9

Optimization Of Logically Rearrangeable Multihop Lightwave Networks With Genetic Algorithms

C. Gazen
Bogazici University, Computer Engineering Department
80815 Bebek, Istanbul, Turkey
fax: (90-212)2631540/1896

C. Ersoy
Bogazici University, Computer Engineering Department
80815 Bebek, Istanbul, Turkey
e-mail: ersoy@boun.edu.tr
fax: (90-212)2631540/1896
tel: (90-212)2631500/1861

Abstract

Multihop lightwave networks are a way of utilizing the large bandwidth of optical fibers. In these networks, each node has a fixed number of transmitters and receivers connected to a common optical medium. A multihop topology is implemented logically by assigning different wavelengths to pairs of transmitters and receivers. By using tunable lasers or receivers, it is possible to modify the topology dynamically, when node failures occur or traffic loads change.

The reconfigurability of logical multihop lightwave networks requires that optimal topologies and flow assignments be found. In this paper, optimization by genetic algorithms is investigated. The genetic algorithm takes topologies as individuals of its population, and tries to find optimal ones by mating, mutating and eliminating them. During the evolution of solutions, minimum hop routing with flow deviation is used to assign flows, and evaluate the fitness of topologies.

The algorithm is tested with different sets of parameters and types of traffic matrices. The solutions found by the genetic algorithm are comparable to the solutions found by the existing heuristic algorithms.

S. Fdida et al. (eds.), *Data Communications and their Performance*

Keywords
multihop architecture, genetic algorithms, flow deviation, logically rearrangeable networks

1 INTRODUCTION

Although the optical fiber has been used very efficiently in long distance communications, its use as the medium of transmission in multi-access applications is still a problem. There are two factors limiting the use of optical fibers in local area networks (Henry, 1989). The first is the power problem. In long distance communications, the optical fiber is used between two points, the sender and the receiver, whereas in local area networks there are many users sharing a single line, and the energy associated with a signal is divided among the users. Since the successful reception of a signal requires a certain amount of energy, the number of users sharing a line is limited. The second problem is the electronic-bottleneck. Although lightwave technology can support many Tb/s of throughput, the electronic devices that carry the signals from one fiber to another, such as in a ring architecture, are at best capable of speeds of a few Gb/s. Among the various approaches for avoiding both of these problems is wavelength division multiplexing (Henry, 1989)(Goodman, 1989).

Wavelength division multiplexing is the lightwave counterpart of frequency division multiplexing for electronic transmission (Goodman, 1989). The available bandwidth on an optical fiber is divided into channels. When a sender and a receiver want to communicate, they first decide on a channel and then use that channel for transmission. Although various architectures are possible with this approach, most require quickly-tunable lasers and receivers, and this is a disadvantage.

The multihop architecture overcomes the disadvantages of relying on quickly-tunable lasers, by assigning fixed channels to sets of transmitters and receivers. In this architecture, each node has a number of receivers and transmitters that are connected to a common multi-wavelength optical network. The multihop topology is implemented by assigning channels to transmitters and receivers that are logically connected. A message may be transmitted at different wavelengths through many nodes before reaching its destination, if the sender and receiver are not assigned to a common channel.

In the original multihop architecture, nodes are connected in a perfect-shuffle topology (Acampora, 1988). This topology decreases the average number of nodes a packet has to pass through before reaching its destination to $\log n$ from $n/2$, the average for a ring topology. As a result, the total traffic capacity of the network is increased. Although the perfect shuffle topology is very efficient for uniform traffic patterns, it does not perform so well with unbalanced traffic loads (Eisenberg, 1988). Other logical connection patterns and design methods that can handle unbalanced traffic loads have also been proposed in (Sivarajan, 1991), (Bannister, 1990), and (Labourdette, 1991).

The logical multihop architecture has a number of advantages. Since the topology is logical, it can be modified to respond to node failures and traffic load changes, as long as the optical network functions properly. Compared with a simple wavelength division multiplexing architecture, this topology requires either tunable lasers or tunable receivers, neither of which need to be quickly-tunable.

To use the logically rearrangeable multi-hop lightwave networks as efficiently as possible, it is necessary to find some procedure that will find an optimum topology and flow assignment for a given traffic load. The problem has been formulated as an optimization problem and solved with an heuristic algorithm in (Labourdette, 1991). In this paper, the use of genetic algorithms on optimizing the logically rearrangeable multi-hop lightwave networks will be studied.

2 PROBLEM DEFINITION

The problem, as formulated in (Labourdette, 1991), is to find a topology, and an assignment of flows, given the traffic matrix, such that the maximum ratio of the flow to the capacity of the link it is assigned to, is minimized. Such a solution allows the traffic matrix to be scaled up optimally without exceeding the capacity of the links. However, the original problem is slightly modified so that representation and manipulation of solutions by the genetic algorithm is easier. The original problem allows two nodes to be connected by more than one link as long as the maximum number of transmitters and receivers is not exceeded. In this paper, either two nodes are connected by one link or they are not connected. The resulting formulation is:

$$\text{minimize} \quad \max_{i,j} \left\{ f_{ij} / C_{ij} \right\} \tag{1}$$

$$\text{subject to} : \quad \sum_j f_{ij}^{st} - \sum_j f_{ji}^{st} = \begin{cases} \lambda_{st}, & \text{if } s = i; \\ -\lambda_{st}, & \text{if } t = i; \\ 0, & \text{otherwise} \end{cases} \quad \forall i, s, t \tag{2}$$

$$f_{ij} = \sum_{s,t} f_{ij}^{st} \quad \forall i, j \tag{3}$$

$$\sum_j Z_{ij} = T \quad \forall i \tag{4}$$

$$\sum_i Z_{ij} = T \quad \forall j \tag{5}$$

$$Z_{ij} = 0, 1 \text{ integer} \quad \forall i, j \tag{6}$$

$$f_{ij}^{st} \geq 0 \quad \forall i, j \tag{7}$$

where f_{ij} = total flow on link from node i to node j (8)

f_{ij}^{st} = traffic flowing from node s to node t on the link from node i to node j (9)

λ_{st} = traffic due from node s to node t (10)

C_{ij} = capacity of link from node s to node t (11)

Z_{ij} = number of directed links from node i to node j (12)

T = number of transmitters / receivers per node (13)

Equations (2) and (3) are flow conservation constraints, and equations (4) and (5) are the degree constraints that set the number of transmitters and receivers per node to T. The problem minimizes the maximum ratio of flows to capacities of links, but because the topology is logical, the capacities can be assumed to be constant and C_{ij} can be removed from the objective function (1).

3 GENETIC ALGORITHMS

Genetic algorithms are a method of searching solutions in the solution space by imitating the natural selection process (Androulakis, 1991)(Holland, 1991). In genetic algorithms, a population of solutions is created initially. Then by using genetic operators, such as mutation and crossover, a new generation is evolved. The fitness of each individual determines whether it will survive or not. After a number of iterations, or some other criterion is met, it is hoped that a near optimal solution is found.

In simple genetic algorithms, solutions are represented by strings. Each string consists of the same number of characters from some alphabet. Initially, the population is a random collection of such strings. At each iteration, individuals from the population are selected for breeding with a probability proportional to their fitness. Individuals are mated in pairs by the crossover operator to generate offsprings. The crossover operator divides each parent string at the same random position and then combines the left substring from the first parent with the right substring from the second parent to produce one of the offspring. The remaining substrings are concatenated to create the second offspring. After the population is replaced with the new generation, the mutation operator is applied. If an individual is to go under mutation, then one of the characters of its string representation is selected at random, and changed to some other character. This process is repeated until the termination criterion is satisfied.

As the above description shows, the simple genetic algorithm assumes that the crossover and mutation operators produce a high proportion of feasible solutions. However, in many problems, simply concatenating two substrings of feasible solutions, or modifying single characters do not produce feasible solutions. In such cases, there are two alternatives. If the operators produce sufficient numbers of feasible solutions, it is possible to let the genetic algorithm destroy the unfeasible ones by assigning them low fitness values. Otherwise, it becomes necessary to modify the simple operators so that only feasible individuals result from their application.

4 OPTIMIZATION USING GENETIC ALGORITHMS

4.1 Strategy

The optimization of logically rearrangeable multihop lightwave networks is a difficult problem, because the physical network and therefore the mathematical formulation imposes only one restriction on the topology, that the number of incoming and outgoing arcs is fixed. Not considering the assignment of flows, the size of the solution space grows exponentially as the number of nodes in the system increases. The problem is made even more difficult by the fact that given a fixed topology, finding an optimum flow assignment is itself a complex problem. The only constraint on flow assignment is the flow conservation criterion. Therefore, attacking the problem as a whole by trying to solve both the topology and assignment problems at the same time, is almost hopeless.

A more promising approach is to divide the problem into two independent problems: the connectivity problem and the assignment problem. Solving the connectivity problem yields optimal topologies and solving the assignment problem yields optimal flow

assignments on fixed topologies. Since the genetic algorithm is much better suited for discrete optimization problems, it will be used to solve the connectivity problem. The flow assignment problem will be solved on fixed topologies with minimum-hop routing and flow-deviation. The overall algorithm is:

create initial population of topologies
while stopping criterion is not met
    evaluate each topology using minimum-hop routing
    mate individuals to create new generation
    mutate generation

The division of the problem reduces the size of the solution space that each algorithm must work on. The genetic algorithm works only on topologies without considering the optimization of flows. Therefore the individuals of the population need to satisfy conditions (4), (5), and (6) only.

The individuals of the genetic algorithm's population are graph topologies. The second problem, assignment of flows, is solved on these topologies generated by the genetic algorithm. The minimum hop routing is applied to each individual to assign flows and evaluate its fitness for survival. Since the topologies that the routing algorithm works on satisfy conditions (4), (5), and (6) already, only conditions (2), (3), and (7) are considered during routing.

4.2 Representing Graphs as Genes

Manipulation of topologies with the genetic algorithm requires that they are represented in some suitable format. Although more compact alternatives are possible, the characteristic matrix of a graph is quite an adequate representation. A graph is represented by $n*n$ bits arranged in an n by n matrix, where n is the number of nodes of the graph. A '1' in row i and column j of the matrix stands for an arc from node i to j and a '0' represents that node i and node j are not connected. The most important advantage of this representation is that it is very easy to check if a graph satisfies the degree constraints. If the number of '1's in every row and every column of the matrix equals the fixed value, then the graph satisfies the constraint. To accelerate the convergence of the genetic algorithm, the genetic operators will be defined so as to generate only feasible individuals.

4.3 The Genetic Operators

Generating Random Graphs

Generating random graphs for the initial population is not very easy because of the degree constraints. To generate a random graph, it is initialized to contain no links. At each iteration, an available bit in the matrix is chosen at random which is then changed to a '1'. In other words, a new link is added to the graph. Addition of a graph can make other bits unavailable future links, because the node might have reached its capacity for incoming or outgoing links. The unavailability of links, in turn, can necessitate that some other links be created, so that the fixed-number of links criterion is met (Figure 1).

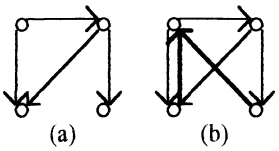


Figure 1 The links that *should* be added after random addition of some links.

The Mutation Operator

This operator takes a single graph and then modifies it by removing one of its links and adding a different link. The result is a different graph still satisfying the constraints. The algorithm is:

let (u,v) be some link chosen at random
find (x,y) randomly such that (u,y) and (x,v) do not exist
remove (u,v) and (x,y)
add (u,y) and (x,v)

The algorithm works well with the representation because the graph matrix can easily be searched to find the links and then be modified.

The Crossover Operator

This operator takes two graphs and produces two offsprings. First, it superimposes the matrices of the two graphs to find the different entries. The graphs can differ at a position in two ways. Either the first has a link and the other does not, or the first does not have a link and the second does. The positions of the differing entries are marked as '01' or '10' on a separate matrix. On this graph, a closed path with corners at the differing bits is found at random. The path should also have the property that, along the path consecutive corners are of different types: A '10' corner should be followed by a '01' corner and a '01' corner by a '10' corner. Flipping the original graphs' bits that are at the same positions as the corners of the path, create two offsprings that still satisfy the constraints. An example will clarify the procedure:

$$g=\begin{matrix}0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\\1&0&0&0&1\\1&1&0&0&0\end{matrix},\qquad h=\begin{matrix}0&0&1&1&0\\1&0&1&0&0\\1&0&0&1&0\\0&1&0&0&1\\0&1&0&0&1\end{matrix}$$

To mate graphs g and h, a temporary graph matrix is created to show the positions of the differing bits:

$$t = \begin{matrix} 0 & 10 & 0 & 01 & 0 \\ 01 & 0 & 0 & 10 & 0 \\ 01 & 0 & 0 & 0 & 10 \\ 10 & 01 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 01 \end{matrix}$$

On this graph, a closed path, such as (1,2)-(1,4)-(2,4)-(2,1)-(4,1)-(4,2)-(1,2), is randomly found. Flipping the bits at these positions in the original matrices produces the two offsprings:

$$offg = \begin{matrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{matrix}, \quad offh = \begin{matrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{matrix}$$

4.4 Evaluating the Graphs

At each iteration, the individuals need to be evaluated to determine their fitness. Since the problem is to minimize the maximum flow on any link, evaluation requires that flows be assigned to the links. The flow assignment problem is solved with the minimum-hop routing and the flow deviation method. During minimum-hop routing, the flow matrix is built as the shortest paths are found. This allows the algorithm to choose the least used path, when alternate paths exist. Still, minimum-hop routing is not adequate because it does not allow flows to be split between alternate paths. To overcome this deficiency, flow deviation is used.

The flow deviation method works with a flow assignment algorithm, in this case minimum-hop routing. Given a flow assignment, the flow deviation method removes the most heavily used link and then the assignment problem is solved on the resulting graph. The two solutions are then combined linearly so as to improve the result. The best linear combination constant is found by fibonacci search (Zangwill, 1969). The algorithm is:

while there is significant improvement
given graph g and flow assignment matrix f, find the most heavily used link (i,j)
remove link (i,j) from g
find the flow assignment matrix fr on reduced graph g
*find, by fibonacci search, the constant k such that k*f+(1-k)*fr is optimal*
*assign f = k*f+(1-k)*fr*
add link (i,j) back to g

This algorithm, however, is not very effective on graphs with a small number of links, because at the end of the first iteration, the flow assignment matrix contains two very

close entries, and removal of any of them in the second iteration causes the other to increase, making improvement impossible. Therefore, the flow deviation method is modified so that the most heavily used link and any other links with close amounts of flow are removed. The flow deviation continues until the removal of the highly loaded links causes the graph to be disconnected.

Although the flow deviation method is successful in decreasing the maximum flow, it has the disadvantage of being a number of times slower than simple minimum hop routing. Each evaluation by flow deviation requires a number of calls to the minimum-hop routing algorithm, and optimization by Fibonacci search between these calls. Instead of simply using flow deviation to evaluate every individual in the population at each generation, it is also possible to improve the flow assignment of the fittest solution without affecting its survival probability.

One problem with assigning flows for evaluating graphs is the case of disconnected graphs. If the traffic matrix is not disconnected, then a disconnected graph cannot carry all the traffic. In such a case, one possibility is to make the survival probability of the disconnected topology equal to zero, and therefore eliminate it in the next generation. However, it may be the case that mating the disconnected graph generates a proper offspring, so it is better to assign a minimal selection probability to disconnected graphs.

4.5 Stopping Criteria

A disadvantage of optimization with genetic algorithms is the difficulty of deciding when to stop. Although statistical variables, such as average fitness of a generation and best fitness value in a generation, are available, their values change almost erratically as generations evolve. Stopping after a certain number of iterations with no improvement or when the change in average fitness is small may cause the algorithm to stop too early or too late, and yet introduces even more parameters that depend on problem size. The stopping criterion used in this paper uses an exponential average of the average fitness values. At each generation, the current value of the exponential average and the average fitness value are combined with the equation $exp\_avg=\alpha*exp\_avg+(1-\alpha)*avg\_fit$, $0<\alpha<1$. Assigning small values (0.05) to α, makes it possible to trace the general behavior of the average fitness value. A good stopping criterion is then to stop when the change in the exponential average is small.

5 RESULTS AND DISCUSSION

5.1 Counting the Graphs

Before running the genetic algorithm, it is a good idea to study the number of distinct individuals that can be generated. A recursive function that yields the distinct number of graphs that satisfy equations (4), (5), and (6) for T=2 will be defined here. The function has three parameters: the number of source nodes still to be connected, the number of destination nodes still requiring two links, and the number of destination nodes still requiring one link. Adding two outgoing links to a node has three outcomes. The two links may both be connected to nodes with no incoming links yet. This decreases the second parameter by two, and increases the third parameter by two. Another possibility is that, one of the links may be

connected to a node with no incoming links and the other to a node with one incoming link. This decreases the second parameter by one, and does not affect the third parameter. Finally, both links may be connected to nodes with one incoming link. This decreases the third parameter by two, and does not affect the second. In all cases, the first parameter is decreased by one. To account for selecting different combinations of destination nodes, each of the three possibilities is multiplied by the number of combinations possible:

$$\mathrm{f}(nodes,twos,ones)=\binom{twos}{2}\mathrm{f}(nodes-1,twos-2,ones+2)+$$
$$\binom{twos}{1}\binom{ones}{1}\mathrm{f}(nodes-1,twos-1,ones)+\binom{ones}{2}\mathrm{f}(nodes-1,twos,ones-2) \tag{14}$$

$$\mathrm{f}(1,twos,ones)=\begin{cases}1 & \text{if } twos=0 \text{ and } ones=2\\ 0 & \text{otherwise}\end{cases} \tag{15}$$

$$\binom{n}{r}=\begin{cases}\dfrac{n!}{(n-r)!r!} & \text{if } n\geq r\geq 0\\ 0 & \text{otherwise}\end{cases} \tag{16}$$

The values of the function for n=2-8 are shown in Table 1. Even for the small network of eight nodes, the number of different topologies is very large. Therefore, it is not possible to try all the possibilities exhaustively.

Table 1 Number of distinct graphs for different graph sizes

Number of nodes	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Number of graphs	1	6	90	2040	67950	3110940	187530840

5.2 Experimenting with the Genetic Algorithm

The traffic data used in testing the genetic algorithm was taken from (Labourdette, 1991). The networks contained eight nodes, each with two receivers and two transmitters, and the traffic data consisted of four different specific types of non-uniform traffic data. To test the algorithm's performance with different graph sizes, the matrices were extended without modifying their underlying structures.

Parameters and Performance

The performance of the genetic algorithm is greatly influenced by a number of parameters, namely the number of individuals in the population, crossover rate and mutation rate. To achieve the best of the genetic algorithm, it is necessary to experiment with these parameters and determine an optimum set. Figure 2 shows the effect of mutation rate on the population. Normally the genetic algorithm converges by settling on some best-fit individuals, but when

the mutation rate is too low, the genetic algorithm will converge too soon, before the global optimum solution is found. As the mutation rate is increased, the standard deviation of the fitnesses of the individuals remains at a high level even after some number of generations. Again, too high a rate may be a problem if the number of mutations does not permit the population to become stable.

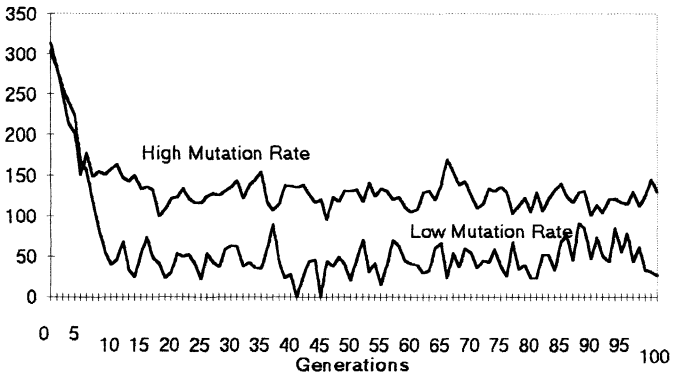


Figure 2 Graph showing effects of mutation rate.

Crossover rate is a parameter closely related to the mutation rate and has similar effects on the evolution of the population. With high crossover rates, the probability that the fittest individual of a generation will mate with some other individual increases, and the best fitness values are not very stable. On the other hand, low crossover rates have the disadvantage of being slow in exploring the solution space.

Using flow deviation method for evaluating each individual in every generation improves the quality of results in early generations but also results in a drastic slow-down, approximately by a factor of ten. Running the algorithm with minimum hop routing with more generations, and improving the final result with flow deviation provides a solution as good as the solution produced with pure flow deviation method.

Time Complexity

The genetic algorithm is almost a random search and its time performance can only be studied experimentally. One important factor, affecting the execution time, is the number of nodes in the input graph. Experiments run with different sized graphs, show that the execution time per generation increases approximately as fast as n^3 (Figure 3). However, this does not necessarily mean that the running time of the algorithm increases as n^3, since the number of generation required to find an optimal solution also varies.

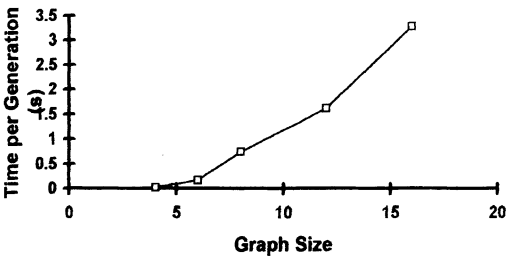


Figure 3 Time behavior of the genetic algorithm with respect to graph size.

In the basic genetic algorithm crossover over rate and mutation rate have negligible effects on running time. However, it is better to modify the genetic algorithm so that only new individuals that created by mutations and crossovers are evaluated at each generation. With this modification increases in crossover and mutation rates also increase the running time slightly.

5.3 Comparison of the Genetic and Heuristic Algorithms

To compare the genetic algorithm with the heuristic algorithm in (Labourdette, 1991), in optimizing logically rearrangeable networks, the genetic algorithm was tested with the traffic data in (Labourdette, 1991). The genetic algorithm was run with a population of 100, 60% crossover rate and 10% mutation rate on a 486DX2-66 PC. The values were collected in 20 runs with each type of traffic.

Table 2 Comparison of optimization algorithms

	Quasi-uniform traffic	*Ring*	*Centralized*	*Disconnected*
Average	6.42	15.11	34.14	32.28
Best solution	6.22	13.63	33.50	29.32
Worst solution	6.55	16.58	34.80	35.67
Running time (s)	38.64	34.63	30.64	31.28
Solution by heuristic algorithm	6.42	13.17	33.50	31.22

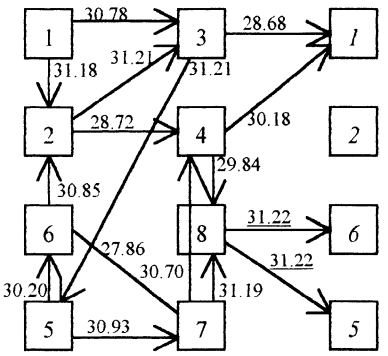
The genetic algorithm almost always performs well with quasi-uniform traffic. With other types of traffic, the algorithm produces good results also, but not in every run. This results both from the fact that most topologies behave well with quasi-uniform traffic therefore allowing the genetic algorithm to investigate the solution space freely and also from the fact that the stopping criteria cannot really decide if an optimal solution is reached.

The genetic algorithm's performance in quality of the results was comparable with those of the heuristic algorithm (Table 2). Although, most of the runs did not result in a more optimal solution, the solutions found by the genetic algorithm were within close proximity

improvements of about 30% compared with the perfect shuffle topology, the genetic algorithm outperformed the heuristic method (Figure 4).

0	10	11	9	0.9	0.8	1	1
9	0	11	9	1	0.8	1	0.9
10	12	0	8	0.9	0.9	1.1	1
8	9	10	0	1.1	1	0.8	0.7
0.7	0.8	1.1	1	0	10	11	8
1.2	0.8	0.9	0.9	9	0	9	8
0.8	1.1	1	1.1	10	11	0	11
0.9	1.1	1	1	11	8	9	0

(a)



(b) (c)

Figure 4 Comparison of the algorithms. (a) Traffic matrix (b) Flows assigned by heuristic algorithm. (c) Flows assigned by genetic algorithm.

6 CONCLUSION

The optical fiber will be playing even a more important role in communications during the next years. Its use as the medium of transmission in local area networks is made possible with multi-wavelength multi-hop architectures. To use this architecture efficiently, it is necessary to find efficient algorithms that solve the topology and flow assignment problems specific to multi-hop lightwave networks.

In this paper, genetic algorithms are used to find optimal topologies that are further improved with the flow deviation method. To apply the genetic algorithm to the problem, a suitable representation and a set of operators are defined. The operators generate only feasible solutions and keep the individuals within the solution space. Although the algorithm operates quite randomly, it produces consistent results that are comparable to the existing heuristic algorithm's results in quality. The availability of alternate configurations is advantageous in cases where node failures and traffic load changes are frequent.

Improvements to the genetic algorithm are still possible. A more compact representation, faster evaluation algorithms and very finely tuned set of parameters will increase the time performance of the genetic algorithm as well as the quality of results. In time, the genetic algorithm will prove to be even more successful in competing with other optimization algorithms.

7 REFERENCES

Acampora, A.S., Karol, M.J. and Hluchyj, M.G. (1988) Multihop Lightwave Networks: A New Approach to Achieve Terabit Capabilities. *Proc. of IEEE Int. Conf. on Comm.*, pp. 1478-1484, 1988.

Androulakis, I.P. and Venkatasubramanian, V. (1991) A Genetic Algorithmic Framework for Process Design and Optimization. *Computers and Chemical Engineering*, Vol. 15, No.4.

Bannister, J.A., Fratta, L. and Gerla, M. (1990) Topological Design of the Wavelength-Division Optical Networks. *Proc. of the IEEE INFOCOM*, pp. 1005-1013.

Eisenberg, M. and Mehravari, N. (1988) Performance of the Multichannel Multihop Lightwave Network under Nonuniform Traffic. *IEEE JSAC*, pp. 1063-1077, Aug. 1988.

Goodman, M.S. (1989) Multiwavelength Networks and New Approaches to Packet Switching. *IEEE Communications Magazine*, Oct. 1989.

Henry, P.S. (1989) High-Capacity Lightwave Local Area Networks. *IEEE Communications Magazine*, Oct. 1989.

Holland, J.H. (1991) Genetic Algorithms. *Scientific American*, Vol. 267, Jul. 1991.

Labourdette, J.F.P. and Acampora, A.S. (1991) Logically Rearrangeable Multihop Lightwave Networks. *IEEE Transactions on Communications*, Vol. 39, Aug.1991.

Sivarajan, K. and Ramaswani, R. (1991) Multihop Lightwave Networks based on De Brujin Graphs. *Proc. of the IEEE INFOCOM*, pp. 1001-1011.

Zangwill, W.I. (1969) *Nonlinear Programming, A Unified Approach.* Prentice Hall, Englewood Cliffs, NJ.

8 BIOGRAPHY

C. Ersoy is an assistant professor at the Department of Computer Science in Bogazici University. After receiving the B.S. and M.S. degrees in electrical engineering from Bogazici University in 1984 and 1986, respectively, he earned the PhD degree in electrical engineering from Polytechnic University, Brooklyn, NY in 1992. During his M.S. studies between 1984-1986, he worked as an R&D engineer in Northern Telecom, Istanbul. His research interests include optimization and networks.

C. Gazen was born in Istanbul, Turkey in 1972. He received the B.S. degree in computer science from Bogazici University in 1995. He will start studying for the PhD degree in computer science in the University of Southern California in Fall 1995, where he will be employed as a graduate research assistant, working on molecular robotics.