International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 2 Issue: 3 435 – 440

Railway Route Optimization System Using Dijkstra Method

Pramod Pandey,
Student, Information Science and Engineering Department, DSCE
Bangalore, India
pramodpandeyy@gmail.com

Sunanda Dixit
Assistant Professor, Information Science and Engineering Department, DSCE
Bangalore, India
sunanda.bms@gmail.com

***Abstract*—** Nowadays, the development of 'smart cities' with a high level of quality of life is becoming a prior challenge to be addressed. In this framework, promoting the model shift towards more reliable, greener and in general more sustainable transportation modes, namely towards a 'smart mobility' could significantly contribute to achieve this goal. The aim of this paper is to provide users with more regular and reliable rail system by optimizing the route among stations.

Keywords-*component; Cost Matrix; Dijkstra algorithm; shortest route; vertex*

*****

I. INTRODUCTION

Nowadays, the development of 'smart cities' with a high level of quality of life is becoming a prior challenge to be addressed. In this framework, promoting the model shift towards more reliable, greener and in general more sustainable transportation modes, namely towards a 'smart mobility' could significantly contribute to achieve this goal[1]. The aim of this paper is to provide users with more regular and reliable rail system by optimizing the route among stations. Here, we make use of Dijkstra's Algorithm to optimize the routes among station and present the user with the shortest route. If a user has to travel from one station to another, the model will give the shortest route between those stations along with the train information between those stations.

II. PROBLEM

Railway system poses a multitude of interesting optimization problem. Today, when you search for a train that runs from a specific station of yours choice to another station, either you get a list of trains between those stations or you will see a line mentioning that there is no direct train. The problem is to provide the users with the information of all those trains, in case of no direct trains, along with the stations that goes from source to non-final station and then from non- final to destination station.

III. LITERATURE SURVEY

There is an enormous wealth of publication on the related vehicle routing problem. The vehicle routing problem itself has been studied in many variants, the closest in spirit to our problem being the distance constrained vehicle routing problem, which has received comparatively little attention. A[5] survey paper by Assad [1980] gives the early applications of simulation and operations research techniques developed to solve the train scheduling problem. In the last two decades, few researchers have shown interest in solving the train scheduling problem. Haghani [1989] presents the formulation and solution of a combined train routing, makeup, and empty car distribution model. This formulation results in a large-scale, mixed integer programming problem with nonlinear objective functions and linear constraints. The problem is then solved using a heuristic decomposition technique. Keaton [1989, 1992] formulates the train scheduling problem as an integer programming problem and applies a Lagrangian relaxation approach along with heuristics to solve the problem. Gorman [1998] applies metaheuristics – genetic algorithms and tabu search – to develop an operating plan for a US railroad. Newman and Yano [2000, 2001] solve the train scheduling problem in specific intermodal settings. Carpara et al. propose a graph theoretic formulation for the train scheduling problem using a directed multigraph. This formulation is used to derive an integer linear programming model that is relaxed using Lagrangian relaxation technique. The relaxation is embedded within a heuristic algorithm which makes extensive use of the dual information associated with the Lagrangian multipliers.

IV. ALGORITHM

In this section, we consider the single-source shortest-paths problem: for a given vertex called the source in a weighted

connected graph, find shortest paths to all its other vertices. It is important to stress that we are not interested here in a single shortest path that starts at the source and visits all the other vertices. This would have been a much more difficult problem.

The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may, of course, have edges in common.

A variety of practical applications of the shortest-paths problem have made the problem a very popular object of study. The obvious but probably most widely used applications are transportation planning and packet routing in communication networks, including the Internet. Multitudes of less obvious applications include finding shortest paths in social networks, speech recognition, document formatting, robotics, compilers, and airline crew scheduling. In the world of entertainment, one can mention path finding in video games and finding best solutions to puzzles using their state-space graphs.

There are several well-known algorithms for finding shortest paths, including Floyd's algorithm for the more general all-pairs shortest-paths problem. Here, we consider the best-known algorithm for the single-source shortest-paths problem, called Dijkstra's algorithm. This algorithm is applicable to undirected and directed graphs with nonnegative weights only. Since in most applications this condition is satisfied, the limitation has not impaired the popularity of Dijkstra's algorithm. Dijkstra's algorithm finds the shortest paths to a graph's vertices in order of their distance from a given source. First, it finds the shortest path from the source to a vertex nearest to it, then to a second nearest, and so on. In general, before its ith iteration commences, the algorithm has already identified the shortest paths to i − 1 other vertices nearest to the source. These vertices, the source, and the edges of the shortest paths leading to them from the source form a sub tree Ti of the given graph (Figure 1).

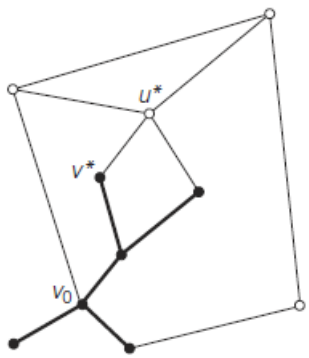


FIGURE 1. Idea of Dijkstra's algorithm.

The sub tree of the shortest paths already found is shown in bold. The next nearest to the source v0 vertex, u, is selected by comparing the lengths of the sub tree's paths increased by the distances to vertices adjacent to the sub tree's vertices.

Since all the edge weights are nonnegative, the next vertex nearest to the

source can be found among the vertices adjacent to the vertices of Ti .

The set of vertices adjacent to the vertices in Ti can be referred to as "fringe vertices"; they are the candidates from which Dijkstra's algorithm selects the next vertex nearest to the source. (Actually, all the other vertices can be treated as fringe vertices connected to tree vertices by edges of infinitely large weights.) To identify the ith nearest vertex, the algorithm computes, for every fringe vertex u, the sum of the distance to the nearest tree vertex v (given by the weight of the edge (v, u)) and the length dv of the shortest path from the source to v (previously determined by the algorithm) and then selects the vertex with the smallest such sum. The fact that it suffices to compare the lengths of such special paths is the central insight of Dijkstra's algorithm.

To facilitate the algorithm's operations, we label each vertex with two labels. The numeric label d indicates the length of the shortest path from the source to this vertex found by the algorithm so far; when a vertex is added to the tree, d indicates the length of the shortest path from the source to that vertex. The other label indicates the name of the next-to-last vertex on such a path, i.e., the parent of the vertex in the tree being constructed. (It can be left unspecified for the source s and vertices that are adjacent to none of the current tree vertices.) With such labeling, finding the next nearest vertex u* becomes a simple task of finding a fringe vertex with the smallest d value. Ties can be broken arbitrarily.

After we have identified a vertex u* to be added to the tree, we need to perform two operations:

- Move u* from the fringe to the set of tree vertices.
- For each remaining fringe vertex u that is connected to u* by an edge of weight w(u*, u) such that du*+ w(u*, u) < du, update the labels of u by u* and du* + w(u*, u), respectively.

Figure 2. Demonstrates the application of Dijkstra's algorithm to a specific graph.

Dijkstra's algorithm compares path lengths and therefore mus tad dedge weights.

Now we can give pseudo code of Dijkstra's algorithm. The set VT of vertices for which a shortest path has already been found and the priority queue Q of the fringe vertices. (Note that in the following pseudo code, VT contains a given source vertex and the fringe contains the vertices adjacent to it after iteration 0 is completed.)

ALGORITHM Dijkstra(G, s)

//Dijkstra's algorithm for single-source shortest paths

//Input:

A weighted connected graph G = 〈V , E〉 with nonnegative weights and its vertex s

//Output:

The length dv of a shortest path from s to v

//and its penultimate vertex pv for every vertex v in V

Initialize (Q)

//initialize priority queue to empty

for every vertex v in V

dv ← ∞; pv ← null

Insert(Q,v,dv)

//initialize vertex priority in the priority queue

ds ← 0; Decrease(Q, s, ds) //update priority of s with ds

VT ← ∅

for i ← 0 to |V | − 1 do

u∗ ← DeleteMin(Q)

//delete the minimum priority element

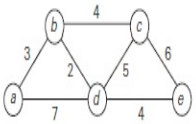
VT ← VT ∪ {u∗ }

for every vertex u in V − VT that is adjacent to u∗ do if du∗ + w(u∗ , u) < du

du ← du∗ + w(uu);

pu ← u∗ Decrease(Q, u, du)

The time efficiency of Dijkstra's algorithm depends on the data structures used for implementing the priority queue and for representing an input graph itself.



Tree vertices	Remaining vertices	Illustration
a(−, 0)	b(a, 3) c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3+4) d(b, 3+2) e(−, ∞)	
d(b, 5)	c(b, 7) e(d, 5+4)	
c(b, 7)	e(d, 9)	
e(d, 9)		

Figure 2 and 3: Dijkstra's algorithm

The shortest paths (identified by following nonnumeric labels backward from a
destination vertex in the left column to the source) and their lengths (given by
numeric labels of the tree vertices) are as follows:
from a to b : a − b of length 3
from a to d : a − b − d of length 5
from a to c : a − b − c of length 7
from a to e : a − b − d − e of length 9

Figure3. Application of Dijkstra's algorithm. The next closest vertex is shown in bold.

The time complexity is in 0 (|V |2) for graphs represented by their weight matrix and the priority queue
implemented as an unordered array. For graphs represented by their adjacency lists and the priority queue implemented as a min-heap, it is in O(|E| log |V |). A still better upper bound can be achieved for Dijkstra's algorithms if the priority queue is implemented using a sophisticated data structure called the Fibonacci heap. However, its complexity and a considerable overhead make such an improvement primarily of theoretical value.

 ISSN: 2321-8169
435 – 440

In routing problems, we search for short routes from station to station. In this model n x n cost matrix consisting of n rows and n columns is used. Every pair of (row, column) is depicted by the name of the station which shows whether a path exists between these stations or not. Distance is the corresponding value for a pair of row and column if there exists a path otherwise it is initialized to some pre- defined value. Cost matrix is shown below:

```
Dijktra::cost_matrix [14][14] =

0 ,      358 ,   1213 ,     1805 ,  99999 , 1072 ,  1960 ,  99999 , 99999 , 99999 , 2630 ,  99999
358 ,    0 ,     99999 ,    99999 , 99999 , 99999 , 1650 ,  99999 , 99999 , 99999 , 99999 , 99999
1213 ,   99999 , 0 ,        493 ,   99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999
1805 ,   99999 , 493 ,      0 ,     99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999
99999 ,  99999 , 99999 ,    99999 , 0 ,     708 ,   99999 , 99999 , 99999 , 1546 ,  99999 , 99999
1072 ,   99999 , 99999 ,    99999 , 708 ,   0 ,     99999 , 99999 , 99999 , 99999 , 99999 , 99999
1960 ,   1650 ,  99999 ,    99999 , 99999 , 99999 , 0 ,     980 ,   532 ,   99999 , 99999 , 99999
99999 ,  99999 , 99999 ,    99999 , 99999 , 99999 , 980 ,   0 ,     880 ,   99999 , 99999 , 99999
99999 ,  99999 , 99999 ,    99999 , 99999 , 99999 , 532 ,   880 ,   0 ,     529 ,   99999 , 99999
99999 ,  99999 , 99999 ,    99999 , 1546 ,  99999 , 99999 , 99999 , 529 ,   0 ,     401 ,   99999
2630 ,   99999 , 99999 ,    99999 , 99999 , 99999 , 1650 ,  99999 , 99999 , 401 ,   0 ,     1084
99999 ,  99999 , 99999 ,    99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 1084 ,  0 ,
2382 ,   99999 , 99999 ,    99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999
99999 ,  99999 , 99999 ,    99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999 , 99999
```

Figure4 : Cost matrix

All the numbers shown above Figure 4, expect 99999 represents real time distances between stations.

These numbers are stored in 2-D arrays. Here the number 0 represents that there is no direct route or train between these stations. The schematic view of basic working model is shown:

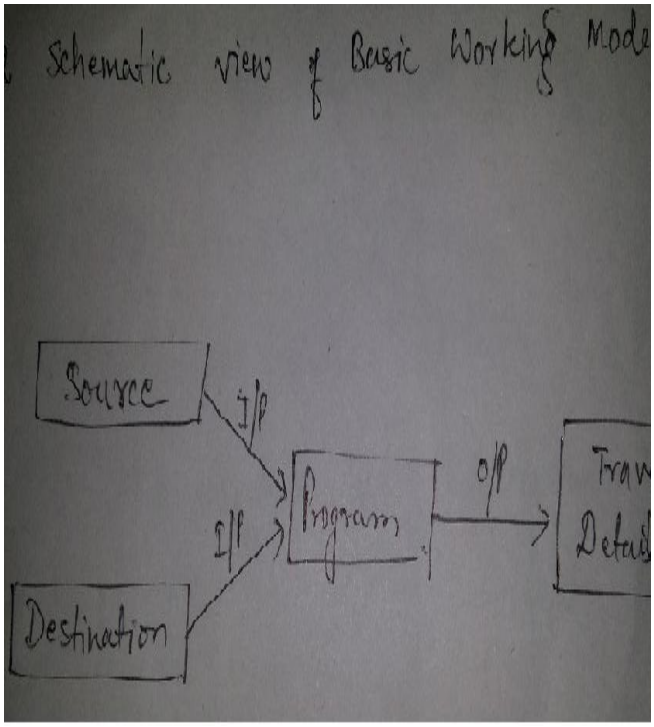
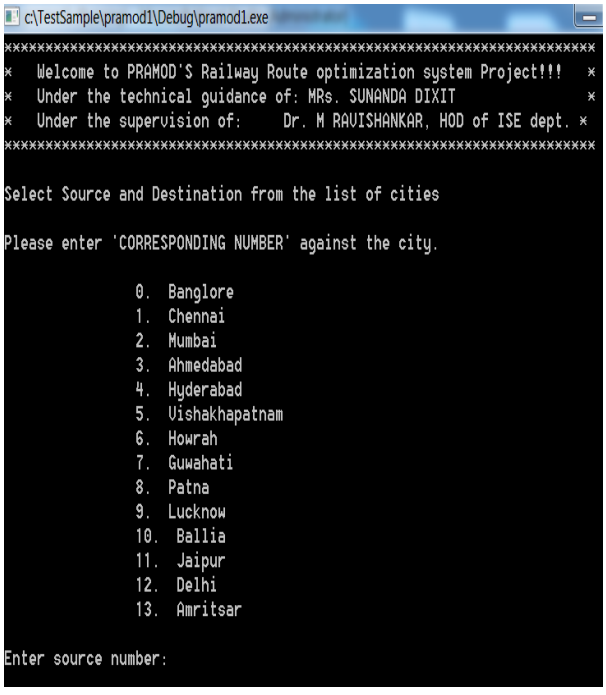


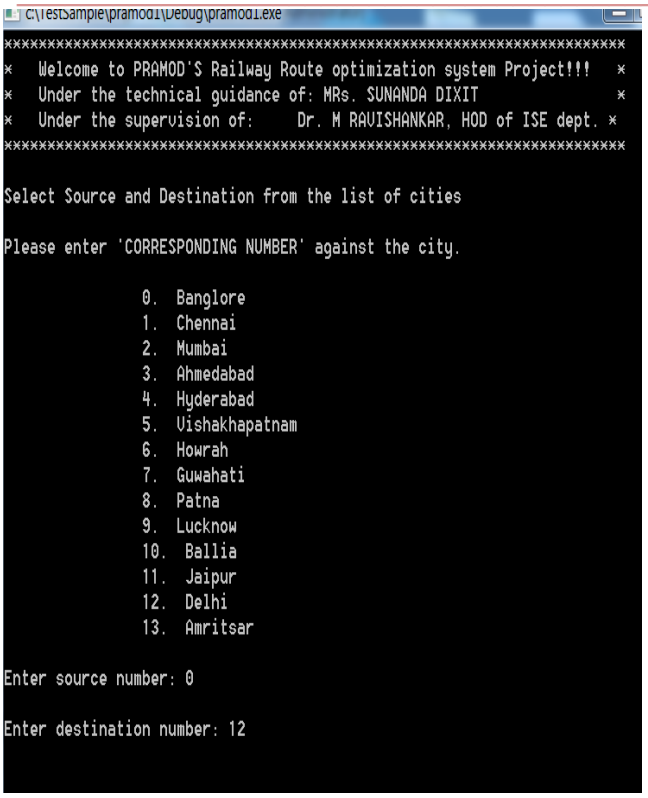
Figure5 : Working Model of proposed method

V. EXPERIMENTAL RESULTS

Here we are shown some snapshots of the basic working MODEL.

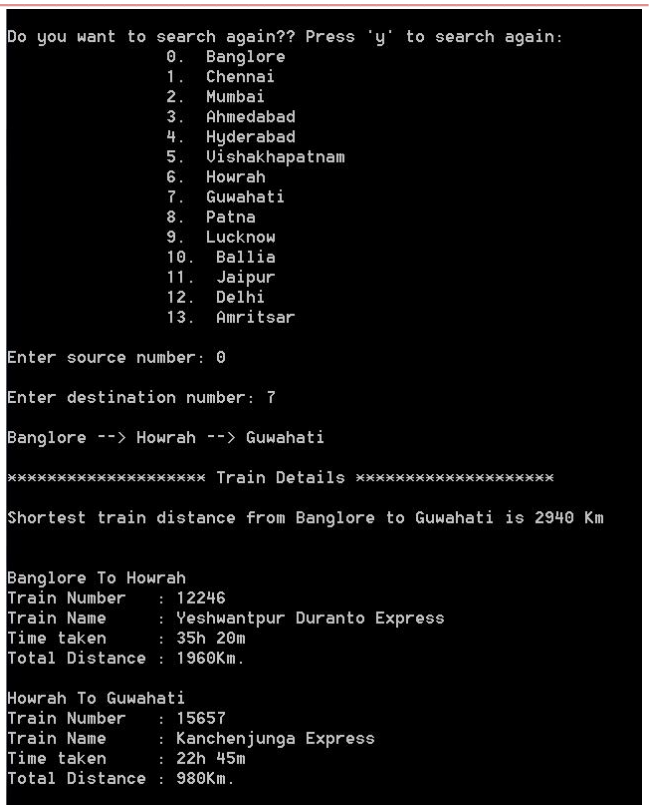


Now, we enter the source and destination number . .



The basic working model will present the user with travel details stored in it.

Another example in case of no direct train. .



Here shortest route is in terms of distance between the two stations and not the time taken to travel from one station to another. In the basic working model, We have included only few station and trains to prove my points.

VI. CONCLUSION

In this paper, I describe simple aspects of optimization. It can provide user with an additional option to find shortest route between any two stations without any undue effort. Our solution approach for routing problem may prove beneficial in near future.

Nevertheless, our current version needs to be improved in order to solve optimization problems more efficiently.

VII. REFERENCES

[1] A. Carbone, F. Papa, N. Sacco, “An optimization approach for delay recovery in urban metro transportation system”.

[2] Anany Levitin, “Introduction to the Design & analysis of Algorithms”, 2nd edition, Pearson Education, 2007.

[3] Thomas H. Cormen, Charles E.Leiserron ,Ronal L. Rivest, Clifford Stein :Introduction to Algorithm, 3rd edition, PHI, 2010

[4] R.C.T. Lee, S.S. Tseng, R.C. Chang & Y.T. Tsai, “ Introduction to the Design and analysis of Algorithm”, A strategic Approach, Tata McGraw Hill, 2005.

[5] Ravindra K. Ahuja, Mohammad Jaradat, Krishna C. Jha, Arvind Kumar And Pooja Dewan "An Optimization-Based Decision Support System for Train Scheduling"

[6] Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, "Fundamentals of Computer Algorithms", 2nd Edition, Universities Press, 2007.