AN OPTIMIZED AND ENERGY EFFICIENT MULTICAST ROUTING BASED ON GENETIC ALGORITHM FOR WIRELESS MOBILE AD HOC NETWORKS

Shenbagaraj R*

Dr. K. Ramar**

ABSTRACT

In Wireless Mobile Ad hoc networks, mobile node battery energy is limited and represents one of the important constraints for designing multicast routing protocols. In regards to the battery lifetime limitation in supporting multicast routing, some studies have given a Genetic algorithm solution for saving power. These proposed methods have always considered several techniques considering only a static scenario. We propose an energy-efficient genetic algorithm that is tested in a dynamic scenario. The simulation results are taken by considering a dynamic scenario which is appropriate for wireless Mobile Ad hoc networks.

*Assistant Professor, Information Technology Department, Mepco Schlenk Engineering, Schlenk Engineering College, Tamil Nadu.

******Principal, Einstein College of Engineering, Tamil Nadu.

INTRODUCTION

Mobile Ad Hoc Networks (MANETs) are groups of self-organizing mobile nodes in dynamic topology networks. MANETs do not need infrastructure units such as base stations or access points in advance. Each movable node in the MANETs has a routing function whereby it communicates by forwarding datagrams via intermediate nodes. If two movable nodes are located within the forwarding range, they communicate with each other directly. Otherwise, they need another node to forward their datagrams and require a datagram forwarding operation using a multi-point hopping method. MANETs [16] are characterized by non-restricted mobility and easy deployment, which makes them very popular.

In general, the design of a QoS multicast routing protocol [7] with multi-constrained metrics has not always taken into consideration the consumption of battery energy. Thus, upon operation of the whole network, some mobile nodes can have problems with energy overhead due to a lack of balance in their battery energy consumption. Once these selected nodes in the multicast tree run out of residual battery energy, an interruption condition arises that is generated in the link during packet forwarding. Thus, there are numerous interruptions that can occur in selected packet forwarding routing paths. The multicast service lifetime will not be maintained continuously until the completion of packet forwarding. The idea is to maximize the duration so that all mobile nodes are up until one of them is drained of energy. The QoS items that are called metrics include available bandwidth, end-to-end delay, probability of packet loss, delay variance (jitter), expense, and so on. Since different items have different properties, they could be classified into three categories, namely additive, multiplicative, and minimal properties. Many relevant references which have been cited in multicast routing topics usually tackle some of the general QoS metrics such as the bandwidth, packet loss rate, and propagation delay.

A genetic algorithm (GA) is a searching algorithm that utilizes the genetic operators, for examples, crossover and mutation. It emulates the evolution idea using natural selection and the survival of the fittest concept. In each generation, a new population of solutions is created by exchanging and combining the information obtained from the solutions through the previous generation. Crossover is one of the genetic operators in which genes from two chromosomes are exchanged and the genotypes of two selected parents are merged to make two new offspring. Crossover is also referred to as recombination and sometimes called mating. Two chromosomes with greater fitness values are picked from the chromosome pool. The starting point and length of the portion to be exchanged are randomly selected. The two new offspring are created and put back into the chromosome pool. Mutation is another genetic operator in which new genetic structures are inducted into the population by randomly modifying some of the genes. Mutation also maintains the search algorithm to escape from local optimum and prevents converging too fast. In other words, mutation operation gives the genetic algorithm an opportunity to search for new and more feasible chromosomes in new areas of solution space. After the mutation operation, the multicast tree will be modified because the mutation operator can destroy the tree structure and outgoing degree constraints. The other genetic algorithm operations include encoding, initial population, evaluation, reproduction, crossover and mutation.

The remainder of this paper is organized as follows: Section 2 does a Literature Survey. Section 3 describes our proposal. Section 3 discusses the analysis and simulation results and the last section discusses our conclusion.

2. LITERATURE SURVEY

A **genetic algorithm** (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. Genetic algorithms are implemented in a computer simulation in which a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions.

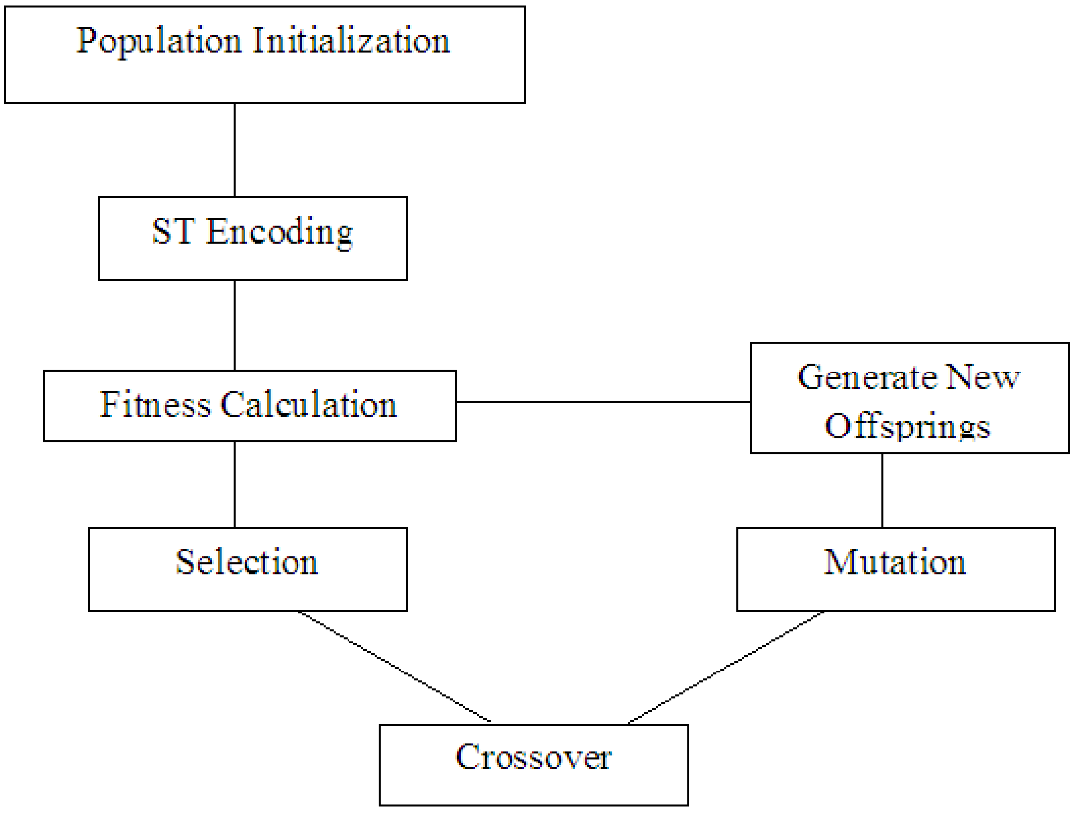
The genetic algorithm object determines which individuals should survive, which should reproduce, and which should die. It also records statistics and decides how long the evolution should continue. Typically a genetic algorithm has no obvious stopping criterion. We must tell the algorithm when to stop. Often the number-of generations is used as a stopping measure, but we can use goodness-of-best-solution, convergence-of-population, or any problem-specific criterion if we prefer.

Each generation the algorithm migrates some of the individuals from each population to one of the other populations. In addition to the basic built-in types, GAlib[20] defines the components we'll need to derive our own genetic algorithm classes. The examples include a few of these derivations including a genetic algorithm that uses multiple populations and 'migration' between populations on multiple CPUs, and a genetic algorithm that does 'deterministic crowding' to maintain different species of individuals during the evolution. The base genetic algorithm class contains operators and data common to most flavors of genetic algorithms. When we derive our own genetic algorithm we can use these member data and functions to keep track of statistics and monitor performance.

The genetic algorithm contains the statistics, replacement strategy, and parameters for running the algorithm. The population object, a container for genomes, also contains some statistics as well as selection and scaling operators. A typical genetic algorithm will run forever. The library has built in functions for specifying when the algorithm should terminate. These include terminate-upon generation, in which we specify a certain number of generations for which the algorithm should run, and terminate-upon-convergence, in which we specify a value to which the best of-generation scores should converge. We can customize the termination function to use our own stopping criterion. The number of function evaluations is a good way to compare different genetic algorithms with various other search methods. The GAlib genetic algorithms keep track of both the number of genome evaluations and population evaluations.

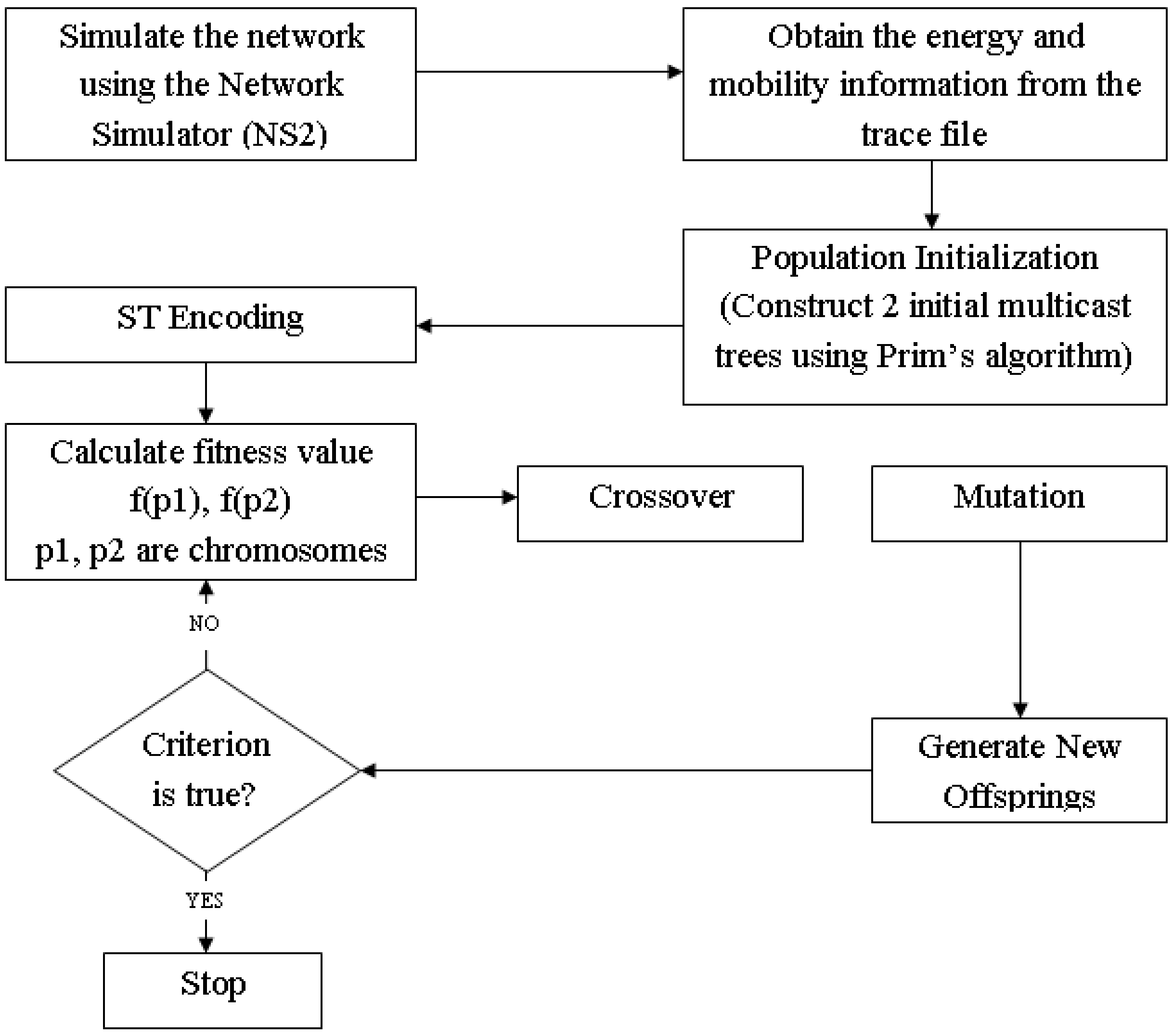
3. OUR PROPOSAL

3.1 OUR ARCHITECTURAL DESIGN



3.1 Architectural Design

3.2 FLOW CHART


Simulate the network using the Network Simulator (NS2)
Obtain the energy and mobility information from the trace file
Population Initialization (Construct 2 initial multicast trees using Prim's algorithm)
ST Encoding
Calculate fitness value f(p1), f(p2) p1, p2 are chromosomes
Crossover
Mutation
Generate New Offsprings
Criterion is true?
NO
YES
Stop

4.2.1 Flow Chart

3.3 NETWORK SIMULATION

The mobile nodes are simulated using NS-2. After simulating the mobile nodes trace file is generated. The network topology used in the simulation is tree network with nine mobile nodes. The routing protocol used here is dsdv multicast routing protocol. The movement trace is set to ON.

3.4 OBTAINING INFORMATION FROM THE TRACE FILE

The trace file contains the energy, mobility, packet transmission details. From the trace file we obtain the energy and mobility information at two different time instances.

3.5 POULATION INITIALIZATION

By executing prim's algorithm for the information we got from NS-2 at two different time instances we construct two multicast tree. The two multicast tree we got is used as initial population.

3.6 ST ENCODING

In this encoding method, the multicast tree data structure is adopted as the chromosome structure. Each pair of S and T chromosomes represents a multicast tree. The advantages of the extended sequence and topology approach are not just to save encoding memory space, but also to reduce the decoding operation. The proposed extended sequence and topology encoding is easy for genetic operations.

3.7 FITNESS VALUE CALCULATION

The fitness function formula is

$$\mathrm{Fit}(T(s,M)) = \alpha*(f_{delay}.\Omega)*(f_{repair})$$

where,

α is the weighting co-efficient of the positive real number

Ω is a penalty function[4]

The propagation delay cost function is

$$f_{delay} = c * \frac{1}{Delay(T(s,M))},$$

where,

c is the coefficient of the positive real number.

The mutation for the repair function is

$$f_{repair}=\mathrm{Tot\_bty}(T(s,M))$$

The total residual battery energy in a multicast tree is defined as

$$\mathrm{Tot\_bty}(T(s,M)) = \sum_{v\in T(s,M)} (Bt_v(t)),\ \text{where, } Bt_v(t) = \zeta_v * \left(\frac{Res\_Bt_v(t)}{Full\_Bt_v}\right)^{\gamma}$$

Table 1. Parameters and Description for fitness value calculation

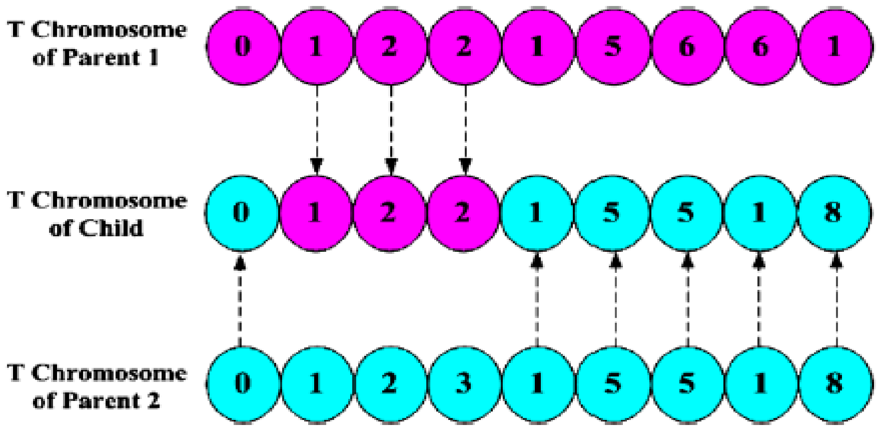
Parameters	Description
$Bt_v(t)$	Battery energy percentage in node v at time t
$Res\_Bt_v(t)$	Residual battery energy capacity of node v at time t
$Full\_Bt_v$	Full battery capacity of node v
ζ_v	Transmission power of node v
γ	Positive weighting factor

3.8 CROSSOVER

We utilize the two point crossover, and randomly select two chromosomes as parents; the genome would be selected from the two points and inherited by the offspring. The remaining genome of the offspring will be inherited from parent 2.

T-Crossover

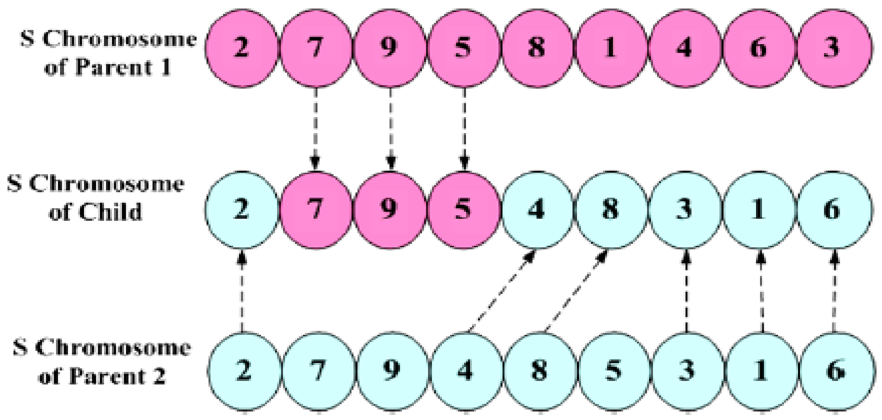
In the two-crossover, we randomly select two chromosomes as parents; the genome would be selected from the two points and inherited by the offspring. The remaining genome of the offspring will be inherited from parent 2.



4.8.1 The crossover of T-Chromosome

S-Crossover

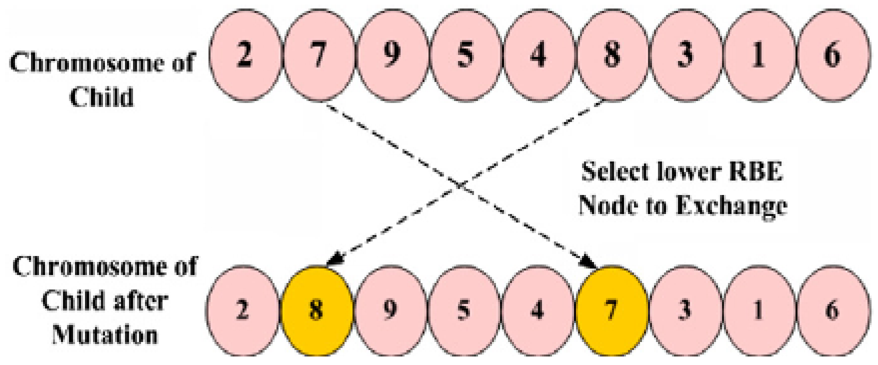
The gene of the S chromosome has unique and unrepeatable characteristics and hence, assuming that the S chromosome of the parent has a higher heredity priority. The priority genome is inherited by offspring from the two selected crossover points and consequently the remaining genome in the offspring is inherited from parent 2.



4.8.2 The crossover of S-Chromosome

3.9 MUTATION AND EVALUATION

The deg(v) denotes the degree of a node. The higher the deg(v), the greater is the energy consumption. The genetic algorithm approach is used to restrain the deg(v). Meanwhile, when considering the problem of equally balancing the consumption of battery energy, we utilize the Mutation-Replace function wherein the term residual battery energy acts as a mutation factor to control the mutation process. The mutation operation of the S chromosome allows the node with lower energy to be replaced by a leaf node with greater energy.


Chromosome of Child
2 7 9 5 4 8 3 1 6
Select lower RBE Node to Exchange
Chromosome of Child after Mutation
2 8 9 5 4 7 3 1 6

5.2.1 The Mutation of S-Chromosome

4. IMPLEMENTATION METHODOLOGY

The files that are used to simulate the mobile nodes are,

- Tcl file
- Scenario file
- CBR file

TCL FILE

The Tcl file includes the script that is used to create nodes, to create link between nodes, to set the position of the nodes and to set the movement for the nodes in the simulated network. The routing protocols, the type of channel are also set.

SCENARIO FILE

This scenario file is used for setting the position of each and every node and the distance between the nodes. This file can be created by the user or otherwise by the user or otherwise by executing the **setdest** command in the terminal.

Steps

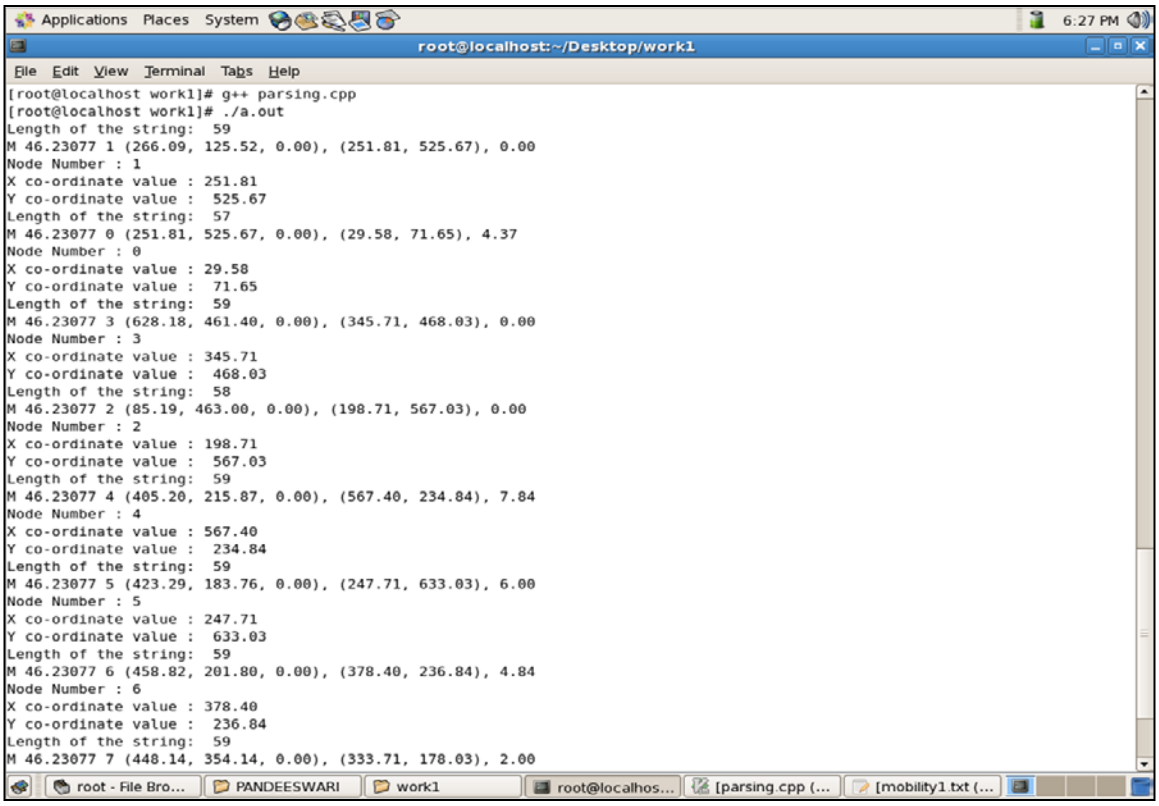
1.Go to the directory /usr/ns-allinone-2.3.1/indep-utils/cmu-scengen/setdest

2.Then type ./setdest –v 1 –n <no of nodes> -p <pause time> -M <maxspeed> -t<simulation time> -x <max X coordinate> -y <max Y coordinate> filename

CBR file

This file is used to create connection between two nodes whenever needed.
We can set which is the source node and which is the destination node. The type of protocol used and the type of traffic used can be set using this file.

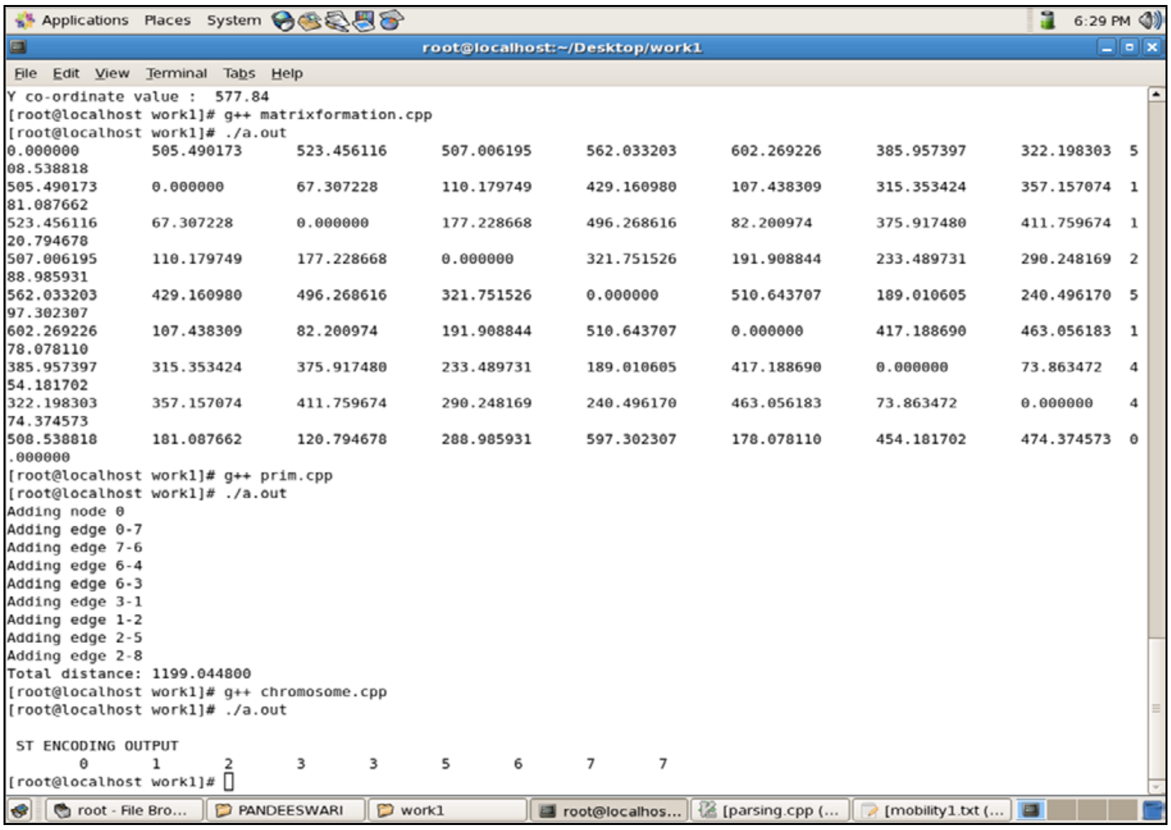
Getting Information from trace file



```
[root@localhost work1]# g++ parsing.cpp
[root@localhost work1]# ./a.out
Length of the string:  59
M 46.23077 1 (266.09, 125.52, 0.00), (251.81, 525.67), 0.00
Node Number : 1
X co-ordinate value : 251.81
Y co-ordinate value :  525.67
Length of the string:  57
M 46.23077 0 (251.81, 525.67, 0.00), (29.58, 71.65), 4.37
Node Number : 0
X co-ordinate value : 29.58
Y co-ordinate value :  71.65
Length of the string:  59
M 46.23077 3 (628.18, 461.40, 0.00), (345.71, 468.03), 0.00
Node Number : 3
X co-ordinate value : 345.71
Y co-ordinate value :  468.03
Length of the string:  58
M 46.23077 2 (85.19, 463.00, 0.00), (198.71, 567.03), 0.00
Node Number : 2
X co-ordinate value : 198.71
Y co-ordinate value :  567.03
Length of the string:  59
M 46.23077 4 (405.20, 215.87, 0.00), (567.40, 234.84), 7.84
Node Number : 4
X co-ordinate value : 567.40
Y co-ordinate value :  234.84
Length of the string:  59
M 46.23077 5 (423.29, 183.76, 0.00), (247.71, 633.03), 6.00
Node Number : 5
X co-ordinate value : 247.71
Y co-ordinate value :  633.03
Length of the string:  59
M 46.23077 6 (458.82, 201.80, 0.00), (378.40, 236.84), 4.84
Node Number : 6
X co-ordinate value : 378.40
Y co-ordinate value :  236.84
Length of the string:  59
M 46.23077 7 (448.14, 354.14, 0.00), (333.71, 178.03), 2.00
```

6.1 Getting Information from trace file

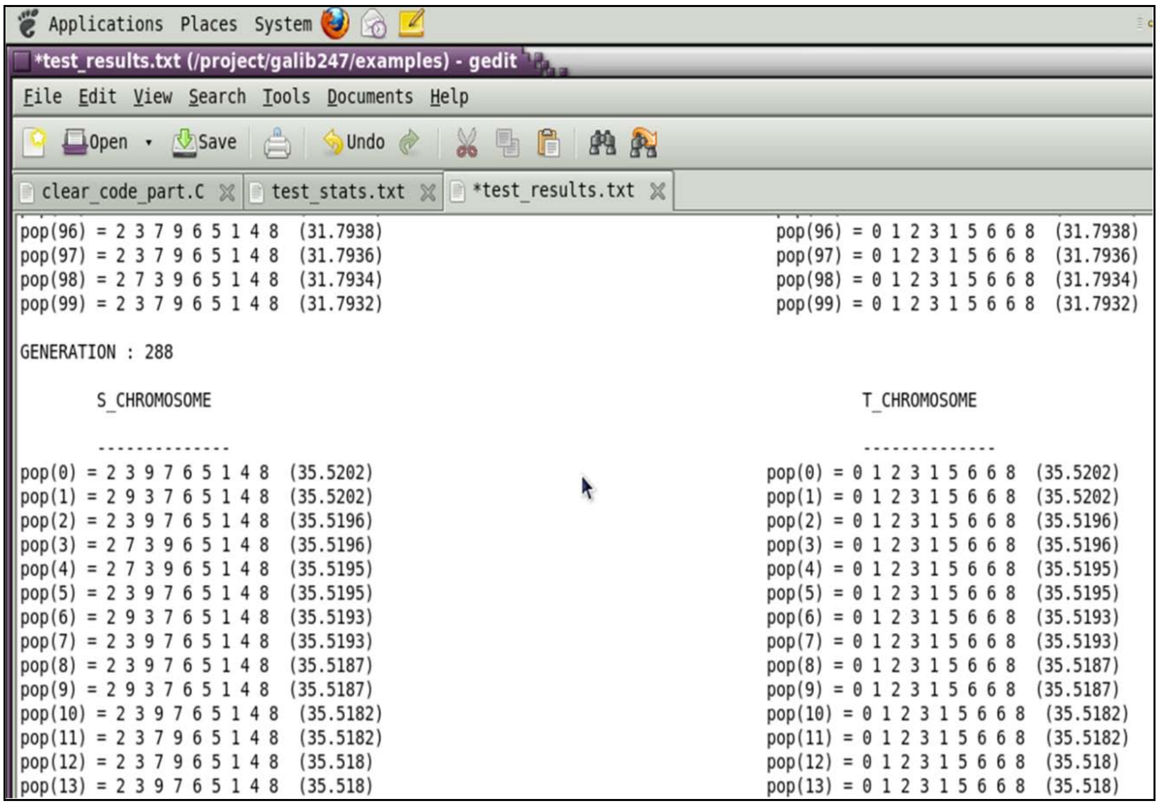
Prim's algorithm and ST Encoding Output


Applications Places System
6:29 PM
root@localhost:~/Desktop/work1
File Edit View Terminal Tabs Help
root - File Bro... PANDEESWARI work1 root@localhos... [parsing.cpp (... [mobility1.txt (...

```
Y co-ordinate value :  577.84
[root@localhost work1]# g++ matrixformation.cpp
[root@localhost work1]# ./a.out
0.000000        505.490173      523.456116      507.006195      562.033203      602.269226      385.957397      322.198303  5
08.538818
505.490173      0.000000        67.307228       110.179749      429.160980      107.438309      315.353424      357.157074  1
81.087662
523.456116      67.307228       0.000000        177.228668      496.268616      82.200974       375.917480      411.759674  1
20.794678
507.006195      110.179749      177.228668      0.000000        321.751526      191.908844      233.489731      290.248169  2
88.985931
562.033203      429.160980      496.268616      321.751526      0.000000        510.643707      189.010605      240.496170  5
97.302307
602.269226      107.438309      82.200974       191.908844      510.643707      0.000000        417.188690      463.056183  1
78.078110
385.957397      315.353424      375.917480      233.489731      189.010605      417.188690      0.000000        73.863472   4
54.181702
322.198303      357.157074      411.759674      290.248169      240.496170      463.056183      73.863472       0.000000    4
74.374573
508.538818      181.087662      120.794678      288.985931      597.302307      178.078110      454.181702      474.374573  0
.000000
[root@localhost work1]# g++ prim.cpp
[root@localhost work1]# ./a.out
Adding node 0
Adding edge 0-7
Adding edge 7-6
Adding edge 6-4
Adding edge 6-3
Adding edge 3-1
Adding edge 1-2
Adding edge 2-5
Adding edge 2-8
Total distance: 1199.044800
[root@localhost work1]# g++ chromosome.cpp
[root@localhost work1]# ./a.out

 ST ENCODING OUTPUT
        0       1       2       3       3       5       6       7       7
[root@localhost work1]# 
```

6.2 Prim's algorithm and ST Encoding Output

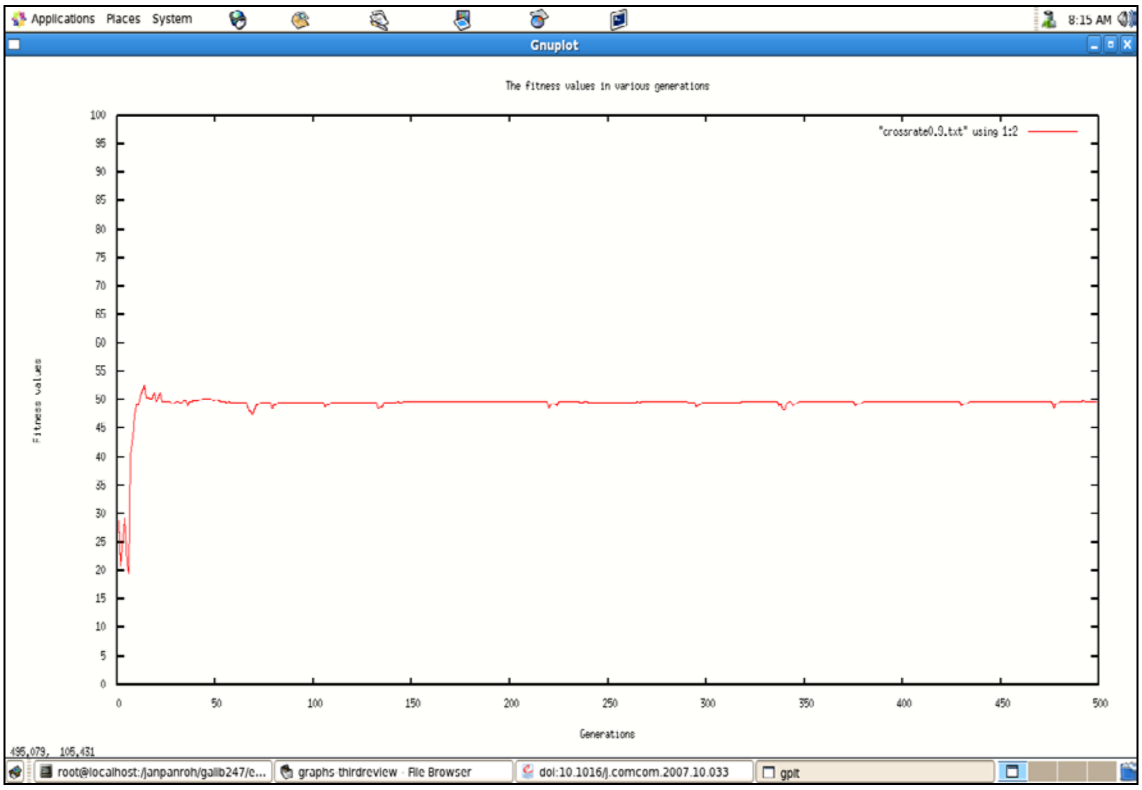
Genetic Algorithm Output


Applications Places System
*test_results.txt (/project/galib247/examples) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
clear_code_part.C test_stats.txt *test_results.txt

```
pop(96) = 2 3 7 9 6 5 1 4 8  (31.7938)                                    pop(96) = 0 1 2 3 1 5 6 6 8  (31.7938)
pop(97) = 2 3 7 9 6 5 1 4 8  (31.7936)                                    pop(97) = 0 1 2 3 1 5 6 6 8  (31.7936)
pop(98) = 2 7 3 9 6 5 1 4 8  (31.7934)                                    pop(98) = 0 1 2 3 1 5 6 6 8  (31.7934)
pop(99) = 2 3 7 9 6 5 1 4 8  (31.7932)                                    pop(99) = 0 1 2 3 1 5 6 6 8  (31.7932)

GENERATION : 288

        S_CHROMOSOME                                                              T_CHROMOSOME

        -------------                                                             -------------
pop(0) = 2 3 9 7 6 5 1 4 8  (35.5202)                                    pop(0) = 0 1 2 3 1 5 6 6 8  (35.5202)
pop(1) = 2 9 3 7 6 5 1 4 8  (35.5202)                                    pop(1) = 0 1 2 3 1 5 6 6 8  (35.5202)
pop(2) = 2 3 9 7 6 5 1 4 8  (35.5196)                                    pop(2) = 0 1 2 3 1 5 6 6 8  (35.5196)
pop(3) = 2 7 3 9 6 5 1 4 8  (35.5196)                                    pop(3) = 0 1 2 3 1 5 6 6 8  (35.5196)
pop(4) = 2 7 3 9 6 5 1 4 8  (35.5195)                                    pop(4) = 0 1 2 3 1 5 6 6 8  (35.5195)
pop(5) = 2 3 9 7 6 5 1 4 8  (35.5195)                                    pop(5) = 0 1 2 3 1 5 6 6 8  (35.5195)
pop(6) = 2 9 3 7 6 5 1 4 8  (35.5193)                                    pop(6) = 0 1 2 3 1 5 6 6 8  (35.5193)
pop(7) = 2 3 9 7 6 5 1 4 8  (35.5193)                                    pop(7) = 0 1 2 3 1 5 6 6 8  (35.5193)
pop(8) = 2 3 9 7 6 5 1 4 8  (35.5187)                                    pop(8) = 0 1 2 3 1 5 6 6 8  (35.5187)
pop(9) = 2 9 3 7 6 5 1 4 8  (35.5187)                                    pop(9) = 0 1 2 3 1 5 6 6 8  (35.5187)
pop(10) = 2 3 9 7 6 5 1 4 8  (35.5182)                                   pop(10) = 0 1 2 3 1 5 6 6 8  (35.5182)
pop(11) = 2 3 7 9 6 5 1 4 8  (35.5182)                                   pop(11) = 0 1 2 3 1 5 6 6 8  (35.5182)
pop(12) = 2 3 7 9 6 5 1 4 8  (35.518)                                    pop(12) = 0 1 2 3 1 5 6 6 8  (35.518)
pop(13) = 2 3 9 7 6 5 1 4 8  (35.518)                                    pop(13) = 0 1 2 3 1 5 6 6 8  (35.518)
```

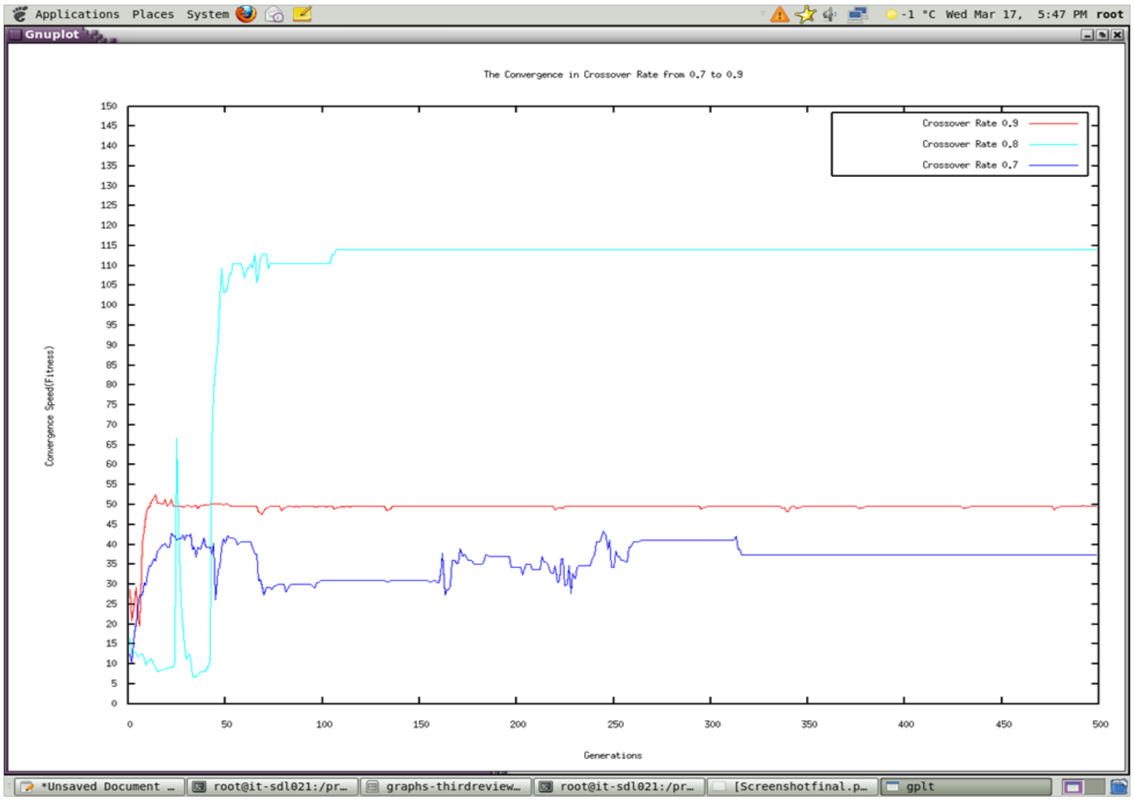
6.3 Genetic Algorithm Output

The Fitness values in various generations



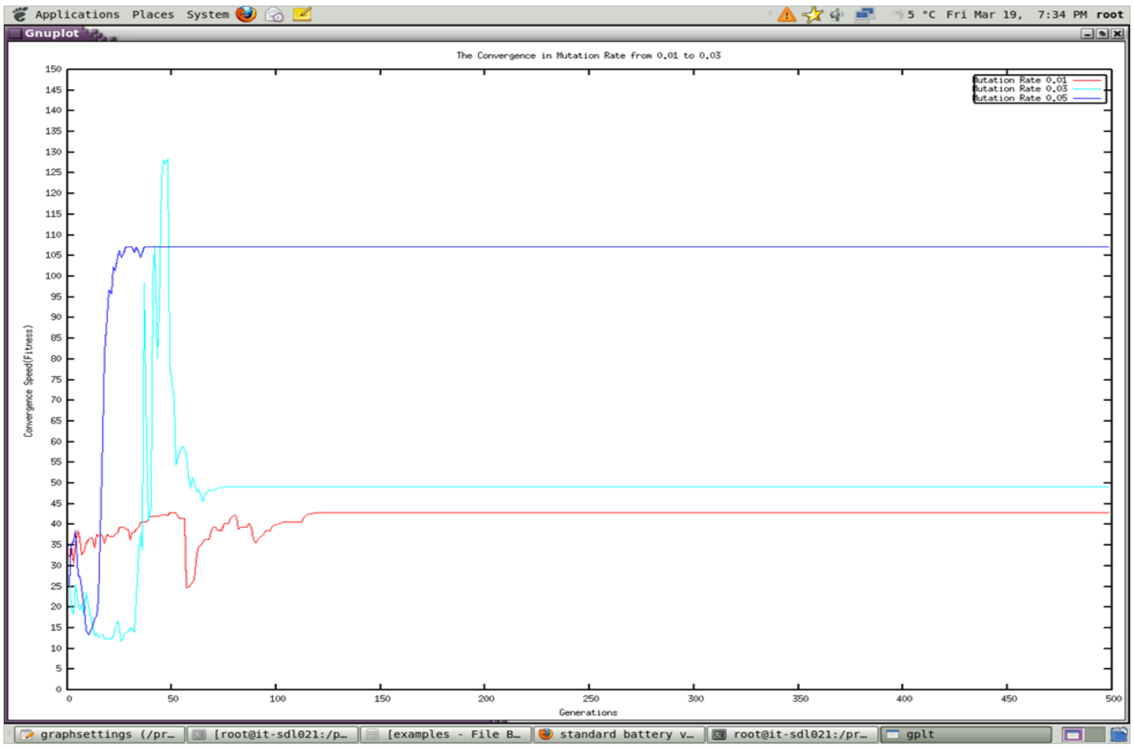
6.4 The Fitness values in various generations

The Convergence in Crossover Rate from 0.7 to 0.9



6.5 The Convergence in Crossover Rate from 0.7 to 0.9

The Convergence in Mutation Rate from 0.01 to 0.05



6.6 The Convergence in Mutation Rate from 0.01 to 0.05

CONCLUSION

We are focused on energy consumption efficiency which generally means selecting the node with the minimum energy consumption in the route selection. When this mechanism generates an improper phenomenon, some nodes with superior conditions are selected, resulting in rapid battery energy consumption. In this excessive usage situation, such schemes will further shorten the lifetime of the multicast tree and quickly end the multicast service. We adopted a strategy for balancing battery energy consumption in the entire multicast tree to prolong its maximum lifetime.

Retaining an energy-efficient genetic algorithm, we improved the mating mechanism and introduced energy efficient mutation as the repair function. The proposed method is applicable for reducing the node degree by replacing nodes with lower energy with nodes with higher energy. The total battery energy consumption for the multicast service is distributed using higher residual battery energy nodes. We also proposed an extended sequence and topology encoding scheme which allows us to directly calculate the cost value for the multicast tree between two arrays, S and T. It is no longer necessary to rebuild a new multicast tree.

Mutation for the repair function exchanges leaf and core nodes. The time complexity for this operation is O (k_Ln), where k is the number of nodes, and Ln is the number of leaf nodes. The reproduction of each generation comes only from crossover and mutation operators. In this way, battery energy and dramatic reduction in convergent time are achieved and the longest multicast service duration is obtained. Our research on genetic algorithms for multicast route selection is more suitable for the mobile ad hoc networks.

The proposed method can be extended to include the data security constraints on MANETs. Newer encoding methods can be proposed in future to get improve security.

REFERENCES

[1] T.C. Chiang, Y.M. Huang, A sequence and topology encoding for multicast protocol (STMP) in wireless ad hoc networks, in: Proceedings of the IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies, August 2003, pp.351–355.

[2] T.C. Chiang, C.H. Liu, Y.M. Huang, A near-optimal multicast scheme for mobile ad hoc networks using a hybrid genetic algorithm, Expert Systems With Applications (in press), vol. 33 (3), October 2007, pp. 734–742.

[3] S. Ci, M. Guizani, H.H. Chen, H. Sharif, Self-regulating network utilization in mobile ad-hoc wireless, in: IEEE Transaction on Vehicular Technology (IEEE TVT), vol. 55 (4), 2006, pp. 1302–1310.

[4] D. Devaraj, S. Veerakumar, Data communication network design using genetic algorithm, in: Proceedings of IEEE International Conference on Intelligent Systems, vol. 2, June 2004, pp. 612–616.

[5] G. Guo, S. Yu, Using the Markov chain of the best individual to analyze convergence of genetic algorithms, in: IEEE, Proceedings of the 3rd World Congress on Intelligent Control and Automation, vol.1, 28 June–2 July, 2000, pp. 512–515.

[6] M. Gen, K. Ida, J.R. Kim, A spanning tree-based genetic algorithms for bicriteria topological network design, in: Proceedings of IEEE International Conference on Evolutionary Computation, May 1998, pp. 15–20.

[7] A.T. Haghighat, K. Faez, M. Dehghan, GA-based heuristic algorithms for QoS based multicast routing, Journal of Knowledge-Based Systems 16 (5–6) (2003) 305–312.

[8] A.T. Haghighat, K. Faez, M. Dehghan, A. Mowlaei, Y. Ghahremani, A genetic algorithm for steiner tree optimization with multiple using Pru¨ fer number, in: Proceedings of the First

EurAsian Conference on Information and Communication Technology, vol. 2510, 2002, pp.272–280.

[9] C.C. Lo, W.H. Chang, A multi-objective hybrid genetic algorithm for the capacitated multipoint network design problem, Transactions of IEEE on Systems, Man, and Cybernetics 30 (3) (2000) 461–470.

[10] L. Li, C. Li, Genetic algorithm-based QoS multicast routing for uncertainty in network parameters, in: Proceedings of Asia-Pacific Web Conference, vol. 2642, April 23–25, 2003, pp. 430–441.

[11] Q.C. Meng, T.J. Feng, Z. Chen, C.J. Zhou, J.H. Bo, Genetic algorithms encoding study and a sufficient convergence condition of gas, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, vol. 1, pp. 649–652, October 1999.

[12] Y. Qiu, Y. Liu, Notes on the convergence of genetic algorithms, in: IEEE, Proceedings of the 3rd World Congress on Intelligent Control and Automation, vol. 1 (2000), pp. 508–511.

[13] G. Rudolph, Convergence analysis of canonical genetic algorithms, IEEE Transactions on Neural Networks 5 (1) (1994) 96–101.

[14] S. Ronald, Robust encodings in genetic algorithms: a survey of encoding issues, in: Proceedings of IEEE International Conference on Evolutionary Computation, April 1997, pp. 43–48.

[15] J. Suzuki, A. Markov, Chain analysis on simple genetic algorithms, in: IEEE Transactions on Systems, Man, and Cybernetics, April 1995, pp. 655–659.

[16] S. Sesay, Z. Yang, J. He, A survey on mobile ad hoc wireless network, Information Technology Journal 3 (2004) 168–175.

[17] H.T. Tran, R.J. Harris, QoS multicast routing with delay constraints, in: Proceedings of INOC, November 2003.

[18] H.T. Tran, R.J. Harris, Solving QoS multicast routing with genetic algorithms, in: Proceedings of IEEE International Conference on ICICS-PCM, vol. 3, pp. 1944–1948, December 2003.

[19] S.Y. Tseng, Y.M. Huang, C.C. Lin, Genetic algorithm for delay and degree-constrained multimedia broadcasting on overlay networks, Computer Communications 29 (17) (2006) 3625–3632.

[20] M. Wall, GAlib: A C++ Library of Genetic Algorithm Components. Available at: <http://lancet.mit.edu/ga>, August 1996.

[21] B. Wang, S.K.S. Gupta, On maximizing lifetime of multicast trees in wireless ad hoc networks, in: Proceedings of IEEE International Conference on Parallel Processing, October 2003, pp. 333–340.

[22] Y. Xiao, M. Guizani, Optimal paging load balance with total delay constraint in macrocell–microcell hierarchical cellular networks, in: IEEE Transaction on Vehicular Technology, vol. 55 (5), 2006, pp.1309–2202.

[23] G. Zhou, Y. Zhu, X. Weng, F. Ye, The generalized approaches of genetic algorithms on constrained minimum spanning tree problems, in: Proceedings of Fifth World Congress on Intelligent Control and Automation, vol. 3, June 2004, pp. 2141–2145.