TRANSPORT
ISSN 1648-4142 / eISSN 1648-3480
2020 Volume 35 Issue 2: 156–167
https://doi.org/10.3846/transport.2020.12477

DYNAMIC TRIP PLANNER FOR PUBLIC TRANSPORT USING GENETIC ALGORITHM

Abhishek BASU, Bharathi RAJA, Rony GRACIOUS, Lelitha VANAJAKSHI*

Dept of Civil Engineering, Indian Institute of Technology Madras, India

Received 15 December 2016; revised 3 April 2017, 12 October 2017; accepted 26 November 2017

Abstract. This paper reports the development of a public transport trip planner to help the urban traveller in planning and preparing for his commute using public transportation in the city. A Genetic Algorithm (GA) approach that handles real-time Global Positioning Systems (GPS) data from buses of the Metropolitan Transport Corporation (MTC) in Chennai City (India) has been used to develop the planner. The GA has been shown to provide good solutions within the problem's computation time constraints. The developed trip planner has been implemented for static network data first and subsequently extended to use real-time data. The "walk mode" and Chennai Mass Rapid Transit System (MRTS) have also been included in the geospatial database to extend the route-planner's capabilities. The algorithm has subsequently been segmented to speed up the prediction process. In addition, a temporal cache has also been introduced during implementation, to handle multiple queries generated simultaneously. The results showed that there is promise for scalability and citywide implementation for the proposed real-time route-planner. The uncertainty and poor service quality perceived with public transport bus services in India could potentially be mitigated by further developments in the route-planner introduced in this paper.

Keywords: dynamic trip planner, genetic algorithm, global positioning system, public transportation, route-planner, static network, real-time data.

Notations

CSA – connection scan algorithm;
ETA – expected time of arrival;
GA – genetic algorithm;
GPS – global positioning systems;
LB – lower bound;
MRTS – mass rapid transit system;
MTC – metropolitan transport corporation;
SF – solution file;
STT – service and time table;
TSF – temporary solution file;
UB – upper bound.

Introduction

The ever-increasing number of commuting people in cities has resulted in serious traffic congestion problems in urban roads. A common-sense approach to avoiding traffic snarls is to encourage people to use public transportation services. Although there are extensive public transportation services in almost all cities in India, they remain under-utilized because of poor service quality; public buses that connect various parts of the city efficiently, for example, are often not time-bound and are unreliable. Frequent delays in bus travel are caused by various reasons, the predominant among them being traffic congestions, and such delays make travel planning very tedious. While the delay itself may not be avoidable, what makes for nuisance value is that the travel status is not communicated to the commuter, who has to wait indefinitely for the service.

Providing the potential commuter with a real-time schedule/plan of the bus trip and travel can, to a large extent, improve the reliability of public transportation and help with better travel planning. Beyond obtaining information about delays and expected trip duration, real-time route schedule and planner can help passengers choose optimum routes to their destination, taking into account, parameters such as number of stops, possible waiting times in bus stops and service frequency, which decide the duration and comfort of the trip.

A route-planner is typically designed using travel data obtained from the road. This data could be historic (static)

*Corresponding author. E-mail: *lelitha@iitm.ac.in*

or real-time. A static route-planner can assist the passenger in planning the trip in advance. A real-time route-planner could help the passenger optimize a trip based on real-time updates; for example, unexpected changes in bus schedules and operation can be communicated in real-time, which can enable the commuter to make on-the-spot alterations to the trip details. The best route-planner is one that combines both static and real-time operation.

Most earlier studies on route optimization of multimodal travel have treated route optimization as non-deterministic polynomial-time hard problems that are time consuming to solve using exact algorithms. To enable faster solution, Deng and Hu (2011) used GA for route optimization. The stochastic search techniques of GAs use natural selection and evolution to solve optimization problems (Hiu 1996; Zhao, Zeng 2006).

Real-time route-planners, though available for traffic conditions with lane discipline and homogeneity, are not pervasive for the public transit systems in countries like India. This is mainly due to the high level of uncertainty involved under such traffic conditions. The variability in such traffic conditions with heterogeneity and lack of lane discipline is much higher than the lane-following, homogeneous traffic. The route-planner developed in this study, takes into account the aforementioned inaccuracies and variability by applying suitable bounds to the input received from the prediction algorithm to ensure the solutions presented to the user are stable and do not change rapidly with time. Moreover, to ensure that solutions are produced within a short period of time, the algorithm of choice here is the GA (Kumar *et al.* 2010), which has proven its efficacy under similar conditions in other studies.

1. Methodology

This real-time dynamic route-planner presented in this study provides alternatives that could help a passenger to seek the itinerary that requires the shortest time to travel from one node (origin) to the other (destination) within the network, under real-time conditions. The planner seeks to provide the solutions in the shortest possible time. Secondary objectives include a framework for scalability of the algorithm and ensuring that the solutions are reasonably stable to absorb the errors inherent in the prediction algorithm whose output the route-planner utilizes to provide the solutions. The assumptions of the model are:

- the bus capacity is considered to be infinite. This assumption is acceptable since the acceptable crowding level varies greatly among different passengers; while some passengers limit their choice to sparsely occupied buses, others might be willing to travel when the bus is occupied close to its capacity. This assumption is also convenient since currently bus crowding data is not available for most bus routes in the system considered;
- the current study assumes all passengers to be homogeneous and therefore value of time is considered to be the same across all passengers. Moreover, passengers are trying to minimize time. The repercussions of this assumption are minimized by the design of the result display – by ensuring sufficient alternatives are presented to the user and it is the user's choice to select the alternative that satisfies the user's goals;
- the exact itinerary is not fixed "apriori" and it is assumed that the user is willing to change their itinerary if the solution presented during the initial query ceases to be optimal. The user's attractive set is governed by the solutions presented by the route-planner and the user's pre-defined preferences such as maximum number of transfers;
- in the static implementation of the route-planner, it is assumed that the buses arrive as per a schedule based on past data. For the real-time implementation of the route-planner, the assumption is that buses arrive randomly, and the estimated arrivals are based on predicted values at the time of query. In case the query is made before the particular run of the bus, then values used in the static implementation are used instead.

In GA, the search space comprises the space of all feasible solutions, each point in which represents a unique solution (Obitko 1998). The fitness of this point is defined according to the objectives of the problem. A string of elements (values) in a particular order, called the chromosome, represents each solution of the GA. The Chromosome representation used in this study is shown in Figure 1, where nodes 2, 12, 14 and 20 represent bus stops, and 51000, 54000, 56000 represent unique bus routes within the network. The elements themselves are called "genes" and the specific representation of the chromosomes is the "encoding". A "generation" of population is the set of chromosomes generated during a single iteration of GA. The application of genetic operators such as "selection", "crossover" and "mutation" on successive generations of population, enhances the fitness of individual solutions.

GA has been used sporadically in bus network optimization problems. Borole *et al.* (2013), for example, developed route-planners using *K*-shortest path algorithm with GPS data, to obtain multi-criterion shortest path. Nanayakkara *et al.* (2007) developed GA-based route-planners for large urban street networks in Singapore. In their work, chromosome encoding with each gene represented a node ID. Abbaspour and Samadzadegan (2011) described the use of GA-based multimodal path computation in time-dependent personal tour planning and scheduling. This study developed a non-binary chromosome encoding scheme with alternate genes representing node and mode. Chen *et al.* (1999) also reported the development of multimodal route-planner using offline data. Jariyasunant

2	51000	12	54000	14	56000	20

Figure 1. Chromosome representation

et al. (2011) showed that pre-computing paths expedites generation of solution.

The suitability of GA in time-constraint shortest path problems indicates that they can be used for bus-based route-planning using dynamic bus GPS data. GA could solve the problem by encoding the chromosome in a suitable scheme. Effective storing and retrieval can enable rapid, real-time solutions. In our work, a GA-based travel time prediction algorithm developed in-house (Vanajakshi *et al.* 2009) has been used for developing the route-planner, as described in the following sections.

In this work, each step of the GA was modified to meet the computational requirements of route-planning. In the chromosome encoding scheme used, alternate genes represented bus stops and bus routes wherein, odd genes were bus stops and even genes were the route ID of the bus. The algorithm was first initialized for population and then evaluated and ranked according to cost value. The genetic operators were then applied on successive generations of solutions after which, infeasible chromosomes were repaired. This was followed by evaluation and ranking. The second step was repeated until the fixed limit on the number of generations was reached. The pseudo-code for the algorithm is shown below:

```
Initialize the population
Evaluate and rank initial population
Repeat
    Perform competitive selection
    Apply genetic operators on selected solutions from population
    Repair infeasible solutions
    Evaluate and rank the solutions in this generation
Until convergence criteria is satisfied
```

During initialization, the chromosome was first constructed starting at the source node. The node adjacent to the source and the corresponding mode of travel were then randomly chosen, this step being repeated with the chosen node, until the destination node was reached. The number of intermediate nodes determined the length of a chromosome; hence, the chromosomal length was directly related to the number of transfers allowed by the user. When a particular node was reached, it was added to a scan list that tracked all the nodes already visited, to avoid repetitions and cycles. It was possible during initialization that a solution did not reach the destination, for example, when a node had no outwardly directed arc, or if all nodes adjacent to the current node were already in the scan list. A counter that counted the number of attempts to select an adjacent node was maintained and when it exceeded the limit, the partial solution was discarded. A pre-defined limit was necessary because GA is probabilistic and independent of the node chosen. The process was repeated multiple times until the predefined population size is reached. The pseudo-code for the initialization step is shown here:

```
Scan = {scanned nodes}
Repeat
Start at O - Origin and pick up an adjacent node randomly (say i)
Repeat
    Add this node to Scan
    Repeat
        Counter = 0
        Pick up an adjacent node randomly (say i)
        Counter = Counter + 1
    Until a node is picked or Counter limit is reached
Until node D - Destination is reached
Until predefined population size is reached
Evaluate and rank initial population
```

The computation time was related to population size – low population size was associated with lower computation time. However, in such cases, the diversity in solution set was also low, which could lead to convergence to local minima. If the population size were high, however, the computation duration was longer, but the diversity of the solution was higher, resulting in higher probability of the existence of the global minimum.

The cost associated with each chromosome was then computed and the validity examined according to the schedule data. Cost function was taken as the duration of the trip, which is the sum of travel and waiting times. Since the study involved cost minimization, the fitness function was inversely related to cost function (Goldberg 1989). There was high cost penalty if a solution was invalid and this made the solution infeasible.

The initialization step was followed by the mating step, in which, the genetic operators (crossover, selection and mutation) were applied on successive generations of the population until the termination criterion was fulfilled. In GA, individuals with higher fitness values in the population were selected. The selection operator allowed only the fittest individual chromosomes to be taken forward to the next mating pool, and by this, the population limit on chromosomes was ensured. Hence, for selection operation, the solutions were ordered according to their cost and only those solutions with low cost function values ("fit" solutions) were carried forward to the next generation. Common genes in two parent chromosomes were detected by the crossover operator, parts of which were then swapped to produce two new daughter chromosomes. The first step of the crossover mechanism chose two parent chromosomes; in this work, the choice was random. Whether or not crossover happened during a particular iteration was decided by the crossover probability P_c. For this, nodes or potential sites for crossover in the parent chromosomes were first identified. In the presence of multiple nodes, one of them was randomly chosen. This was followed by crossover, during which two new daughter chromosomes were generated. The post-processing procedure removed cycles after this process. Figure 2 shows an example of the crossover operation.

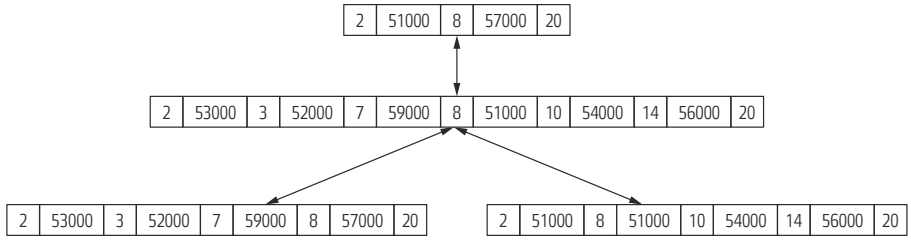

Figure 2. A single point crossover operation (at node 8)

The pseudo-code for the mating step (crossover and selection operators) is shown here:

```
Generate a random number in (0,1)
If random number generated < Pc
    Halt and pass on the same population to next generation
Else
    Select two chromosomes randomly from {Chromosome-set}_i
    Parents = randomly chosen chromosomes
    N = similar odd genes (potential sites for crossover)
    Case: N = 0
        Halt
    Case: N > 1
        Select particular site for crossover among potential sites
        (randomly)
        Go to N = 1 case with chosen site
    Case: N = 1
        Perform crossover at chosen site to produce offspring
        chromosome
        Evaluate offspring chromosome and add them to population
Rank the population {Chromosome-set}_i+1
Elitism Strategy: Retain only the fittest chromosomes in the set
```

Mutation refers to the random change in the chromosome in an effort to prevent convergence of the solution to a local minimum. In this study, mutation operation has been applied as follows: a randomly chosen odd gene (except the first and last gene) is changed and is assigned a new random value. Since this solution is not valid in its present state, a new path connecting the origin to the destination through the randomly chosen gene is then computed and duplicate nodes are removed (the repair process). This was followed by computation of the cost associated with the chromosomes generated after the mating step and rejection of infeasible solutions. The parameter governing the mutation operation is denoted by mutation probability P_m. Figure 3 shows an example of the mutation followed by repair operation.

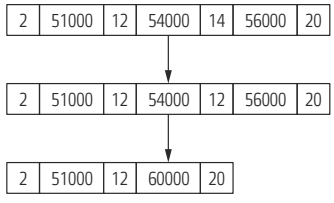

Figure 3. A mutation (node 14) followed by repair operation (remove duplicate node 12)

The pseudo-code for the mating step (mutation) is shown below:

```
Generate a random number in (0,1)
If random number generated < Pm
    Halt and pass on the same population to next generation
Else
    Select a chromosome randomly from {Chromosome-set}_i
    Swap a set of odd genes (barring first and last gene)
    Repair the chromosome
    Evaluate the chromosome
    Add the chromosome to the population set
Rank the population {Chromosome-set}_i+1
```

2. Algorithm implementation

Python was used in this work to implement the GA-based framework. The parameters were chosen based on user preferences obtained through a survey conducted in Chennai City (India). GPS data from buses operated by MTC, Chennai were used for computation. Each log (or record) included a timestamp and the corresponding longitude and latitude of the bus.

The imposed constraints were divided into two broad categories: hard constraints and soft constraints. Hard constraints used during computation of results, was again of two types: network constraint and maximum number of transfers in a particular solution. The former meant that buses were constrained to fixed routes and not allowed deviations. The bus network was represented by a directed graph comprising links to a particular bus route ID. The onward and return directions along the same bus route were taken into consideration and the directions within the same bus route were assigned different IDs. The second hard constraint of maximum number of transfers in a particular solution was chosen in order to eliminate too many solutions, thereby limiting the computational time involved.

Soft constraints, imposed after computation of results, were used to filter the solutions in order to obtain the best and most relevant solutions. Soft constraints included maximum number of transfers allowed by the user, maximum walking distance and maximum travel duration that the user preferred. These constraints enable user-specific planning solutions.

A temporal geospatial STT database was used to store network and schedule data for the algorithm. The data-

base had two components: the network structure data and temporal data for bus routes. The former was stored as a modified adjacency matrix containing information on nodes adjacent to a given node, and arcs representing the various bus routes. The temporal data for the bus routes consisted of GPS data obtained from the buses and corresponding predictions. The departure time from each bus stop was found and this was stored in the database as independent trip files for each bus route.

3. Evaluation of algorithm on a static network

Static networks provide invariant, pre-determined and historic temporal information for computation. The input data provided by static network was observed schedule of buses. Bus routes with IDs 5A, 5B, 5E, 21D, 21L, 23C and 154 were chosen for this study. A schematic of the network is presented in Appendix. The GA parameters were tuned as follows: chromosome length limit of 9 and predefined population limit of 10 were chosen for the initialization step. Crossover probability value P_c was chosen as 0.8, mutation probability value P_m as 0.01 and the total number of generations (including the initialization step) was chosen as 10 for the mating step. The parameter values used in this study are representative values, that have been fine-tuned empirically based on the network data available. In the application of the route-planner framework to a larger network, the ideal GA parameter values would vary with time since the network itself is dynamic in nature – depending on the time of the day, there could be several or few links/arcs present within the network, thereby changing the sparsity and requiring a similar change in the GA parameters as well. The authors suggest to optimize and store GA parameter values after the complete network information is available based on time period of the day. The focus in the study was on selecting representative GA values that would allow us to study the ever-growing dynamic network and focus on producing solutions within time constraints.

Ten iterations of genetic operator application were carried out for each generation. A sample solution for a query where origin was set as "Broadway", destination as "ESI Hospital" and time of travel as 10:30 am is shown in Figure 4. There were four solutions for the above query, and each solution specifies the bus routes to be taken and the bus transfer stations. For each solution, the total duration of trip in seconds, called cost value is also shown and the solutions are presented in ascending order of cost value.

Computation time measurement under static condition

One hundred nodes were chosen randomly from within the network in order to measure the computation time to arrive at solutions for a generic query. Start time of a trip was arbitrarily chosen and all feasible solutions between these nodes were computed at this time. The time taken for each iteration was recorded and the time to compute each solution is shown in Figure 5.

```
Solution #1
Bus Stop >>> Broadway
Take Bus >>> 21L
Bus Stop >>> Madhya Kailash/CLRI
Take Bus >>> 5E
Bus Stop >>> ESI Hospital
Cost Function Value is 4500

Solution #2
Bus Stop >>> Broadway
Take Bus >>> 21L
Bus Stop >>> Santhome Church
Take Bus >>> 21D
Bus Stop >>> Vannandurai
Take Bus >>> 5E
Bus Stop >>> ESI Hospital
Cost Function Value is 5891

Solution #3
Bus Stop >>> Broadway
Take Bus >>> 21D
Bus Stop >>> Ashtalakshmi Koil/Velankanni Koil/Besant Nagar Church
Take Bus >>> 5E
Bus Stop >>> ESI Hospital
Cost Function Value is 5940

Solution #4
Bus Stop >>> Broadway
Take Bus >>>> 21D
Bus Stop >>> Vannandurai
Take Bus >>> 5E
Bus Stop >>> ESI Hospital
Cost Function Value is 5940
```

Figure 4. Program output for sample query: static case

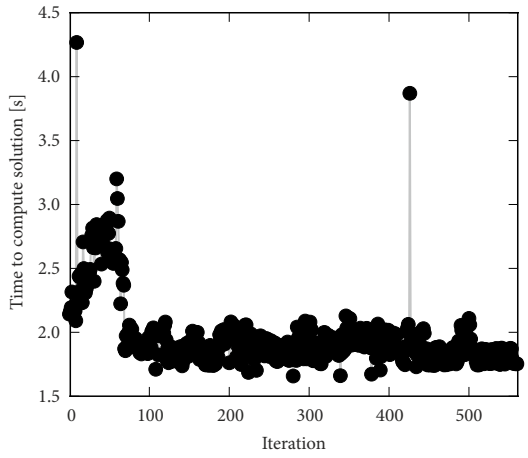

Figure 5. Time taken per iteration: static testing

The GA framework produces acceptable solutions. Although the mean time to produce solutions is high, the algorithm is robust. This paves way for the discussion in the subsequent sections where real-time data is used with the identified GA parameters from the static case. A total of 561 valid iterations were recorded. The average time taken to compute a particular solution was found to be 1.97 s, the standard deviation was 0.28 s. The limit on the number of generations was used as the termination criterion because the network chosen was sparse. As observed

from the iteration time scatter plot, barring a few cases, the GA was able to produce solutions in most cases within a short period of time. In practice, this hints at capping the maximum time for generating results to a query since this covers the solution generation for most cases. As part of the study, the cap on maximum time was enforced indirectly, by maintaining a cap on the number of generations.

4. Real-time implementation

To provide relevant information, the route-planner needs to work with real-time bus data because traffic networks are dynamic in nature and static data based information may not be sufficient to provide the ideal itineraries in real-time. The distinction of this from static planner is that ETAs of buses obtained from real-time data are utilized to optimize the user's itinerary in the real-time implementation of the planner. Therefore, in this work, the algorithm was integrated with the real-time data available for a subset of buses in 16 routes: 5A, 5B, 5E, 7B, 19B, 21L, 221H, 23C, 47A, G18, M119A, M14, M7A, M7, M70 and 154. Schematic of the network is presented in Appendix. A prediction algorithm generated the ETAs at the various bus stops along the bus route using raw data from buses. A separate file, called the ETA file was generated for each bus in a route and for specific bus stop. Predictions were made at small intervals of time until as specific bus stop was reached, after which, the prediction process was stopped.

Although this study focused on developing a route-planner for bus routes, in order to extend the route-planner's capabilities, walk mode and Chennai MRTS, which is an elevated railway line system; were also included in the geospatial database. The two-way MRTS stretch between "Velacherry" and "Chennai Beach" were included as a separate mode. A distance calculation using the GPS coordinates was proposed for the walk mode. A connector was defined as the shortest link between a bus stop and the nearest MRTS station, which would be traversed by walk. The ticket purchase time at the MRTS station was also included as a fixed time period. The parameters for the GA in real-time planning were the same as those used in the static planner.

4.1. Computation time measurement under real-time condition

A hundred nodes were chosen at random from within the network in order to obtain the computation time needed to generate solutions for a generic query, and all feasible solutions between the nodes were computed using a trip start time chosen arbitrarily. The time taken for all iterations was noted. The time taken to compute each solution is shown in Figure 6. Forty valid iterations were produced. The average time taken to compute a solution was 2.29 s, the standard deviation was 0.58 s.

4.2. Segmentation

In order to narrow down the search space during real-time query, the base algorithm was segmented to further reduce the time required to compute solutions. For this, network solutions pre-generated for all feasible node pairs were stored in SFs and could be retrieved during real-time query. Thus, the algorithm could skip the initialization steps and proceed to the mating step to apply genetic operators on the retrieved solutions. Two modules – "archiver" and "retriever"– were used to perform the segmentation. The "archiver" calculated solutions for the entire network and stored them in a database, while the "retriever" retrieves data from the database as input during a query. Figure 7 shows a schematic summarizing the working of the route-planner.

As can be observed from the schematic diagram, the mating step is now operated in isolation, instead of as part of the complete GA algorithm, which is the effect of segmentation of the algorithm. The input to the mating step is usually from the initialization step. In this, the segmentation step enables the route-planner to go directly to the mating step forgoing the necessity to initialize a population. Although the initialization step requires only a small fraction of the total time, it could sometimes fail to yield a solution, due to the fact that GA is a probabilistic tech-

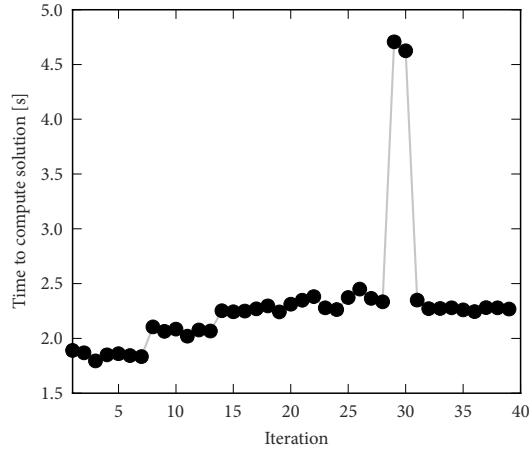

Figure 6. Time taken per iteration: real-time testing

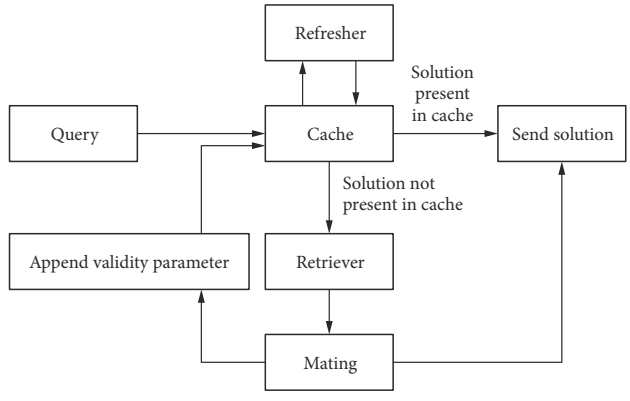

Figure 7. Schematic diagram of the route-planner

nique. Without segmentation, the common way to resolve this would be to run the initialization for multiple iterations until suitable candidate solutions are available for the next steps. However, with segmentation, it is ensured that suitable candidate solutions are available for all runs of the algorithm.

To summarize, in segmented algorithm, solutions of the network problem were generated and stored, to be retrieved when a query was made to seed the GA. After retrieving the stored solutions, they are assigned a cost value based on the real-time data available at the time of query.

4.3. Computing time in segmented algorithm

A hundred nodes were chosen from the network and base algorithm was evaluated under real-time conditions. All feasible solutions between these nodes at the same time instance were obtained as explained before. The time duration of each iteration was recorded and the time of computation of each solution is shown in Figure 8.

Forty valid iterations were produced as before. However, for segmented algorithm, the average time taken to compute a single solution was 0.12 s, the standard deviation was 0.003 s. Thus, segmentation produced approximately 95% faster results than the base algorithm.

4.4. Handling multiple queries in real-time

Delays can be caused by simultaneous generation of multiple requests to the route-planner. Computing time in such cases can be optimized by maintaining a cache, a repository-of-sorts, to store recent queries. In this work, results of earlier queries were stored in a TSF database and assigned a validity parameter to indicate the duration of validity of the SF. At every fresh query, the TSF was checked for similar (same destination and origin) queries in the queue. To evaluate the performance, three queries were sent to the route-planner, with similar first and last queries, while the intermediate query was for a different origin and destination. Due to the cache, solution to the repeated query (query 3) was generated 99% faster than when there was no cache. Thus, the use of cache improves the efficiency of real-time field implementation.

4.5. Need for error correction

Consider the following scenario: a query was generated at 11:48 am for a trip required at 12:15 pm between "Karapakkam" bus stop (origin) and "Chennai Beach" MRTS station (destination). The solution obtained using this program, as shown in Figure 9 (program output) and Figure 10 (map depiction), had both bus and MRTS modes and required a transfer.

The map (Figure 10) shows the transfer points for the sample case. Here, a passenger arrived via bus 19B at the bus stop (marked red) and walks over (blue) to the MRTS station (marked green).

In the next step, multiple queries were generated at the following times:

- about thirty minutes before journey;
- ten minutes before the start time of the journey, as estimated from the first step;
- after the journey started;
- ten minutes before ETA at the destination – "Madhya Kailash" bus stop;
- the actual time of arrival.

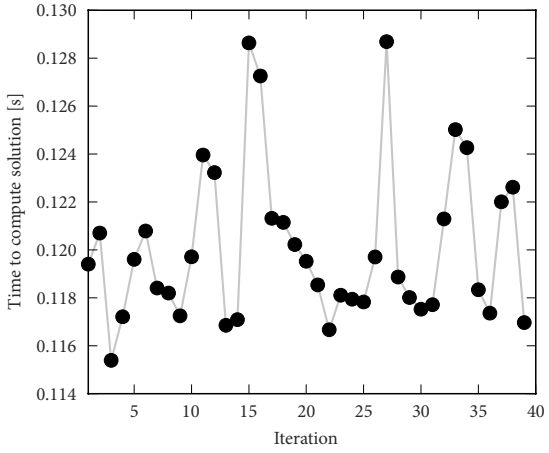

Figure 8. Time taken per iteration: segmented algorithm testing

```
Solution #1
Bus Stop >>> Karapakkam
Take Bus >>> 19B-Towards-saidapet
Bus Stop >>> Madhya Kailash
Walk >>> Madhya Kailash to Kasturbai Nagar MRTS Station
MRTS >>> Kasurbai Nagar MRTS Station
Take Train >>> Towards Chennai Beach Station
MRTS >>> Chennai Beach Station
```

Figure 9. Program output for sample query: real-time case

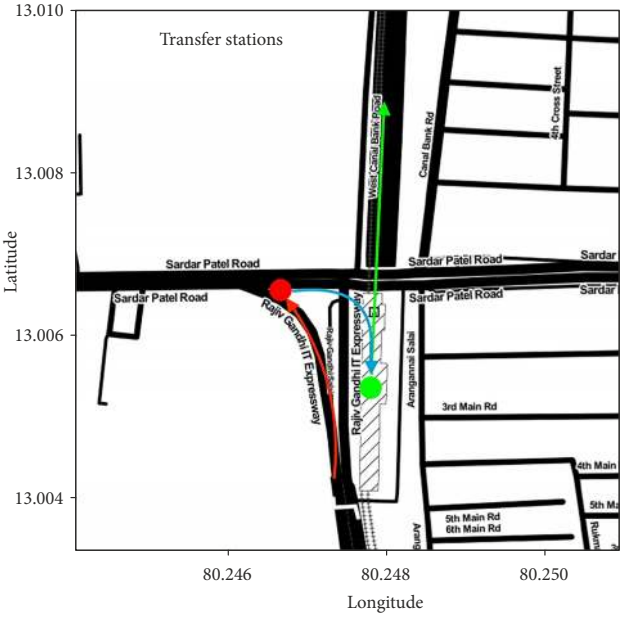

Figure 10. A map denoting the transfer points for a sample query

Table 1. ETAs at various times of query and the associated cost

Scenario	Time of query	Bus stops		Walk and ticket time	Boarding: MRTS		Cost
		Karapakkam	Madhya Kailash		Kasturbai Nagar	Chennai Beach	
1	11:48	12:17	12:39	12:48	12:51	13:20	3747
2	12:09	12:19	12:46	12:53	13:11	13:40	4860
3	12:30	n/a	12:42	12:51	13:11	13:40	4843
4	12:32	n/a	12:40	12:49	12:51	13:20	3643
5	actual data	12:19	12:43	12:52	13:11	13:40	4843

Table 1 lists the ETAs at the above times of query and the corresponding cost function value. There was significant variation in the cost value beyond the Scenario 3 (after start of journey), which implies that the user would get different values for the journey duration after the journey starts. The high variation in cost function is an unfavourable attribute for the route-planner because of the possibility of changes to the chosen solution after journey starts, due to a missed transfer.

The route-planner utilizes the output of a prediction algorithm (exogenous and not covered in the present study). This prediction algorithm uses GPS feed to generate ETA. These predicted ETAs are then used within the route-planner framework to generate candidate solutions. However, due to inconsistent GPS feed and inherent errors in predictions, these ETAs can have errors and these were primarily found to depend on the time of the day, the location and the time of query. A good route-planner should not significantly alter the solution and its cost function value during the journey, to avoid inconvenience to the commuter. It is therefore necessary to minimize the error effects in our route-planner. The errors in predicted values should be captured within the route-planner so that buffers are provided around the predicted time. Hence, these errors were observed over an extended period of time and analysed to find the bounds on the predicted values in order to generate conservative solutions within the route-planner. These bounds, to an extent, help to generate solutions that do not change rapidly over time. This is discussed in the subsequent sections.

4.6. Integration of an error function

When a user generates a query, the algorithm uses predicted travel time data available at that instant. Prediction errors in the predicted travel time may, in extreme cases, even invalidate the solution generated by the route-planner. The errors largely depend on the bus stop, time of day and bus route and must be accounted for during route-planning. We have categorized the error into two groups, depending on the time of the day and the time to reach the bus stop and the mean value of errors were calculated for each category. When there was a positive error, the bus arrives earlier than predicted and in the case of negative error – later. The bus route 19B and "Madhya Kailash" bus stop were selected to calculate the error function, and the results are shown in Figure 11.

The error function constructed, took into account the bus stop ID, the bus route ID, time of day, and the time of query. Given t_{ai} as the ETA at time t_i, t_{ai} is the actual/observed time of arrival at the bus stop. To buffer the ETA, error limits were imposed:

$$t_{ai} - \varepsilon^+ \le t_a \le t_{ai} + \varepsilon^-, \quad (1)$$

where: t_{ai} represents ETA at time t_i; t_a represents actual/observed time of arrival at the bus stop; ε^+ represents was taken as the magnitude of mean positive error; ε^- that of the mean negative error.

Since t_a was not known at the time of prediction, it was difficult to choose the error value. Hence, the values of $(t_{ai} - t_i)$ from historical error data were used. Bounds were placed on the predicted data, to ensure that the final solution set contained individual valid solutions with high probability. This also ensured that the solution or the cost function value did not fluctuate, an upper limit value of total time of the journey was obtained – leading to generation of conservative solutions.

The sample real-time query described in the earlier subsection was explored again, with bounds on the ETAs, using the error function. Table 2 shows the bounds on the ETAs. The cost function value against scenario plot (with and without the error function) is shown in Figure 12. The cost function value did not fluctuate after the trip commenced due to inclusions of bounds. Hence, the error function was useful to reduce fluctuations in the solution.

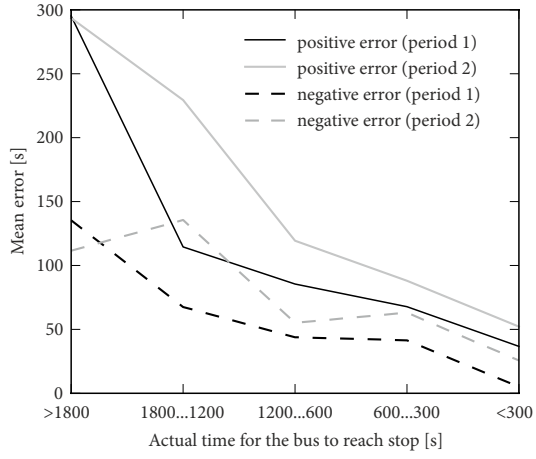

Figure 11. Mean error values v/s actual time for the bus to reach stop

Table 2. Bounds on ETAs: real-time sample query

Scenario	Time of query	Bus stops				Walk and ticket time	Boarding: MRTS		Cost
		Karapakkam		Madhya Kailash			Kasturbai Nagar	Chennai Beach	
		LB	UB	LB	UB				
1	11:48	12:14	12:19	12:38	12:45	12:54	13:11	13:40	5104
2	12:09	12:17	12:21	12:41	12:48	12:57	13:11	13:40	5430
3	12:30	n/a	n/a	12:38	12:44	12:53	13:11	13:40	5104
4	12:32	n/a	n/a	12:36	12:43	12:52	13:11	13:40	5104
5	actual	12:19		12:43		12:52	13:11	13:40	5104

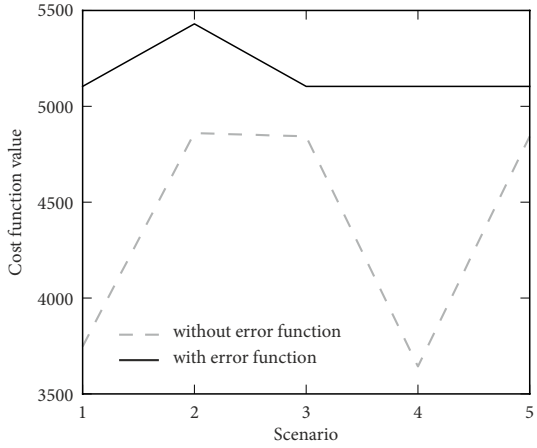

Figure 12. Cost function value against scenario with and without the error function

Conclusions

The present study developed a route-planner that can operate with both static data and real-time data from the road, with bus mode of travel being the primary focus. Bus routes of the MTC, Chennai, India were taken as a case study. The route-planner used GA framework on a set of bus routes to test its performance first on a static network, and was later extended to real-time predictions. Real-time prediction values were associated with inherent errors that sometimes led to erroneous solution dispensed to the user during the journey. To resolve this issue, an error function was created and was shown to be effective in providing reliable data to the traveller. Therefore, two key features of the route-planner presented as a part of this study are its ability to generate reasonable results in a relatively short period of time, as exemplified by the cases discussed in the paper, and the ability of the error-function incorporated in the planner to trim the solution set by getting rid of solutions with high uncertainty (due to high probability of missing transfers).

In order to extend the route-planner's capabilities, walk mode and the Chennai MRTS were also included in the geospatial database and this showed that this framework could potentially be integrated with other modes of public transport. The algorithm was subsequently segmented to speed up prediction process. In addition, a temporal cache was also introduced during implementation, to handle multiple queries generated simultaneously. Overall, it was seen that there is great promise for scalability and citywide implementation for the real-time route-planner developed in this study. The uncertainty and poor service quality perceived with public transport bus services in India could be mitigated by further developments in the route-planner introduced in this paper.

Acknowledgements

The authors acknowledge the support for this study as a part of the sub-project CIE/10-11/168/IITM/LELI under the Centre of Excellence in Urban Transport project funded by the Ministry of Urban Development, Government of India, through letter No N-11025/30/2008-UCD.

Appendix

1) All tests were performed using Python 3.5.1-0, on a computing system with the following configuration: processor – 2.6 GHz Intel Core i5, memory – 8 GB 1600 MHz DDR3, operating system – OS X (10.11.4).
2) A comparison with Dijkstra's algorithm on a dummy network was done prior to implementation and the results are presented below.

   Dijkstra's algorithm is deterministic and is assured to give the minimal cost solution. GA on the other hand is probabilistic in nature. In order to check if the GA framework suggested in the present study could generate comparable solutions, a dummy network was chosen, as shown in Figure A1. On this network, both

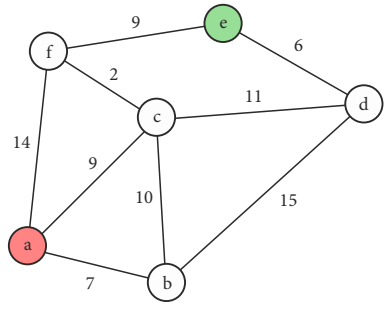

Figure A1. The dummy network (*note:* the nodes are denoted by lower-case alphabets and the numbers on the links represent cost associated with it)

Dijkstra's algorithm and the modified GA algorithm were tested to find the shortest path from node "a" to node "'e". The Dijktra's algorithm could find the shortest path solution [a–c–f–e] in approximately 0.00097 s. The parameters for GA framework were chosen to be same as earlier. The GA framework was tested for 100 separate test runs for different initial population sizes, and it was checked for the number of times, it could find the best solution. It was found that for an initial population size of 10, the GA framework was able to find the best solution in 85 out of 100 test runs (85%), which was satisfactory. Moreover, in 12 out of remaining 15 test runs, it could generate the second-best solution. The average time taken by the algorithm was 0.00042 s per test run. For an initial population size of 30, the GA framework was able to find the ideal solution always. Table A1 shows the variation of the average computation time per test run and percentage of optimal solutions with the initial population size that is chosen.

Based on the mentioned comparison, it was concluded that the present GA framework is well suited to the problem presented in this study.

3) Figure A2 shows the various bus routes and MRTS route, which were utilized in the study.
4) To demonstrate the efficacy of the GA method used, the CSA was utilized (Dibbelt *et al.* 2017). Comparison of the time required and the cost function value for both CSA- and the GA-based implementations of the route was carried out. Comparison was done for both (1) static implementation: a sample query with origin as "Broadway" and destination as "ESI Hospital" and (2) real-time implementation: a sample query with origin and destination as "Karapakkam" and "Chennai Beach" station was generated with scheduled time of travel as 12:15 pm. Results obtained are presented in Table A2.

Table A1. Variation of average computation time and percentage of optimal solutions with the initial population size

Initial population size	% of optimal solutions	Average computation time (test run) [s]
3	35	0.00018
4	47	0.00021
5	60	0.00025
6	65	0.00026
7	69	0.00030
8	74	0.00031
9	80	0.00035
10	85	0.00042
30	100	0.00104

It can be seen that for the static implementation, the CSA is better. On the other hand, segmented GA performs better than CSA for the real-time implementation. This shows the advantage of using GA for more complex real-time problems. To further illustrate the above, five additional examples were evaluated under real-time implementation. The results obtained are presented in Table A3.

Table A2. Performance comparison of GA and CSA

Algorithm used	Static implementation		Real-time implementation	
	Computation time [s]	Cost function value	Computation time [s]	Cost function value
GA	1.97	4527	0.12	3747
CSA	1.77	4500	0.24	3747

Note: the values for GA presented in Table are averaged over 100 runs of the query.

Table A3. Comparison of average computation time for additional examples (real-time implementation)

Source	Destination	Time of query [h:min]	Cost value for CSA	Computation time for CSA [s]	Average cost value for GA	Average computation time for GA [s]
ESI	D.M.S.	9:45	2027	0.20	2029	0.13
Mettupalayam	Errikarai	9:30	886	0.17	886	0.10
Vadapalani Temple	Ajantha	10:10	981	0.20	981	0.12
Guindy Race Course	Maduvinkarai	12:15	329	0.20	329	0.14
Saidapet	Malar Hospital	19:05	1253	0.29	1260	0.12
Post Office	Sidco	18:30	423	0.21	423	0.11

Note: the values for GA presented in Table are averaged over 100 runs of the query.

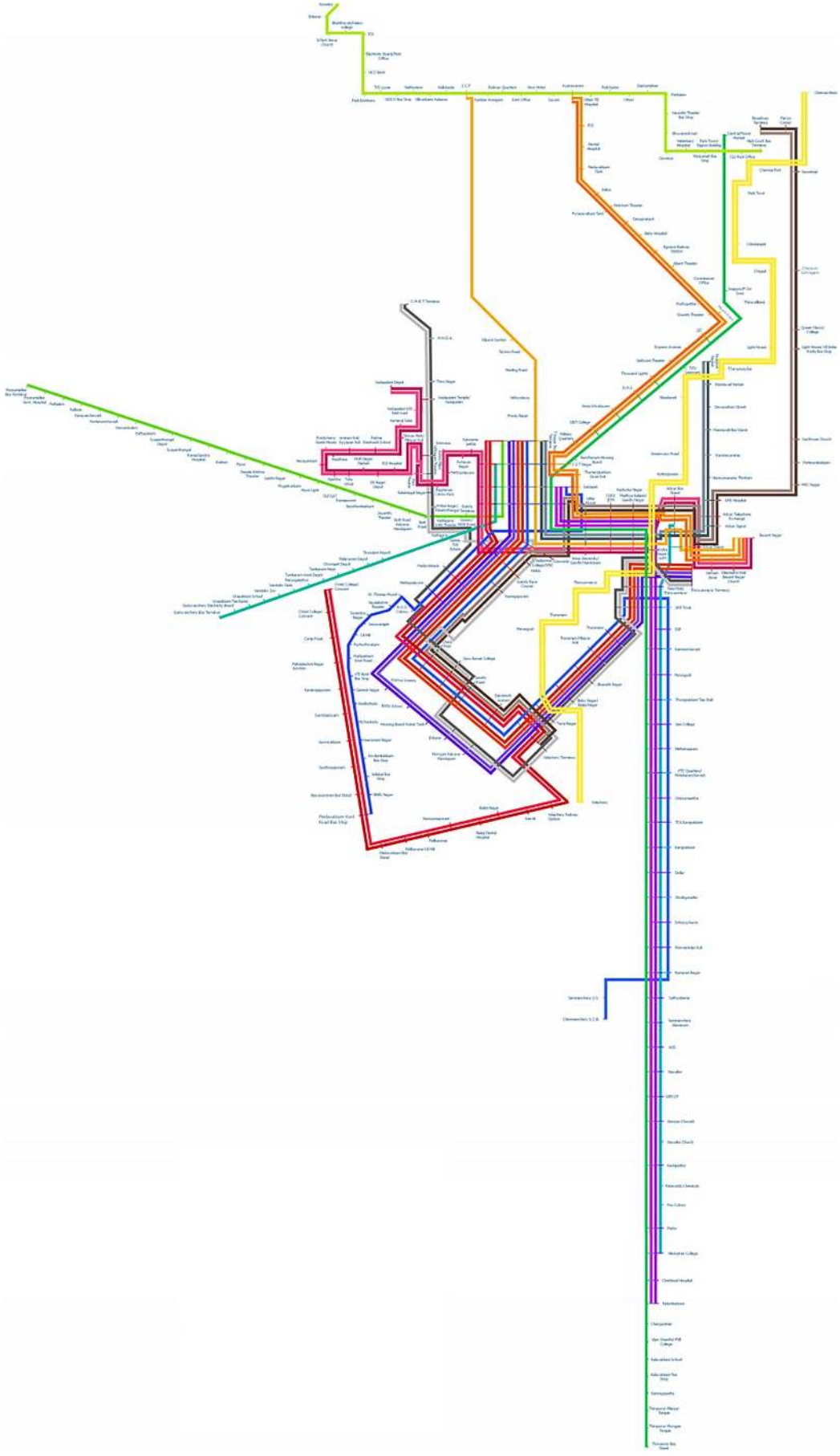

Figure A2. Schematic map depicting the bus routes utilized in this study

References

Abbaspour, R. A.; Samadzadegan, F. 2011. Time-dependent personal tour planning and scheduling in metropolises, *Expert Systems with Applications* 38(10): 12439–12452. https://doi.org/10.1016/j.eswa.2011.04.025

Borole, N.; Rout, D.; Goel, N.; Vedagiri, P.; Mathew, T. V. 2013. Multimodal public transit trip planner with real-time transit data, *Procedia – Social and Behavioral Sciences* 104: 775–784. https://doi.org/10.1016/j.sbspro.2013.11.172

Chen, C.; Kitamura, R.; Chen, J.; 1999. Multimodal daily itinerary planner: interactive programming approach, *Transportation Research Record: Journal of the Transportation Research Board* 1676: 37–43. https://doi.org/10.3141/1676-05

Deng, Y.; Hu, S. 2011. Route optimization of multi-modal travel based on improved genetic algorithm, in *Proceedings 2011 International Conference on Transportation, Mechanical, and Electrical Engineering* (*TMEE*), 16–18 December 2011, Changchun, China, 1701–1704. https://doi.org/10.1109/TMEE.2011.6199539

Dibbelt, J.; Pajor, T.; Strasser, B.; Wagner, D. 2017. *Connection Scan Algorithm*. 49 p. Available from Internet: https://arxiv.org/abs/1703.05997v1

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional. 432 p.

Hiu, W. 1996. *Genetic Algorithms*, in *Surveys and Presentations in Information Systems Engineering* (*SURPRISE*), Imperial College London, UK.

Jariyasunant, J.; Mai, E.; Sengupta, R. 2011. Algorithm for finding optimal paths in a public transit network with real-time data, *Transportation Research Record: Journal of the Transportation Research Board* 2256: 34–42. https://doi.org/10.3141/2256-05

Kumar, M.; Husian, M.; Upreti, N.; Gupta, D. 2010. Genetic algorithm: review and application, *International Journal of Information Technology and Knowledge Management* 2(2): 451–454. https://doi.org/10.2139/ssrn.3529843

Nanayakkara, S. C.; Srinivasan, D.; Lup, L. W.; German, X.; Taylor, E; Ong, S. H. 2007. Genetic algorithm based route planner for large urban street networks, in *2007 IEEE Congress on Evolutionary Computation*, 25–28 September 2007, Singapore, 4469–4474. https://doi.org/10.1109/CEC.2007.4425056

Obitko, M. 1998. *Genetic Algorithms*. Available from Internet: https://www.obitko.com/tutorials/genetic-algorithms/index.php

Vanajakshi, L.; Subramanian, S. C.; Sivanandan, R. 2009. Travel time prediction under heterogeneous traffic conditions using global positioning system data from buses, *IET Intelligent Transport Systems* 3(1): 1–9. https://doi.org/10.1049/iet-its:20080013

Zhao, F.; Zeng, X. 2006. Simulated annealing–genetic algorithm for transit network optimization, *Journal of Computing in Civil Engineering* 20(1): 57–68. https://doi.org/10.1061/(ASCE)0887-3801(2006)20:1(57)