2009 International Conference on Future Computer and Communication

A Parallel Genetic Algorithm for Shortest Path Routing Problem

Salman Yussof
Department of System and Networking
E-mail: salman@uniten.edu.my

Rina Azlin Razali
Department of System and Networking
E-mail: rina@uniten.edu.my

Ong Hang See
Department of Electronic and Communication Engineering
E-mail: ong@uniten.edu.my

***Abstract*— Shortest path routing is the type of routing widely used in computer networks nowadays. Even though shortest path routing algorithms are well established, other alternative methods may have their own advantages. One such alternative is to use a GA-based routing algorithm. Based on previous research, GA-based routing algorithm has been found to be more scalable and insensitive to variations in network topologies. However, it is also known that GA-based routing algorithm is not fast enough for real-time computation. In this paper, we proposed a parallel genetic algorithm for solving the shortest path routing problem with the aim to reduce its computation time. This algorithm is developed and run on an MPI cluster. Based on experimental result, there is a tradeoff between computation time and the result accuracy. However, for the same level of accuracy, the proposed parallel algorithm can perform much faster compared to its non-parallel counterpart.**

Keywords-component ***Coarse grained, message passing interface, parallel genetic algorithm, shortest path routing.***

I. INTRODUCTION

Routing in a computer network refers to the task of finding a path from a source node to a destination node. Given a particular network, it is very likely that there is more than one path that can be used. The task of a routing algorithm is to find the shortest path. Shortest path routing algorithms such as Dijkstra's algorithm and Bellman-Ford algorithm are commonly used in computer network nowadays [1].

Even though shortest path routing algorithms are already well established, there are researchers who are trying to find alternative methods to find shortest paths through a network. One such alternative is to use genetic algorithm (GA). GA is a multi-purpose search and optimization algorithm that is inspired by the theory of genetics and natural selection [2]. The problem to be solved using GA is encoded as a chromosome that consists of several genes. The solution of the problem is represented by a group of chromosomes referred to as a population. In each iteration of the algorithm, the chromosomes in the population will undergo one or more genetic operations such as crossover and mutation. The result of the genetic operations will become the next generations of the solution. This process continues until either the solution is found or a certain termination condition is met. The idea behind GA is to have the chromosomes in the population to slowly converge to an optimal solution. At the same time, the algorithm is supposed to maintain enough diversity so that it can search a large search space. It is the combination of these two characteristics that makes GA a good search and optimization algorithm.

One of the earliest GA-based shortest path routing algorithms is the one proposed by Munetomo et al. [3], [4]. Munetomo proposed a GA-based routing algorithm to generate alternate paths that can be quickly used in the case of link failures. In the proposed algorithm, the algorithm chromosome is encoded as a list of node IDs that are on the path from the source node to the destination node. Since different paths can have different number of nodes, the chromosomes are of variable length. This algorithm employs crossover, mutation and migration genetic operators in generating the next generation of solutions. Chang et. al [5] also proposed a GA-based routing algorithm for solving the shortest path routing problem. Similar to Munetomo's algorithm, the chromosome in this algorithm consists of a sequence of node IDs that are on the path from source to destination. However, there are several differences in the details of the GA implementation such as in the crossover and mutation operations. Some researchers implemented a hybrid GA algorithm where GA is combined with another algorithm to solve the shortest path routing problem. One example of this is the algorithm proposed by Hamdan et. al [6] who combined GA with the Hopfield network. Another example would be an algorithm proposed by Riedl [7] who combined GA with a local heuristics search.

Chang et. al [5] has shown that using GA for the shortest path routing problem has several advantages. The first advantage is that GA is insensitive to variations in network topologies with respect to route optimality and convergence speed. The second advantage is that GA-based routing algorithm is scalable in the sense that the real computation size does not increase very much as the network size gets larger. However this literature also pointed out that GA is not fast enough for real-time computation and in order to achieve a really fast computation time in GA, a hardware implementation of GA is required.

In this paper, we are proposing a parallel genetic algorithm for the shortest path routing problem. The motivation behind this proposal is that parallel implementation of GA should be able to improve its computation time. The proposed algorithm is implemented on a message passing interface (MPI) cluster.

978-0-7695-3591-3/09 $25.00 © 2009 IEEE
DOI 10.1109/ICFCC.2009.36

II. Parallel genetic algorithm

GA is generally able to find good solutions in reasonable amount of time but as they are applied to harder and bigger problems, there is an increase in the time required to find adequate solutions. To tackle this problem, there have been multiple efforts to make GA faster and one of the promising choices is to use parallel implementation [8].

In parallel GA (PGA), there are multiple computing nodes. The task of each computing node depends on the type of parallel GA used. There are four major types of parallel GAs: (1) Master-slave GA, (2) Coarse-grained GA, (3) Fine-grained GA and (4) Hierarchical hybrids. In master-slave GA, one computing node will become the master and the other computing nodes will become the slaves. The master node will hold the population and perform most of the GA operations. However, the master can assign one or more computing-intensive tasks (i.e. fitness evaluation or crossover) to the slaves. This is done by sending one or more chromosomes to the slaves and the master would then wait for the slaves to return their results. In coarse-grained GA, the population is divided to the computing nodes. Each computing node would then have a sub-population and each node executes GA on its own sub-population. To ensure that good solutions can be spread to other nodes, the nodes will occasionally (with certain probability) exchange chromosomes with each other. This exchange is called migration and it involves a node sending its best chromosome to other nodes. The other nodes would then replace their worst chromosome with the one received. In fine-grained GA, each computing node only has a single chromosome. The computing nodes are normally arranged in a spatial structure where each node can only communicate with several neighboring nodes. The population would be the collection of all the chromosomes in each node. To execute a genetic operation, a computing node will have to interact with its neighbors. Since the neighborhood overlaps, eventually the good traits of a superior individual can spread to the entire population. Fine-grained GA has the highest level of parallelism among the four types of parallel GAs. But it also has a large communication overhead due to the high frequency of interactions between neighboring nodes. The final PGA type is the hierarchical hybrid which is structured in two levels. At the higher level, the algorithm operates as a coarse-grained GA while at the lower level the algorithm operates as a fine-grained (or master-slave) GA.

For each of the parallel GA type, there are multiple variations proposed by researchers to improve its performance or to suite a particular problem. For example, Golub et. al [9] proposed a master-slave GA where the master only creates the population and let the slaves perform the whole evolution process. Ling Tan et. al [10] proposed a coarse-grained GA which has a special computing node assigned to collect the best chromosomes from all the nodes and distribute one or more of the fittest ones to the other nodes. This is done to reduce the delays in propagating the globally fittest chromosome to all the computing nodes. Toro et. al [11] also proposed a modification to the coarse-grained GA where a master node is assigned to gather the whole population from all the computing nodes once in several generations. The master node would then sort the chromosomes according to some objective function and then distribute them again to the computing nodes.

Parallel GA has been successfully used in various problems such as design optimization [12], transport routing [13], time series forecasting [14], network design [15] and sorting [16].

III. Proposed parallel genetic algorithm for shortest path routing

A. Overview

For the proposed algorithm, the coarse-grained GA has been chosen to be adopted. The main reason for this choice is due to the use of MPI cluster. In an MPI environment, communication between computers has a large overhead. Therefore, coarse-grained GA would be the most suitable type of PGA to be used since it has the lowest communication overhead compared to the other PGA types.

In our PGA implementation, all the computing nodes will randomly create their own sub-population and each of them will execute GA on its own sub-population. However, one of the computing nodes will be assigned a special task to gather results from all the other nodes and then choose the best result (the one that gives the shortest path) to be the output of the PGA. This node is called the collector node.

The operations done by the computing nodes are outlined below:

1. Randomly initialize the initial sub-population.
2. Evaluate fitness of each chromosome in the population.
3. Create the mating pool which consists of all the chromosomes in the current population.
4. Apply crossover operator several times to create *n* new children for the new sub-population (where *n* is the size of the sub-population). The parents are selected using the selection operator. Crossover is only performed if one or both of the parents have not yet mated.
5. Apply mutation operator on the chromosomes in the mating pool. Each chromosome has a certain probability to be mutated.
6. Repeat step 2 until the sub-population converges or the maximum number of iterations has been achieved. The value for maximum iteration used in all the experiments is 100.
7. Send the best chromosome to the collector node.

The operations done by the collector node are outlined below:

1. Receive the best chromosomes from each of the computing nodes.
2. Choose the best chromosome from the ones received. This would then be the shortest path found by the parallel algorithm.

In the proposed algorithm, the migration operation, as commonly employed in a coarse-grained PGA implementation, is not implemented to minimize the interaction between computing nodes. This is because in an MPI environment, interaction between computing nodes is very costly in terms of computation time. This decision would, however, result in a slight performance decrease in terms of accuracy.

B. Genetic Encoding

A communication network can be modeled as a directed graph *G(N,E)*, where *N* is the set of nodes representing routers and *E* is the set of edges connecting the links that connect between the routers [1]. Each edge *(i,j)* is associated with an integer representing the cost of sending data from node *i* to node *j* and vice versa.

In the proposed algorithm, each chromosome is encoded as a series of node IDs that are in the path from source to destination. The first gene in the chromosome is always the source and the last gene in the chromosome is always the destination. Since different paths may have different number of intermediate nodes, the chromosomes will be of variable length. However, the maximum length of a chromosome cannot exceed the total number of nodes in the network. Any repeated nodes in the chromosome signify that the path represented by the chromosome contains a loop and in network routing, any loop should be eliminated.

C. Initial sub-population

In the beginning, the sub-population is filled with chromosomes that represent random paths. Even though the paths are random, they are supposed to be valid paths, where the chromosomes consist of a sequence of nodes that are in the path from sender to receiver. The number of chromosomes generated for each sub-population, S_n, depends on the total population size and the number of computing nodes, as depicted in Eq. 1.

$$S_n = \frac{P}{N} \tag{1}$$

where S_n represents the sub-population size of the *n*th node, *P* represents the total population size and *N* represents the total number of computing nodes.

The algorithm used to generate the random paths is as follows:

1. Start from the source node.
2. Randomly choose, with equal probability, one of the nodes that are connected to the current node.
3. If the chosen node has not been visited before, mark that node as the next node in the path. Otherwise, find another node.
4. If all the neighboring nodes have been visited, go back to step 1.
5. Otherwise, repeat step 2 by using the next node as the current node.
6. Do this until the destination node is found.

D. Fitness Function

Each chromosome in the population is associated with a fitness value that is calculated using a fitness function. This value indicates how good the solution is for a particular chromosome. This information is then used to pick the chromosomes that will contribute to the formation of the next generation of solution. The fitness function used in the proposed algorithm is defined as follows:

$$f_i = \frac{1}{c_i} \tag{2}$$

where f_i represents the fitness value of the *i*th chromosome and c_i represents the total cost of the path represented by the *i*th chromosome. This would give a higher fitness value for shorter paths.

E. Selection

Selection is used to choose the parent chromosomes for the crossover operation. The selection scheme used in the algorithm is the pairwise tournament selection with tournament size, s = 2. In this selection scheme, a parent for the crossover operation is selected by randomly choosing two chromosomes from the population. The one with the higher chromosome between the two will be selected as a parent. To select two parents, this operation is performed twice.

F. Crossover

Crossover is performed on the two parent chromosomes selected using the selection scheme described above. To ensure that the paths generated by the crossover operation are still valid paths, the two chromosomes selected must have at least one common node other than the source and destination nodes. If more than one common node exists, one of them will be randomly chosen with equal probability. The chosen node is called the crossover point. For example, assume that we have the following parent chromosomes:

Parent chromosome 1 = [A B C G H **I** X Y Z]
Parent chromosome 2 = [A K L M I T U Z]

where A and Z are the source node and destination node respectively. In this example, the common node is node I. Therefore, crossover operation will exchange the first portion of chromosome 1 with the second portion of chromosome 2 and vice versa. As a result, the following child chromosomes will be generated:

Child chromosome 1: [A B C G H **I** T U Z]
Child chromosome 2: [A K L M **I** X Y Z]

These two chromosomes would then become new members of the population.

G. Mutation

Each chromosome produced by the crossover operation has a small chance to be mutated based on the mutation probability, p_m. For all the experiments, the value for p_m is set to 0.05. For each chromosome that is chosen to be mutated, a mutation point will be chosen randomly, with equal probability, among the intermediate nodes in the path from sender to receiver (i.e. the sending and receiving node cannot be chosen as the mutation point). Once the mutation point is chosen, the chromosome will be changed starting from the node after the mutation point and onwards. For example, assume that the following chromosome has been chosen to be mutated:

Original chromosome: [A C E F **G** H I Y Z]

where A and Z are the sending node and the receiving node respectively. Assume also that the node G has been chosen as the mutation point. The mutated chromosome would become like this:

Mutated chromosome: [A C E F **G** x_1 x_2 x_3 … Z]

The mutated chromosome now contains a new path from G to Z where x_i is the ith new node in the path. The new path is generated randomly; the same way as the paths in the initial population is generated.

IV. Experiment and Discussion

A. *Experiment setup*

The proposed algorithm has been implemented as a C++ program. The program is run on an MPI cluster which has six machines. Each machine has a dual-core processor. A computing node is associated with a single processing core. Therefore, up to 12 computing nodes can be run on this parallel computer.

The objective of this experiment is to measure the performance of the proposed algorithm with respect to accuracy and computation time. Accuracy measures the percentage of the shortest paths returned by the algorithm that are actually shortest paths (as obtained from Dijkstra's algorithm). Computation time measures the execution time taken by the algorithm to obtain all the results from the beginning to the end of the simulation. The performance of the algorithm will be compared to the non-parallel version of the same algorithm.

There are two types of network used in the simulation, the n x n mesh network and the Waxman network [17]. The Waxman network is actually a random graph where the existence of link between two nodes, i and j, is defined by the following probability:

$$p_{ij} = \alpha \exp(-(\frac{d_{i,j}}{\beta L})), \; 0 < \alpha, \beta < 1 \qquad (3)$$

where $d_{i,j}$ is the distance between the two nodes, and L is the maximum inter-nodal distance in the topology. A larger value of α would generate a graph with higher density and a smaller value of β increases the density of short edges relative to longer ones. In all the experiments, the values for both α and β are set to 0.2 and 0.1 respectively. However, to avoid having disconnected nodes, each node must be connected to at least one other node. The network topologies used are 10x10 mesh network 15x15 mesh network, 100-node Waxman network and 225-node Waxman network. Each link in the network is given a randomly generated cost value, $c_k(i,j)$ ~ uniform[1,20].

The result reported in the next section is averaged over 50 runs. For each run, a new network with a new set of link metrics is randomly generated using different seeds. For the Waxman network, this also means that a different network topology is generated on each run. In each run, a total of 1000 source-destination pairs are randomly chosen and the shortest path for each of them is calculated.

B. *Result*

Fig. 1 and Fig. 2 show the accuracy of the proposed algorithm for the n x n mesh network and the Waxman network respectively. Table I on the other hand, shows the accuracy for the non-parallel version of the algorithm. It is obvious that as the number of computing node increases, the accuracy decreases linearly. This is because as the number of computing node increases, each computing node would have smaller sub-population size. In GA, smaller population size would result in lower performance. However, the benefit of PGA is apparent when comparing the computation time as depicted in Fig. 3 and Fig. 4. For both types of network, as the number of computing node increases, the computation time decreases exponentially. However, it is also observed that for small sub-population size, having too many computing nodes may eventually increase the computation time again. This is because with small sub-population size, the parallelism is not fully exploited due to the computing nodes are not doing enough work and the communication overhead would then cause the computation time to increase. For comparison, Table II shows the computation time for the non-parallel version of the algorithm.

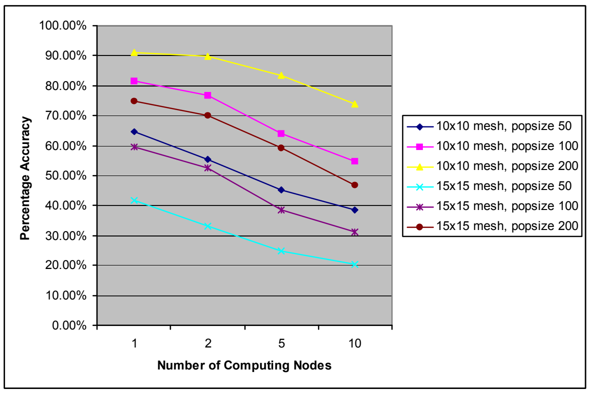

Figure 1. Accuracy for n x n mesh network

This experiment also shows that having a higher total population size would increase the accuracy. This effect is consistent regardless of the network topology, network size or the number of computing nodes. Having a higher population size would also increase the computation time. However, with larger number of computing nodes, the increase in computation time becomes less apparent.

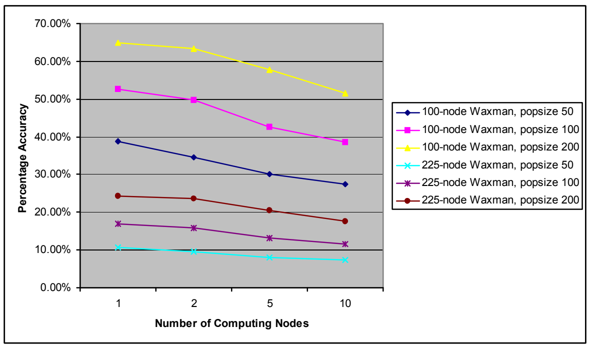

Figure 2. Accuracy for Waxman network

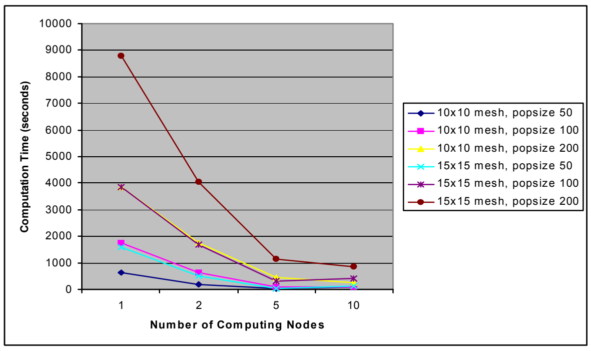

Figure 3. Computation time for *n* x *n* mesh network

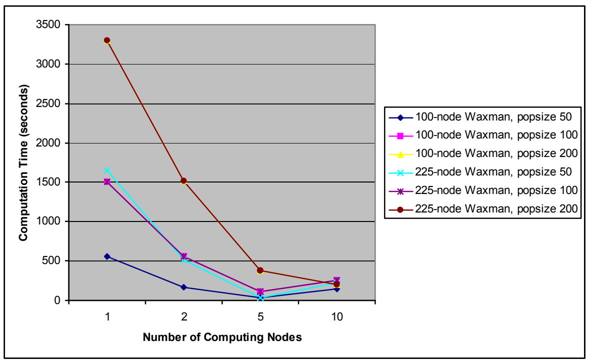

Figure 4. Computation time for Waxman network

TABLE I: RESULT ACCURACY FOR NON-PARALLEL VERSION OF THE ALGORITHM

Network Topology	Population Size	Accuracy
10 x 10 mesh	50	64.35 %
10 x 10 mesh	100	80.83 %
10 x 10 mesh	200	91.34 %
15 x 15 mesh	50	41.59 %
15 x 15 mesh	100	59.03 %
15 x 15 mesh	200	74.56 %

TABLE II: COMPUTATION TIME FOR NON-PARALLEL VERSION OF THE ALGORITHM

Network Topology	Population Size	Computation Time (seconds)
10 x 10 mesh	50	634.325
10 x 10 mesh	100	1847.4
10 x 10 mesh	200	5304.58
15 x 15 mesh	50	1849.26
15 x 15 mesh	100	5335.11
15 x 15 mesh	200	8884.98

TABLE III: COMPUTATION TIME REQUIRED BY BOTH THE PROPOSED PGA AND ITS NON-PARALLEL COUNTERPART TO GET THE SAME RESULT ACCURACY

Network Topology	T_{PGA} (seconds)	T_{GA} (seconds)	T_{PGA} / T_{GA} * 100
10x10 mesh	1179.04	8675.82	13.59 %
15x15 mesh	2672.4	8884.98	30.08 %
20x20 mesh	3447.12	16381.1	21.04 %
100-node Waxman	1780.62	7303.43	24.38 %
225-node Waxman	2312.92	19501.1	11.86 %
400-node Waxman	5129.28	44431.8	11.54 %

The advantage of the proposed algorithm can be more clearly seen by comparing the computation time required by both the proposed PGA and the non-parallel version of the algorithm to get the same accuracy. This is done by running the non-parallel version of the algorithm several times, each for different population size. The population size used varies from 50 to 500, with an increment of 50. The result is then compared with the proposed PGA with total population size of 500 and 10 computing nodes (each computing node would then have a sub-population of 50). Table III shows the percentage of computation time required by the proposed PGA (T_{PGA}) and the computation time of its non-parallel counterpart (T_{GA}) to get relatively the same accuracy. Based on the result, the computation time of PGA is only around 11% to 30% out of the computation time required by the non-parallel version of the algorithm. This reduction in computation time seems to be independent of the network type and size.

V. CONCLUSION

This paper proposed a coarse-grained parallel genetic algorithm for solving the shortest path routing problem. The aim is to produce a faster GA-based routing algorithm. Based on experiment done on an MPI cluster, the accuracy and computation time of the algorithm is dependant on population size and the number of computing nodes. With larger number of computing nodes, accuracy decreases linearly but computation time decreases exponentially. With larger population size, the accuracy is higher but the execution time is also higher. However, the increase of computation time due to the increase in population can be reduced by having larger number of computer nodes. When compared to the non-parallel version of the same algorithm, the developed parallel algorithm is observed to have a greatly reduced computation time.

REFERENCES

[1] J. F. Kurose, K. W. Ross, *Computer Networking: A Top-down Approach*, Addison-Wesley, 4th Edition, 2007.

[2] D. E. Goldberg, *Genetic Algorithm in Search , Optimization and Machine Learning, Addison Wesley*, 1989.

[3] M. Munetomo, Y. Takai, Y. Sato, “A migration scheme for the genetic adaptive routing algorithm”, in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2774 – 2779, October 1998.

[4] M. Munetomo, N. Yamaguchi, K. Akama, Y. Sato, “Empirical investigations on the genetic adaptive routing algorithm in the Internet”, in *Proc. Congress on Evolutionary Computation*, vol. 2, pp. 1236 – 1243, May 2001.

[5] Chang Wook Ahn, R. S. Ramakrishna, “A genetic algorithm for shortest path routing problem and the sizing of populations”, *IEEE Transactions on Evolutionary Computation*, vol. 6, issue 6, pp. 1 – 7, February 1999.

[6] M. Hamdan, M. E. El-Hawary, “Hopfield-genetic approach for solving the routing problem in computer networks”, in *Proc. Canadian Conference on Electrical and Computer Engineering*, vol. 2, pp. 823 – 827, May 2002.

[7] A. Riedl, “A hybrid genetic algorithm for routing optimization in IP networks utilizing bandwidth and delay metrics”, in *Proc. IEEE Workshop on IP Operations and Management*, pp. 166 – 170, 2002.

[8] Erick Cantu Paz, D. E. Goldberg, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic Publishers, 2001.

[9] M. Golub, D. Jakobovic, “A new model of global parallel genetic algorithm”, in *Proc. 22nd International Conference on Information Technology Interfaces*, pp. 363 – 368, June 2000.

[10] Ling Tan, D. Taniar, K. A. Smith, “A new parallel genetic algorithm”, in *Proc. International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 284 – 289, May 2002.

[11] F. de Toro, J. Ortega, J. Fernandez, A. Diaz, “PSFGA: a parallel genetic algorithm for multiobjective optimization”, in *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pp. 384 – 391, January 2002.

[12] M. M. Atiqullah, “Problem independent parallel genetic algorithm for design optimization”, in *Proc. International Symposium on Parallel and Distributed Processing*, pp. 204 – 211, April 2002.

[13] N. Meghanathan, G. W. Skelton, “Intelligent transport route planning using parallel genetic algorithm and MPI in high performance computing cluster”, in *Proc. International Conference on Advanced Computing and Communications*, pp. 578 – 583, December 2007.

[14] S. E. Eklund, “Time series forecasting using massively parallel genetic programming”, in *Proc. International Symposium on Parallel and Distributed Processing*, April 2003.

[15] Runhe Huang, Jianhua Ma, T. L. Kunii, E. Tsuboi, “Parallel genetic algorithm for communication network design”, in *Proc. 2nd Aizu International Symposium on Parallel Algorithms/Architecture Synthesis*, pp. 370 – 377, March 1997.

[16] Myung-Mook Han, “Applying parallel genetic algorithm for sorting problem”, in Proc. IEEE International Conference on Fuzzy Systems, vol. 3, pp. 1796 – 1801, August 1999.

[17] B.M. Waxman, “Routing for multipoint connections”, *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617 – 1622, December 1988.