zones in Nigeria comprising all the 36 states including the Federal Capital Territory (FCT) Abuja, and the 774 local government areas making up Nigeria, spanning approximately 990,000 $km^2$ as shown in the map below.


**Figure 7:** Map of Nigeria

**10. Review of Related Literature**

Considerable empirical studies on the performance of shortest path algorithms have been reported in the literature, interested readers are referred to [1, 27, 72]. The first polynomial-time algorithm for SPPs was developed by Dijkstra [11]. Others include: Dial *et al*. [22-23]; Glover *et al.* [24]; Gallo and Pallottino [25]; Hung and Divoky [26]; Ahuja *et al*. [27]; Mondou *et al*. [28]; Cherkassky *et al*. [29]; Goldberg and Radzik [30] respectively. In all of these cited works, there is no clear answer as to which algorithm, or a set of algorithms that runs fastest on real road networks. In a recent study conducted by Zhan & Noon [31], a set of three shortest path algorithms that run fastest on real road networks has been identified. These three algorithms are: *1)* the *Gallo and Pallottino's graph growth algorithm implemented with two queues*, 2) the *Dijkstra algorithm implemented with approximate buckets*, and 3) the *Dijkstra algorithm implemented with double buckets*. As a follow up to that study, this paper reviews and summarizes these three algorithms and others, and demonstrates the data structures and implementation strategies related to the algorithms. Other SPAs to be considered include: Bellman-Ford-Moore [32, 33, 34]; Floyd-Warshall [35, 36, 37, 38, 39]; A* - search [40, 41, 42, 43] and Johnson's algorithms [44, 45] respectively. Dijkstra's algorithm runs in space and time complexities of $O(|V^2|)$ and $O(|E| + V \log |V|)$ respectively, when implemented with I/O-efficient *priority queues* and method. Bellman-Ford-Moore runs in $O(|V^2|)$ and $O(|V.E|)$; Floyd-Warshall in $O(n^3)$ for both space and time bounds, Johnson's has a time bound of $O(|V^2| \log |V| + |V.E|)$ and A*-search algorithms depends on the problem heuristic respectively. The aim of the review is to select out of the numerous algorithms and their variants the best-fit, efficient and fastest to be used in the implementation of our proposed application. Shortest-path problems, logistics and transportation problems have received tremendous attention from research communities in Computer Science (theoretical), Operations Research and Mathematics. The literature on shortest-path problems and algorithms is large, quite long and detailed [17, 30]. [3, 4, 5, 6, 7, 8, 9] describes a number of applications of the SPAs, as well as efficient algorithms for the same problem. For more detailed review and analysis the interested reader should consult Dijkstra [11]; Dial *et al*. [22-23]; Glover *et al.* [24]; Gallo and Pallottino [25]; Hung and Divoky, [26]; Ahuja *et al*. [27]; Mondou *et al*. [28]; Cherkassky *et al*. [29]; Goldberg and Radzik [30], Dreyfus [58]; Fu et al. [59] Gallo and Pallottino [60]; Duckham & Kulik [61]; Mark [62]; Zhan & Noon [63]; Sanders & Schultes [64] and Golledge & Gärling [65] respectively. The review of related works and lessons learnt informed the choice of methodologies, technologies and paradigms employed in the advancement of this research problem. Here we briefly outline the various algorithmic paradigms and their advancements from new design paradigms, data structures improvements and parameterization to input restrictions so as to contextualize our work [10].

There have been a number of reported evaluations of shortest path algorithms in the literature (e.g., Glover *et al*. 1985; Gallo and Pallottino 1988; Hung and Divoky 1988), a recent study by (Cherkassky *et al*. 1993 & 1996 [29, 74]) is one of the most comprehensive evaluations of shortest path algorithms to date. They evaluated a set of 17 shortest path algorithms. In their experiment, Cherkassky *et al*. coded the 17 algorithms using the C programming language, and tested the C programs on a SUN Sparc-10 workstation. One-to-all shortest paths can be

computed by these C programs. Cherkassky *et al*. used a number of simulated networks with various degrees of complexity for evaluating the algorithms. The results of their studies suggest that no single algorithm and concluded that no single algorithm consistently beat all others over all problem classes. Overall, they suggested that the Dijkstra algorithm implemented with a double-level bucket structure (DKD) is the best algorithm for networks with non-negative arc lengths. The double-bucket structure represents the approach used to track and identify the next node to add to the shortest path tree (i.e. the next closest node to the origin). Using the open-source codes written by Cherkassky et al. (1996), Zhan & Noon (1998), [75] conducted an evaluation of 15 of the 17 algorithms on a variety of real road networks. They concluded that TWO-Q, DKD and the Dijkstra algorithm incorporating approximate buckets (DKA, which is a different variant of the bucket approach than DKD) are the three fastest one-to-all shortest path algorithms. In a subsequent study, Zhan and Noon (2000) compared these three algorithms for the one-to-one shortest path problem on 10 different road networks. They suggested that DKA is the best choice for the case when shortest paths are somewhat short and that Pallottino's TWO-Q is the best choice in situations where the shortest paths are relatively long. Given the work of Cherkassky *et al.* (1996) and Zhan and Noon (1998, 2000), one can conclude that the top three candidates for SPA application on real road networks are two versions of Dijkstra's algorithm (DKA and DKD), and Pallottino's TWO-Q algorithm. This past research, although meticulous, falls short in terms of three major areas. First, all real networks that were tested can be considered relatively small when compared to many GIS networks. Second, all tests performed on large networks involved random graphs, which lack geographical integrity. These two factors together could certainly skew the overall results and therefore the conclusions as well. Third, no attempt was made to test A*-search as a viable approach. Therefore, a balanced test of the top SPA candidates does not exist and conclusions drawn in the previous work may be misguided. In the next section, we will introduce the A*-search approach. We will also provide what we think is a plausible explanation as to why it has been ignored in previous tests of solving the one-to-one shortest path problem (SPP).

More recently, Zhan and Noon (1996, 1997 & 1998) [31, 63, 75] tested 15 of the 17 shortest path algorithms using real-world road networks. In their evaluation, Zhan and Noon dropped two of the 17 algorithms tested by Cherkassky *et al* (1993). They did not consider the special-purpose algorithm for acyclic networks because an arc on real road networks can be treated bi-directional, and hence real-world road networks contain cycles. They also dropped the implementation using a stack to maintain labeled nodes (see the next section for descriptions about stack and labeled nodes) because they found that this algorithm is many times slower than the rest of the algorithms on real-world road networks during their preliminary testing. These 15 algorithms are summarized in Table 11.1 below. It is not the intention of this paper to review these 15 algorithms thoroughly. Detailed description of the algorithms can be found in Cherkassky *et al*. (1993) and the references therein. Based on their evaluation, Zhan and Noon suggested that the best performing implementation for solving the one-to-all shortest path problem is Pallottino's graph growth algorithm implemented with two queues (TQQ). They further suggested that when the goal is to obtain a one-to-one shortest path or one-to-some shortest paths, the Dijkstra algorithm offers some advantages because it

can be terminated as soon as the shortest path distance to the destination node is obtained. Zhan and Noon recommended two of Dijkstra implementations. The choice between the two implementations depends on the maximum network arc lengths. They recommended the approximate buckets implementation of the Dijkstra algorithm (DKA) for computing one-to-some shortest paths over networks whose maximum arc length is less than 1500. For networks whose maximum arc length is greater than 1500, they recommended that the double buckets implementation of the Dijkstra algorithm (DKD) should also be considered. Table 1 below summarize the various algorithms tested.

The first polynomial time, greedy-based algorithm for the SPA problem, was conceived by E.W. Dijkstra in 1956 and 'published' in 1959 [1, 11, 78]. Though it is argued that it was first "published" in 1956 by Lester Ford [20], others are of the view that it was in 1957 by a team of researchers at the Case Institute of Technology, in an annual project report for the Combat Development Department of the US Army Electronic Proving Ground [79]. The same algorithm was later independently rediscovered and actually publicly published by Edsger Wybe Dijkstra in 1959. A nearly identical algorithm was also described by George Dantzig in 1958 [18], who formulated its linear programming model as shown in section 2. In this section, we will give the simplest possible description of Dijkstra's algorithm (DA) as provided in Cormen *et .al* [35] and other classical books: Sedgewick and Wayne [80], Aho, Hopcroft, and Ullman [81, 82]). In the same vein, many scientific papers on Dijkstra's algorithm use these frameworks: Sniedovitch [83]; Cherkassky, Goldberg, and Radzik [74] respectively. Dijkstra's Algorithm delivers the shortest path from a given node i to a single destination node or all other nodes within a graph with non-negative edge path costs (Dijkstra, 1959). The idea is to set up a set of unvisited nodes and the tentative distance from node i to all other nodes j, denoted as Dist[j]. Furthermore, a list Prev[j] stores the previous node on the path from node i to node j. All distances are initially set to infinity. The start node i is the initial current node. For the current node all of its unvisited neighbors (connected by an edge) are considered next and the tentative distances to these neighbors are calculated. After all neighbour nodes of the current node have been considered, the current node is deleted from the set of unvisited nodes. The next node selected as the current node is the one with the shortest distance to node i (i.e., the node which has the minimum value in Dist[j]). The algorithm terminates once the destination node has been deleted from the set of unvisited nodes, or once all nodes have been considered and the set of unvisited nodes is empty. The calculation of the tentative distances is done by checking if the currently stored distance from node i to node j (given by Dist[j]) is greater than Dist[k] + d(k, j). In this case, Dist[j] is overwritten with Dist[k] + d(k, j) and Prev[j] is set to node k. The simplest implementation of Dijkstra's algorithm stores nodes in a linked list or an array, and the operation to find the minimum value in list Dist is a linear search through all nodes in Dist. In this case, the time complexity is $O(n^2)$. However, the complexity can be reduced for sparse graphs, i.e., for graphs with far fewer than $n^2$ edges, with a more intelligent strategy to store the graph in form of adjacency lists and using different types of heaps to implement the operation to find the minimum value in Dist efficiently. With a strategy of using the *Fibonacci heap* the running time can be reduced to O(m+nlog(n)), Cormen *et al.* (2001); Fredman and Tarjan (1987). The bi-directional approach of Dijkstra's algorithm, also called two-side Dijkstra is

based on the idea that finding the shortest path from start node i to destination node j might be carried out faster, if the search is started from both sides in parallel. As Dijkstra's algorithm can be interpreted as a breadth-first search (BFS), a lot of "unnecessary" steps might be carried out before reaching the destination node. Starting from both sides might reduce the computational effort significantly, Berge and Ghouila-Houri (1965). However, if the two-sided procedure is terminated as soon as a node has been processed from both directions, there is no guarantee that this node is actually on the shortest path from the start node *i* to destination node *j*, Vahrenkamp and Mattfeld (2007). Dijkstra's algorithm, as it is universally known, is particularly well-behaved if the graph has no negative-weight edges. In this case, it's not hard to show (by induction, of course) that the vertices are scanned in increasing order of their shortest-path distance from *s*. It follows that each vertex is scanned at most once and thus each edge is relaxed at most once. Since the key of each vertex in the heap is its tentative distance from *s*, the algorithm performs a *DecreaseKey* operation every time an edge is relaxed. Thus, the algorithm performs at most *E DecreaseKeys*. Similarly, there are at most *V* Insert and *ExtractMin* operations. Thus, if we store the vertices in a Fibonacci heap, the total running time of Dijkstra's algorithm is $O(|E| + |V| \log|V|)$; if we use a regular *binary heap*, the time bound is $O(|E| \log |V|)$ [85]. Efficient implementations of Dijkstra's shortest path algorithm using a new data structure, called the *radix heap, o*n a network with *n* vertices, *m* edges, and non-negative integer arc costs bounded by *C, on* a one-level form of radix heap gives a time bound for Dijkstra's algorithm of *O(m + n* log C*).* A two-level form of radix heap gives a bound of *O(m + n* log C/log log C*).* A combination of a *radix heap* and a previously known data structure called a *Fibonacci heap* is due to [86, 87] gives a bound of *O(m + n*/log C*).* The best previously known bounds are *O(m* + *n* log C*)* using *Fibonacci heap,* alone and O(m log n log C*)* using the *priority queue structure* of Van Emde Boas *et al.* [88]. Computer networks provide an application for the shortest-path problems. The goal is to find the cheapest way for one computer to broadcast a message to all other computers on the network. The network can be modeled by a graph with edge weights indicating time or cost to send a message to a neighbouring computer. Dijkstra's algorithm does not use a *min-priority queue* and has a running time of $O(|V|^2)$ but when implemented in a Fibonacci heap, it runs in O(|E|+|V|log|V|), where |E| is the number of edges, |V| the number of vertices and O the upper bound running time. This makes the algorithm asymptotically the fastest known shortest path algorithm for directed graphs with unbounded non-negative weights. For large networks, the classical Dijkstra's algorithm [11] is used to compute shortest paths in robot path-planning [12]; logistics distribution lines [13]; link-state routing (LSR) protocols [14]; open-shortest path first (OSPF) [15] and intermediate system to intermediate system (IS-IS) routing algorithms in computer and telecommunication networks, [16] respectively. The idea of this algorithm is also explained in [23]. An implementation of Dijkstra's pseudo-code and algorithms is shown below in algorithms 1 and 2 respectively:

**Algorithm 1:** Dijkstra's algorithm C++ implementation with priority queue

```
// Dijkstra's shortest paths algorithm with priority queue
void Dijkstra(Graph* G, int* D, int s) {
int i, v, w; // v is current vertex
DijkElem temp;
DijkElem E[G->e()]; // Heap array with lots of space
temp.distance = 0; temp.vertex = s;
E[0] = temp; // Initialize heap array
heap<DijkElem, DDComp> H(E, 1, G->e()); // Create heap
for (i=0; i<G->n(); i++) { // Now, get distances
do {if (H.size() == 0) return; // Nothing to remove
temp = H.removefirst();
v = temp.vertex;
} while (G->getMark(v) == VISITED);
G->setMark(v, VISITED);
if (D[v] == INFINITY) return; // Unreachable vertices
for (w=G->first(v); w<G->n(); w = G->next(v,w))
if (D[w] > (D[v] + G->weight(v, w))) { // Update D
D[w] = D[v] + G->weight(v, w);
temp.distance = D[w]; temp.vertex = w;
H.insert(temp); // Insert new distance in heap
        }
    }
}
```

The classic problem of finding the shortest path over a network has been the target of many research efforts over the years. These research efforts have resulted in a number of different algorithms and a substantial amount of empirical findings with respect to performance. Unfortunately, prior research does not provide a clear direction for choosing an I/O-efficient algorithm when one faces the problem of computing shortest paths on real road networks. Most of the computational testing on shortest path algorithms has been based on randomly generated networks, which may not have the characteristics of real road networks. There are two classes of algorithms to solve shortest path problems viz: Label-setting algorithm (LSA) [11, 76, 77], such as: Dijkstra [11], and Label-correcting algorithm (LCA) such as Gallo and Pallottino's [25, 60] graph growth algorithm with two queues (TWO-Q) data structures and threshold based algorithms such as: Glover & Klingman [24]). The label-correcting (LC) algorithm [20, 32, 34, 94] uses a list structure to manage the scan eligible node set that needs to be examined during the shortest path tree building process. It is the variations of the list operation policy that are used to differentiate the LC algorithms such as LC with *queue* [34], LC with *double-ended queue* [95], and LC with *threshold lists* [24]. The major feature of an LC algorithm is that it cannot provide the shortest path between two nodes before the shortest path to every node in the network is identified. The necessity of this type of operation (referred as *one-to-all* search mode) makes the LC algorithms more suitable in situations when many

shortest paths from a root node need to be found. Based on empirical evidence the LC algorithm is often used in transportation planning applications where multiple routes have to be identified. Both algorithms are iterative and assign tentative distance labels to nodes at each step, which are estimates of the upper bounds complexity on the shortest path distances. Label setting algorithms designate one label as permanent (optimal) at each iteration. Label correcting algorithms consider the labels as temporary until the final step when they all become permanent. According to Gallo & Pallottino, [25, 63] stated that almost all SPT algorithms of practical interest can be derived from one single prototype method. This prototype can be structured as follows:

**Step 1**: Begin with any directed tree *T* rooted at *s* and for each $v \in N$, let *d(v)* be the length of the path from *s* to *v* in T;

**Step 2:** Let $(i,j) \in A$ be an arc for which $d(i) + l(i,j) < d(j)$, then adjust the vector *d* by setting $d(j) = d(i) + l(i,j)$, and update the tree *T* by replacing the current arc incident into node *j* by the new arc *(i,j);*

**Step 3:** Repeat step 2 until $d(i) + l(i,j) \geq d(j)$ for every $(i,j) \in A$.

The basic idea behind this prototype is that one first begins with a tree T rooted at origin s and then iteratively updates this tree by replacing the arcs until all the arcs in the tree meet Bellman's optimality conditions Bellman (1958), which is to say that for every arc (i,j), the path length from origin s to node i plus the length of arc (i,j) is no less than the path length from s to j. Most of the shortest path algorithms follow the above prototype. In fact, all of the SPAs that are considered to be efficient in past tests can be defined within the prototype structure. The basic differences only occur in determining the order in which the improvements of step 2 are made and in the data structure used to store the list of candidates (arcs or nodes) for possible improvement. The famous Dijkstra algorithm, which is the best currently known bounds for SPAs, in combination with priority queues data structures [27, 87, 88, 89, 90, 91, 92] is also consistent with this prototype. It applies a best-first selection strategy at step 2. Essentially, with a best-first strategy, nodes of the shortest path tree are identified in order of the distance from the origin. Thus, the optimality criterion is satisfied in order of distance from the origin. The standard steps of the shortest path algorithms are as follows (assuming that the algorithm starts from the origin node):

***Step* 1:** Initialization: Set $i = o$; $L_{(i)} = 0$; $L_{(j)} = \infty \; \forall \; j \neq i$; $P_{(i)}$ = NULL.
Define the scan eligible node set $Q = \{i\}$;

***Step* 2:** Node Selection: Select and remove a node ($i$) from $Q$.

***Step* 3:** Node Expansion: Scan each link emanating from node $i$. For each link $a = (i, j)$
If
$L_{(i)} + c_a < L_{(j)}$
then
$L_{(j)} = L_{(i)} + c_a$; $P_{(j)} = a$
Insert node $j$ into $Q$

***Step* 4:** Stopping Rule: If $Q = \varnothing$ then STOP.
Otherwise: goto step 2.

The original Dijkstra's algorithm did an unordered search when finding the next candidate node, and has been shown to have a complexity of $O(n^2)$, where $n$ is the number of nodes. Various improvements have been made on the performance of the algorithm by storing candidate nodes in a heap or priority queue structure, and retrieving the node at the front of the queue as the next ***current node***, which speeds up the node selection and has a complexity of $O(m \log (n))$ (Johnson, 1972), where $m$ is the number of arcs. This is the approach that we intend to use in TransRoute. Other improvements include the S-bucket algorithm (Dial, 1969) [22] and an algorithm that uses a Fibonacci heap. This last algorithm has a run time complexity of $O(m+n\log(n))$ Ahuja *et al*., 1990 [27]. Dijkstra's algorithm is shown below:

---

**Algorithm 2:** Dijkstra's Algorithm

---

**INITIALIZATION:**
**for all** $v \in V$ **do**
$dist[v] \leftarrow \infty$
$final[v] \leftarrow false$
$pred[v] \leftarrow -1$
**end for**
$dist[s] \leftarrow 0$
$final[s] \leftarrow true$
$recent \leftarrow s$
*{// vertex s is permanently labeled with 0. All other vertices are temporarily labeled with ∞. Vertex s is the most recent vertex to be permanently labeled //}*

**ITERATION:**
**while** $final[t] = false$ **do**
**for** every immediate successor $v$ of $recent$ **do**
**if** not $final[v]$ **then** *{// update temporary labels //}*
$newlabel \leftarrow dist[recent] + w_{recent,v}$
**if** $newlabel < dist[v]$ **then**
$dist[v] \leftarrow newlabel$
$pred[v] \leftarrow recent$
*{// re-label v if there is a shorter path via vertex recent and make recent the predecessor of v on the shortest path from s //}*
**end if**
**end if**
**end for**
let $y$ be the vertex with the smallest temporary label, which is = ∞
$final[y] \leftarrow true$
$recent \leftarrow y$

*{// y*, the next closest vertex to *s* gets permanently labeled //*}*
**end while**

The original Dijkstra algorithm partitions all nodes into two sets: temporarily and permanently labelled nodes. At each iteration, it selects a temporarily labelled node with the minimum distance label as the next node to be scanned Dijkstra [11]; Ahuja *et al*. [72]. Once a node is scanned, it becomes permanently labelled as stated in section 9. In Dijkstra's original algorithm, temporarily labelled nodes are treated as a non-ordered list. This is equivalent to treating the priority queue Q in the above general procedure for shortest path tree construction as a non-ordered list. This is of course a bottleneck operation because all nodes in Q have to be visited at each iteration in order to select the node with the minimum distance label. A natural enhancement of the original Dijkstra algorithm is to maintain the labelled nodes in a data structure in such a way that the nodes are sorted by distance labels. The bucket data structure is just one of those structures. Buckets are sets arranged in a sorted fashion (Figure 11.4). Bucket *k* stores all temporarily labelled nodes whose distance labels fall within a certain range. Nodes contained in each bucket can be represented with a doubly linked list. A doubly linked list only requires O(1) time to complete an operation in each distance update in the bucket data structure. These operations include: 1) checking if a bucket is empty, 2) adding an element to a bucket, and 3) deleting an element from a bucket, Zhan *et al*. [31, 63, 75]. Dial (1969) was the first to implement the Dijkstra algorithm using buckets. In Dial's implementation, bucket *k* contains all temporarily labelled nodes whose distance labels are equal to *k*. Buckets numbered 0, 1, 2, 3,..., are checked sequentially until the first non-empty bucket is identified. Each node contained in the first nonempty bucket has the minimum distance label by definition. One by one, these nodes with the minimum distance label become permanently labelled and are deleted from the bucket during the scanning process. The position of a temporarily labelled node in the buckets is updated accordingly when the distance label of a node changes. For example, when the distance label of a temporarily labeled node is changed from d(1) to d(2), this node is moved from bucket d(1) to bucket d(2). This process is repeated until all nodes are permanently labelled. Dial's original implementation of the Dijkstra algorithm (DKB) requires *nC+1* buckets in the worst case, where *C* is the maximum arc length of a network. However, it has been proven that for a network with a maximum arc length of *C*, only *C+ 1 bucket* are needed to maintain all temporarily labelled nodes (Ahuja *et al*. 1993, pp.113-114). In the *approximate bucket* implementation of the Dijkstra algorithm (DKA), a bucket *i* contains those temporarily labelled nodes with distance labels within the range of [*i*b*, (*i+1*)* *b-1*], where *b* is a chosen constant. Here approximate means that the values of the distance labels in a bucket are not exactly the same as in the case of DKB, but are within a certain range. Nodes in each bucket are maintained in a FIFO queue. Algorithm DKA requires a total of *largerInteger(C/b)+1* buckets. The worst case complexity of DKA is *O(mb+n(b+C/b))*. It can be seen that this algorithm trades speed for space. Each node can be scanned more than once, but a node cannot be scanned more than b times. *The Dijkstra's algorithm implemented with double buckets (DKD)*: The *double bucket* implementation of the Dijkstra's algorithm (DKD) combines the ideas of the above two algorithms DKM and DKA. Two levels of buckets, high- level and low- level, are

maintained in the DKD implementation. A total of *d* buckets in the low- level buckets are used. A bucket *i* in the high- level buckets contains all nodes whose distance labels are within the range of [*i*d, (i+1)* d-1*]. In addition, a nonempty bucket with the smallest index *L* is also maintained in the high- level buckets. A low- level bucket *d(j)-L*d* maintains nodes whose distance labels are within the range of [*L*d, (L+1)* d-1*]. Nodes in the low- level buckets are examined during the scanning process. After all nodes in the low-level buckets are scanned, the value of *L* is increased. When the value of *L* increases, nodes in the non-empty high-level buckets are moved to its corresponding low- level buckets, and the next cycle of scanning process begins, Zhan *et al*. 1996, 1997 & 1998 [31, 63, 75].

According to Gutenschwager [2], the routing of vehicles or personnel in complex logistics systems is a task that needs to be solved in numerous applications, e.g., detailed models of transport networks or order picking areas. The number of relevant nodes in such networks can easily exceed 10,000 nodes. Often, a basic task is finding the shortest path from one node (start) to another (destination). Within the last years various simulation tools have been extended by respective algorithms. However, the execution time of the simulation model may significantly depend on the number of nodes in the network. Present algorithms from the literature and a comparison of these algorithms with respect to execution time and model size for different scenarios. We further present an approach to work with so called sub-networks, i.e., the network is separated into areas, where finding the shortest path includes the task of starting at a node in one sub-network with a destination in another one. There is a variety of applications where simulation models are set up to analyze systems in which the routing of personnel, e.g., workers or order pickers, and vehicles is a relevant part of the modeled process. A basic task in this field is to find the shortest path from a current position to a given destination (within a graph of nodes). For this problem, a number of algorithms are known. Several simulation systems offer an automatic routing of vehicles, such that a vehicle takes the shortest path from the current position (node) to a destination node automatically. Due to implementation details and the used algorithm, the performance of simulation tools used in the domain of production and logistics such as: Technomatix Plant Simulation from Siemens, AutoMod from Applied Materials, and Enterprise Dynamics from INCONTROL Simulation Solutions [2, 71] for finding a shortest path may differ in some most relevant aspects viz:

- Maximum size of transport networks (number of nodes) that can be simulated
- Average computational time to fulfill a certain number of transport orders during a simulation run
- Average computational time and resulting model size to initialize the corresponding graph and matrices where shortest paths are stored

**Table 1:** Evaluated Algorithms

| CLASS | ABBREVIATION | IMPLEMENTATION DATA STRUCTURES |
|---|---|---|
| A*-Search | ASH | A*-Search algorithm with k-array Heap |
| | ASBD | A*-Search algorithm with Double Buckets |
| | ASBA | A*-Search algorithm with Approximate Buckets |

| | | |
|---|---|---|
| | | data structures |
| Bellman-Ford-Moore | BFM<br>BFP | Bellman-Ford-Moore<br>Bellman-Ford-Moore with parent- - checking |
| Dijkstra | DKQ<br>DKB<br>DKD<br>DKA<br>DKM<br>DKF<br>DKH<br>DKR | Dijkstra's Naive Implementation<br>Dijkstra's Buckets - - Basic Implementation<br>Dijkstra's Buckets - - Double Implementation<br>Dijkstra's Buckets - - Approximate<br>Dijkstra's Buckets - - Overflow Bag<br>Dijkstra's Heap - - Fibonacci<br>Dijkstra's Heap - - k- - array<br>Dijkstra's Heap - - R- - Heap |
| Graph Growth Model | TQQ (Two-Q)<br><br>PAP | Graph Growth - - Gallo & Pallottino algorithm with two queues data structures<br>Graph Growth - - Pape |
| Threshold Algorithm | THR | Threshold Algorithm |
| Topological Ordering | GR1 | Topological Ordering - - Basic |
| Topological Ordering | GR2 | Topological Ordering - - Distance Updates |

**Table 2:** Complexity Analysis of different shortest paths algorithms

| **Algorithm** | **Negative Edge** | **Single Source** | **All Sources** | **Time Complexity** | **Space Complexity** |
|---|---|---|---|---|---|
| Dijkstra | | √ | | $O(\|E\| + \|V\| \log\|V\|)$ | $O(V^2)$ |
| Bellman-Ford | √ | √ | | $O(\|V\|.\|E\|)$ | $O(V^2)$ |
| Floyd-Warshall | √ | | √ | $O(V^3)$ | $O(V^3)$ |
| A*-Search | | √ | | A function of the heuristic | A function of the heuristic |
| Johnson's | √ | | √ | $O(V^2 \log V + VE)$ | A function of the heuristic |
| Prim's | | √ | √ | O (VLog\|V\|+ELog \|V\|) | $O(\|E\| + \|V\|^2)$ |
| Kruskal's | | √ | √ | $O(\|E\| \log\|V\|)$ | $O(\|E\|\log\|E\|)$ |

## References


[1] Ahuja, R.K. and Orlin, J.B. (1993). Graph and Network Optimization. In Journal of Optimization and Operations Research, Vol. II.

[2] Gutenschwager, K; Volker, S; Radtke, A; and Zeller, G. (2012). The Shortest Path: Comparison of Different Approaches and Implementations for the Automatic Routing of Vehicles. In Proceedings of the 2012 Winter Simulation Conference, IEEE, Munich, Germany. pp. 3312-3323.

[3] Schaffer, C.A. (1998). A Practical Introduction to Data Structures and Algorithm Analysis, Prentice Hall, pp. 401-404.

[4] Singh, R.P., and Vandana (2014). Application of Graph Theory in Computer Science and Engineering, International Journal of Computer Applications (0975 – 8887), Volume 104 – No.1, pp. 10-12.

[5] S.G. Shrinivas *et al.* (2010). Applications of Graph Theory in Computer Science: An Overview, International Journal of Engineering Science and Technology, Vol. 2(9), pp. 4610-4621.

[6] Narasingh D. (1990). Graph theory with applications to engineering and computer science, Prentice Hall of India.

[7] Venugopal, D. (2015). Application of Graph Theory in Computer Science Engineering, International Journal of Science, Technology & Management, Volume No. 04, Special Issue No. 01, pp.1192-1198.

[8] Tosuni, B. (2005). Some interesting topics of graph theory in modern Computer Science, European Journal of Mathematics, Technology and Computer Science, ISSN 1946-4690, pg.

[9] Grama, A; Gupta, A; Kampis, G and Kumar, V. (2003). Graph Algorithms.

[10] Orlin, J.B; Madduri, K; Subramani, K and Williamson, M. (2010). A faster algorithm for the single source shortest path problem with few distinct positive lengths. In Journal of Discreet Algorithms-Elsevier, Vol. 8, pp.189-198.

[11] Dijkstra, E.W. (1959). A note on two problems in connexion with Graphs, available at: (http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf). Numerische Mathematik**1**.pp.269-271.DOI:10.1007/BF01386390 (http://dx.doi.org/10.1007/BF01386390) [Accessed: Wednesday, June 1, 2016.]

[12] Huijuan, W. et al. (2011). Application of Dijkstra algorithm in robot path-planning, Second International Conference on Mechanic Automation and Control Engineering (MACE), pp. 1067-1069.

[13] Liu, X-Y and Yan-Li, C. (2010). Application of Dijkstra Algorithm in Logistics Distribution Lines, Proceedings of the Third International Symposium on Computer Science and Computational Technology (ISCSCT '10), pp. 048-050.

[14] Abujassar, S.R. and Ghanbari, M. (2011). Efficient algorithms to enhance recovery schema in link state protocols, International Journal of Ubicomp (IJU), Vol. 2, No. 3, Doi: 10.5121/IJU.2011.2304 53.

[15] Coltun, R., Ferguson, D., Moy, J.A. and Lindem. (2008). OSPF for IPv6. The Internet Society. OSPFv3. (https://en.wikipedia.org/wiki/Open_Shortest_Path_First).

[16] Katz, D. (2000). North American Network Operators Group NANOG 19. Albuquerque. OSPF vs IS-IS.

[17] Zeng, W., & Church, R.L. (2009). Finding shortest paths on real road networks: the case for A*. In International Journal of Geographic Information Science, Vol. 23, No. 4, pp.531-543.

[18] Dantzig, G.B. (1957). Discrete-variable extremum problems. Operations Research, 5, pp. 266–277.

[19] Minty, G. (1957). A comment on the shortest route problem. Operations Research, 5, pp. 724.

[20] Ford, L.R. (1956). Network flow theory. Technical Report P-923 (Santa Monica, CA: The Rand Corporation).

[21] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100–107. DOI:10.1109/TSSC.1968.300136.

[22] Dial, R. B. (1969). Algorithm 360: Shortest Path Forest with Topological Ordering. C*ommunications of the ACM,* 12, 632-633.

[23] Dial, R. B., Glover, F., Karney, D., and Klingman, D. (1979). A Computational Analysis of Alternative Algorithms and Labelling Techniques for Finding Shortest Path Trees. *Networks*, 9, 215-248.

[24] Glover, F., Klingman, D., and Philips, N. (1985). A New Polynomially Bounded Shortest Paths Algorithm. *Operations Research*, 33, 65-73.

[25] Gallo, G., and Pallottino, S. (1988). Shortest Paths Algorithms. *Annals of Operations search*, 13, 3-79.

[26] Hung, M. H., and Divoky, J. J. (1988). A Computational Study of Efficient Shortest Path Algorithms. *Computers & Operations Research*, 15, 567-576.

[27] Ahuja, R. K., Mehlhorn, K., Orlin, J. B., and Tarjan, R. E. (1990) Faster algorithms for the Shortest Path Problem. *Journal of the Association for Computing Machinery*, Vol. 37, No. 2, pp. 213-223.

[28] Mondou, J.-F., Crainic, T. G., and Nguyen, S. (1991) Shortest Path Algorithms: A Computational Study with the C Programming Language. *Computers & Operations Research*, 18, 767-786.

[29] Cherkassky, B. V.; Goldberg, A. V.; and Radzik, T. (1993). Shortest paths algorithms: Theory and experimental evaluation. Mathematical Programming 73:129–174.

[30] Goldberg, A. V., and Radzik, T. (1993). A Heuristic Improvement of the Bellman-Ford Algorithm. *Applied Mathematics Letter,* 6, 3-6.

[31] Zhan, F. B., and Noon, C. E. (1996). Shortest Path Algorithms: An Evaluation Using Real Road Networks. *Transportation Science, pp.65-73* (in press).

[32] Bellman, R. (1958). On a routing problem. Quarterly of Applied Mathematics 16: 87–90. MR 0102435 (http://www.ams.org/mathscinet-getitem?mr=0102435). [Accessed: Wednesday, June 1, 2016].

[33] Ford Jr., L.R. (1956). Network Flow Theory (http://www.rand.org/ pubs/papers/P923.html). Paper P-923. Santa Monica, California: RAND Corporation, U.S.A.

[34] Moore, E.F. (1959). The Shortest Path through a Maze. Proc. Internat. Sympos. Switching Theory 1957, Part II. Cambridge, Mass.: Harvard Univ. Press. pp. 285– 292. MR 0114710 (http:/ / www. ams. org/mathscinet-getitem?mr=0114710).

[35] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Clifford, C. (2001). Introduction to Algorithms (1st ed.). MIT Press and McGraw-Hill.

- Section 26.2, The Floyd–Warshall algorithm, pp. 558–565;
- Section 26.4, a general framework for solving path problems in directed graphs, pp. 570–576.

[36] Floyd, R.W. (1962). Algorithm 97: Shortest Path. Communications of the ACM **5** (6): 345. DOI:10.1145/367766.368168 (http://dx.doi.org/10.1145/367766.368168).

[37] Ingerman, P.Z. (1962). Algorithm 141: Path Matrix. Template: Communications of the ACM **5** (11): 556. Doi: 10.1145/368996.369016 (http://dx.doi.org/10.1145/368996.369016).

[38] Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In C.E. Shannon and J. McCarthy. Automata Studies. Princeton University Press. pp. 3–42.

[39] Warshall, Stephen (1962). A Theorem on Boolean Matrices. Journal of the ACM**9** (1): 11–12. Doi: 10.1145/321105.321107 (http://dx. doi. org/10.1145/ 321105.321107).

[40] Pearl, J. (1984). Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley.

[41] Russell, S. J.; Norvig, P. (2003). Artificial Intelligence: A Modern Approach. Upper Saddle River, N.J.: Prentice Hall. pp. 97–104.

[42] Doran, J. (1967). An Approach to Automatic Problem-Solving. Machine Intelligence, 1:105–127.

[43] Klein, D., Manning, C.D. (2003). A* parsing: fast exact Viterbi parse selection. Proc. NAACL-HLT.

[44] Johnson, D.B. (1977). Efficient algorithms for shortest paths in sparse networks. Journal of the ACM 24:1-13.

[45] Black, P.E. (2004). Johnson's Algorithm, Dictionary of Algorithms and Data Structures, National Institute of Standards and Technology.

[46] Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: A System for Large-Scale Graph Processing. In proceedings of SIGMOD'10, ACM 978-1-4503-0032-2/10/06, Indianapolis, Indiana, USA.

[47] Jonathan, L.G., and Yellen, J. (2005). Graph Theory and Its Applications. (2nd Edition), Chapman and Hall/CRC.

[48] Lumsdaine, A., Gregor, D., Hendrickson, B., and Berry, J.W. (2007). Challenges in Parallel Graph Processing. Parallel Processing Letters 17, 5{20}.

[49] George, B. (?). Königsberg Bridge Problem – First Draft, UM ID #2582042, CSCi 8701, G04

[50] http://mathforum.org/isaac/problems/bridges1.html [Accessed: June 2, 2016].

[51] http://mathworld.wolfram.com/KoenigsbergBridgeProblem.html [Accessed: June 6, 2016].

[52] http://en.wikipedia.org/wiki/Seven_Bridges_of_Konigsberg [Accessed: June 1, 2016].

[53] Chartrand, G. (1985). The Königsberg Bridge Problem: An Introduction to Eulerian Graphs. In Introductory Graph Theory. New York: Dover, 1985.

[54] http://www.cut-the-knot.org/do_you_know/graphs.shtm [Accessed: June 10, 2016].

[55] Willhalm, T. (2005). Engineering Shortest Paths and Layout Algorithms for Large Graphs. Ph. D. Thesis, Karlsruhe University, Germany.

[56] Sturtevant, N.R., and Geisberger, R. (2010). A Comparison of High-Level Approaches for Speeding up Path finding. In Proceedings of the Sixth Conference on Artificial Intelligence and Interactive Digital Entertainment, 76-82. Palo Alto, California, U.S.A.

[57] Sanan, S; Jain, L and Kappor, Bharti. (2013). Shortest Path Algorithm, In International Journal of Application or Innovation in Engineering and Management (IJAIEM), Vol. 2, Issue 7. Available at: http://www.ijaiem.org [Accessed: May 29, 2016].

[58] Dreyfus, S.E. (1969). An appraisal of some shortest-path algorithms. Operations Research, 17, pp. 395–412.

[59] Fu, L., Sun, D. and Rilett, L.R. (2006). Heuristic shortest path algorithms for transportation applications: state of the art. Computers and Operations Research- Elsevier, 33, pp. 3324–3343.

[60]Gallo, G. and Pallottino, S. (1986). Shortest path methods: a unifying approach. Mathematical Programming Study, Vol. 26, pp. 38–64.

[61] Duckham, M. and Kulik, L. (2003). Simplest paths: automated route selection for navigation, In: W. Kuhn, M. F. Worboys and S. Timpf (Eds.), *Spatial Information Theory: Foundations of Geographic Information Science*, pp. 182 – 199.

[62] Mark, D. M. (1985). Finding simple routes: 'ease of description' as an objective function in automated route selection, in: *Proceedings, Second Symposium on Artificial Intelligence Applications (IEEE)*, Miami Beach, 577-581.

[63] Zhan, F. B. and Noon, C. E. (1998). Shortest path algorithms: an evaluation using real road networks, *Transportation Science*, 32.1, 65-73.

[64] Sanders, P. and Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries, in: Brodal G.S. and Leonardi S. (eds.), *Proceedings of the 13th Annual European Symposium*, Palma de Mallorca, Spain, October 3-6, 2005, 568–579.

[65] Golledge, R. G. and Gärling, T. (2004). Cognitive maps and urban travel, In: D. A. Hensher, K. J. Button, K. E. Haynes and P. R. Stopher (Eds.), *Handbook of Transport Geography and Spatial Systems*, Elsevier: Amsterdam, 501 - 512.

[66] Wijesinghe *et al.* (2013). GIS-Enabled Travel Planner System with TSP Implementation–PNCTM; Vol. 2.

[67] Sommer, C. (2006). Approximate shortest path and distance queries in networks, Graduate school of information science and technology, University of Tokyo, Computer Science Department, M.Sc thesis pg. 12.

[68] Uri, Z. (2001). All pairs shortest paths in weighted directed graphs exact and almost exact algorithms. In Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), Vol. 12, pp. 51-54.

[69] Zhao, L.J and Hsing, K.C. (1996). Shortest path algorithms: An evaluation using real road networks, Journal of Transportation Science and Technology, Multi-Science publishing Co. Ltd, Hockey Essex SS54AD UK, Vol.11, pp. 24-25.

[70] Bjorn, P., and MadsKehlet, J. (2006). Partial Path Column Generation for the Vehicle Routing Problem with Time Windows, European Journal of Operational Research, Elsevier Online publishers Ltd, Boulevard Langford lane Kindlington Oxford United Kingdom.Vol. 114, pp.302 – 305.

[71] Olivier, G., Lemaire, P., Eric, P., and Rivreau, R. (2010). Cut generation for an integrated employee timetabling and production scheduling problem. European Journal of Operational Research, Boulevard Langford lane Kindlington Oxford, U.K 201:557 – 567.

[72] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). Network Flows: Theory, Algorithms and Applications. Englewood Cliffs, NJ: Prentice Hall. Available at: http://publish.uwo.ca/~jmalczew/gida_1/Zhan/Zhan.htm (accessed 13/6/2016 5:12:03 PM).

[73] Nagi, R. (1994). IE 680 – Special Topics in Production Systems: Networks, Routing and Logistics, Department of Industrial Engineering, University of Buffalo, School of Engineering and Applied Sciences.

[74] Cherkassky, B.V., Goldberg, A.V. and Radzik, T. (1996). Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming: Series A and B, 73, pp. 129–174.

[75] Zhan, F. B. (1997). Three fastest shortest path algorithms on real road networks, *in Journal of Geographic Information and Decision Analysis,* vol.1, No.1, pp. *70*-82.

[76] Dantzig, G.B. (1960). On the shortest route through a road network. Operational Research Quarterly, 6, pp. 187–190.

[77] Whiting, P.D, Hillier, J.A. (1960). A method for finding the shortest route through a road network. Operational Research Quarterly; 11:37–40.

[78] Geisberger, R; and Shieferdecker, D. (2013). Advanced Route Planning in Transportation Networks, In Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'13), Universitat des Landes Baden Wurttemberg Germany. Available at: www.algo2.iti.kit.edu/publications.php (accessed on: Wednesday, June 1, 2016).

[79] Erickson, J. (2014). Algorithms. Lecture 21: Shortest Paths [Fa'14], p.4. Available via: http://www.cs.uiuc.edu/~jeffe/teaching/algorithms (accessed on: June 16, 2016).

[80] Sedgewick, R., and Wayne, K. (2011). Algorithms. Pearson Education. 4th edition.

[81] Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, U.S.A.

[82] Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D. (1987). Data Structures and Algorithms. Addison-Wesley, Reading, MA, U.S.A.

[83] Sniedovich, M. 2006. Dijkstras algorithm revisited: the dynamic programming connexion. Control and Cybernetics 35(3):599–620.

[84] Knuth, D.E. (1977). A Generalization of Dijkstra's Algorithm. Information Processing Letters **6** (1): 1–5.

[85] http://www.cs.uiuc.edu/~jeffe/teaching/algorithms [Accessed: June 1, 2016.]

[86] Ralston, B., Tharakan, G., and Liu, C. (1994). A Spatial Decision Support System for Transportation Policy Analysis. *Journal of Transport Geography*, Vol. 2, pp. 101- 110.

[87] Fredman, M.L; Tarjan, R.E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms In *Journal of the Association for Computing Machinery (ACM)* 34(3):596–615. Available at: (http://portal.acm.org/citation.cfm?id=28874). Doi: 10.1145/28869.28874 [Accessed: June 1, 2016].

[88] Van Emde Boas, P., Kaas, R., and Zijlstra, E. (1977). Design and implementation of an efficient priority queue. *Math. Syst.Theory 10*, pp. 99- 27.

[89] Cherkassky, B.V., and Goldberg, A.V. (1996). Heap-on-Top Priority Queues. Technical Report 96-042, NEC Research Institute, Princeton, NJ, U.S.A.

[90] Fredman, M.L; and Willard, D.E. (1990). Trans-dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. In *Proc. 31stst IEEE Annual Symposium on Foundations of Computer Science*, pp. 719-725.

[91] Thorup, M. (1996). On RAM priority queues. *SIAM J. Comp.*, 30(1):86–109, 2000.

[92] Thorup, M. (2002). Equivalence between priority queues and sorting, 2002. in Proc. FOCS'02.

[93] Vahrenkamp, R., and Mattfeld, D. C. (2007). Logistiknetzwerke: Modelle f¨ur Standortwahl und Tourenplanung. Wiesbaden: Gabler.

[94] Ford Jr, L.R; and Fulkerson, D.R. (1962). Flows in networks. Princeton, NJ: Princeton University Press, U.S.A.

[95] Pape, U. (1974). Implementation and efficiency of Moore algorithms for the shortest root problem. Mathematical Programming; 7: 212–22.

[96] Knopp, S., Sanders, P., Schultes, D., Schulz, F., and Wagner, D. (2007). Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In Proceedings of the Workshop on Algorithm Engineering and Experiments, 36–45. New Orleans, Louisiana.

[97] Thorup, M. (2008). Undirected single-source shortest paths with positive integer weights in linear time", Journal of the ACM Vol. 154. pp. 362- 394.

[98] Goldberg, A.V. (1993). Scaling Algorithms for the Shortest Paths Problem. *In Proc. 4nd ACM-SIAM Symposium on Discrete Algorithms*, pp. 222-231.

[99] Yen, J.Y. (1970). An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics* **27**: 526–530.MR0253822 (http://www.ams.org/mathscinet-getitem?mr=0253822). [Accessed: June 1, 2016].

[100] Bannister, M. J.; Eppstein, D. (2012). Randomized speedup of the Bellman–Ford algorithm. (http://siam.omnibooksonline.com/2012ANALCO/data/papers/005.pdf). *Analytic Algorithmics and Combinatorics (ANALCO12), Kyoto, Japan*. pp. 41–47. arXiv: 1111.5414 (http://arxiv.org/abs/1111.5414). [Accessed: June 1, 2016].

[101] Erkut, E. (1996) The Road Not Taken. *ORMS Today*, 23, 22-28.

[102] Mehlhorn, K & Sanders, P. (2007). Algorithm and Data Structures: The Basic Toolbox, Springer-Verlag, pp. 74-97.

[103] Ojaswa, S; Darka, M; Francois, A; and Girija, D. (2005). Travelling Salesperson Approximation Algorithm for Real Road Networks, Technical Report 388, Department of Geomatics Engineering, University of Calgary, 2500, University Drive NW, Calgary, AB, Canada, T2N 1N4.