CHAPTER 3

THE APPLICATIONS OF METAHEURISTICS IN POINT TO MULTIPOINT ROUTING PROBLEM

3.1 Introduction

The field of telecommunications is one of the rapidly evolving areas of all disciplines as the growth of data flowing in a telecommunications network has increased drastically. Data flow through the network can be seen in daily business such as sending messages, emails, paying bills, banking services, airline bookings and even shopping which all can be done using network services. Designing the network to be more efficient is necessary as the activity demand for it is getting higher daily. The efficiency of the telecommunications network can be measured by some criteria such as blocking, delay (waiting system), congestion and traffic.

Blocking can occur if all the devices are occupied when the demand of services is initiated. A call is said to be delayed if the waiting time is longer than some specified length of time. Congestion is said to happen in a situation when the subscriber is unable to obtain a connection immediately. It is also known as time congestion or call congestion due to the unavailability of a free server. Traffic is measured based on the occupancy of server(s) in the network.

Before the telecommunications network is defined, it is best to define the two words of telecommunication and network separately. Telecommunication is a term encompassing the transmissions or receptions of signals, images, or information over a distance. *Tele* is a Greek word that means *far* and network is defined as interconnected group of nodes (Held, 1995). In general, a network comprises of transmitters, receivers, or other devices that support communications.

The telecommunications network can be seen as a collection of terminals, links and nodes which connect together to enable telecommunication between the users of the terminals. The most important function needed of a telecommunications network is to be able to carry audio, visual, and data communications. Understanding the telecommunications network would give a picture of how data flow and also how data interchange is carried out.

Data transfer or information transfer must be established and maintained at an acceptable level based on some criteria such as speed of connection, speed of information transfer, freedom from error and also cost (Horak, 2007). The data information can be preserved in the original form, or it can be altered during transmission either by compressing for efficiency or encrypting for security reason.

The simplest example of a telecommunications network can be seen in telephone services. It begins with a call at one point and the call's signal will be routed through a node or multiple nodes until it reaches its destination. The process only takes a few seconds to be carried out. Usually, telecommunications network such as telephone line service is often referred to as circuit switching, while data routing is known as packet switching. In packet switching, the network is able to store messages (data) and pass them to the destination(s) when available. Specifically, the data source is divided into packets (bit sizes) and sent over the network. The data (for example messages, video or audio) are also divided into packets and sent over the network depending on the traffic. Each packet (after the divide) does not necessarily use the same route to reach the destination; it can also use a different route. Packets are then reconnected again after the whole packets arrive.

A network that transfers a stream of data (data flow) from a source to destination(s) will be assigned a route of sequence of links that connects the source and destination(s). This network is responsible in assigning a route to a sequence of links

that connects the source and destination. The network should be capable in allocating a portion of its capacity along the route to be used, where the decision to route is done by routers (switches). This process is called switching (Gananasivam, 2006).

Data are any materials which are represented in a formalized manner so that they can be stored, manipulated, and transmitted by the machine, while routing can be defined as the assignment of the communications path by which a message or telephone call can reach its destination (Held, 1995). Hence, data routing is a process of moving or transferring data (packet of data) from source to destination.

One of the problems in data routing is Message Scheduling Problem (MSP) or also known as request scheduling. MSP is defined as a process of scheduling a set of requests such that the entire route is optimal. A set of requests may include two or more requests. Request here can be pictured as a call or a message. In this problem, each request has a single source and multiple destinations. Specifically each request may have different sources and destinations. Each request is characterized as one source node, one or multiple destinations and a specific request capacity. Request capacity is the capacity (sometimes referred to as weight) of the message or call.

3.2 Literature Review

The telecommunications network is modeled as a graph that has a capacity and cost assigned to its edges, as shown in Figure 3.1. The network consists of 8 nodes and 11 edges. MSP is often solved by treating the problems as a Point to Point Routing Problem (PPRP) and the PPRP is known to be NP-complete. In PPRP there is only one single source and one single destination. However, in the telecommunications practice, a message is often sent to multiple destinations. In PPRP, if there are several destination nodes, the same message will be sent many times depending on the number of destinations. This increases the cost since the same message is sent many times and frequently it shares the same path/link, which is neither efficient nor capacity effective.

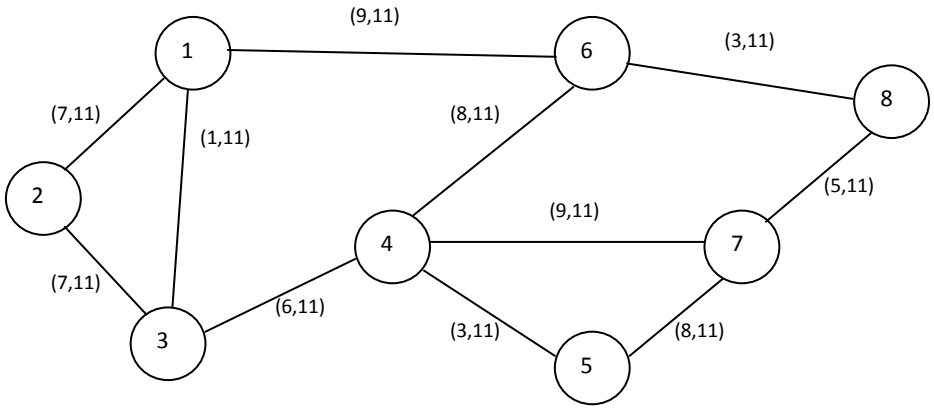

Figure 3.1: An example of Telecommunications Network with the association (cost, capacity) for each edge.

PPRP has been studied comprehensively by Cox *et al.*(1991) where call requests for circuit switching in a telecommunication network arrive randomly at any of the nodes in the network. A scheduler is described for the network, where it assigns a single dedicated path through the network from the source node to the destination node during the entire requested time interval. The scheduler must also consider the capacities of the request that might concurrently use the link. Each call request is specified by six attributes: source node, destination node, requested start time, requested duration, bandwidth requirement or capacity, and priority class. The schedule is feasible if the sum of capacities of the requests assigned to a link at any given time is less than the link's capacity. If no particular schedule can accommodate a call request, the request is said to be blocked and infeasible.

The cost of using a network is calculated as the sum of the costs of each path used. For the infeasible schedule, a penalty is incurred. The scheduler calculates the cost of each feasible schedule. The cost is the sum of the net cost of successfully carrying scheduled requests through their assigned paths. For an infeasible schedule, one includes penalties associated with requests that are blocked. The scheduler's task is to choose a set of paths for all call requests that minimizes the total cost while taking into account the time varying nature of the call requests. Cox *et al.* (1991) addressed this

problem through the use of a statistical approach that makes rapid initial assignments of incoming call requests to paths, based upon statistical information about the current network traffic. The authors then used evolutionary programming techniques to find alternative routing assignments that were nearly optimal than the previous ones or that were able to accommodate additional requests.

Christensen *et al.* (1997) proposed solving PPRP by treating the set of requests as a whole, instead of treating it as a one to one request. This assumption made the problem no longer a PPRP, but as a Point to Multipoint Routing Problem (PMRP). PMRP attempts to find an optimal route that routes a set of requests from one source node (point) to several destination nodes (multipoint). This is achieved by duplicating (copying) the message when it reaches the intermediate node(s); that is a node that connects two or more destination nodes. This is determined by finding the minimum cost Steiner tree. In Christensen *et al.*, minimum cost Steiner tree is found by using a heuristics algorithm. In Christensen *et al.*, the research was undertaken for a telecommunication company, LDDS WorldCom which is a locally based internationally prominent telecommunications vendor. It is very common for a telecommunications company to transmit the same signal to many different destinations.

Zhu *et al.* (1998) extended the work of Christensen *et al.* (1997) in which the study considered dealing with subsets of requests, when dealing with the whole requests was infeasible. Using the same metaheuristics method, the order based GA is evaluated with some modifications in the objective function where only $k, k \epsilon N$number of requests, and$N$ is the total number of requests. This is done by assuming that all requests yield the same profit and the main aim is to maximize the *k*. PMRP here is solved by assuming that the request could be split in at most two; resulting in two new features which find paths for sub-problems and capacity (bandwidth) of each corresponding path. Consequently, the chromosome representation is divided into two

segments known as *request segment* and *percentage segment*, where *request segment* is simply permutations of partial requests and *percentage segment* is determining the transmission of the partial requests. Galiasso and Wainwright adopted the Partially Mapped Crossover (PMX) and double point crossover as their crossover operators. Galiasso and Wainwright (2001) improved the work of Zhu *et al.* (1998) and they were able to solve 19 request problems out of the 20 requests, compared to Zhu *et al.* with only 18 requests.

Kampstra *et al.* (2006) provided a comprehensive overview where the authors classified telecommunications network design into node locations, topology design, tree design, routing, restoration, network dimensioning, admission control problems, and frequency assignment and wavelength allocation. PMRP is classified under the routing classification along with adaptive routing, shortest path routing, multicast routing and intelligent agents.

3.3 Point to Multipoint Routing Problem (PMRP)

As described earlier PMRP is similar to PPRP except that instead of routing to a single destination node, PMRP determines the route to a multiple destination nodes for each of the requests. The problem is to find an optimal routing to send a message from a single source to multiple destinations efficiently.

As mentioned earlier, a request is characterized by the source node, requested start time, requested duration, bandwidth requirement or capacity and priority class. Similar assumptions made in (Christensen *et al.*, 1997) are adopted in this research in which a special case of PMRP is considered where all requests are assumed to have the same start time, duration, and priority.

3.4 Solving PMRP

PMRP is solved by considering the single source and multiple destinations as a whole. The first step of the algorithm is to determine the minimum Steiner tree that will provide the possible connections of all the nodes available including the source and destination nodes. The Kou Markowsky Berman Method (KMB) algorithm is employed (Kou *et al.*, 1981) to determine a near optimal minimum spanning tree. Other methods to solve the minimum spanning tree are also available, for example ZEL (Zelikovsky, 1993) and IZEL or *Iterated Zelikovsky*. However, KMB provides the lowest order of computational complexity of $O(E \log V)$. Details of the KMB algorithm are given in Section 3.4.1. The KMB algorithm requires solving the shortest path problem and determining the minimum spanning tree. An established algorithm of Dijkstra (Dijkstra, 1959) is used to find the shortest path and Kruskal's algorithm (Kruskal, 1956) is employed to find the minimum spanning tree. It is noted that in Christensen *et al.*, (1997) the authors adopted Prim's algorithm instead. However, in this research Kruskal's algorithm is adopted due to better time complexity as Kruskal's algorithm has a time complexity of $O(E \log V)$ as opposed to Prim's algorithm of $O(V^2)$.The details of the Dijkstra's and Kruskal's algorithms have been described in Sections 2.4 and 2.5.

3.4.1 Kou Markowsky Berman Method

The algorithm of Kou, Markowsky and Berman (KMB) (Kou *et al.*, 1981) which is used to determine the minimum Steiner tree is outlined. The algorithm is embedded both in the VNS and GA. It is noted that the algorithm requires finding the minimum spanning tree using Kruskal's algorithm and solving the shortest path problem using Djiktra's algorithm. The general algorithm of KMB is given as follows:

INPUT: an undirected distance graph $G = (V, E, d)$ and a set of Steiner points $S \subseteq V$. d is equals to the distance of the shortest path.

OUTPUT: a Steiner tree, T_H for G and S

Step 1 *Construct the complete undirected distance graph* $G_1 = (V_1, E_1, d_1)$ *from G and S.*

Step 2 *Find the minimal spanning tree,* T_1*, of* G_1*. (If there are several minimal spanning trees, pick an arbitrary one).*

Step 3 *Construct the sub-graph,* G_s*, of G by replacing each edge in* T_1 *by its corresponding shortest path in G. (If there are several shortest paths, pick an arbitrary one).*

Step 4 *Find the minimal spanning tree,* T_s *of* G_s*. (If there are several minimal spanning trees, pick an arbitrary one).*

Step 5 *Construct a Steiner tree,* T_H*, from* T_s *by deleting edges in* T_s*, if necessary so that all leaves in the* T_H *are Steiner points.*

3.4.2 An illustrative example in solving PMRP

Consider a network consisting of 6 nodes labeled [A B C D E F] with a maximum capacity of 11 units is given in Figure 3.2a. Assume that the network is to schedule 2 requests from the source nodes to their respective destinations and the details on the source nodes, their destinations and their respective capacities are given in Table 3.1.

Table 3.1: Dataset for routing.

Request	Source	Destination	Capacity
1	A	D,E	5
2	B	A,D,F	6

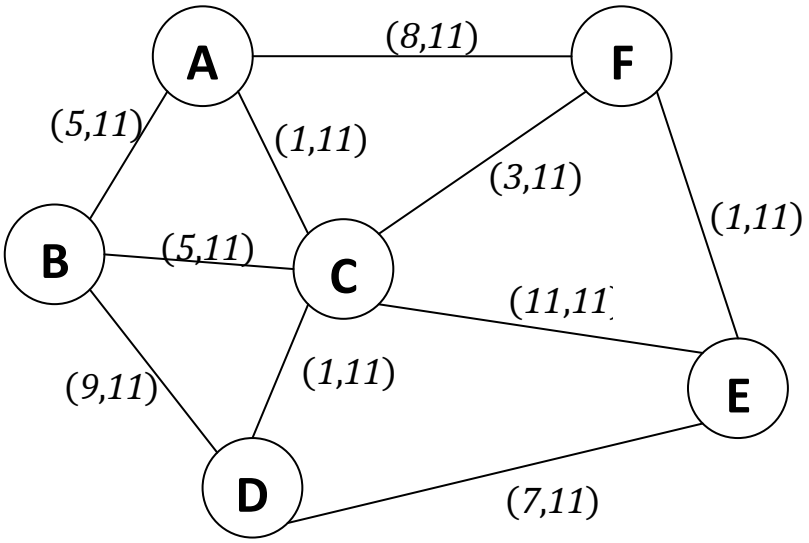

Figure 3.2a: Example of telecommunications network with 6 nodes.

Figure 3.2a shows the network with the cost of using the edge and the maximum capacity of the edge. The nodes include the source node(s), destinations node(s), and intermediate node(s). Assuming that the requests are to be routed in the following order of [1 2], which means that Request 1 is to be considered first then followed by Request 2.

Request 1 with a capacity of 5 units is to be sent from source node A to destination nodes D and E. The shortest path from source node A to destination node D is $A \rightarrow C \rightarrow D$ with a minimum cost 10 $(1 \times 5 + 1 \times 5)$. The shortest path from source node A to destination node E is $A \rightarrow C \rightarrow F \rightarrow E$ with a minimum cost of 25. Figure 3.2b shows the shortest path between source node A (red) to destination node D (orange) represented by the dashed-dotted lines with A→C→D and the cost is 10. The calculation is [capacity request×(path(A,C)+path(C,D))]=[5 × (1+1)]. The remaining capacities on respective edges are 6 units each. Figure 3.2c gives the shortest path (given in dashed-dotted lines) for $A \rightarrow E$ is to route the request A→C→F→E and the cost of routing is 25.

Figure 3.2d shows the final routing for Request 1 with a capacity 5 from source node A to destination nodes D and E represented by the dotted line. Steiner tree for routing Request 1 is shown by the dotted lines with the routing cost of 30 units. Figure 3.2e shows the routing of Request 2. Note that the request is duplicated at an intermediate node C (green) and sent to destination nodes F and D respectively. The cost of the entire Request 2 routing is 84 units only. So the overall total cost of routing Table 3.1 with order [1 2] is 114 units.

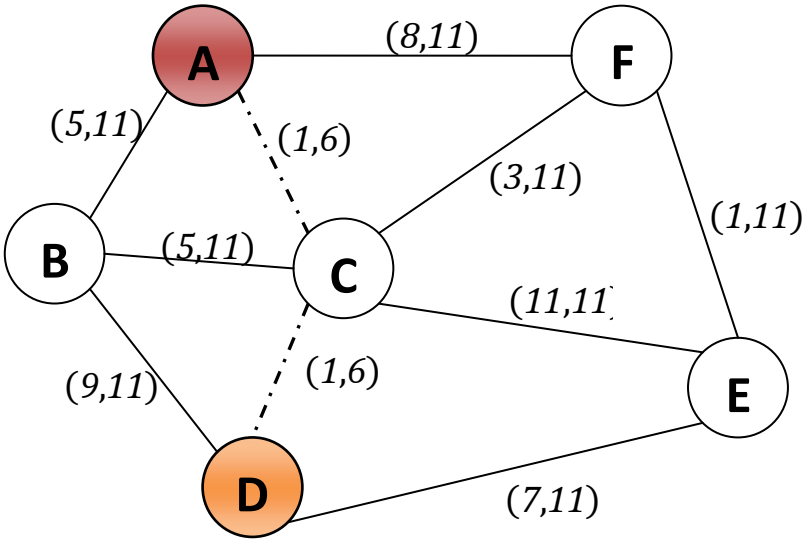

Figure 3.2b: Shortest path from source node A (red) to destination node D (orange) is represented in dashed dotted lines.

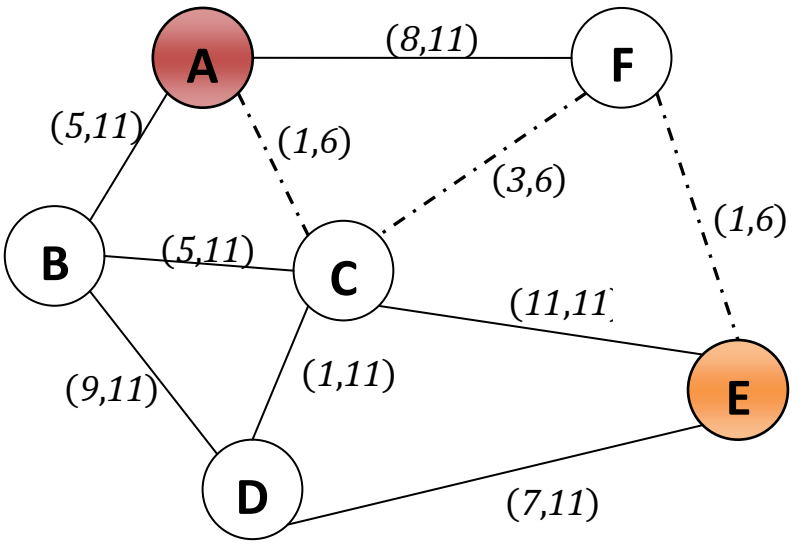

Figure 3.2c: Shortest path from A to E is represented by the dashed-dotted lines.

It is noted that Request 1 uses the same edge that is (A, C) and the message is duplicated at an intermediate node C to be sent to destination nodes D and E respectively with a total cost of 30 units (as shown if Figure 3.2d). However, if the PPRP case is considered using the same routes (Figures 3.2b and 3.2c) it will traverse an additional 5 units cost as the edge (A, C) will be used twice. The remaining capacity of edge (A, C) is reduced to 1 while the remaining capacities on other edges are not affected and they remain the same.

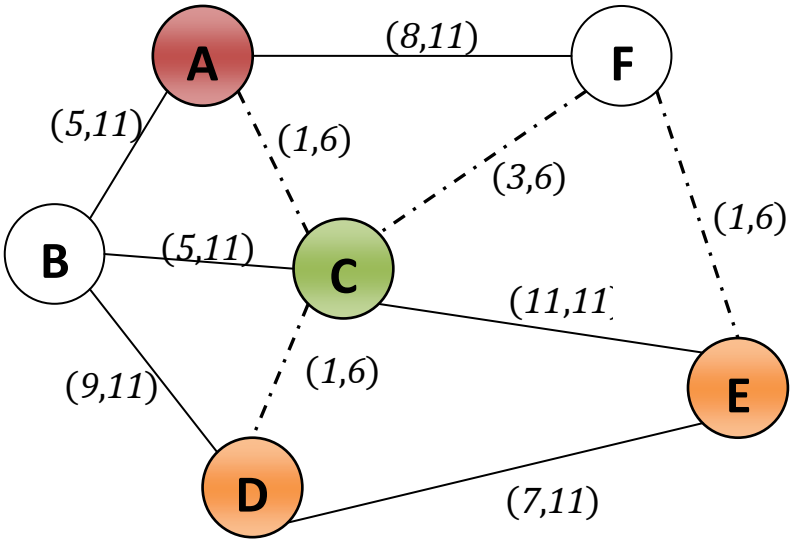

Figure 3.2d: The final route for routing request 1.

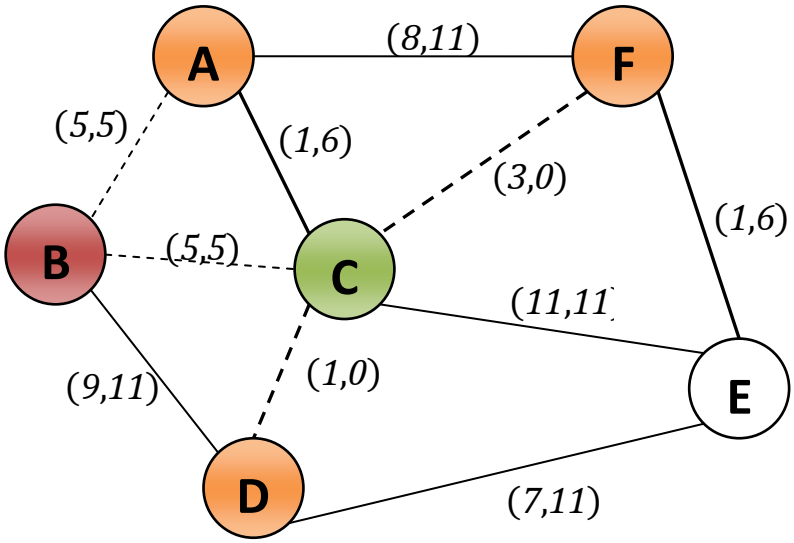

Figure 3.2e: Dotted line represents Steiner tree of Request 2 with a total cost of 84 units.

3.5 Implementation of Metaheuristics for PMRP

Two metaheuristics methods are proposed based on GA and VNS to solve PMRP. Both of these methods embed the algorithm of KMB algorithm (Kou *et.al*, 1981) which will be used to find the minimum spanning tree for each request.

3.5.1 Genetic Algorithm for PMRP

The same approach as in Christensen *et al.'s* (1997) paper is adopted, where the solution is represented as an order where a chromosome consists of a permutation of integers and each integer represents a request number. The requests are routed according to the order they appear in the chromosome. To route a given request, GA will first inspect the edges with a capacity of less than the capacity of the request; for edges that do not have enough capacity, the cost for that particular edge is made sufficiently high (artificial cost) temporarily so that it will not be considered in the route. Then, the (near) optimal Steiner tree is determined for each request. The capacities of all the paths are then updated. The steps are repeated until all requests have been routed in the order found in the chromosome. It is noted that the minimum cost Steiner tree is determined by the heuristics method of Kou, Markowsky and Berman's algorithm (KMB) (Kou *et al.*, 1981). The KMB determines the minimum cost tree that spans all the destination nodes including the source node. Dijkstra's algorithm is adopted (Dijkstra, 1959) to find the shortest path in the KMB algorithm. It should be noted that Kruskal's algorithm is chosen instead of Prim's algorithm as adopted by Christensen *et al.* (1997) due to better time complexity where Kruskal's algorithm has a time complexity of $O(E \log V)$ as opposed to Prim's algorithm of $O(V^2)$.

Representation

The representation used for GA is the real representation using integer numbers. Each number represents the request's number. For example in Figure 3.3 , there is chromosome [1 2 3 4]; thus here request number 1 is routed first, followed by request numbers 2 and 3, and then request number 4.

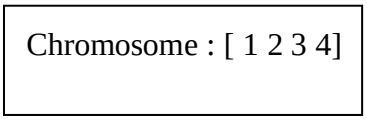

Figure 3.3: Example of coded chromosomes.

Another example is[4 1 3 2], where in the example request number 4 will be routed first, followed by request numbers 1 and 3 and finally request number 2. Note that each request has its own message capacity.

Crossover Operator

Enhanced Edge Recombination Operators (EERO)

Enhanced Edge Recombination Operators (EERO) basically emphasizes the edges. This was an improvement of the Edge Recombination Operator (ERO) introduced by Grefenstette (1987). It was then enhanced by Starkweather *et al.* (1991). Basically the inspiration has fastened the next selection by marking the common subsequence.

The basic ER crossover has been discussed in Chapter 2. However, it is also summarized here for better understanding in the later examples. The steps of the ERO are given as follows:

Step 1 *Randomly select a city to be the current city of the offspring*.

Step 2 *Select four edges (two from each parent) incident to the current city c.*

Step 3 *Define a probability distribution over selected edges based on their cost. The probability for the edge associated with a previously visited city is 0.*

Step 4 *Select an edge. If at least one edge has non-zero probability, selection is based on the above distribution; otherwise, selection is random (from unvisited cities).*

Step 5 *The city on 'the other end' of the selected edge becomes the current city.*

Step 6 *If the tour is complete, stop; otherwise, go to step 2.*

Example of EERO

An example of the EERO is given as follows, where a set of 5 request orders are used to describe it.

Parent 1=[1 2 3 4 5]

Parent 2=[3 2 4 1 5]

The edge request list is listed as:

Request 1: edge to other requests: 2, -5, 3

Request 2: edge to other requests: 1, -3, 4

Request 3: edge to other requests: -2, 4, 5

Request 4: edge to other requests: 3, 5, 1, 2

Request 5: edge to other requests: 4, -1, 3

The edge request list is listed based on the edge that connects both parents. For example in the Request 1 list, there is 2, -5 and 3 where it can be seen that request 1 is connected to request 2 and 5 in parent1, then connected to 4 and 5 in parent 2. Here it is noted that as request 5 is connected to request 1 for both parents, it is emphasized by

marking 5 with the symbol '-'. The same goes to edge list for request 2. As seen in parent 1, request 2 is also connected to requests 1 and 3, and in parent 2, request 2 is connected to requests 3 and 4. As 3 appear twice, so 3 is indicated as '-3'.

From there, now the new request (offspring) can be found. It is randomly started with request 1. So,

New Request: [1 ☐ ☐ ☐ ☐]

To find the next request number, it will be chosen from the list. The edge list for request 1 is 2, -5, and 3. As request 5 is marked with the symbol '-', it will be chosen as the next request in the new offspring. The new offspring is indicated as follows:

New Request: [1 5 ☐ ☐ ☐]

Then the next offspring has to be chosen, so the edge list of request 5 is seen. It can be seen that request 5 is connected to request 4, -1 and 3. By right 1 as the next request should be chosen, but it can be seen that 1 is already in the new offspring, so there are the alternatives of 3 and 4. To choose between 3 and 4, their edge lists have to be checked. It can be seen that the list for request 4 is longer than that of request 3; thus request 3 is chosen as the next request in the new offspring. The new offspring is shown as follows:

New Request: [1 5 2 ☐ ☐]

Following the same steps discussed above, it can finally be shown that the new request (offspring) is as follows:

New Request: [1 5 2 3 4]

Mutation Operator

The mutation operator used in this research is inversion. This is simply finding two random positions to be inversed. An example and illustration of this inversion operator is discussed in Chapter 2. It is noted that Christensen *et al.* (1997) employed the Roulette Wheel Selection (RWS) in the selection process and the Partially Mapped Crossover (PMX) as a crossover operator.

The GA algorithm can be summarized as follows:

Step1 *Generate an initial population, where chromosomes are represented as permutations of integers.*

Step 2 *Evaluate all the chromosomes in the initial population (generation 0), by finding the near optimal Steiner tree.*

Step 3 *Record the best chromosome from generation 0.*

Step 4 *Until the maximum number of generations or the stopping condition is attained:*

4.1 *Rank all the chromosomes according to their objective values.*

4.2 *Selection: The resulting chromosomes after ranking will go through the selection operator, Stochastic Universal Sampling (SUS). SUS will select chromosomes for reproduction (crossover) according to their fitness value in the current generation.*

4.3 *Crossover: The resulting chromosomes from Selection will go through this step. Enhanced Edge Recombination Operators (EERO) is used as the crossover operator, where EERO emphasizes on the order of requests.*

4.4 *Mutation: The resulting chromosomes (offspring) from Crossover then will go through this process. Inversion is used as a mutation*

operator, where it is simply invert the part of the chromosome between two randomly selected positions.

4.5 *Reinsertion*: *Reinsert the offspring into the new population, selecting the fittest individual to replace the old population.*

Step 5 *Repeat Step 4.*

3.5.2 Variable Neighborhood Search for PMRP

Similarly to GA, a solution set in VNS is also represented as permutation of integers. As in GA, requests are routed according to the order of the number represented in the solution. As mentioned earlier the important factor is in defining the neighbourhood to be explored. In this study, the neighbourhood k , N_k is defined as a distance function, that is the cardinality of the symmetric difference between any two solutions V_1 and V_2 written as $\rho(V_1, V_2) = |V_1 \setminus V_2|$ or $\rho = |V_1 \Delta V_2|$ where Δ denotes the symmetric difference operator.

Four different local search methods that emphasize the order which are swap, inversion, *or*-opt and *restricted or*-opt are introduced. Swap, Insertion and *or*-opt algorithm had been explained and illustrated in Chapter 2. Restricted *or*-opt algorithm is explained here.

Restricted or-opt

Basically restricted *o*r-opt was introduced to limit the number of removing and reinserting vertices in *Step 2* of or-opt. This is done by introducing a block. Instead of searching all the insertion, we limit the number of search allowed. In this thesis the number of block is set at 10.The algorithm of the restricted *o*r-opt can be described as follows:

Step 1 *Consider an initial tour and set* $t = 1, s = 3$ *and Block=10. Set counter=1;*

Step 2 *Remove from the tour a chain of s consecutive vertices, starting with the vertex in position* t*, and tentatively insert it between all the remaining pairs of consecutive vertices on the tour.*

(i) If the tentative insertion decreases the cost of tour, implement it immediately, thus defining a new tour. Set $t = 1$*, and repeat Step 2.*

(ii) In no tentative insertion decreases the cost of tour, set $t = t + 1$*. If* $t = n + 1$ *then proceed to Step 3, otherwise repeat Step 2.*

Evaluate counter=counter+1; If counter=Block; set $t = n + 1$*. Proceed to Step3.*

Step 3 *Set* $t = 1$ *and* $s = s - 1$*. If* $s > 0$ *go to Step 2, or else Stop.*

The algorithm of VNSPMRP can formally be represented as follows:

Step 1 *Define* k_{max}*, the maximum number of neighbourhood structures to be explored.*

Step 2 *Randomly generate a solution x where the solution is represented as permutation of integers, where each integer represents a request. Each request is characterized by its source node and multiple destination nodes, along with its capacity (request capacity or weight of the message).*

Step 3 *Repeat the following until the stopping condition is met.*

3.1 *Set* $k = 1$.

3.2 *Until* $k = k_{max}$

3.2.1 *Shaking. Select randomly an individual,* x' *from the* k^{th} *neighbourhood of* x $(x' \in N_k(x))$*. Evaluate its corresponding*

minimum Steiner Tree by using the Kou Markowsky Berman (KMB) method for each of the requests.

3.2.2 *Local Search. For* $i = 1$ *until* $maximum\ number\ of\ iterations$. *Apply swap, inversion, or-opt or restricted or-opt separately as the local search. Note that these methods emphasize the request order as these local search used targets the order. Evaluate the corresponding Steiner tree and the total cost. Denote that the best individual found as* x''.

3.2.3 *Move Or Not.*

If $cost\ (x'')$ *is better than* $cost\ (x)$.

Denote x'' *as the new* x, $(x \leftarrow x'')$. *And set* $k = 1$.

Else *Increase k by 1,* $(k = k + 1)$.

Output is the best order with its corresponding minimum Steiner tree.

3.6 Results and Discussion

Both programmes, GA and VNS were written in MATLAB 7.5 and were run on the computer using the Windows platform with Intel Core 2 Duo processor with 4Ghz speed. Applying GA for PMRP is very straight forward. The population size is 20 for small and medium case; and 50 for a large case. The Probability for the crossover and mutation is 0.7 and 0.1 respectively. The maximum numbers of generations are 20, 30 and 50 for small, medium and large respectively.

Both GA and VNS were run on the data set given as an example in Christensen *et al.* (1997). This data set contained four messages and Table 3.2 tabulates the source node, destination nodes and the capacity of each request. The graphical representation of the network is presented in Figure 3.4. All paths (edges) have the same capacity, which is 8 and the cost assigned to each edge is the same as in the main example in Christensen *et al.* (1997).

The results obtained are given in Table 3.3. The results clearly show that these algorithms produced superior results as compared to those reported in Christensen *et al.* (1997). It is noted that VNS1 and VNS2 embedded swap and invert local search, respectively. Also GA1 and GA2 represented the results from Christensen *et al.* (1997) and of this research, respectively. It is noted that these algorithms also found alternative solutions: [1 2 3 4], [1 4 3 2], [1 3 4 2], [2 1 3 4], [2 1 4 3] and [4 1 3 2]. The results reported in Christensen *et al.* (1997) and Zhu *et al.* (1998) were not able to be compared due to the unavailability of the detailed data.

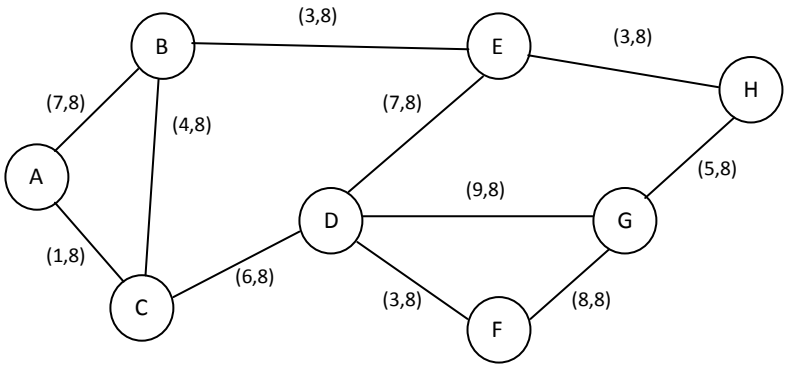

Figure 3.4: A Communications network with (cost, capacity) associated with each edge (Christensen *et al.*, 1997).

Table 3.2: Data for the first test case.

Request	Source	Destination(s)	Capacity
R1	B	C, H	5
R2	D	B, H	3
R3	G	E, F, B	4
R4	A	G	2

Table 3.3 illustrates the results for the example taken from Christensen *et al.* (1997). It is observed that the results from the GA and the two VNS of this research produced slightly superior results. This may be due to the underlying algorithm for the Steiner Tree problem. Better methods were used by embedding in the KMB method that was the Kruskal's algorithm instead of Prim's algorithm as reported in Christensen *et al.* VNS converged in a superior time as compared to GA. Alternative solutions have

also been determined using these algorithms: [1 2 3 4], [1 4 3 2], [1 3 4 2], [2 1 3 4], [2 1 4 3] and [4 1 3 2].

Table 3.3: Results for the first test case.

	GA1 (Christensen *et al.*, 1997)	GA2	VNS1	VNS2
Best Order	[2 4 1 3]	[2 4 1 3]	[4 2 1 3]	[1 3 2 4]
Time (*in seconds*)	-	1.356253	0.303125	0.257813
Cost	253	249	249	249

* GA2: GA with EERO; VNS1: VNS embeds Swap; VNS2: VNS embeds Invert.

The algorithms are extended to look at a second data set consisting of medium sized problems (scheduling 4 to 8 requests). A network consisting of 50 nodes and 83 edges was selected taken from the OR-Library (steinb1 and the last 20 data from steinb2 that did not conflict with steinb1) (Beasley, 1990). The capacity of all the edges/links were fixed at 12. The characteristics of the requests are given in Table 3.4, whilst the results are tabulated in Table 3.5.

Table 3.4: Data for the second test.

Request	Source	Destination(s)	Capacity
1	36	7, 23, 25, 40	8
2	17	15, 30, 31, 40, 41, 46	5
3	48	3, 9	9
4	41	13, 22, 27, 35, 50	6
5	2	6, 14, 18, 23, 27, 33, 47, 49	5
6	13	28	6
7	50	5, 12, 28,31, 44, 45	7
8	20	17, 25, 41	8

Table 3.5: Results for the second test case.

No of Requests	Criteria	GA	VNS1	VNS2
4R	Best Order	[3 1 2 4]	[3 1 2 4]	[3 1 2 4]
	Time (in seconds)	14.21877	6.085938	6.434378
	Cost	1208	1208	1208
5R	Best Order	[5 1 3 4 2]	[5 1 3 4 2]	[5 1 3 4 2]
	Time	19.30315	10.26875	9.723444
	Cost	1703	1703	1703
6R	Best Order	[5 1 3 2 4 6]	[5 1 3 4 2 6]	[5 1 3 4 2 6]
	Time	21.65938	13.53594	11.99063
	Cost	1841	1841	1841
7R	Best Order	[5 1 3 2 4 6 7]	[5 1 2 3 4 6 7]	[5 1 3 4 2 6 7]
	Time	26.02657	14.36407	17.21718
	Cost	2450	2450	2450
8R	Best Order	*	*	*
	Time			
	Cost			

* The algorithm fails to converge

There are also alternative solutions for 4 requests: [3 1 4 2]; 5 requests: [5 1 3 2 4] and [5 1 2 3 4]; 6 requests: [5 1 2 3 6 4]; and 7 requests problems: [5 1 2 3 4 6 7]. All the algorithms failed to converge to a feasible solution for the 8 requests problems. It was observed that the capacity constraint on any of the path could not accommodate the 8 requests.

Encouraged by the results, the algorithms were tested on larger problems of up to 20 requests. The network comprised of 61 nodes and 133 edges. The data are shown in Table 5. The capacity for the network was maintained at 12, and the edges and costs were taken by modifying data steinb from the OR-Library (Beasley, 1990). The results obtained are given in Table 6. It was observed that the two VNS (VNS1 and VNS2)

failed to converge to feasible solutions for the 19 and 20 requests problems. It is noted that the population size for the GA algorithm were increased to 50 individuals and the algorithms were run for 100 generations. The local search for the VNS was improved to include the *or*-opt which proved to perform better for the Travelling Salesman and routing problems. This was at the expense of the larger CPU time. The algorithm was denoted as VNS3.

Then encouraged by the results obtained in VNS3, we then tested the algorithm for restricted *or*-opt as local search, VNS4 in hope of better computational running time. However it is noted that VNS3 converge in lower time for Requests 4 to 9 because it fails to improve the results as it may gets stuck in local minimum compared to VNS4, where better results were found at the expense of higher computational time. The results improved in term of *cpu* time until 8R, then continue with a slightly higher *cpu* time for the other requests. However in terms of quality solution, it is proven that restricted *or*-opt perform better, except for 16R where results obtained using GA is slightly better.

Table 3.6: Data for the third test.

Request	Source	Destination(s)	Capacity
1	36	7, 23, 25, 40	8
2	17	15, 30, 31, 40, 41, 46	5
3	48	3, 9	9
4	41	13, 22, 27, 35, 50	6
5	2	6, 14, 18, 23, 27, 33, 47, 49	5
6	13	28	6
7	50	5, 12, 28,31, 44, 45	7
8	24	30, 29, 20	5
9	52	13, 55, 22, 9	4

Table 3.6: Continued.

Request	Source	Destination(s)	Capacity
10	53	28, 52, 13, 55, 41, 14	6
11	10	31, 20, 5, 40	5
12	15	30, 20, 23, 22, 18	4
13	14	9, 35, 16	4
14	61	15, 33, 38, 20,	3
15	55	4, 41, 21	5
16	14	16, 43, 44, 31, 9	7
17	60	28, 14	2
18	9	6, 35, 30, 7, 4, 31	4
19	51	54, 40, 10	6
20	51	23, 10	5

Table 3.7: Results for the third test case.

No of Requests	Criteria	GA	VNS1	VNS2	VNS3	VNS4
4R	Cost	**813**	**813**	**813**	**813**	**813**
	Time (in seconds)	393.5125	38.2125	30.98282	51.42346	15.20937
5R	Cost	**1028**	**1028**	**1028**	**1028**	**1028**
	Time	549.7438	60.74844	51.67971	117.9142	49.53752
6R	Cost	**1135**	**1135**	**1135**	**1135**	**1135**
	Time	599.7266	61.21408	65.7047	166.7283	117.6078
7R	Cost	**1600**	**1600**	**1600**	**1600**	**1600**
	Time	719.1938	78.63288	88.00941	279.6203	267.5781
8R	Cost	**1768**	**1768**	**1768**	**1768**	**1768**
	Time	799.7219	106.4251	87.81569	529.4485	359.8924

Table 3.7: Continued 1.

No of Requests	Criteria	GA	VNS1	VNS2	VNS3	VNS4
9R	Cost	1955	1975	1969	**1951**	**1951**
	Time	897.5828	102.0751	121.9547	544.1922	692.625
10R	Cost	**2272**	**2272**	2293	**2272**	**2272**
	Time	1024.33	132.7064	119.2828	774.9266	972.1284
11R	Cost	**2568**	2611	2646	2572	**2568**
	Time	1147.95	145.2829	112.3704	708.6957	1720.97
12R	Cost	2808	2897	2879	2768	**2752**
	Time	1252.26	188.3596	118.183	1029.794	1770.81
13R	Cost	2970	3058	3060	2959	**2941**
	Time	1330.3800	158.9971	168.8579	980.7216	3541.7
14R	Cost	3190	3252	3285	3192	**3113**
	Time	1427.73	187.5609	146.1902	1312.548	7043.61
15R	Cost	3412	3422	3479	3316	**3294**
	Time	1491.8889	206.558	167.0469	1674.065	4720.351
16R	Cost	**3673**	3889	4184	3900	3844
	Time	1620.04	203.2931	219.6407	2050.567	4424.944
17R	Cost	3902	4131	4042	4036	**3834**
	Time	1700.60	252.7404	221.2611	1770.141	3891.36
18R	Cost	4053	4479	4386	4201	**3969**
	Time	1826.00	238.3081	222.8828	2442.993	9385.284
19R	Cost	4974	*	*	4601	**4476**
	Time	1906.65	*	*	2739.76	9511.067

Table 3.7: Continued 2.

No of Requests	Criteria	GA	VNS1	VNS2	VNS3	VNS4
20R	Cost	4744	*	*	5019	**4534**
	Time	1958.175	*	*	2230.875	14301.43

* The algorithm fails to converge

Table 3.7 presents the results for the larger problems and the best solutions found are highlighted. It is noted that GA (achieving 8 out of 20 best solutions), VNS with *or*-opt (achieving 7 best solutions out of 20 problems) and VNS with *or*-opt (achieving 19 best solutions out of 20 problems), where VNS with *or*-opt (VNS3) were almost comparable with GA performing slightly better in larger problems in terms of solution quality and CPU time. However it is shown that VNS4, that is VNS with restricted *or*-opt perform better compared to GA and VNS3. The algorithm for VNS1 and VNS2 fails to converge for larger request because of the underlying local search, swap and invert, are not powerful enough.

It is noted that out of the ten runs, GA produced smaller numbers of feasible solutions compared to VNS with *or*-opt and VNS with *restricted or*-opt

3.7 Conclusion

In this study, a slightly better GA (GA2) was designed as compared to GA1 proposed by Christensen *et al.* (1997) when tested on the examples given in Christensen *et al*. The developed GA2 produced better results in terms of minimum cost. Besides that, alternative solutions for several cases were also presented. These differences may have been due to the different operators for selection and crossover that were used. The VNS method was successfully developed and it is noted that VNS had the potential to perform better if the underlying local search was good. For the first test case it was

proven that VNS with a simple local search produced the same results as GA2 in a slightly better running time.

Encouraged by the results in the first test case, the problems were tested for up to 8 requests by successfully applying the GA and VNS methods for the case. It was shown that the final results were comparable with the GA method in terms of cost. However, VNS produced better *cpu* time.

The study was extended by testing the algorithms for up to 20 requests test problem. The results showed that VNS produced similar results to GA at a significant time. All the algorithms, GA, VNS1 and VNS2 failed to route the 19 requests and 20 requests problems.

Many researchers had successfully reported that o*r-opt* produced better results in the TSP. Encouraged by that, the VNS3 which embedded the *or-opt* as a local search was developed. However, the results obtained did not surpass GA as VNS3 produced a higher percentage of feasible solutions even for larger problems. VNS3 successfully routed all 20 requests, with a trade off to the running time.

We then apply restricted *or*-opt as local search in VNS, named VNS4. It is observed that VNS4 surpass GA and VNS3 with a good solution, however the running time is higher compared to VNS3 for bigger case. VNS4 successfully find best solution for all 20R except for 16R where GA produces better results.

It is concluded that the VNS performed better compared to the GA with a good underlying method as a local search. However, as the complexity of the local search algorithm increased, the time taken to obtain the solutions also increased.