Nigerian Journal of Technology (NIJOTECH)
Vol. 36, No. 3, July 2017, pp. **876 – 888**
Copyright© Faculty of Engineering, University of Nigeria, Nsukka,
Print ISSN: 0331-8443, Electronic ISSN: 2467-8821
www.nijotech.com
http://dx.doi.org/10.4314/njt.v36i3.30

COMPARATIVE ANALYSIS AND IMPLEMENTATION OF DIJKSTRA'S SHORTEST PATH ALGORITHM FOR EMERGENCY RESPONSE AND LOGISTIC PLANNING

A. H. Eneh[1,*] and U. C. Arinze[2]
[1,2] DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF NIGERIA, NSUKKA, ENUGU STATE, NIGERIA.
E-mail addresses: *[1] agozieh.eneh@unn.edu.ng, [2] uchechukwu.arinze.pg79296@unn.edu.ng*

ABSTRACT
TransRoute: a web-based vehicle route planning application is proposed in this paper. This application leverages existing input-output (I/O) efficient implementations of shortest path algorithms (SPAs) to implement the proposed system that will fundamentally address the problems experienced in moving people, goods and services from one location to another. A number of SPAs are evaluated using landau notations. Main functionalities of the system will be implemented as a web-enabled geographic information system (GIS) application based on open-source technologies and object-oriented software development methodology using unified modeling language. Pilot implementation is done based on spatial data of three selected states in Nigeria, pulled from web-based mapping tools like Google Maps and Microsoft Bings respectively. In conclusion, the Dijkstra's algorithm implemented with double bucket dynamic data structure is selected for implementing the proposed route planning system, as past research efforts has proven that it is the fastest with run-time improvements from O(m + n/log C) to O(m) respectively.

Keywords: *TransRoute, shortest path algorithms, double bucket data structure, GIS*

1. INTRODUCTION

In this paper, we propose *TransRoute*: a web-based, route planning system that will leverage advanced data structures, shortest path algorithms, 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 Nigerian economy, hence substantially helping in saving lives, increase bottom line of logistic firms and improve efficiency in service delivery.

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. A plethora of route queries problems in emergency response, road traffic simulation, logistic planning and transportation sector of the economy exists, occasioned by 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 the need for efficient, effective and timely response to emergencies, facility location, network flows, vehicle routing and delivery of logistic package by courier firms and emergency response agencies like: DHL, FedEx, UPS, EMS, NEMA, hospitals and fire service, from source to their respective destinations at minimal cost and promptly has been a major challenge faced by these firms, agencies and application developers around the world due to its similarity to the travelling salesman problem (TSP) which requires that: *given the cost of travel between n cities or stops, the objective is to minimize the total cost of a single route that visits all n cities/stops and returns to its starting point.* At a glance, this type of problem may appear seemingly trivial, given the relative ease of determining the shortest possible route for single-stop routes using algorithms such as A*-Search, Dijkstra, Bellman-Ford and Floyd-Warshall algorithms respectively. However, any given TSP route is actually one of many possible combinations of multiple single-stop routes, thus making this type of multi-stop routing non-deterministic polynomial time (NP) hard, combinatorial optimization problem that can be modeled using equation 1 below:

$$R = N! \qquad (1)$$

Where R is the number of unique possible TSP routes for a given number, N of cities/stops. Based on the above equation, If the starting city is fixed for all solutions, there are only (N-1)! possible solutions. These expressions make TSP routing a problem that exhibit extreme combinatorial explosion, hence as the number of

* **Corresponding author, tel: +234 - 807 – 675 – 6975**

stops to be solved increases, the problem becomes exponentially more complex. An example of the effects of combinatorial explosion on search space of possible solutions for this type of routing problem is shown in Table 1.

Table 1: Effects of combinatorial explosion on search space of possible solutions

Number of Route Stops	Number of Possible Routes
5	120
10	3,628,800
15	1,307,674,368,000
20	2,432,902,008,176,640,000
30	2.65e+32
40	8.16e+47
50	3.04e+64

The shortest path problem (SPP) is compounded by the slow and inaccurate results of current commercial solutions and the expansion of existing road networks, covering hundreds of thousands and millions of road junctions. Using simple approaches yields very slow query times, which can either be inconvenient for the end-users or expensive for the route service provider. Using advanced heuristic techniques yields inaccurate results. For the client, this can mean waste of time and money and for the service provider; the application development process becomes a difficult balancing act between speed and sub-optimality of the computed routes, hence the considerable interest in the development of more efficient and accurate route planning techniques leveraging robust mapping applications like Google maps, MapQuest and Microsoft Bings respectively.

To achieve the aim of this research, a road network can be represented as a directed, weighted graph, $G = (V, E)$, as shown in figure 1, with a vertex set V with n vertices, where each vertex refers to a city and an edge set $E \subseteq V \times V$ with m edges, where edges connect one city to another city, a weight function $w : E \longrightarrow \mathbb{R}_0^+$ assigns a non-negative weight $w((u, v))$ to each edge (u, v). The weights on the edges indicate the distance between two connected cities, the 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, [1].

SPAs have 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.

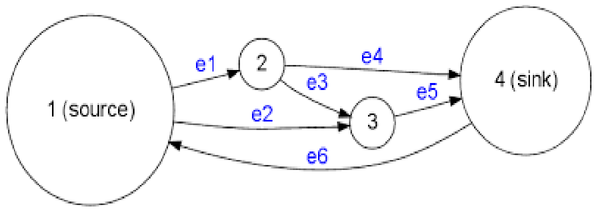
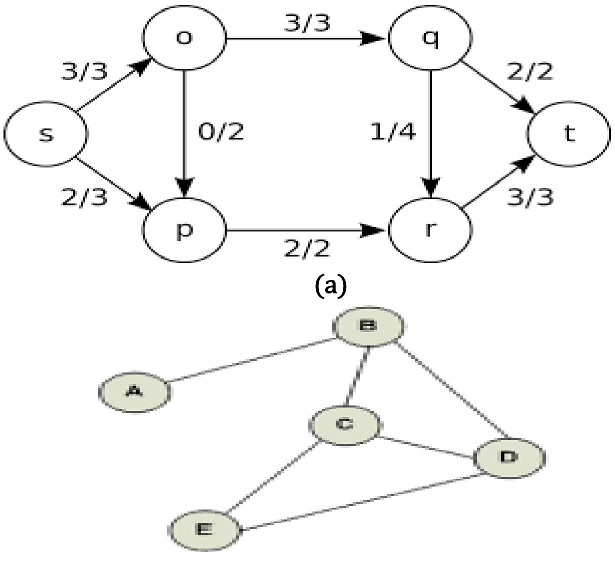


Figure 1: Showing source and sink nodes for a directed, weighted graph

It is also relatively easy, solutions in typical graphs like road networks visit a tiny fraction of vertices, with [2] observing visits to 80,000 vertices out of 32 million in one example [3]. 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 [4], started with the problem of Königsberg Bridge in 1735, [5 – 10]. This problem led to the concept of Eulerian graph. A graph as shown in figure 2 (a-b), is a pair **G = (V, E)**, [11], [12]; 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 $e = U^*V$ is given the value $f(U^*V) = f(u)^*f(v)$, 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 $|V|$, and the number of edges is written as $|E|$. $|E|$ can range from zero to a maximum of $|V|^2 - |V|$.



(b)

***Figure 2:** Examples of graphs: (a). Directed graph (digraph), (b). Undirected graph*

The linear programming (LP) formulation [13] for the shortest path problem (SPP) is:

$$\text{Min} \sum_{(i,j)\in A} d_{i,j}x_{i,j} \tag{2}$$
$$\text{subject to} \sum_{(i,k)\in A} x_{i,k} - \sum_{(j,i)\in A} x_{j,i} = 0 \quad \forall i \neq s, t \tag{3}$$
$$\sum_{(s,i)\in A} x_{s,i} = 1 \tag{4}$$
$$\sum_{(j,t)\in A} x_{j,t} = 1 \tag{5}$$
$$0 \leq x_{i,j} \leq 1 \tag{6}$$

There are two classes of algorithms to solve shortest path problems viz: Label-setting algorithm (LSA) [14 - 16], and Label-correcting algorithm (LCA) such as [17 - 19] graph growth algorithms with two queues (TWO-Q) data structures and threshold based algorithms such as: [20]. The LCA uses a list data 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 LCA such as LCA with *queue* [21], and LCA with *threshold lists* [20]. The major feature of an LCA 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 LCA more suitable in situations when many shortest paths from a root node need to be found. Based on empirical evidence the LCA 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. LSA designate one label as permanent (optimal) at each iteration. LCA consider the labels as temporary until the final step when they all become permanent. According to [22, 23] almost all SPAs of practical interest can be derived from one single prototype method viz:

```
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 $Q = \emptyset$ then STOP
        Otherwise: goto step 2
```

In the view of [24], 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 [14, 24]. Once a node is scanned, it becomes permanently labelled. 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 . 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: a). checking if a bucket is empty; b). adding an element to a bucket; and c). deleting an element from a bucket, [16, 25, 26]. Table 1 and 2 summarize the various algorithms tested and various SPAs runtime complexities obtained using Landau notation.

The routing of vehicles or personnel in complex logistic systems is a task that needs to be solved in numerous applications, e.g., detailed models of transport networks or order picking areas, whose number of relevant nodes can easily exceed 10,000 nodes [27]. Graphs provide the ultimate in data structure flexibility, because they can model both real-world systems and abstract problems; hence find wide applications [28 – 34].

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 [14] to compute shortest path in robot path-planning [35]; logistics distribution lines [36]; link-state routing (LSR) protocols [37]; open-shortest path first (OSPF) [38] and intermediate-system-to-intermediate-system (IS-IS) routing protocols and algorithms in computer and telecommunication networks, [39] respectively are used.

The rest of this paper is organised as follows. Section 2 details selected related works in the field of SPAs. Systems composition, methodology and architecture are outlined in section 3. The results of the various computational algorithmic simulations and software implementations are stated in Section 4. Discussion and analysis of the results obtained are discussed in section 5 and a brief conclusion is provided in section 6.

2. RELATED WORKS

The review of related works 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 input restrictions so as to contextualize our work [34]. Considerable empirical studies on the performance of shortest path algorithms have been reported in the literature, interested readers are referred to [18 - 22, 25 – 27, 34].

The first polynomial time, greedy-based algorithm for the SPA problem was conceived by [14] in 1956 and 'published' in 1959 [1, 14, 40, 41]. The implementation and simplest possible description of Dijkstra's algorithm has been provided in section 4 and other classical books such as Cormen *et al* [23], Sedgewick and Wayne [42], Aho, Hopcroft, and Ullman [43] and need no further description. The run time of the algorithm is dependent on the priority queue implementation data structure which is a part of the algorithm. 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. A combination of a *radix heap* and a previously known data structure called a *Fibonacci heap* is due to [44], [45] 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.* [46].

In another early contribution, [47] developed a search strategy, called A*-Search, to solve for minimum cost paths, but a recent study by [48] is one of the most comprehensive evaluations of shortest path algorithms to date. They evaluated a set of 17 SPAs. In their experiment, they coded the 17 algorithms using 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. The results of their studies 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. Using the open-source codes written by [49, 50] conducted an evaluation of 15 of the 17 algorithms on a variety of real road networks. They concluded that TWO-Q, DKD and Dijkstra algorithm incorporating approximate buckets (DKA, which is a variant of the bucket approach than DKD) are the three fastest one-to-all SPAs.

In a subsequent study, [23, [50 – 52] 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.

Fu *et.al* [53], proposed an optimal heuristic search strategy to find shortest path for Vehicle Navigation System by physically cutting off area within which the shortest path is not supposed to appear, hence known as the restricted search strategy.

Given the works of [3, 49 – 52] 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 as shown in table 1. Table 2 shows the performance of the SPAs. For the sake of this research we will analyse Dijkstra's SPA with double bucket [14], Hart *et al* A*-Search algorithm [47] and Fu *et.a*l [53] restricted search heuristic respectively.

3. SYSTEM COMPOSITION, METHODOLOGY AND ARCHITECTURE

A web-based, GIS-enabled model is selected as the implementation approach, as we envisioned the realized system would be accessible online by any web user from any location within the geographical test area of Anambra, Enugu and Abuja, F.C.T. A three-tier architectural framework is adopted towards implementing the proposed system. Hence, the system will have-user interface, application logic, data and server layers respectively as shown in figure 3 and 4 respectively. PHP which is widely used as a general-purpose scripting language that is especially suited for web application development is used as the implementation language, alongside model-view-controller (MVC) and Laravel code framework. It can be embedded in (X)HTML or XML markup languages respectively. It serves as a tool for creating dynamic data-driven web pages. Google maps mapping service is integrated to render spatial data into map data for route visualization so as to enhance user experience and look-and-feel of the application. Apache MySQL 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. Object-oriented software development methodology with unified modeling language (UML) architectural framework for designing the system components, as shown in Figures 5, 6, 7 and 8 respectively.

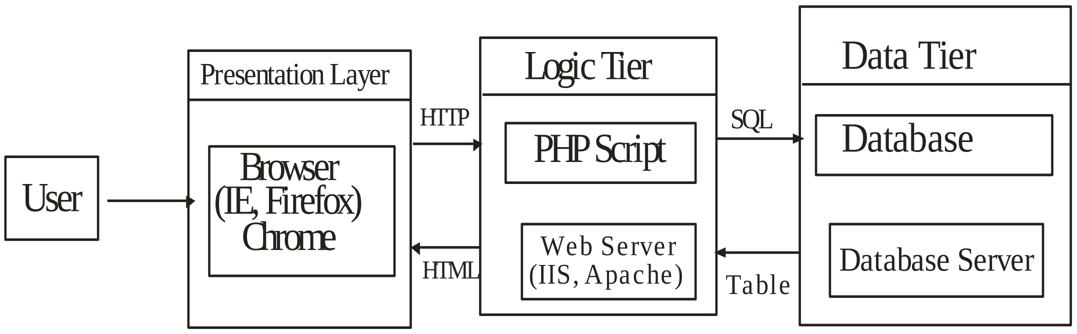


Figure 3: Three-tier architectural framework of TransRoute application

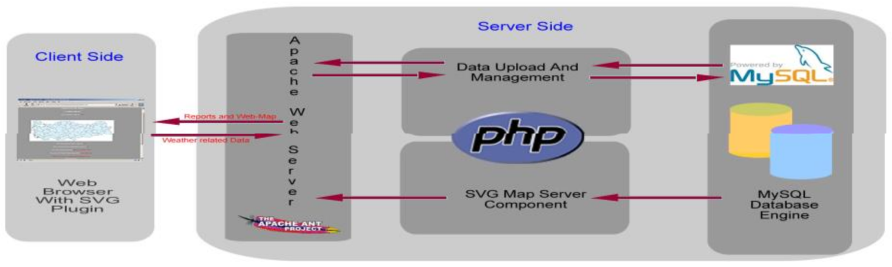


Figure 4: Architectural layout of the proposed system

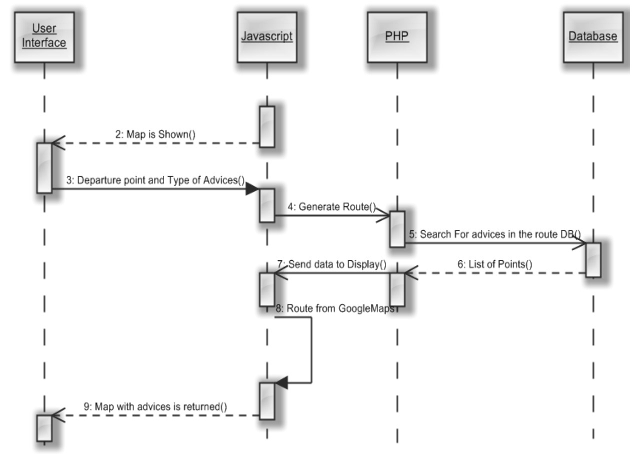

User Interface
Javascript
PHP
Database
2: Map is Shown()
3: Departure point and Type of Advices()
4: Generate Route()
5: Search For advices in the route DB()
7: Send data to Display()
6: List of Points()
8: Route from GoogleMaps
9: Map with advices is returned()

Figure 5: *Class diagram of the proposed system*

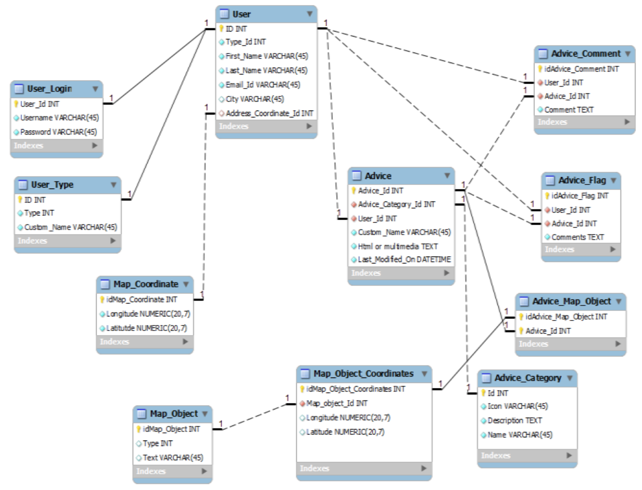


Figure 6: *Sequence diagram of the proposed system*

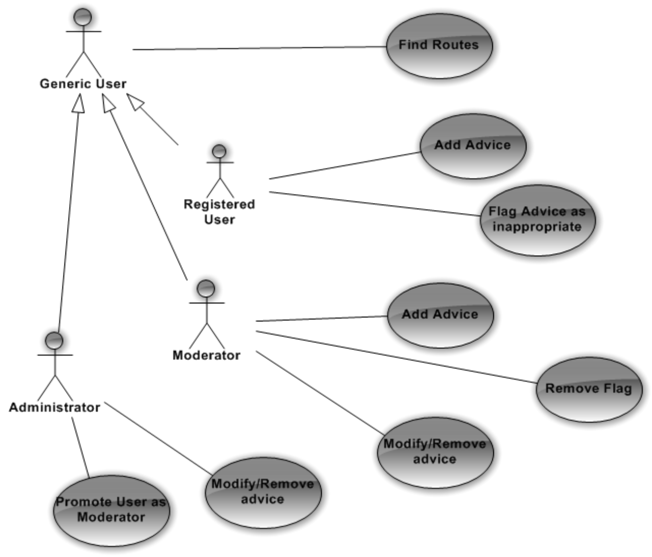


Figure 7: *Use-Case diagram of the proposed system*

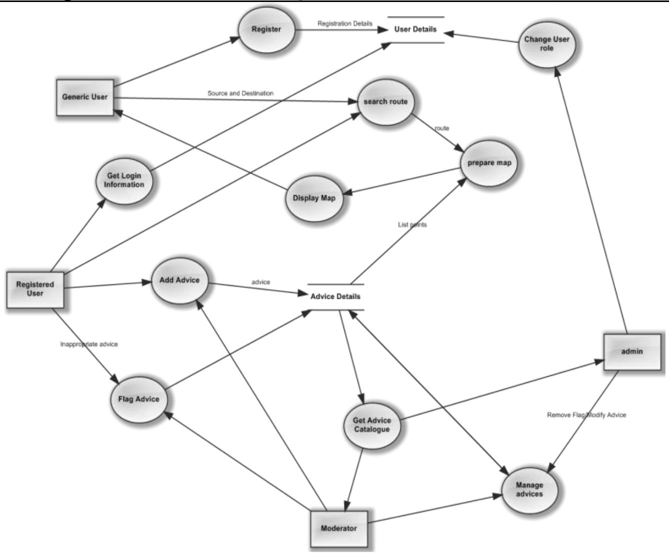


Figure 8: *Data flow diagram of the proposed system*

4. RESULTS AND SOFTWARE IMPLEMENTATION DETAILS

Computer simulation run of the three selected algorithms were performed on Toshiba PC with Intel Core i5 86x 64-bit processor architecture, CPU running at 2.50 GHZ, and 8 GB RAM memory on Windows 8.1 platform. MATLAB software version 7.11.0 R2010b was used in performing the algorithm simulation so as to test the optimality of the three selected algorithms. The 2-D plot obtained is shown in figure 10. The various algorithms evaluated, implementation data structures and their run time complexities are provided in tables 1 and 2. Algorithms 1, 2, and 3 provides details of Dijkstra, A*-Search and Restricted Search algorithms analyzed respectively.

The SPA was implemented using PHP 2.1, Laravel mapping application programming interface (API) framework version 5.0, Google map 1.0.7 and HTML 5.0 respectively. The choice is justified by the robust and scalable application development environment they provide.

Algorithms of the three fastest SPAs are shown below. PHP code implementation of Dijkstra's SPA (the algorithm of choice) is also shown below with its different modules.

Table 1: *Evaluated Shortest Path Algorithms (SPAs)*

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

Source [43]

Table 2: *Complexity Analysis of different shortest paths algorithms [10]*

Algorithm	Negative Edge	Single Source	All Sources	Time Complexity	Space Complexity
Dijkstra		√		$O(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$	$O(V^2)$
Bellman-Ford	√	√		$O(\lvert V\rvert . \lvert E\rvert)$	$O(V^2)$
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|)	$O(\lvert E\rvert + \lvert V\rvert^2)$
Kruskal's		√	√	$O(\lvert E\rvert \log \lvert V\rvert)$	$O(\lvert E\rvert \log \lvert E\rvert)$

Algorithm 1: 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] ← true
                 recent ← 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 ← dist[recent] + $w_{recent,v}$
            if newlabel < dist[v] then
                dist[v] ← newlabel
                pred[v] ← 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] ← true
         recent ← y
        {// y, the next closest vertex to s gets permanently labeled //}
     end while
```

Algorithm 2: A*-Search Algorithm

```
                              f(V) = distance from S to V + estimate of the distance to D.
                                  = d(V) + h(V,D)
                                  = d(V) + sqrt( (x(V) – x(D))2 + (y(V) – y(D))2)
                               where x(V), y(D) and x(V), y(D) are the coordinates for node V and the destination
                               node D.
                               The A* Search algorithm:
                               for each u G:
                             d[u] = infinity;
                         parent[u] = NIL;
                         End for
                        d[s] = 0;
                        f(V) = 0;
                        H = {s};
                     while NotEmpty(H) and targetNotFound:
                  u = Extract_Min(H);
              label u as examined;
            for each v adjacent to u:
          if d[v] > d[u] + w[u, v] , then
         d[v] = d[u] + w[u, v];
       p[v] = u;
      f(v) = d[v] + h(v, D);
     DecreaseKey[v, H];
```

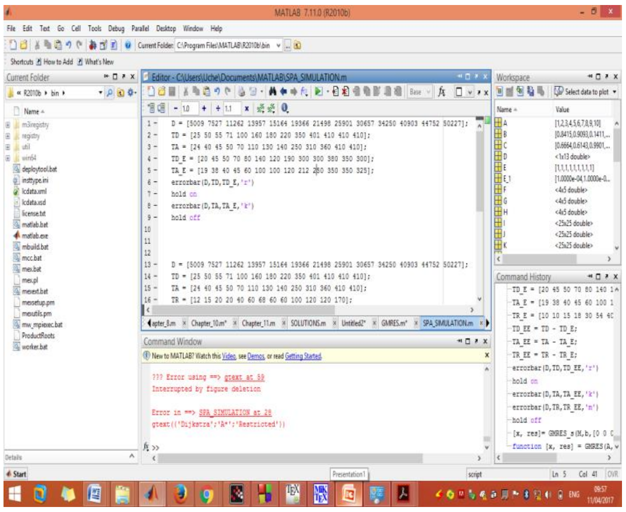
Algorithm 3: Restricted Search Algorithm

```
For each u G:
                              d[u] = infinity;
                                parent[u] = NIL;
                               End for
                              d[s] = 0;
                             H = {s};
                         while NotEmpty(H) and targetNotFound:
                         u = Extract_Nin(H);
                       label u as examined;
                         for each v adjacent to u:
                         if outOfRange(v), then continue;
```

```
                                    if d[v] > d[u] + w[u, v], then
                                    d[v] = d[u] + w[u, v];
                                    parent[v] = u;
                                DecreaseKey[v, H];
                            Procedure outOfRange(Constraint Area A, Vertex v):
          //A is a polygon given;
          //v is a Vertex being checked;
          Make a straight-line L from v to the right of v;
          Counter = 0;
          For each edge e of A
          if L intersects with e
          increase Counter by one;
          if Counter is even
     return true;
   else
return false;
```

5. DISCUSSION OF RESULTS

The results of the algorithm analysis and implemented application were verified and validated using different approaches. First, the algorithms were tested using real data. The data were transformed into MATLAB codes and executed on MATLAB 2010 code environment as shown in figure 9. The 2-D plot shown in figure 10 was obtained. From the plot Dijkstra and A*-search algorithms are observed to converge, while restricted search algorithm diverges. Since Dijkstra's algorithm offered better performance as indicated the plot it was selected as the algorithm of choice for implementing our application. The realized application user interface home screen and simulated runs for three selected Nigerian cities of Abuja, Anambra and Enugu are shown in figure 11, 12, 13 and 14 respectively.



```
D = [5009 7527 11262 13957 15164 19366 21498 25901 30657 34250 40903 44752 50227];
TD = [25 50 55 71 100 160 180 220 350 401 410 410 410];
TA = [24 40 45 50 70 110 130 140 250 310 360 410 410];
TD_E = [20 45 50 70 80 140 120 190 300 300 380 350 300];
TA_E = [19 38 40 45 60 100 100 120 212 250 350 350 325];
errorbar(D,TD,TD_E,'r')
hold on
errorbar(D,TA,TA_E,'k')
hold off



D = [5009 7527 11262 13957 15164 19366 21498 25901 30657 34250 40903 44752 50227];
TD = [25 50 55 71 100 160 180 220 350 401 410 410 410];
TA = [24 40 45 50 70 110 130 140 250 310 360 410 410];
TR = [12 15 20 20 40 60 68 60 60 100 120 120 170];
```

```
??? Error using ==> gtext at 59
Interrupted by figure deletion

Error in ==> SPA_SIMULATION at 28
gtext({'Dijkstra';'A*';'Restricted'})
```

```
TD_E = [20 45 50 70 80 140 1
TA_E = [19 38 40 45 60 100 1
TR_E = [10 10 15 18 30 54 40
TD_EE = TD - TD_E;
TA_EE = TA - TA_E;
TR_EE = TR - TR_E;
errorbar(D,TD,TD_EE,'r')
hold on
errorbar(D,TA,TA_EE,'k')
errorbar(D,TR,TR_EE,'m')
hold off
[x, res]= GMRES_s(M,b,[0 0 C
function [x, res] = GMRES(A,
```

MATLAB 7.11.0 (R2010b)
File Edit Text Go Cell Tools Debug Parallel Desktop Window Help
Current Folder: C:\Program Files\MATLAB\R2010b\bin
Editor - C:\Users\Uche\Documents\MATLAB\SPA_SIMULATION.m
Workspace
Command Window
Command History
Start

Figure 9: *MATLAB code simulation*

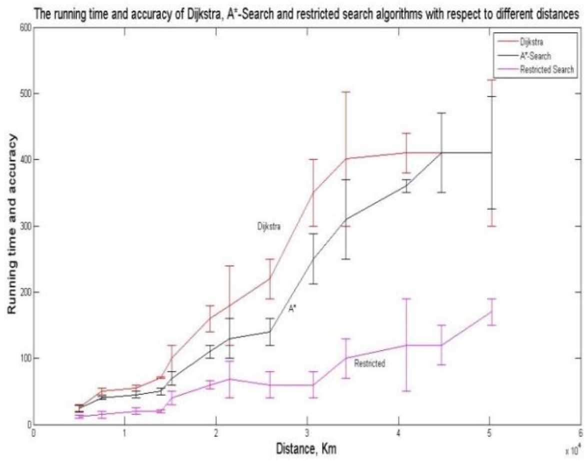


Figure 10: *Run time/accuracy vs. distance plot*

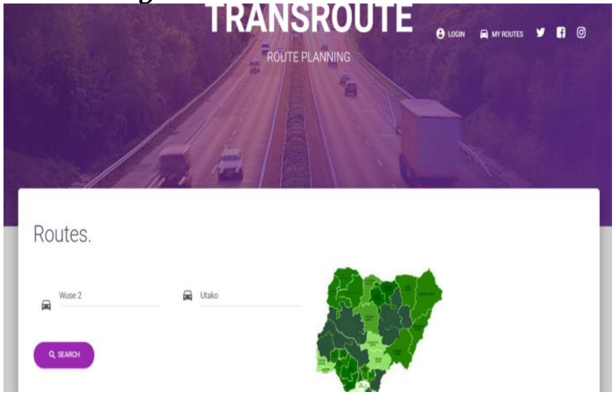
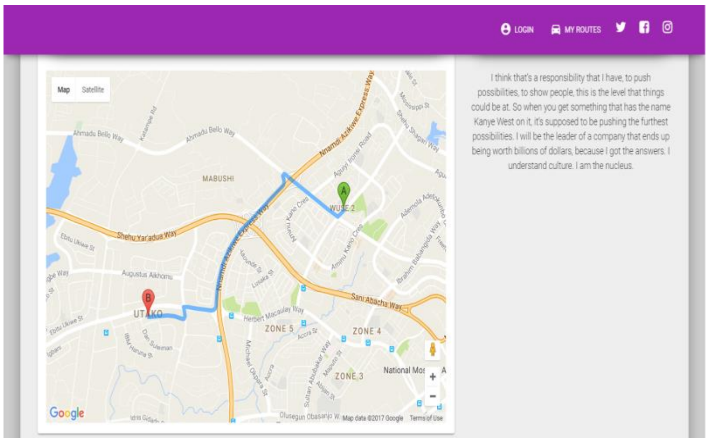


Figure 11: *Realized user interface of TransRoute application*



I think that's a responsibility that I have, to push possibilities, to show people, this is the level that things could be at. So when you get something that has the name Kanye West on it, it's supposed to be pushing the furthest possibilities. I will be the leader of a company that ends up being worth billions of dollars, because I got the answers. I understand culture. I am the nucleus.

Figure 12: *Route Visualization within F.C.T*

Third-Party Package Defnition

```
{
  "name": "laravel/laravel",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "type": "project",
  "require": {
    "laravel/framework": "5.0.*",
    "alexpechkarev/google-maps": "1.0.7",
    "cornford/googlmapper": "2.*",
    "guzzlehttp/guzzle": "~4.0",
    "laravelcollective/html": "5.0"
  },
  "require-dev": {
    "phpunit/phpunit": "~4.0",
    "phpspec/phpspec": "~2.1"
  },
  "autoload": {
    "classmap": [
      "database"
    ],
    "psr-4": {
      "App\\": "app/"
    }
  },
  "autoload-dev": {
    "classmap": [
      "tests/TestCase.php"
    ]
  },
  "scripts": {
    "post-install-cmd": [
      "php artisan clear-compiled",
      "php artisan optimize"
    ],
    "post-update-cmd": [
      "php artisan clear-compiled",
      "php artisan optimize"
    ],
    "post-create-project-cmd": [
      "php -r \"copy('.env.example', '.env');\"",
      "php artisan key:generate"
    ]
  },
  "config": {
    "preferred-install": "dist"
  }
}
```

Site Controller

```
<?php namespace App\Http\Controllers;
use Cornford\Googlmapper\Facades\MapperFacade as Mapper;
use GuzzleHttp\Client;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\URL;
class WelcomeController extends Controller {
  /*
  |--------------------------------------------------------------------------
  | Welcome Controller
  |--------------------------------------------------------------------------
  |
  | This controller renders the "marketing page" for the application and
  | is configured to only allow guests. Like most of the other sample
  | controllers, you are free to modify or remove it as you desire.
  |
  */
  /**
   * Create a new controller instance.
   *
   * @return void
   */
  public function __construct()
  {
    $this->middleware('guest');
  }
  /**
   * Show the application welcome screen to the user.
   *
   * @return Response
   */
  public function index()
  {
//    $client = new Client();  $response = $client-
>get('https://maps.googleapis.com/maps/api/directions/json?origin=Enugu&desti
nation=Abuja&key=AIzaSyD7aFBwZSIrS5EvwS7d71fkZYzpV4eZbts')->getBody();
//
//    $obj = \GuzzleHttp\json_decode($response,true);
//
//    $startLat = $obj['routes'][0]['bounds']['northeast']['lat'];
//    $startLng =$obj['routes'][0]['bounds']['northeast']['lng'];
//    $endLat = $obj['routes'][0]['bounds']['southwest']['lat'];
//    $endLng =$obj['routes'][0]['bounds']['southwest']['lng'];
//    Mapper::map($startLat,$startLng)->polyline(
//               [
//                 ['latitude' => $startLat, 'longitude' => $startLng],
//                 ['latitude' => $endLat, 'longitude' => $endLng]
//               ]);
//dd(url());
    return view('welcome');
  }
  public function test()
  {
    $client = new Client();
    $response = $client-
>get('https://maps.googleapis.com/maps/api/directions/json?origin=Enugu&destina
tion=Abuja&key=AIzaSyD7aFBwZSIrS5EvwS7d71fkZYzpV4eZbts')->getBody();
    $obj = \GuzzleHttp\json_decode($response,true);
    return view('test',$obj);
  }
  public function search(Request $request){
    if ($request->has('end_location') && $request->has('start_location'))
    {
      $endpoints = [
              'search_criteria'=>
                [
                  'start'=> $request['start_location'], 'end'=>$request['end_location']
                ],
              ];
      return view('route',$endpoints);
    }else {
    //get data and map routes
     //dd(base_path());
      return view('route');
    }
  }
```

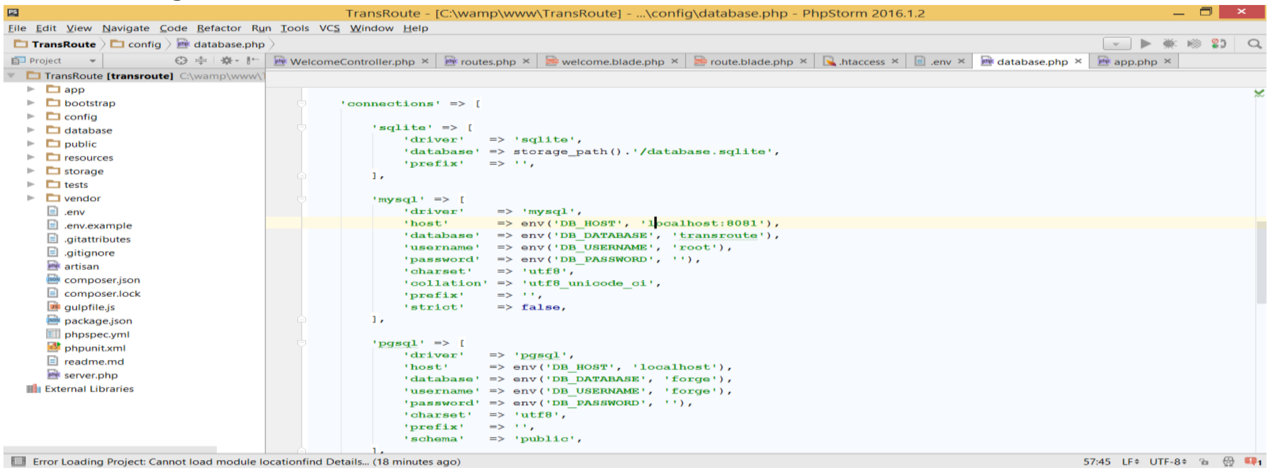
Home Page – Search controls

```
@extends('site')
@section('content')
  <div class="section section-basic">
    <div class="container">
      <div class="title">
        <h2>Routes.</h2>
      </div>
      {!! Form::open(array('url' => url().'/index.php/search', 'method' =>
'POST')) !!}
      <div id="inputs">
        <div class="row">
          <div class="col-sm-3">
            <div class="input-group">
            <span class="input-group-addon">
              <i class="material-icons">directions_car</i>
            </span>
              {!! Form::text('start_location','',array('class' => 'form-
control','placeholder'=>'Leaving')) !!}
              <label class="control-label" id="starterror" style="visibility:
hidden">Empty input</label>
            </div>
            <div class="row">
              <div class="col-sm-1">
                <button id="submit" class="btn btn-primary btn-round"
type="submit" onclick="validate(event);">
                  <i class="material-icons">search</i> Search
                </button>
              </div>
            </div>
          </div>
          <div class="col-sm-3">
            <div class="input-group">
            <span class="input-group-addon">
              <i class="material-icons">directions_car</i>
            </span>
              {!! Form::text('end_location',old('end_location'),array('class'
=> 'form-control','placeholder'=>'Arriving')) !!}
            </div>
          </div>
          <div class="col-sm-3">
            <img src="{{asset('images/nigeria_map.jpg')}}" alt="Rounded
Image"
              class="img-container img-responsive">
            <div>Google Maps and Nigeria Logo</div>

          </div>
        </div>

        <div class="row">
          <div class="col-sm-3">

          </div>
        </div>
      </div>

      {!! Form::close() !!}
    </div>
  </div>

<script>
  function validate(e) {
    if(!$('#start_location').val())
   {
     console.log($('#start_location'))
     $('#starterror').visible();
     e.preventDefault();
   }
   }
</script>
@endsection
```

Search Result – Map View

```
@extends('site')
@section('content')
  <div class="section section-tabs">
    <div class="container">
      <div class="row">
        <div class="col-md-8">
          <div class="title">
            <h3> Route : {!! $search_criteria['start'] !!} To {!!
$search_criteria['end'] !!}</h3>
          </div>
          <!-- Tabs with icons on Card -->
          <div class="card card-nav-tabs">
            <div class="header header-success">
              <!-- colors: "header-primary", "header-info", "header-
success", "header-warning", "header-danger" -->
              <div class="nav-tabs-navigation">
                <div class="nav-tabs-wrapper">
                  <ul class="nav nav-tabs" data-tabs="tabs">
                    <li class="active"><a href="#" data-
toggle="tab">Save Route</a></li>
                    <li><a href="#" data-
toggle="tab">History</a></li>
                  </ul>
                </div>
              </div>
            </div>
            <div class="content">
              <div id="map" style="height: 500px"></div>
            </div>
          </div>
          <!-- End Tabs with icons on Card -->
        </div>
        <div class="col-md-4">
          <div class="title">
            <h3>Advice</h3>
          </div>
          <!-- Tabs on Plain Card -->
          <div class="card card-nav-tabs card-plain">
            <div class="header header-danger">
              <!-- colors: "header-primary", "header-info", "header-
success", "header-warning", "header-danger" -->
              <div class="nav-tabs-navigation">
                <div class="nav-tabs-wrapper">
                  <ul class="nav nav-tabs" data-tabs="tabs">
                    <li class="active"><a href="#home" data-
toggle="tab">Recent</a></li>
                    <li><a href="#history" data-
toggle="tab">New</a></li>
                  </ul>
                </div>
              </div>
            </div>
            <div class="content">
              <div class="tab-content text-center">
                <div class="tab-pane active" id="home">
```

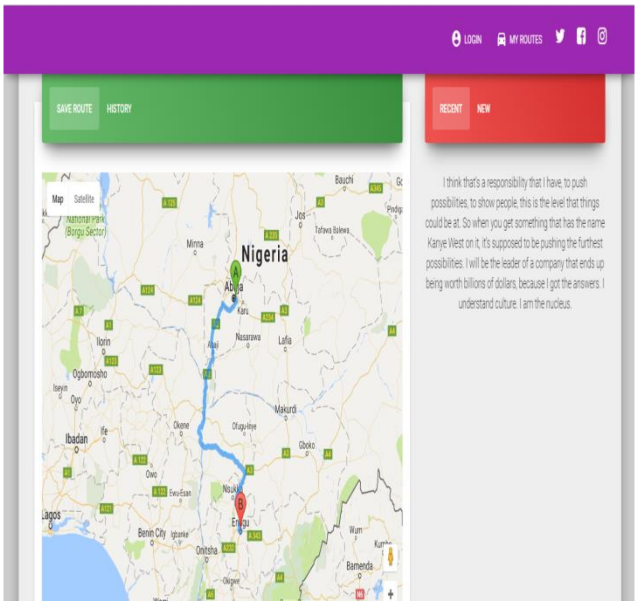
Database Configuration files and codes for TransRoute


TransRoute - [C:\wamp\www\TransRoute] - ...\config\database.php - PhpStorm 2016.1.2
File Edit View Navigate Code Refactor Run Tools VCS Window Help
TransRoute config database.php
Project
WelcomeController.php routes.php welcome.blade.php route.blade.php .htaccess .env database.php app.php
TransRoute [transroute] C:\wamp\www\
app
bootstrap
config
database
public
resources
storage
tests
vendor
.env
.env.example
.gitattributes
.gitignore
artisan
composer.json
composer.lock
gulpfile.js
package.json
phpspec.yml
phpunit.xml
readme.md
server.php
External Libraries
57:45 LF UTF-8

```
    'connections' => [

        'sqlite' => [
            'driver'   => 'sqlite',
            'database' => storage_path().'/database.sqlite',
            'prefix'   => '',
        ],

        'mysql' => [
            'driver'    => 'mysql',
            'host'      => env('DB_HOST', 'localhost:8081'),
            'database'  => env('DB_DATABASE', 'transroute'),
            'username'  => env('DB_USERNAME', 'root'),
            'password'  => env('DB_PASSWORD', ''),
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
            'strict'    => false,
        ],

        'pgsql' => [
            'driver'   => 'pgsql',
            'host'     => env('DB_HOST', 'localhost'),
            'database' => env('DB_DATABASE', 'forge'),
            'username' => env('DB_USERNAME', 'forge'),
            'password' => env('DB_PASSWORD', ''),
            'charset'  => 'utf8',
            'prefix'   => '',
            'schema'   => 'public',
        ],
```

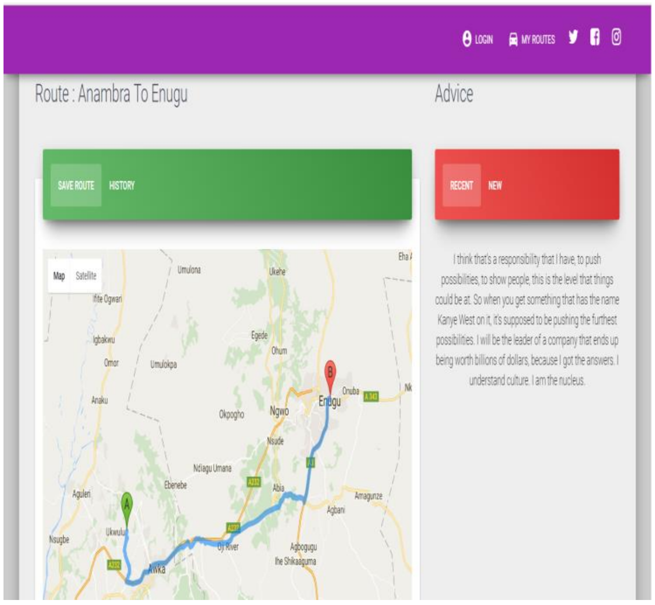
Error Loading Project: Cannot load module locationfind Details... (18 minutes ago)

Database Configuration files and codes for TransRoute



I think that's a responsibility that I have, to push possibilities, to show people, this is the level that things could be at. So when you get something that has the name Kanye West on it, it's supposed to be pushing the furthest possibilities. I will be the leader of a company that ends up being worth billions of dollars, because I got the answers. I understand culture. I am the nucleus.

Figure 13: Route Visualization for F.C.T to Enugu



I think that's a responsibility that I have, to push possibilities, to show people, this is the level that things could be at. So when you get something that has the name Kanye West on it, it's supposed to be pushing the furthest possibilities. I will be the leader of a company that ends up being worth billions of dollars, because I got the answers. I understand culture. I am the nucleus.

Figure 14: Route Visualization for Anambra to Enugu

6. CONCLUSION

Shortest path problems are one of the basic problems within Computer Science and specifically Operations Research and Programming sub-fields respectively. In this paper, we evaluated selected shortest path algorithms such as Dijkstra; A*-search; restricted search, Bellman-Ford; Floyd-Warshall; Gallo Pallottino graph growth algorithm, *et cetera* and eventually selected Dijkstra's SPA implemented with double bucket data structure due to its fast and robust performance attributes with linear run time complexity to implement our route guidance application for optimal result. While the A*-search algorithm is the most popular heuristic algorithms, its computational efficiency in real transportation networks is bounded by a factor of 2 or 50% saving in computational time as compared to an ordinary label search algorithm. This application is intended to be deployed in emergency response and logistic planning.

7. REFERENCES

[1] Ahuja, R.K., and Orlin, J.B. "Graph and Network Optimization", *Journal of Optimization and Operations Research,* UNESCO-EOLSS, Vol. II, 1993.

[2] Lumsdaine, A., Gregor, D., Hendrickson, B., and Berry, J.W. "Challenges in Parallel Graph Processing". *Parallel Processing Letters* Volume 17, No. 5, p.20, 2007.

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

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

[5] George, B. "Königsberg Bridge Problem" First Draft, UM ID# 2582042, CSCi 8701, G04

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

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

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

[9] Chartrand, G. *"The Königsberg Bridge Problem: An Introduction to Eulerian Graphs"* Introductory Graph Theory*.* New York: Dover, 1985.

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

[11] Schaffer, C.A. "A Practical Introduction to Data Structures and Algorithm Analysis", Prentice Hall, pp. 401-404, 1993.

[12] Venugopal, D. "Application of Graph Theory in Computer Science and Engineering, International *Journal of Science, Technology & Management,* Volume No. 04, Special Issue No.1, pp.1192-1198, 2015.

[13] Dantzig, G.B. "Discrete-Variable Extremum Problems"*. Operations Research,* Volume 5, pp. 266–277, 1957.

[14] Dijkstra, E.W. "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik* 1. pp. 269-271, 1959.

[15] Dantzig, G.B. "On the Shortest Route through a Road Network". *Operational Research Quarterly,* 6, pp. 187–190, 1960.

[16] Whiting, P.D., Hillier, J.A. "A Method for finding the Shortest Route through a Road Network". *Operational Research Quarterly;* Vol 11, pp. 37–40, 1960.

[17] Ford, L.R. "*Network Flow Theory*". Technical Report P-923, Santa Monica, CA: The Rand Corporation, 1956.

[18] Gallo, G., and Pallottino, S. "Shortest Path Methods: A Unifying Approach". *Mathematical Programming Study,* Vol. 26, pp. 38–64, 1986.

[19] Moore, E.F. "The Shortest Path through a Maze". In Proceedings of International Symposium on Switching Theory, 1957, Part II. Cambridge, Massachusetts, Harvard Univ. Press. pp. 285–292. MR 0114710.

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

[21] Moore, E.F. "The Shortest Path through a Maze*". In Proceedings of International Symposium on Switching Theory,* Part II, Cambridge, Massachusetts, Harvard Univ Press, pp. 285-292, MR 0114710,1957.

[22] Gallo, G., and Pallottino, S. "Shortest Paths Algorithms". *Annals of Operations Research,* Vol.13, pp. 3-79, 1988.

[23] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Clifford, C. "Introduction to Algorithms", 2nd ed., MIT Press and McGraw-Hill, 2001.

[24] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. "*Network Flows: Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993

[25] Zhan, F. B., and Noon, C. E. "Shortest Path Algorithms: An Evaluation Using Real Road Networks", *Transportation Science*, 32.1, 65-73, 1998.

[26] Zhan, F. B. "Three Fastest Shortest Path Algorithms On Real Road Networks", *Journal of Geographic Information and Decision Analysis*, Vol.1, No.1, pp.*70*-82, 1997.

[27] Gutenschwager, K., Volker, S., Radtke, A., Ulm, H., and Zeller, G. "The Shortest Path: Comparison of Different Approaches and Implementations for the Automatic Routing of Vehicles", *Proceedings of the 2012 Winter Simulation Conference,* IEEE, Munich, Germany. pp. 3312-3323, 2012.

[28] Schaffer, C.A. "*A Practical Introduction to Data Structures and Algorithm Analysis"*, Prentice Hall, pp. 401-404, 1998.

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

[30] S.G. Shrinivas *et al.* "Applications of Graph Theory in Computer Science: An Overview", International Journal of Engineering Science and Technology, Vol. 2, No.9, pp. 4610-4621, 2010.

[31] Narasingh D. "*Graph Theory with Applications To Engineering And Computer Science,* Prentice Hall of India, 1990.

[32] Venugopal, D. "Application of Graph Theory in Computer Science and Engineering", *International Journal of Science, Technology and Management*, Vol. No. 04, Special Issue No. 01, pp.1192-1198, 2015.

[33] Tosuni, B. "Some Interesting Topics Of Graph Theory In Modern Computer Science, *European Journal of Mathematics, Technology and Computer Science,* ISSN 1946-4690, 2005.

[34] Orlin, J.B; Madduri, K; Subramani, K and Williamson, M. "A Faster Algorithm For The Single Source Shortest Path Problem With Few Distinct Positive Lengths". *Journal of Discrete Algorithms* - Vol. 8, pp.189-198, 2010.

[35] Huijuan, W. *et al.* "Application of Dijkstra Algorithm In Robot Path-Planning, Second International Conference on Mechanic Automation and Control Engineering (MACE), pp. 1067-1069, 2011.

[36] Liu, X-Y and Yan-Li, C. "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, 2010.

[37] Abujassar, S.R., and Ghanbari, M. "Efficient Algorithms To Enhance Recovery Schema In Link State Protocols", *International Journal of Ubicomp* (IJU), Vol. 2, No. 3, 2011.

[38] Coltun, R., Ferguson, D., Moy, J.A., and Lindem. "OSPF for IPv6". The Internet Society, OSPFv3, 2008. Available at: https://en.wikipedia.org/wiki/Open_Shortest_Path_First [Accessed: Wednesday, June 1, 2016 5:12:03].

[39] Katz, D. "OSPF vs IS-IS". North American Network Operators Group NANOG 19. Albuquerque, New Mexico, U.S.A, 2000.

[40] Geisberger, R; and Shieferdecker, D. "Advanced Route Planning in Transportation Networks", In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'13), Universitat des Landes Baden Württemberg, Germany, 2013.*

[41] Erickson, J. "Algorithms, Lecture 21: Shortest Paths" [Fa'14], p.4, 2014. Available at: http://www.cs.uiuc.edu/~jeffe/teaching/algorithms [Accessed on: June 16, 2016].

[42] Sedgewick, R., and Wayne, K. "Algorithms". Pearson Education, 4th edition, 2011.

[43] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. "*Data Structures and Algorithms"*, Addison-Wesley, Reading, MA, U.S.A., 1987.

[44] 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.

[45] Fredman, M.L., Tarjan, R.E. Fibonacci Heaps and their Uses In Improved Network Optimization Algorithms, *Journal of the Association for Computing Machinery* (ACM), Vol. 34, No. 3, pp. 596–615, 1987.

[46] Van Emde Boas, P., Kaas, R., and Zijlstra, E. "Design and Implementation of an Efficient Priority Queue". *Math. Syst. Theory* 10, pp. 99-27, 1977.

[47] Hart, P. E.; Nilsson, N. J., Raphael, B. A. "Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* SSC4 4 (2), pp. 100–107, 1968.

[48] Cherkassky, B. V., Goldberg, A. V., and Radzik, T. "Shortest Paths Algorithms: Theory and Experimental Evaluation". *Mathematical Programming: Series A and B*, Vol.73, pp.129–174, 1993.

[49] Doran, J. "An Approach to Automatic Problem-Solving". Machine Intelligence, Vol.1, pp. 105-127, 1967.

[50] Klein, D., and Manning, C.D. "A* parsing: Fast Exact Viterbi Parse Selection*. Proc. NAACL-HLT,* 2003.

[51] Johnson, D.B. "Efficient Algorithms For Shortest Paths In Sparse Networks, *Journal of the Association of Computing Machinery* (ACM), Vol. 24, pp. 1-13, 1977.

[52] Black, P.E. "Johnson's Algorithm, Dictionary of Algorithms and Data Structures", National Institute of Standards and Technology, U.S.A., 2004.

[53] Fu, L., Sun, D. and Rilett, L.R. "Heuristic Shortest Path Algorithms For Transportation Applications: State Of The Art". *Computers and Operations Research*, Vol. 33, pp. 3324–3343, 2006.