CHAPTER 3

ROUTE OPTIMIZATION METHODS IN INTELLIGENT TRANSPORTATION SYSTEMS USING THE SPATIAL NETWORK DATA MODEL

3.1 INTRODUCTION

Spatial network database is an important component of Spatial Database. Transportation planning is one of the important applications of spatial network database. The Spatial network data models have three levels of Database design that are conceptual data models (ie Graphs), logical data models (ie., data types, operations) and physical data models. Spatial network application is first modeled at the conceptual level and then subsequently mapped into a logical model (Greene 1989).

Conceptual Data Model in Spatial Networks is a Graph network. It is defined as a directed graph $G = (N, A)$ consisting of a set of $N$ nodes and a set of arcs ‘$A$’ with associated numerical values, $n=|N|$, the number of arcs, $m=|A|$, and the length of an arc connecting nodes ‘$i$’ and ‘$j$’, are denoted as $l(i,j)$. The shortest path problem can be stated as follows: given a network, find the shortest distance (least costs) from a source node to all other nodes or to a subset of nodes on the network. These shortest paths represent a directed tree $T$ rooted from a source node with the characteristic that a unique path from’s’ to any node ‘$i$’ on the network. The length of the shortest path from’s’ to any node ‘$i$’ is denoted as $d(i)$. This directed tree is called as a shortest path.
tree. Logical data model for spatial networks is a common data type. Spatial Data types are Point, LineString, Polygon, MultiPoint, MultiPolygon, and MultiLineString. Spatial network consists of Vertex, Directed Edge, Graph and Path. The detailed spatial data types is shown in Figure 3.1.

![Geometry Type Diagram](image)

**Figure 3.1 Geometry type**

Road-map databases contain a large amount of complex information, including street names, street address ranges, turn restrictions, one-way street information and others. In addition, these databases are likely to contain a large amount of time-varying data, which need to be checked for accuracy on a periodic basis. More and more applications are based on map databases, so establishing quality evaluation procedures and assurance programs for these databases is of great importance. In Figure 3.2 Roads are represented as lines and Location of the place are represented as points and Figure 3.3 shows the Distance from one Location to many Locations (Theodoridis 1998).
Figure 3.2 Roads and location

With the development of geographic information systems (GIS) technology, network transportation analysis within a GIS environment has become a common practice in many application areas. A key problem in network and transportation analysis is the computation of shortest paths between different locations on a network. Sometimes this computation has to be done in real time. In some cases the fastest route has to be determined in a few seconds in order to ensure the safety of a patient. Moreover, when large real road networks are involved in an application, the determination of shortest paths on a large network can be computationally very intensive. Because many applications involve real road networks and because the computation of a fastest route (shortest path) requires an answer in real time, a natural question to ask is: Which shortest path algorithm runs fastest on real road networks? (Yangjun Chen 2003).
Although considerable empirical studies on the performance of shortest path algorithms have been done, there is no clear answer as to which algorithm, or a set of algorithms runs fastest on real road networks (Cherkassky 1993).

3.2 GRAPH THEORY

A (directed) graph \( G = (N, E, C) \) consists of a node set \( N \), a cost set \( C \) and an edge set \( E \). The edge set \( E \) is a subset of the cross product \( N \times N \). Each element \((u, v)\) in \( E \) is an edge joining node \( u \) to node \( v \). Each edge \((u,v)\) is associated with a cost \( C(u, v) \). Cost \( C(u, v) \) takes values from the set of real numbers. A node \( v \) is a neighbor of node \( u \) if edge \((u, v)\) is in \( E \). Degree of a node is the number of neighboring nodes. A path in a graph from a source node ‘s’ to a destination node ‘d’ is a sequence of nodes \((v_0, v_1, v_2, ..., v_k)\) where \( s = v_0, d = v_k \), and the edges \((v_0, v_1), (v_1, v_2), ..., (v_k, d)\) are present in \( E \).
The number of edges in a path is called the cardinality of the path. The cost of the path is the sum of the cost of edges.

\[ \sum_{i=1}^{k} C(v_{i-1}, v_i) \]

An optimal path from node u to node v is the path with smallest cost. The set of path from node u to node v is denoted by paths (u,v). A suffix of a path is obtained by removing nodes and edge from the beginning of path. For example path via \((v_1, v_2, ..., v_k)\) is a suffix of path via \((v_0, v_1, v_2, ..., v_k)\). A shortest\_path\_tree(s) is a collection of shortest paths from source node s to all nodes in the graph, with s being the root node. Diameter of a graph is the largest cardinality of the shortest path between any pair of nodes. If there exists a path from all nodes to all other nodes in the graph, then the graph is called connected. A path cost estimator in a graph is a function \(f(u, v)\) which computes estimated cost of an optimal path between the two nodes u and v in the graph. A path cost estimator is admissible, if it always under estimates the cost of the path, i.e. \(f(u, v) \leq \text{path}(u,v)\) for all path in paths(u,v). A path cost estimator is called monotonic iff(suffix(path)) \(\leq f(path)\) for all paths and all suffixes. Single pair path problems can be divided into shortest path and any path problems. Shortest path problems can be defined as follows: Given a graph \(G = (N, E)\) and nodes u and v in N, find the shortest path between u and v. The shortest path is the path with smallest cost. Any path problems can be defined by removing the shortest path constraint from shortest path problem (Fu 2006).
3.3 FORMULATION OF THE PROBLEM

In real road network, the route optimization method is the problem of finding a path between two nodes such that the sum of the distance of its constituent edges is minimized. Genetic Algorithm is used to find the shortest path and the result of this algorithm has been compared with the existing Dijkstra’s Algorithm and Ant Colony Optimization. Four parameters applied on shortest path algorithms are Network representations, Labeling Method, Selection Rules and Data Structures. Execution time is obtained using the set of one-to-one, one-to-all and all-to-all shortest paths and using the road data sets with different number of nodes and links.

Four parameters that effect on shortest path algorithms are Network Representations, Labeling Method, Selection Rules and Data Structures (Zhan and Noon 1996).

3.3.1 Network Representations

The network model contains logical information such as connectivity relationships among nodes and links, directions of links, and costs of nodes and links. There are several straightforward ways of representing a general network for computational purposes. Past research has proven that the Forward Star representation is the most efficient data structure for representing networks (Gallo and Pallottino 1988).

Three sets of arrays are used in the forward star data structure. The first array is used to store data associated with arcs, the second array is used to store data related to nodes and the third array is used to store data associated with road information (road name).
Figure 3.4 Transport network model
All arcs of a network in question are maintained in a list and are ordered in a specific sequence. That is, arcs emanating from nodes 1, 2, 3, ... are ordered sequentially. Arcs emanating from the same node can be ordered arbitrarily, however. All information associated with an arc, such as starting node, ending node, cost and arc length are stored with the arc in corresponding arrays. Transport Network model is shown in the Figure 3.4. First array contains node ID (ie. A, B, C, D, E and F). Second array contains Link ID, Start node, end node and Road Id. Third array contains various Road ID and Road Name

3.3.2 Labeling Method

Labeling method provide an alternative approach for solving transport problem. Labeling techniques can be used to solve a wide variety of network problems, such as shortest-path problems, maximal-flow problems and minimum spanning tree problem. The basic idea behind the labeling procedure is to systematically attach labels to the nodes of a network until the optimum solution is reached. Three pieces of information are maintained for each node ‘i’ in the labeling method while constructing a shortest path tree: the distance label, d(i), the parent node, p(i), and the node status, S(i). The distance label, d(i), stores the upper bound of the shortest path distance from ‘s’ to ‘i’ during iteration. Select and edge (s,i) such that d(s) + length (s,i) < d(i). Upon termination of an algorithm, d(i) represents the unique shortest path from ‘s’ to ‘i’. The parent node p(i) records the node that immediately precedes node ‘i’ in the out-tree. The node status, S(i), can be one of the following: unreached, temporarily labeled and permanently labeled. When a node is not scanned during the iteration, it is unreached.

Normally the distance label of an unreached node is set to positive infinite. When it is known that the currently known shortest path of getting to
node ‘i’ is also the absolute shortest path will ever be able to attain, the node is called permanently labeled. When further improvement is still expected to be made on the shortest path to node ‘i’, node ‘i’ is considered only temporarily labeled. It follows that \( d(i) \) is an upper bound on the shortest path distance to node \( i \) if the node is temporarily labeled; and \( d(i) \) represents the final and optimal shortest path distance to node ‘i’ if the node is permanently labeled.

### 3.3.3 Selection Rules and Data Structures

“The performance of a particular shortest path algorithm partly depends on how the basic operations in the labeling method are implemented. Two aspects are particularly important to the performance of a shortest path algorithm: 1) the strategies used to select the next temporarily labeled node to be scanned, and 2) the data structures utilized to maintain the set of labeled nodes” (Ahuja 1993). Strategies commonly used for selecting the next temporarily labeled node to be scanned are FIFO (First In First Out): the oldest node in the set of temporarily labeled nodes is selected first in this search strategy, LIFO (Last In First Out): the newest node in the set of temporarily labeled nodes is selected first in this search strategy, BFS (Best-First-Search): the minimum distance label from the set of temporarily labeled nodes is considered the best node (Gallo and Pallottino 1988).

A number of data structures can be used to manipulate the set of temporarily labeled nodes in order to support these strategies. These data structures include arrays, singly and doubly linked lists, stacks, buckets and queues. A singly linked list contains a collection of elements. Each element has a data field and a link field. The data field contains information to be stored, and the link field contains a pointer pointing to the next element in the list.
A doubly linked list contains two pointers. One pointer points to the previous element in the list, and another pointer points to the next element in the list. Stack is another special type of list which only allows removal and addition of an element at one end of the list. The queue is a special type of list which allows the addition of an element at the tail and the deletion of an element at the head (Ahuja 1993). Data are representing in the form of points. Points contain two values x and y. Each point contain label (i.e) name of the node. Then name of the node is stored in Singular linked list. Each element in the singular linked list has data field to store the value x, y and a link field that points to the next element in the list.

3.4 ALGORITHM

Time of transportation system has been obtained by using Genetic Algorithm and compared with the existing Dijkstra's Algorithm and Ant Colony Optimization algorithm.

3.4.1 Genetic Algorithm

Genetic Algorithm is a Meta heuristic technique (Diaz, 1996) that could provide robust tools with optimal or quasi-optimal designing, programming of transportation networks and node location. Because of its excellent flexibility, robustness and adaptability characteristics, it has been successfully applied in the non-linear and complex optimization problem solutions, and also it is much appropriate to face the noisy combinatorial problems associated with the real systems optimization and transportation networks.
Individual representation as paths, composed of routes (stretch joining two adjacent nodes), conforming a trajectory from an origin node to a goal node, and operators utilization on sub-paths are novel features. Different solutions with Genetic Algorithms to the same problems have been well studied in (Gen 1997). A route is the two-neighbor nodes union and is represented as a matrix containing the connecting arcs of each node. The possible amount of paths in each generation corresponds to the population.

Fitness is assigned comparing each path with the others, and is the method to determine if a path has the possibility of survival to the next generation. Fitness is computed using the sum of products of the route lengths and their associated costs. Selection operation extracts first allocated elements of the sorted population and they are transferred to the first positions of the next generation population, based on a pre-established selection rate.

A simple crossing operator is used in this algorithm. Two paths from the previous generation are randomly chosen. Two cross points are randomly picked in each path string and then the marked fragments of the paths are extracted. They are interchanged in both the paths. Mutation operator is used in a similar way of crossing one. A path and its mutation points are randomly chosen. In the case of ‘n’ nodes, a number of ‘n’ generations are selected. The more adequate selection, crossing and mutation rates for the proposed model are found to be 32%, 38%, and 30% of the population paths, respectively.

The shortest path problem to solve is a generalization of the traditional problem of shortest path. This is, optimal (upon fitness function evaluation) path searching connecting an origin node with a goal node, passing through an intermediary node set, linking each other by non directed routes, from which, length, transportation costs, average altitude, time attributes and others, are known, and used to evaluate the fitness function.
The genetic algorithm allows an initial path population by random generation. Each path-individual has a fitness function facilitating to be differentiated with others, and then through genetic operators participate in next generation’s development of better paths, producing each time better paths for the trajectories with the best fitness from the previous population.

**Methodology**

Following is the procedure for finding the shortest path from a starting node to a destination node in a real road network using genetic algorithm. This is compared with existing Dijkstra’s algorithm and Ant colony optimization algorithm.

Genetic algorithms (GA) are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. Initial populations of parent solutions are generated at random. Here the evolution starts from a population of randomly generated individuals. In each generation, Population of strings is created, which is further processed by three operators: Reproduction, Crossover, and Mutation. Reproduction is a process in which individual strings are copied according to their fitness function. Crossover is the process of swapping the content of two strings at some points. Finally, Mutation is the process of flipping the value at a particular location in a string with a very low probability.

**Algorithm using GA**

Procedure Iterative( N, E, s, d);
begin
  foreach u in N do
    begin
      C(s,u) = \infty;
      C(u,u) = 0;
path(u,v): = null;
end
frontierSet: = [ s ];
while not_empty(frontierSet) do
begin
foreach u in frontierSet do
begin
frontierSet: = frontierSet - [u];
fetch( u.adjacencyList);
foreach <v, C(u,v)> in u.adjacencyList
begin
if C(s,v) > C(s,u) + C(u,v) then
begin
C(s,v): = C(s,u) + C(u,v);
path(s,v): = path(s,u) + edge(u,v);
if not_in(v, frontierSet) then
frontierSet: = frontierSet + [v];
end; end
end
end
end
end procedure

procedure Fetch(u,adjacencyList)
begin
calculate fitness values of individuals
select two individuals at random
single-point crossover
return highfitness individual
end procedure
3.4.2 Dijkstra’s Algorithm

Dijkstra’s Algorithm (Golden 1976) for computing the shortest path is based upon the calculation of values in three one-dimensional arrays, each of size equal to the number of nodes in the network. Each row of each array corresponds to one of the nodes of the network. As the algorithm proceeds, paths are calculated from the start node to other nodes in the network. The paths are compared and the best (minimum weight) paths are chosen. This process is done continuously.

Dijkstra( N, E, s, d);
foreach u in N
do
begin
   C(s,u) = \infty;
   C(u,u) = 0;
   path(u,v): = null;
end
frontierSet: = [ s ];
explodedSet: = emptySet;
while not_empty(frontierSet)
do
begin
   select u from frontierSet with minimum C(s, u);
   frontierSet: = frontierSet - [u];
explodedSet: = explodedSet + [u];
if (u = d) then
   terminate
else
begin
fetch( u.adjacencyList);
foreach <v, C(u,v)> in u.adjacencyList
if C(s,v) > C(s,u) + C(u,v) then
begin
   C(s,v): = C(s,u) + C(u,v);
   path(s,v): = path(s,u) + (u,v);
   if not_in(v, frontierSet U exploredSet) then
      frontierSet: = frontierSet + [v];
end
end

At each stage in the computation:

- The array distance keeps track of the current minimal distances from the start node to the array nodes. (As the algorithm proceeds, these distances become refined to closer approximations to the shortest distance).

- The array path keeps a record of the preceding nodes on the current best paths from the start node to the array node.

- The array included marks off the nodes as they are used in the calculation of the minimal distance path from start node to the end node.

The algorithm initialized arrays as follows:

- Cells in the distance array are initialized to 0 for the start node cell; infinity for the cells whose nodes are not directly connected to the start node; and to their direct distances from the start node to all the other cells.
• Cells in the path array are initialized to the start node for cells whose nodes are adjacent to start, otherwise undefined.

• Cells in the included array are initialized to ‘no’ for all cells except the start node cell.

3.4.3 Ant Colony Optimization

The Ant System, introduced by Colomi et al. (1991), and Dorigo et al. (1992) with an application on the Traveling Salesman Problem (TSP), is a recent meta heuristic for hard combinatorial optimization problems. Many Ant System algorithms, proven to be very efficient, have been proposed to solve different types of combinatorial optimization problems such as symmetric and asymmetric traveling salesman problems. The idea of imitating the behavior of real ant colonies for solving hard combinatorial optimization problems led to the development of the ant colony algorithms. Real ants communicate with each other via an aromatic essence called ‘pheromone’ in their search of food, where the quantity of pheromone depends on the quality of the food source. This will consequently make all ants choose the paths leading to rich and nearby food sources as the pheromone trails on these paths will grow faster. In time all ants will follow the shortest path. (Montemanni 2005)

Choosing the shortest path can be explained in terms of autocatalytic behavior (i.e. positive feedback) that the more are the ants following a trail the more that trail becomes attractive for being followed. The most interesting aspect of autocatalytic process is that finding the shortest path around the obstacle is the result of the interaction between the obstacle shape and ants distributed behavior. Although all ants move at approximately the same speed and deposit a pheromone trail at approximately the same rate, it takes longer
to go on their longer side than on their shorter side of obstacles. This makes the pheromone trail accumulate quicker on the shorter side.

Ant Colony Optimization (ACO) is a paradigm for designing metaheuristic algorithms for combinatorial optimization problems. The first algorithm which can be classified within this framework was presented in 1991 and, since then, many diverse variants of the basic principle have been reported in the literature. The essential trait of ACO algorithms is the combination of a priori information about the structure of a promising solution with a posteriori information about the structure of previously obtained good solutions.

Metaheuristic algorithms are algorithms which, in order to escape from local optima, drive some basic heuristic: either a constructive heuristic starting from a null solution and adding elements to build a good complete one, or a local search heuristic starting from a complete solution and iteratively modifying some of its elements in order to achieve a better one. The metaheuristic part permits the low level heuristic to obtain solutions better than those it could have achieved alone, even if iterated. Usually, the controlling mechanism is achieved either by constraining or by randomizing the set of local neighbor solutions to consider in local search, or by combining elements taken by different solutions.

The characteristic of ACO algorithms is their explicit use of elements of previous solutions. In fact, they drive a constructive low-level solution, but including it in a population framework and randomizing the construction in a Monte Carlo way. A Monte Carlo combination of different solution elements
is suggested also by Genetic Algorithms, but in the case of ACO the probability distribution is explicitly defined by previously obtained solution components.

The particular way of defining components and associated probabilities is problem-specific, and can be designed in different ways, facing a trade-off between the specificity of the information used for the conditioning and the number of solutions which need to be constructed before effectively biasing the probability distribution to favor the emergence of good solutions. Different applications have favored either the use of conditioning at the level of decision variables, thus requiring a huge number of iterations before getting a precise distribution, or the computational efficiency, thus using very coarse conditioning information.

The first ant finds the food source (F), via any way (a), then returns to the nest (N), leaving behind a trail pheromone (b) (ie. a chemical signal that triggers a natural responses in another member of the same species). Because of the shortest route, Ants indiscriminately follow four possible ways but, the strengthening of the runway makes it more attractive. Ants take the shortest route whereas the long portions of other ways lose their trail pheromones. In a series of experiments on a colony of ants with a choice between two unequal length paths leading to a source of food, biologists have observed that ants are tend to use the shortest route. Figure 3.5 shows the movement of ant F to N.
Figure 3.5 Movement of ant from F to N

ACO Algorithm

procedure ACO_MetaHeuristic
while (not_termination)
    generateSolution( )
    pheromoneUpdate( )
    daemonActions( )
end while
end procedure

Ants generation and activity, pheromone trail evaporation, and
daemon actions of ACO need synchronization. In general, a strictly sequential
scheduling of the activities is particularly suitable for non-distributed
problems, where the global knowledge is easily accessible at any instant and
the operations can be conveniently synchronized. On the contrary, some form
of parallelism can be easily and efficiently exploited in distributed problems
like routing in telecommunications networks (Dorigo et al., 1998).
3.5 RESULTS AND DISCUSSION

The road networks were stored and maintained as a set of nodes in the spatial database. The nodes, links and link lengths were obtained from the remote sensing satellite image and stored in the database. Satellite images captured by Indian Remote Sensing Satellites namely IRS – P6 during the period of March 2005 were taken for analysis. The size of the collected data was 284 MB. The road data regarding the network were stored and maintained in Arcview GIS. This was run on a Pentium IV workstation with 512MB RAM and 2400 MHz of CPU under the Windows NT environment.

The three shortest path algorithms were tested using real road networks. The networks used for testing include road networks of Madurai city (Figure 3.9). Group of Nodes were selected randomly from a set of nodes in the real road network. One Source node and a Destination node were selected from the group of nodes. Identification of the various paths available from source node to destination node was done, and the optimum path from the source node to destination node was selected. The execution time for the optimum path that was selected is obtained. For each network, a sample of 50, 100, 200, 300 and 500 nodes were randomly selected at the outset and designated as the sample source nodes for that network for different experiments.

The aim of finding the shortest path is to find the shortest distance (least cost) path from a source node ‘s’ to every other node in the node set in the real road network ‘N’. Research study focuses on the relative execution speeds of the various algorithms. The computational results for this problem were obtained using the set of public domain java source codes for computing shortest paths. Their implementations proved to be fast with respect to
computation time and efficient with respect to storage requirements. Twenty iterations have been carried out to calculate the results

Real road networks are used for evaluation rather than randomly generated networks. To find the processing time, the implementation of stack order of labeled node is done. These one-to-all shortest paths can be represented as a directed out-tree rooted at the source node ‘s’. This directed tree is referred to as a shortest path tree. All the algorithms are evaluated using the labeling method, but they differ according to the rules used to select labeled nodes for scanning and in the data structures used to manage the set of labeled nodes. This rule ensures that the shortest path tree is constructed by “permanently labeling” one node at a time. Once a node is permanently labeled, its optimal shortest path distance from the source node can be identified.

The results are presented in the Tables 3.1 to 3.3 and Figures 3.6 to 3.8 show the execution time for the algorithms with different conditions. X-axis represent the number of nodes and Y-axis is the time required for reaching the node from the source to destination. Time is the parameter chosen for all the three algorithms.

**Table 3.1 Execution time of algorithms with one to one condition in seconds**

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Genetic</th>
<th>ACO</th>
<th>Dijkstra's</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.46</td>
<td>0.96</td>
<td>0.43</td>
</tr>
<tr>
<td>100</td>
<td>3.96</td>
<td>4.96</td>
<td>3.25</td>
</tr>
<tr>
<td>200</td>
<td>11.13</td>
<td>14.42</td>
<td>12.14</td>
</tr>
<tr>
<td>400</td>
<td>43.55</td>
<td>52.24</td>
<td>49.13</td>
</tr>
<tr>
<td>500</td>
<td>89.45</td>
<td>104.15</td>
<td>102.49</td>
</tr>
</tbody>
</table>
Table 3.2 Execution time of algorithms with one to all condition in seconds

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Genetic</th>
<th>ACO</th>
<th>Dijkstra's</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.58</td>
<td>0.57</td>
<td>0.52</td>
</tr>
<tr>
<td>100</td>
<td>5.39</td>
<td>5.38</td>
<td>5.32</td>
</tr>
<tr>
<td>200</td>
<td>15.12</td>
<td>18.12</td>
<td>17.34</td>
</tr>
<tr>
<td>400</td>
<td>44.43</td>
<td>48.12</td>
<td>50.11</td>
</tr>
<tr>
<td>500</td>
<td>100.37</td>
<td>109.56</td>
<td>111.42</td>
</tr>
</tbody>
</table>

Table 3.3 Execution time of algorithms with all to all condition in seconds

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Genetic</th>
<th>ACO</th>
<th>Dijkstra's</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.62</td>
<td>0.58</td>
<td>1.32</td>
</tr>
<tr>
<td>100</td>
<td>6.61</td>
<td>6.03</td>
<td>8.25</td>
</tr>
<tr>
<td>200</td>
<td>18.03</td>
<td>18.63</td>
<td>32.76</td>
</tr>
<tr>
<td>400</td>
<td>51.76</td>
<td>72.04</td>
<td>102.89</td>
</tr>
<tr>
<td>500</td>
<td>107.78</td>
<td>134.76</td>
<td>207.78</td>
</tr>
</tbody>
</table>
Figure 3.6 Graph with one to one condition for three algorithms

Figure 3.7 Graph with one to many condition for three algorithms
Figure 3.8 Graph with many to many condition for three algorithms

Execution time of three shortest path algorithms using real road networks with different nodes is shown in the Tables 3.1 to 3.3 for different condition.

a) For one to one condition from the Table 3.1, if the number of nodes < 200, then the Execution times of algorithms are as follows:

Dijkstra < Genetic < ACO

For this state, Dijkstra algorithm performs better than other algorithms.

b) For one to one condition from the Table 3.1, if the number of nodes >= 200, then the Execution times of algorithms are as follows:

Genetic < Dijkstra < ACO.

For this state, Genetic algorithm performs better than other algorithms.
c) For one to all condition from the Table 3.2, if the number of nodes < 200, then the Execution times of algorithms are as follows:

Dijkstra < ACO < Genetic

For this state, Dijkstra algorithm performs better than other algorithms.

d) For one to all condition from the Table 3.2, if the number of nodes >= 200, then Genetic algorithm performs better than other algorithms.

e) For all to all condition from the Table 3.3, if the number of nodes < 200, then the Execution times of algorithms are as follows:

ACO < Genetic < Dijkstra.

For this state, ACO algorithm performs better than other algorithms.

f) For all to all condition from the Table 3.3, if the number of nodes >= 200, then the Execution times of algorithms are as follows:

Genetic < ACO < Dijkstra.

For this state, Genetic algorithm performs better than other algorithms.

The road data regarding the network are stored and maintained in Arcview GIS running on a Pentium IV workstation with 512MB RAM and 2400 MHz of CPU under the Windows NT environment. Twenty iterations have been carried out to calculate the results. The term one to one condition refers that the algorithm, which is used to create a graph into a linked list
representation. If the graph is an un-directed then the resultant list will be a singly linked list. In the case of one to all condition the graph is represented as a Tree. To represent a Tree in a graph, there are BFS and DFS methods. Both of them can yield the acyclic graph structure from a graph.

3.6 CONCLUSION

The computation of shortest paths is often a central task because shortest path distances are often needed as input for "higher level" models in many transportation analysis problems. With the advancement of GIS technology and the availability of high quality road network data, it is possible to conduct transportation analysis within a GIS environment. Consequently, these analysis tasks demand high performance shortest path algorithms that run fastest on real road networks. Although there has been considerable reported research related to the evaluation of the performance of shortest path algorithms, there has been no clear answer as to which algorithm or a set of algorithms runs fastest on real road networks in the literature. Evaluations of three shortest path algorithms using real road networks have been identified and listed below.

Execution time of the algorithms depends on the number of nodes and their relations in the real road networks. When the number of nodes and the constraint increases, Meta - Heuristic methods (Genetic algorithm) perform better than others. For the small number of nodes and for one to one and one to all conditions, Dijkstra’s Algorithm performs better than others. ACO algorithm performs better than others for the small number of nodes and for all to all condition. The Genetic Algorithm among execution time is the optimal solution for large number of nodes (greater than 200) with all the three conditions.
Figure 3.9 Madurai city map with roads
CHAPTER 4

MINIMAL GENERATION GAP–GENETIC ALGORITHM
FOR INTELLIGENT TRANSPORTATION SYSTEMS

4.1 INTRODUCTION

In present GIS route finding modules, heuristic algorithms have been used to carry out its search strategy. Due to the lack of global sampling in the feasible solution space, these algorithms have considerable possibility of being trapped into local optima. This chapter addresses the problem of selecting route to a given destination on an actual map under a static environment. The proposed solution uses a genetic algorithm (GA) with Minimal Generation Gap model for Intelligent Transportation system. A part of an arterial road is regarded as a virus. A population of viruses is generating in addition to a population of routes

4.1.1 Genetic Algorithm

Genetic algorithms (GAs) are search algorithms that work via the process of natural selection. They begin with a sample set of potential solutions which then evolves toward a set of more optimal solutions (Gen 1997).

Genetic algorithm is different from more normal optimization and search procedures in four ways
- GAs work with a coding of the parameter set, not the parameters themselves.
- GAs search from a population of points, not a single point.
- GAs use pay off (objective function) information, not derivatives or other auxiliary knowledge
- GAs use probabilistic transition rules, not deterministic rules.

4.1.2 Genetic Algorithm Operators

The basic operations of the genetic algorithm are simple and straight-forward:

Reproduction: The act of making a copy of a potential solution.

Crossover: The act of swapping gene values between two potential solutions, simulating the "mating" of the two solutions.

Mutation: The act of randomly altering the value of a gene in a potential solution.

4.1.3 Crossover

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent. Specific crossover made for a specific problem can improve performance of the genetic algorithm. Many crossover techniques exist for organisms which use different data structures to store themselves (Gen 1997).
Single point crossover

One crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent. Figure 4.1 shows single point crossover.

\[ 11001011 + 11011111 = 11001111 \]

Figure 4.1 Single point crossover

Two point crossover

Two crossover points are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent Figure 4.2 show two point crossover.

\[ 11001011 + 11011111 = 11011111 \]

Figure 4.2 Two point cross over

Partially Cycle Crossover (PCX)

Partially Cycle crossover is described with the help of a simple illustration. Consider two randomly selected parents P1 and P2 as shown in
Figure 4.3 that are solutions to a traveling salesman problem. The offspring C1 receives the first variable (representing city 9) from P1. We then choose the variable that map onto the same position in P2. Since city 9 is chosen from P1 which maps to city 1 in P2, we choose city 1 and place it into C1 in the same position as it appears in P1 (fourth gene), as shown in Figure 4.3. City 1 in P1 now maps to city 4 in P2, so we place city 4 in C1 in the same position it occupies in P1 (sixth gene). We continue this process once more and copy city 6 to the ninth gene of C1 from P1.

At this point, since city 6 in P1 maps to city 9 in P2, we should take city 9 and place it in C1, but this has already been done, so we have completed a cycle; which is where this operator gets its name. The missing cities in offspring C1 is filled from P2. Offspring C2 is created in the same way by starting with the first city of parent P2 (see Figure 4.3).

![Illustration of Partially Cycle crossover.](image-url)
4.1.4 Fitness Functions and Natural Selection

It is necessary to be able to evaluate how "good" a potential solution is relative to other potential solutions. The "fitness function" is responsible for performing this evaluation and returning a positive integer number, or "fitness value", that reflects how optimal the solution is: the higher the number, the better the solution (Heydar 2008).

The fitness values are then used in a process of natural selection to choose which potential solutions will continue on to the next generation, and which will die out. It should be noted, however, that natural selection process does not merely choose the top x number of solutions; the solutions are, instead, chosen statistically such that it is more likely that a solution with a higher fitness value will be chosen, but it is not guaranteed. This tends to correspond to the natural world.

A common metaphor for the selection process is that of a large roulette wheel. Remembering that fitness values are positive integers, imagine that each potential solution gets a number of slots on the wheel equal to its fitness value. Then the wheel is spun and the solution on which it stops is selected. Statistically speaking, solutions with a higher fitness value will have a greater chance of being selected since they occupy more slots on the wheel, but even solutions with just a single slot still have a chance.

In contrast, as genetic algorithms always have solutions in a population during a search, they can provide alternative routes using other solutions in the shortest time. A method has been proposed in this thesis for using a genetic algorithm to find the easiest-to-drive and quasi-shortest route to reach a destination within a given time. It can be used to produce and choose candidate routes that guarantee the meeting of deadlines and satisfy constraints regarding ease of driving (Huang 1996).
4.2 STUDY AREA

The selected area for research is Madurai city. Known as the Athens of the East, Madurai is one of the ancient historic cities in the world. It is the second largest city in the state of Tamilnadu in India with a population of about 18 lakhs. The archeological findings clearly suggest that the city is more than 2500 years old. The dating also matches with the sangam and post sangam age (3rd century B.C to 3rd century A.D) when the city was the commercial and cultural capital of south India. This town has survived as the capital city for more than twenty centuries for the dynasties that ruled the southern part of the India. This long unbroken history has rendered Madurai with a rich, diverse, ethnic, socio-cultural and religious setup, reflected in the settlement, which is based upon caste, religion and occupational systems.

Within each community structure, there is a characteristic topology in built form, spatial definition and settlement pattern, complete with religious and social infrastructure giving a unique character to the town. Moreover, the significance of this town lies in its planning which was the product of 16th century. This city is located at 9° 55’ 59’ N and 78° 7’ 0’ E. The topography of the city is approximately 100.58 meters (330 feet) above the sea level. The location of the study area is shown in Figure 4.4.

![Figure 4.4 Study area](image)
4.2.1 Need for Monitoring Madurai Transportation

The Government of India has recently (year 2005) launched Jawaharlal Nehru Urban Renewable Mission (JnURM). Ten cities in India have been included in the mission and Madurai is one among them. The mission focuses on providing basic services to urban poor and establishing necessary urban infrastructure. The key issues of Madurai are optimal utilization of the available natural resources and retainable physical environment, control of unplanned development in and around the religious sites, conservation of Madurai’s heritage through statutory listing and precincts plans. These issues are directly related to the growth of the city. Further Madurai is a major hub for tourists visiting down south. The tourist population reaches around 30 lakhs in a year. The floating population is around two lakhs per day. On several festival days there is a bulge and this has not been adequately catered for. Negligence and ignorance of these issues play a vital role that lead to the destruction of the historic fabric of the city.

Considering the issues, it has become mandatory for the city administration to monitor the traffic and control the unplanned growth of the city. The administration is determined to protect, promote and sustain the social of the city by settling these traffic issues with the help of technologies such as change detection using Madurai map. This thesis attempts to develop a novel optimization methodology to find the shortest path from an origin to a destination during traffic and other factors of Madurai city.

4.2.2 Integrating Geometry into the DBMS Data Model

The central idea for integrating geometric modeling into a DBMS data model is to represent “spatial objects” (in the sense of application objects such as river, country, city and others) by objects with at least one attribute of a spatial data type. Hence the DBMS data model must be extended by SDTs
at the level of atomic data types (such as integer, string and others), or better
be generally open for user-defined types. So far, most often the relational
model has been used as a basis but the approach can be used as well with any
other, e.g. object-oriented, data model (Francisco 2001). In the relational case
an object is represented by a tuple, so we can define example relations:

relation states (statename: STRING; area: REGION; stateid: INTEGER)
relation cities (cityname: STRING; center: POINT; ext: REGION; cityid: INTEGER)
relation road (rname: STRING; route: LINE)

The modeling of spatially embedded networks has not yet received
much attention in the research literature, although quite a bit of work has been
done for graphs in databases in general. Usually the assumption is that graphs
are represented by the given facilities of a data model. A disadvantage then is
that the graph structure is not visible to the user and can not be supported very
well in system implementation. In GraphDB model, this emphasizes an
explicit modeling of graphs together with a clean integration into a “standard”
object-oriented model. GraphDB offers object classes with inheritance, like
other OO models, but additionally distinguishes three kinds of object classes
called simple classes, link classes and path classes, whose elements
correspond to nodes, edges, and explicitly stored paths of a graph. For
example, in GraphDB use to model a highway network whose nodes are
highway junctions and exits with an associated POINT attribute, whose edges
are highway sections with an associated LINE attribute, and where highways
are explicitly stored paths, as follows:

class vertex = pos: POINT;
vertex class junction = name: STRING;
vertex class exit = nr: INTEGER;
link class section = route: LINE, no_lanes: INTEGER, top_speed: INTEGER
from vertex to vertex;
path class highway = name: STRING;
Here the junction and exit subclasses inherit the pos attribute from the vertex class. A highway is a path over a non-empty sequence of section edges.

4.3 FORMULATION OF THE PROBLEM

In present GIS route finding modules, heuristic algorithms have been used to carry out search strategies. Due to the lack of global sampling in the feasible solution space, these algorithms have considerable possibility of being trapped into local optima. This algorithm tries to address the selection of route to a given destination on an actual map under a static environment. The proposed solution uses genetic algorithm (GA).

The proposed solution using the GA is based on the following basic strategies:

- Any segment of an arterial road is regarded as a main virus. Populations of viruses are generated in addition to a population of routes.
- Only routes that include viruses are generated as the initial population. If two of the same routes are produced in the population, one is removed.
- Minimal Generation Gap (MGG) model is used for alternation of generations. Mutation does not occur in the proposed method.

During the crossover operation, if two routes give the same path then one of the paths is removed from the population. Mutation operation is not used in this method for find the final solution.
4.4 PROBLEM CONSTRAINTS

GA has been successfully applied to many NP-hard problems. When the problem has hard constraints, some candidate solutions are infeasible. This problem is generally addressed in three ways:

- Avoiding infeasible solutions during the search process.
- Penalizing infeasible solutions.
- Extracting feasible solutions after carrying out the search using populations that may contain infeasible candidate solutions.

During the cross over operation, new solution is obtained; the path does not exist between two nodes then assign high cost for the path. It will avoid of choosing the particular path from the population. If the solution is not feasible then remove the particular solution from the population during the search process. The feasible solution is obtained from the population. Sometime it may contain infeasible candidate solution.

The constraints on this particular problem can be grouped into the following two categories:

- Fixed constraints: are always enforced, irrespective of the composition of the current solution. For example, to select arterial and/or wide roads to decrease the number of turns and so on.
- Dynamic constraints: are enforced based on the composition of the current solution. For example, to select the roads where the traffic restrictions are not imposed and to select the roads not congested with traffic and so on. In this stage of the
development, our algorithm uses a directed graph representation that embeds some static constraints.

4.5 METHODOLOGY

Genetic algorithm is used along with Minimal Generation Gap model to find the shortest route to reach a destination within a given time. Variable-length chromosomes (routes) and their genes (intersections) have been used for encoding the problem. To generate the initial population, a virus is randomly selected from the population of viruses.

The fitness of a route is evaluated using the length of the route and the time required to travel along it. The best individual and the random one selection is based on roulette wheel technique. Both the routes from the origin to the virus and the virus to the destination are generated by using an RTA* algorithm. A set of new generation is produced by performing single point crossover on a set of selected routes. Another set of new generation is also produced by performing Partial cycle crossover on the set of same selected routes.

4.5.1 Pseudo Code

procedure GA_MGG
Input map and map database;
Input origin and destination;
Initialize a population of viruses;
Initialize a population of individuals (routes);
Set Generation= 1;
for generation = 1 to Number of Generations (repeat until meeting deadline)
begin
  calculate fitness values of individuals;
  select two individuals at random;
  single-point crossover;
  MGG;
end;
end procedure;

4.5.2 Coding and Fitness

Repairing infeasible candidate solutions may incur significant computational expense, but omitting them from the search process may leave the search space disconnected, preventing satisfactory optima from being reached. Computationally efficient search can be carried out by choosing a representation that implicitly excludes infeasible candidate solutions, without hindering the search process from visiting different parts of the search space.

Variable-length chromosomes (routes) and their genes (intersections) have been used for encoding the problem. A chromosome of the proposed GA consists of sequences of positive integers that represent the IDs of nodes through which a routing path passes. In the other hand, each route regards from an origin to a destination as an individual for the GA and expresses it as a sequence of intersections with directions.

The fitness of a route is evaluated using the length of the route, the time required for a car to travel along it and the ease of driving. Penalties for violations are defined for each constraint. An amenity can be calculated from a penalty if the route violates constraints. The route regards with the lowest number of penalties as the best one for drivers.
4.5.3 Population Initialization

In general, there are two issues to be considered for population initialization of GA: the initial population size and the procedure to initialize population. It was felt that the population size needed to increase exponentially with the complexity of the problem (i.e., the length of the chromosome) in order to generate good solutions. Recent studies have shown, however, that satisfactory results can be obtained with a much smaller population size. To summarize, a large population is quite useful, but it demands excessive costs in terms of both memory and time. As would be expected, deciding adequate population size is crucial for efficiency.

Secondly, there are two ways to generate the initial population: heuristic initialization and random initialization. In this method, we select viruses within the rectangle on the map with a diagonal that links the origin and the destination and we name them ‘main-viruses’. To generate the initial population, a virus is randomly selected from the population of viruses. Next, both the routes from the origin to the virus and the route from the virus to the destination are generated by using an RTA* algorithm. Finally, the route that combines these two routes with the virus becomes an individual. This procedure is repeated for all main viruses.

In the proposed scheme, Minimal Generation Gap (MGG) model has been used for alternation of generations. In this model, two routes are replaced by crossover with each new generation and so, two evaluations of fitness are required between generations. Mutation is not used in this method. MGG model is shown in the Figure 4.5.
Figure 4.5 Minimal Generation Gap model

The flows for the process are:

- Take two individuals at random from the population for use as parents.

- Apply simple crossover to the parents to produce offspring, where the crossover site is at an intersection, which they have in common. If there is no such common intersection, then go to the previous step.

- From the parent and their children, select the best individual (elite route) and the random one using the roulette wheel technique with the original parents are replaced.

In this model, original parents are two individuals and replacing individuals are also two. Then, the elite individual is left for progress in solving a problem, and a random individual is also left for maintaining diversity of population. It should also be noted that this model only uses fitness values of each individual. It makes the computational load of the operation light.
4.6 RESULTS AND DISCUSSION

The shortest path problem can be solved on a given transport network by finding the optimum path from one or some source node to all other nodes or to a subset of nodes. The possible optimum paths in each generation correspond to the total population. The parameter values to be found are in the form of positive integer numbers, so that the chromosome populations are formed using positive integer number encoding method. By introducing partial solutions to problems, that are considered to be viruses, a population of viruses and populations of individuals were created.

Crossover operation is used to manipulate a pair of chromosomes in order to produce one or more child chromosomes. Parents are selected according to their fitness values. Better the fitness values of the chromosomes, more the chances are for them to get selected. Chromosomes are selected based on its fitness values through the roulette wheel technique. If the fitness of the best chromosome is 90% of the entire roulette wheel then the other chromosomes will have very few chances to get selected. The worst will have the fitness value 1, the second worst 2 and so on whereas; the best will have the fitness value N (number of chromosomes in population). Thus chromosomes are evaluated based on their fitness values.

In the MGG-GA, the selection of chromosomes is done on the current population based on the fitness values – chromosomes with higher fitness values are more likely to get selected than those with low fitness values. Chromosomes are selected based on their fitness values through the roulette wheel technique. Selected chromosomes are then included in the next generation of population. After that the population undergoes the crossover (also called recombination) genetic operation, which selects chromosomes from the population to produce offspring. Using selection methods, two
parent chromosomes are chosen for crossover operation. Using single-point and partial cycle crossover, parts of the gene string in each parent chromosomes are swapped to produce two new offsprings, which are included in the next generation of population. The process is repeated for 25 times. The process of selection and crossover are then repeated on the surviving population, until a terminating criteria (maximum number of generation) is reached. The best individual chromosome, based on the elite operation is selected. In this operation, the best chromosome in every generation is copied directly to the next generation. This operation is used to ensure that the best chromosome in each generation has at least the same fitness value as that of the best chromosome in the previous generation. The final population is usually considered as the optimal solution to the problem.

The experiments had been conducted for a group of 30, 60, 90, 120 and 150 nodes in the areas of interest. The optimized path using Simple GA is obtained and the proposed algorithm with MGG- GA is used to find the optimized path for the same set of data. It takes less memory resources, and also, it presents more flexibility to changes in the restrictions applied to the road segments.

The graph has been plotted with the comparison of the optimized path obtained by both the algorithms in addition to the shortest route between the cities in various cases. Comparison of the three algorithms with single point crossover and PCX cross over using real road network is shown in the Figures 4.6 and 4.7. MGG GA and GA have been executed for 25 iterations to calculate the results. Here 1000 generations are carried out for single iteration. The standard deviation obtained for MGG GA is 1.65 and for GA is 1.87.
Figure 4.6 Comparison of DA, GA and MGG GA using PCX crossover

Figure 4.7 Comparison of DA, GA and MGG GA using Single point crossover

Based on the number of nodes from the Figure 4.6, the minimum distance is calculated for the path using PCX crossover technique for all the three methods.
a) For real road network, if the number of nodes = 30, then the distance for algorithms are as follows:

MGG GA < Shortest path algorithm < GA

In this case, MGG GA algorithm performs better than other algorithms.

b) For real road network, if the number of nodes = 60, then the distance for algorithms are as follows:

MGG GA < Shortest path algorithm < GA

In this case, MGG GA algorithm performs better than other algorithms.

c) For real road network, if the number of nodes = 90, then the distance for algorithms are as follows:

MGG GA < Shortest path algorithm < GA

In this case, MGG GA algorithm performs better than other algorithms.

d) For real road network, if the number of nodes = 120, then the distance for algorithms are as follows:

MGG GA < Shortest path algorithm < GA

In this case, MGG GA algorithm performs better than other algorithms.

e) For real road network, if the number of nodes = 150, then the distance for algorithms are as follows:

MGG GA < Shortest path algorithm < GA
In this case, MGG GA algorithm performs better than other algorithms.

Based on the number of nodes from the Figure 4.7, the minimum distance is calculated for the path using Single point crossover technique for all the methods.

a) For real road network, if the number of nodes = 30, then the distance for algorithms are as follows:

   MGG GA < Shortest path algorithm < GA

   In this case, MGG GA algorithm performs better than other algorithms.

b) For real road network, if the number of nodes = 60, then the distance for algorithms are as follows:

   Shortest path algorithm < MGG GA < GA

   In this case, Shortest path algorithm performs better than other algorithms.

c) For real road network, if the number of nodes = 90, then the distance for algorithms are as follows:

   MGG GA < Shortest path algorithm < GA

   In this case, MGG GA algorithm performs better than other algorithms.

d) For real road network, if the number of nodes = 120, then the distance for algorithms are as follows:
MGGA < Shortest path algorithm < GA

In this case, MGGA algorithm performs better than other algorithms.

e) For real road network, if the number of nodes = 150, then the distance for algorithms are as follows:

MGGA < Shortest path algorithm < GA

In this case, MGGA algorithm performs better than other algorithms.

4.7 CONCLUSION

As a high efficient search strategy for global optimization, genetic algorithm demonstrates favorable performance on solving the combinatorial optimization problems. The best route selection problem in network analysis can be solved with genetic algorithm through efficient encoding, selection of fitness function and various genetic operations. Crossover is identified as the most significant operation to the final solution. The experiment shows that the designed implementation method is effective in terms of computation distance and complexity. The algorithms of GA and Dijkstra’s shortest path algorithm have no difference in the iterations. So the distance is not changed into them. But MGGA algorithm yields the minimized as well as accurate solutions of the iterations in the PCX. The ratio of the distance and nodes per iteration also decreased. This denotes that the memory allocations for the resultant are reduced. From the SPX and the PCX, the iterations of the MGGA algorithm is presenting us the minimum solution than the other feasible solutions. This also gives the optimal solution for the given transportation problem using Minimal Generation Gap model.