Algorithm Insurance of Portal for Public Transport Schedules

Georgi Krastev [1], Tsvetozar Georgiev [1]

[1] *Department of Computing, University of Ruse, Ruse, Bulgaria*

Abstract **– An algorithm for automated generating of itinerary between two randomly chosen bus stops has been described in this article. The itinerary may be generated according to the time of departure or according to the desired time of arrival at a certain bus stop.**

Keywords **– Algorithm, Smartphone, Schedule, Public transport.**

1. Introduction

The problem of algorithm insurance especially generating itinerary between two randomly chosen bus stops is the main one in the process of developing a portal for schedules for public transport.

Dijkstra's algorithm named after its discoverer Edsger Dijkstra has been used while developing the application [2, 3]. It determines the shortest itinerary in an oriented graph from a chosen point to all the rest with non-negative weights of the ribs. The algorithm may be modified and used to find some other optimal paths [6, 7, 8]. The algorithm's modification could lead to a possibility for choosing some criteria according to which the traveling may be accomplished [1, 4, 5, 9].

The algorithm may be described as follows: let r be the chosen initial point, * (i, j) be the rib's weight from i to j or to infinity, if such one does not exist. The algorithm works by supporting the length d (i) of the found to the shortest path moment from r to every point i and gradually expands the multitude S from the points for which this length is surely optimal. Initially:

DOI: 10.18421/TEM61-13
https://dx.doi.org/10.18421/TEM61-13

Corresponding author: Georgi Krastev,
Department of Computing, University of Ruse,
Ruse, Bulgaria
Email: GKrastev@ecs.uni-ruse.bg

$$d(j) = c(r,i) \qquad (1)$$

To each iteration the algorithm chooses such a point v that is not in S and d(v) is minimal. The point v is added in S and relaxation is accomplished: for each point j that is not in S.

$$d(j) = \min(d(j), d(v) + c(v, j)) \qquad (2)$$

The algorithm ends when all points of the graph are in multitude S [4]. Transport routes schedules and their graphs can be interpreted as an oriented graph. More than one factor can be used for separate connections' weights. If the aim is to achieve the fastest movement between the initial and the final bus stop, the time necessary to arrive at a definite bus stop may be used for weight. If the aim is to achieve the smallest number of changes of transport routes, the number of changes is used for weight.

2. Search algorithm

It is necessary to create an oriented graph with its weights using the information available in data base. A great amount of data that cannot be charged in the operating memory because of the restricted size of the information terminal that is necessary to fulfill that goal. Supposing the operation is possible, the decision won't be successful because of a lack of available memory for other users that use the site at the same time. That's why the algorithm has to work in iterations to minimize the necessary data charged in the operating memory. The graph won't be available at the time when algorithm starts operating.

The information will be submitted gradually only for the needed connections and sections.

A list with the already visited bus stops has to be supported to make comparison about the speed by which people got to these bus stops and the number of changes they have done to a definite moment.

The search starts from the initial or the final bus stop depending whether the search is according to the time of departure or the time of arrival. If the search is according to the time of departure the time increases on every next stage of search; that is why it is called forward search. If the search is according to the time of arrival the approach proceeds in the

opposite direction: the start is from the last bus stop and the search to the initial bus stop is backward concerning the time; it is called backward search. An excerpt from the process of search with given time of departure is shown in the diagram in figure 1.

Stage 1 shows a transport route that reaches bus stop X1 at a definite time t1o. Then the next stage comes. Stage 2 shows repetition that is part of a cycle that reaches all bus stops. The times when transport routes arrive at a definite bus stop are defined in the first part of stage 2.

Their times of arrival have to be longer than the time of arrival of the current transport at this bus stop. A case has been shown in stage 2 where a current transport route reaches its last bus stop. In that case the search continues concerning the other transport routes. In stage 3 two of the bus stops that reaching the chosen station or if there are no more stations to be visited. The second case may appear if there is no possibility to reach the last bus stop at a given time of departure or arrival.

As a whole the algorithm works as follows: it finds transport routes and their closest passing by a current bus stop at a definite time. Each transport route passes several times by a definite bus stop. That is why it is necessary to find the closest passing to the desired bus stop at the moment for each transport route. The closest passing forward in time is looked for if it is a forward search; if it is a backward one – then it is back in time. After the passing transport routes are defined the next two bus stops towards which they direct are found. A list which is used in the next iteration is composed from the information from these bus stops.

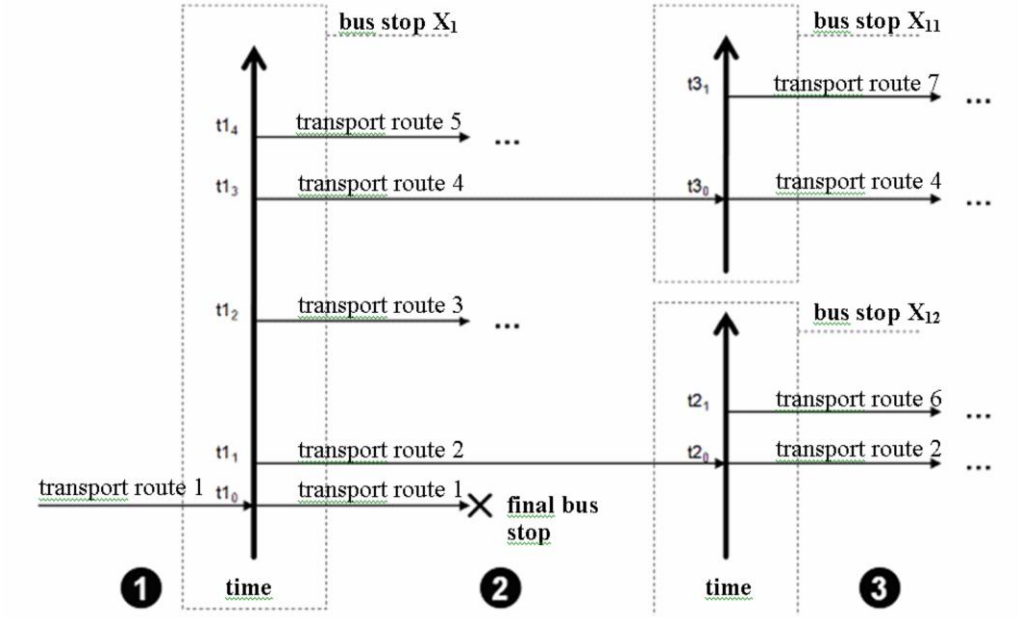

Figure 1. An excerpt from the process of forward search

have been generated in stage 2 could be seen. Every next stage generates more bus stops that have to be reached. The number of bus stops that have to be reached reduces from the list of the already visited bus stops. Thus only new bus stops are allowed to be visited. What is kept as information is not the whole graph but the information from the last iteration and a list with the fastest itineraries or itineraries with the smallest number of changes to a definite bus stop. There is an exemplary graph shown in figure 2. that is built from several successive iterations and that starts form a zero moment.

The graph has the shape of a tree. The information that the algorithm keeps is derived from the last iteration. Thus memory can be saved in comparison to charging the whole tree. The algorithm for visiting is realized recursively being accomplished while

The finding of the last bus stops has been done in two ways: the first one that is the fastest without any control of the number of changes between the separate transport routes, and the second one that is with the smallest number of changes without taking into considerations the speed during the whole passage.

The algorithm's semantic code is:

```
  function recursively_traveling
($current_bus_stops)
  {
  if ($current_bus_stops is empty) {
  exit: the end has been reached;
  }
  for each ($current_bus_stops) {
  find other transports passing by this bus
stop;
  add the_already found ones to $new_branches
```

```
  }
  for each ($new_branches) {
  find next bus stops
  if (there aren't next bus stops) {
  ignore the branch;
  }
  otherwise {
  // found last bus stop
  if (the found bus stop is the one that has
been looked for) {
  // check what type the decision is
  if (the found bus stop has the smallest
number of branches) {
  add as a decision with smallest number of
branches;
  if (there is a decision for fastest
itinerary) {
  exit: return the result from the found
routes;
                              }
  }
  otherwise {
  add as a decision for a fastest itinerary;
  if (there is a decision for a fastest
itinerary) {
  exit: return the result from the found
itineraries;
                              }
                        }
  }
  // visited bus stops check
  if (the bus stop has been visited) {
  if (the current itinerary has less changes
than the available one) {
  put the current itinerary as an available one
with smallest number of changes;
  add to $next_step the current route;
  }
  otherwise if (the current itinerary is faster
than the available one) {
  put the current one as an available itinerary
as fastest;
  add the current itinerary to $next_step;
                        }
                  }
  otherwise {
  add the current itinerary to $next_step;
  }
                  }
  }
  // recursive visiting
  recursive_visiting ($next_step)
  }
```

3. Conclusion

Algorithm insurance of a portal for a mobile information terminal for public transport has been described in this article. The Dijkstra's algorithm for defining the shortest itinerary in an oriented graph from a given point to all the rest with non-negative ribs weights has been used in the current paper. The application whose algorithm has been describes has the possibility for automated generating of itinerary between two randomly chosen bus stops. The itinerary may be generated according to departure and arrival time at a definite bus stop.

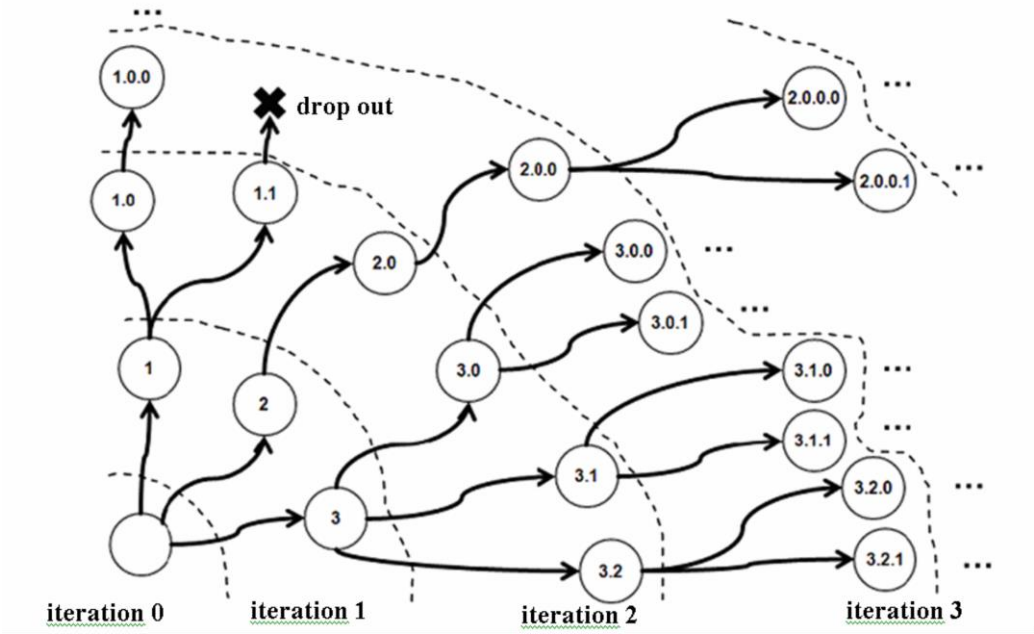

Figure 2. Iterations in the process of generating an itinerary

Acknowledgements

This research was supported by aScientific Project 16-FEEA-03 of the University of Ruse.

References

[1]. Bast, H., Funke, S., Matijevic, D., Sanders, P., & Schultes, D. (2007). *In transit to constant time shortest-path queries in road networks*. In Proceedings of the Meeting on Algorithm Engineering & Expermiments (pp. 46-59). Society for Industrial and Applied Mathematics.

[2]. Biggs, N., Lloyd, E., Wilson, R. (1986), *Graph Theory*, 1736-1936, Oxford University Press.

[3]. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001). *Section 24.3: Dijkstra's algorithm.Introduction to Algorithms*(Second ed.). MIT Press and McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7.

[4]. Fu, L., Sun, D., Rilett, L. R. (2006). *Heuristic shortest path algorithms for transportation applications: state of the art*. Computers & Operations Research, 33(11), 3324–3343.

[5]. Goldberg, A. V., Kaplan, H., & Werneck, R. F. (2006). *Reach for a: Efficient point-to-point shortest path algorithms*. In Proceedings of the Meeting on Algorithm Engineering & Expermiments (pp. 129–143). Society for Industrial and Applied Mathematics.

[6]. Russell, S., Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall. pp. 75–81. ISBN 978-0-13-604259-4.

[7]. Xu, M. H., Liu, Y. Q., Huang, Q. L., Zhang, Y. X., & Luan, G. F. (2007). *An improved Dijkstra's shortest path algorithm for sparse network*. Applied Mathematics and Computation, 185(1), 247–254.

[8]. Zhan, F. B., Noon, C. E. (1998). *Shortest Path Algorithms: An Evaluation Using Real Road Networks.*Transportation Science. 32 (1): 65–73. doi:10.1287/trsc.32.1.65.

[9]. Zhan, F. B. (1997). *Three fastest shortest path algorithms on real road networks: Data structures and procedures.* Journal of geographic information and decision analysis, 1(1), pp. 69-82.