

# Genetic algorithms for designing multihop lightwave network topologies

C. Gazen, C. Ersoy\*

*Bogazici University, Computer Engineering Department, 80815 Bebek, Istanbul, Turkey*

Received 3 September 1996; received in revised form 3 October 1998; accepted 8 October 1998

## Abstract

Multihop lightwave networks are a means 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 article, optimization of these logical topologies 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 and the solutions are compared against histograms of random samples from the solution space. These tests show that the solutions found by the genetic algorithm are comparable with and in some cases better than those found by existing heuristic algorithms. © 1999 Elsevier Science Ltd. All rights reserved.

*Keywords:* Genetic algorithms; Multihop lightwave networks; Logically rearrangeable networks

## 1. Introduction

Telecommunication networks using lightwave technology have become very popular owing to the large bandwidth of the optical fiber. One of the strong candidate architectures for using this large bandwidth is the multihop lightwave network based on wavelength division multiplexing (WDM) [1,2]. In this architecture, all nodes are connected to the same optical medium which may have different physical topologies, such as bus, star or tree as shown in Fig. 1. Although all users are connected to the same medium, WDM scheme takes advantage of the large bandwidth of the fiber by allowing many concurrent channels. Each node has a small number of transmitters and receivers. One wavelength is dedicated to each channel between two users. Because of the limitation on the number of transmitters and receivers at each user, there is no direct channel between all user pairs. Thus, packets destined to a node may reach their destinations by hopping through a sequence of intermediate nodes. Separate channels created by assigning different wavelengths to receiver transmitter pairs defines

the logical connectivity. This logical topology is independent of the underlying physical topology and can be changed by a different wavelength assignment.

The original logical topology, ShuffleNet, was proposed by Acampora et al. [3] and had several stages connected by a perfect shuffle as shown in Fig. 2. Each directional link in this connectivity pattern corresponds to a channel created by a wavelength carrying signals between two user nodes. Acampora et al. have shown that ShuffleNet achieves high efficiency for uniform traffic loads [3].

Eisenberg and Mehravari have proved that the performance of the architecture degrades under nonuniform traffic requirements [4]. Different logical topologies and topological design techniques were proposed for overcoming weaknesses of the perfect shuffle topology under unbalanced traffic requirements. Sivarajan and Ramaswami have shown that de Bruijn graphs as logical topologies can support significantly more nodes than a ShuffleNet [5]. Other logical connection patterns and design methods that can handle unbalanced traffic loads have also been proposed. Some of these proposals consisted of regular topologies such as K-dimensional square lattices [6].

Recent availability of slowly tunable lasers and receivers makes it easier to change the wavelength assignments. This enables the network manager to adapt new logical topologies

\* Corresponding author. Tel.: + 90-212-263-1540; fax: + 90-212-287-2461.

*E-mail address:* ersoy@boun.edu.tr (C. Ersoy)

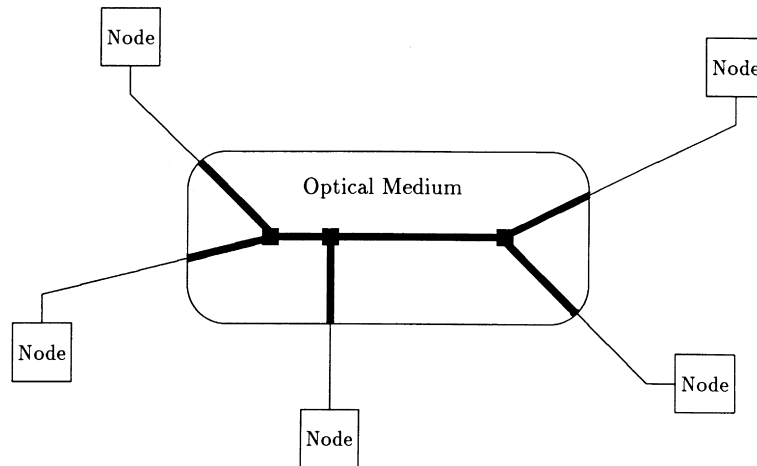


Fig. 1. An example physical topology for lightwave networks.

according to the changes in the traffic requirements or in the case of component failures. However, because of the slow response time of the lasers and receivers, the reconfiguration is also slow and not suitable for instantaneous changes. Labourdette and Acampora introduced a logical topology design problem in which the transmitters and receivers are slowly tunable [7]. They formulated the logical topology design problem as a mixed integer programming problem and proposed a heuristic which reduces the costs owing to congestion [7]. Ersoy and Panwar used simulated annealing for the optimization of the logical topology [8]. With some additional constraints owing to wavelength re-use, Zhang and Acampora proposed a heuristic wavelength assignment algorithm [9]. Ramaswami and Sivarajan proposed design algorithms for maximizing the traffic carried on the logical topology [10].

A key property of the multihop scheme turns out to be the relative independence between the logical interconnection pattern among nodes and the physical topology or fiber layout, allowing any logical connectivity diagram among the nodes to be achieved by properly choosing the transmitter/receiver wavelength assignments [7]. By deploying

optical transmitters and receivers which can be slowly tuned to any of the wavelengths in use, it becomes possible to adaptively optimize the logical connectivity with respect to prevailing non-uniform traffic patterns. The capability of dynamically reconfiguring the network can also be invoked to cope with failures of network elements since the logical connectivity could then be optimized subject to the availability of network equipment.

In this article, we propose a new method based on genetic algorithms (GA) for optimizing the logically rearrangeable multihop lightwave networks. Section 2 contains a more detailed definition and the formulation of the problem. Genetic algorithms in general are briefly described in Section 3. The description of the adaptation of GA to the described problem is given in Section 4. Results of experiments with GA and comparisons of the GA solutions to the solutions found by other algorithms are presented in Section 5. Section 6 concludes the article.

## 2. Problem definition

The problem of designing a multihop lightwave network takes the form of selecting the connection diagram, and partitioning the flow to accommodate each element of the node-to-node traffic intensity matrix among the links created by the connection diagram, in such a way that an objective function is optimized.

The performance measures of interest are the average delay over the network and the throughput achieved by the network. The delay over the network consists of a propagation delay and a transmission delay. The propagation delay is owing to the geographical distances between nodes. The transmission delay is owing to the queuing delay at each node and the access delay to a channel when the channel is shared among several source destination pairs. Classically, one might model the queuing delay at a node using the formula for an M/M/1 queue, arguing that the

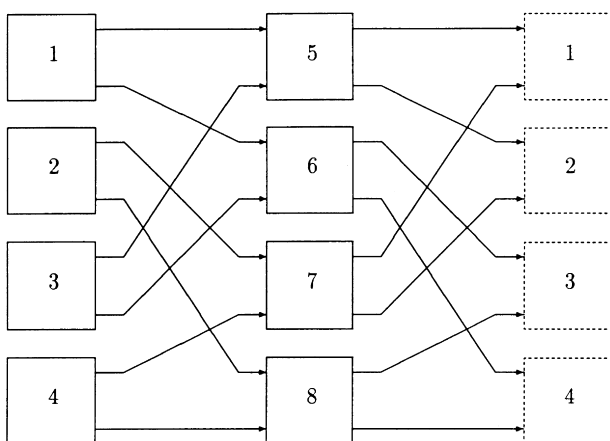


Fig. 2. The perfect shuffle connectivity diagram.

expression captures the qualitative behavior of the real network. The access delay to the channel is dependent upon the media access protocol used by the nodes accessing the common channel or wavelength, and may be very difficult to model. Achievable throughput is a critical performance measure in local and metropolitan area networks [11]. With this criterion, we seek a solution for which the traffic matrix can be scaled up by the greatest factor without exceeding the channel capacity. Hence, our objective is minimizing the largest traffic flowing on any link of the derived connectivity diagram. With such a solution, wavelength assignment and routing remain unchanged when the traffic matrix is scaled up until the channel capacity is reached [7,11]. The resulting formulation is:

Problem P:

$$\text{minimize } \max_{i,j} \{f_{ij}/C_{ij}\} \quad (1)$$

$$\text{subject to: } \sum_j f_{ij}^{st} - \sum_j f_{ji}^{st} = \begin{cases} 1_{st}, & \text{if } s = i; \\ -1_{st}, & \text{if } t = i; \forall i, s, t \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

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

$$\sum_j Z_{ij} = T \quad \forall i, \quad (4)$$

$$\sum_i Z_{ij} = T \quad \forall j, \quad (5)$$

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

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

where

$$f_{ij} = \text{total flow on link from node } i \text{ to node } j \quad (8)$$

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

$$1_{st} = \text{traffic due from node } s \text{ to node } t \quad (10)$$

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

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

$$T = \text{number of transmitters/receivers per node} \quad (13)$$

Eqs. (2) and (3) are flow conservation constraints, and Eqs. (4) and (5) are the degree constraints that set the number of transmitters and receivers per node to  $T$ . Constraints (6) and (7) are non-negativity and integrality constraints. If needed other constraints on the logical topology and routing can be added to the problem. The goal is minimizing 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).

Problem P is a mixed integer optimization problem with a min–max type objective function. It is a difficult optimization problem as noted in [7]. Since search space grows at least as fast as  $N!$  (when  $T = 1$ ), and even faster for larger  $T$ , it is impractical to determine the optimum logical topology by exhaustive search even for small values of  $T$  and  $N$ . We used genetic algorithms for solving this logical topology design problem and compared the solutions found by GA to those of the existing heuristics.

### 3. Genetic algorithms

Genetic algorithms were pioneered by John Holland during the mid-1970s in the field of machine learning [12]. A method of searching solutions in the solution space by imitating the natural selection process [13–16]. 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 offsprings. 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 description given earlier 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 [17]. In such cases, there are two alternatives. If the operators produce sufficient number 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

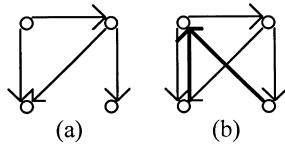


Fig. 3. The links that should be added for not violating the degree constraints are shown with bold lines in (b) after random creation of initial links in (a).

the simple operators so that only feasible individuals result from their application.

Genetic algorithms were used in a wide variety of optimization tasks, including numerical optimization and such combinatorial optimization problems as circuit layout and job–shop scheduling [18]. They have lately been used in a number of signal processing problems such as optimum IIR filter design, optimal selection of nonlinear models, speech processing and image processing [19]. Akarun et al. have applied a GA to the problem of optimally assigning quantization colors in images [20].

#### 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 [21]. 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  $n \times n$  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 node  $j$  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 constraints. 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

###### 4.3.1. 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 link can make other bits unavailable as 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 degree constraints are not violated as shown in Fig. 3.

###### 4.3.2. 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.

#### 4.3.3. 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:

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

To mate graphs  $g$  and  $h$ , a temporary graph matrix,  $t$ , 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 [21].

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 [22]. 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

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	67 950	3 110 940	187 530 840

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 article 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  $\alpha$ , 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 (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:

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

$$f(1, twos, ones) = 1, \quad \text{if } twos = 0 \text{ and } ones = 2, \tag{15}$$

$$f(1, twos, ones) = 0, \quad \text{otherwise,}$$

$$\binom{n}{r} = \frac{n}{(n-r)!r!} \quad \text{if } n \geq r \geq 0, \tag{16}$$

$$\binom{n}{r} = 0, \quad \text{otherwise.}$$

The value of the function counting the number of different graphs for  $n = 2 - 8$  are given in Table 1. Even for the small network of eight nodes, the number of different topologies is very large. Therefore, as we stated in Section 2, it is not possible to try all the possible solutions exhaustively.

5.2. Experimenting with the genetic algorithm

In order to compare our solutions with those of the existing heuristics, we have experimented with GA on the example problems described in [7]. These 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.

5.2.1. 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. Fig. 4 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.

Crossover rate is a parameter closely related to the

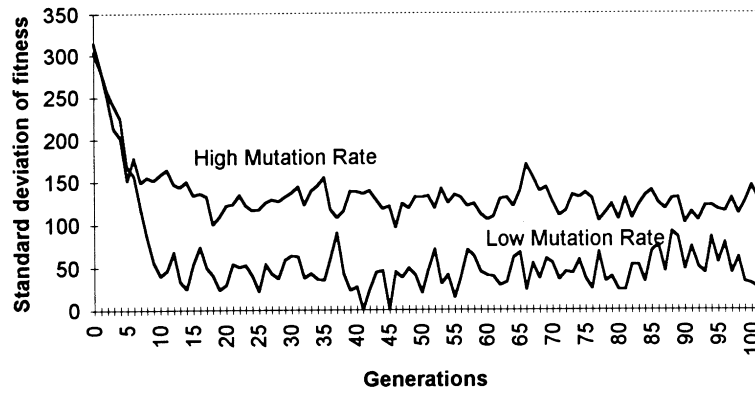


Fig. 4. The graph showing effects of mutation rate on the population.

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. Contrarily, 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.

### 5.2.2. 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 size graphs, show that the execution time per generation increases approximately as fast as  $n^3$  as can be followed in Fig. 5. However, this does not necessarily mean that the running time of the algorithm increases with  $n^3$ , since

the number of generations required to find an optimal solution also varies.

In the basic genetic algorithm, crossover 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 [7], for optimizing logically rearrangeable networks, the genetic algorithm was tested with the four different non-uniform type traffic data sets in [7]. In the *quasi-uniform* type, all requirements are approximately the same; in the *ring* type, the requirements are larger between Nodes 1 and 2, Nodes 2 and 3, Nodes 3 and 4, etc.; in the *centralized* type, all requirements to and from Node 1 are larger than the others; in the *disconnected* type, requirements are larger among Nodes 1, 2, 3, and 4; and among Nodes 5, 6, 7, and 8. 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.

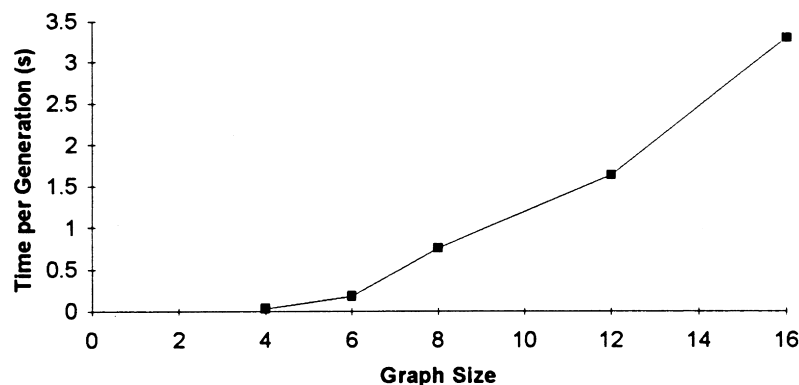


Fig. 5. Time behavior of the genetic algorithm with respect to the connectivity graph size.

Table 2  
Comparison of optimization algorithms

	Quasi-uniform	Ring	Centralized	Disconnected
Best solution (GA)	6.22	13.63	33.50	29.32
Average solution (GA)	6.42	15.11	34.14	32.28
Worst solution (GA)	6.55	16.58	34.80	35.67
Average run time (s) (GA)	38.64	34.63	30.64	31.28
Solution by the heuristic	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.

In general, the genetic algorithm’s performance in quality of the results was comparable with those of the heuristic algorithm as listed in 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 and quite acceptable. In certain cases, where the heuristic algorithm had achieved improvements of about 30% compared with the perfect shuffle topology, the genetic algorithm outperformed the heuristic method in [7] such as in the case with the disconnected type traffic matrix. Fig. 6 shows the disconnected type traffic matrix, the solution found by the heuristic and the solution found in one of the runs of GA.

We were not able to compare the time complexity of GA with the heuristic algorithm since running times were not specified in [7].

5.4. Random search versus genetic algorithm

We have also used the statistical goodness measure described in [24]. In this measure, 5000 random topologies are generated and the objective function is evaluated for each. The histogram of these values is then compared with the solutions found by the genetic algorithm. Fig. 7 summarizes the results for four different traffic matrices given in [7]. The average of the values found by 20 runs of the GA is marked with a vertical line on each histogram in Fig. 7. Histograms show that the majority of the random topologies are significantly worse than the solutions found by the GA. In fact, the best solutions found by the GA were always better than the best of random solutions. The peculiar distribution for the centralized and the disconnected traffic patterns results from the fact that while generating random topologies, a node was allowed to be connected to

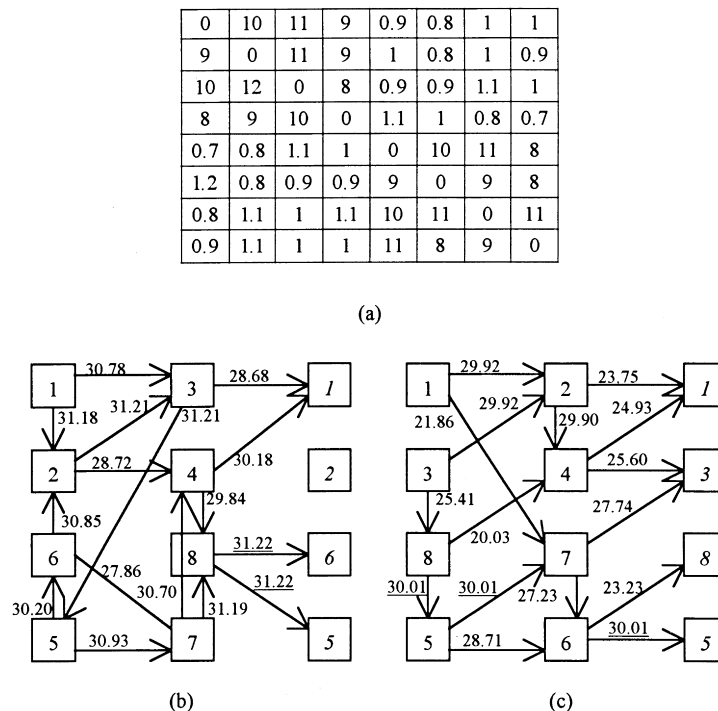


Fig. 6. Comparison of the algorithms: (a) Traffic matrix; (b) Flows assigned by the heuristic algorithm; and (c) Flows assigned by the genetic algorithm.

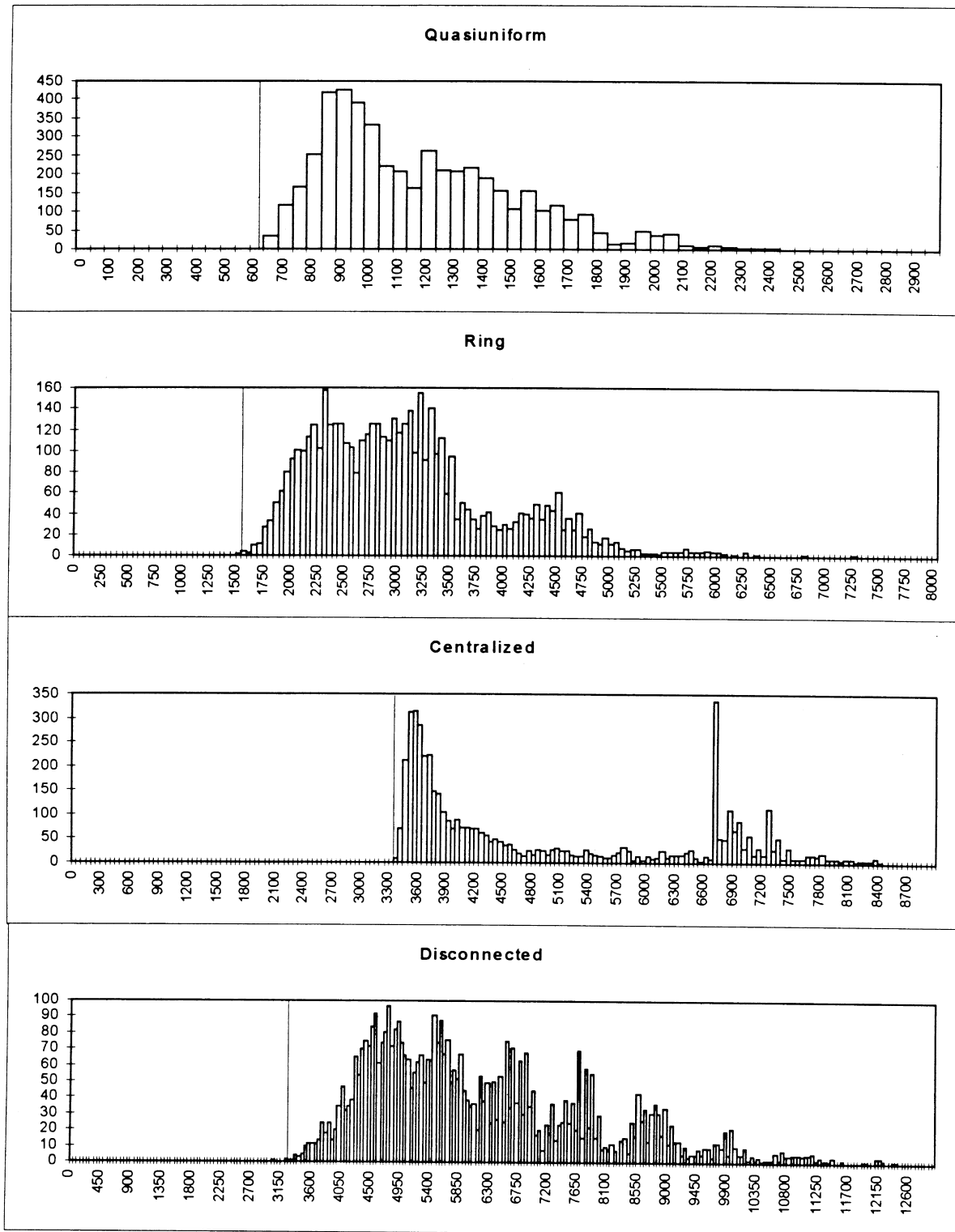


Fig. 7. Histograms of 5000 samples for each type traffic matrix. Horizontal axes show the cost of each sample  $\times 100$ , vertical lines show the average cost of GA solutions.

itself in order not to interfere with the random topology generation process. In the centralized case, wasting one of the potential links of the central node by forming a useless loop, doubles the flow on the remaining link. This causes a

second peak in the histogram, approximately at double the value of the first one. In the disconnected case, any wasted link causes a significant increase on the flow of the remaining link, so eight peaks are observed.

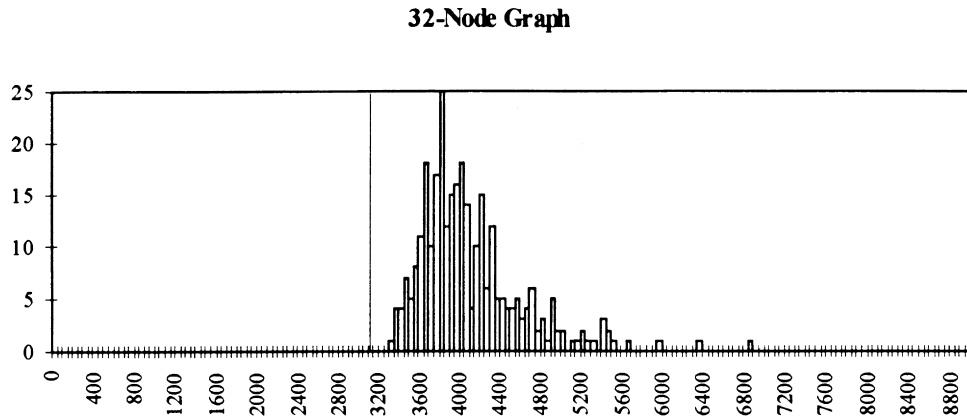


Fig. 8. Histogram of 500 samples for the 32 node problem. Horizontal axis shows the cost of each sample  $\times 100$ , the vertical line shows the average cost of GA solutions.

In order to see the behavior of the algorithm on larger problems, we have experimented with 32 node examples. Fig. 8 shows the histogram for a sample of 500 random topologies evaluated and the best solution located by five different runs of the genetic algorithm for this larger problem. All solutions found by the genetic algorithm in five runs were better than the best random solution.

## 6. Conclusion

The optical fiber will be enjoying an ever increasing importance in communications in the coming years. Its use as the medium of transmission in local area networks is made possible with multi-wavelength multihop architectures. To use this architecture efficiently, it is necessary to find efficient algorithms that solve the topology and flow assignment problems specific to multihop lightwave networks.

In this article, 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 better than solutions found by a random search and that are comparable to the existing heuristic algorithm's results in quality. As in the case with disconnected type traffic requirements, GA may find consistently better solutions than the existing heuristics. Since GA is a parallel search technique, it stops with multiple high quality solutions. The availability of alternate configurations is advantageous in cases where node failures and traffic load changes are frequent.

Because of its flexibility GA can easily be adapted to lightwave network design problems with different objective functions such as maximum throughput and minimum average delay. It is also easy to incorporate additional constraints such as those owing to propagation delay,

physical topology and tunability constraints of receivers and transmitters. Although GA produced high quality solutions for this problem, this required long running times as observed in [17,23] on similar optimization problems. 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 algorithms may prove to be even more successful in competing with other optimization techniques.

## References

- [1] Acampora AS. Intelligent optical networks. *Proceedings of the IEEE* 1993;81:111–131.
- [2] Goodman MS. Multiwavelength networks and new approaches to packet switching. *IEEE Communications Magazine*, October 1989.
- [3] Acampora AS, Karol MJ, Hluchyj MG. Multihop lightwave networks: a new approach to achieve terabit capabilities. *Proc IEEE Int Conf Comm*, 1988, pp. 1478–1484.
- [4] Eisenberg M, Mehravari N. Performance of the multichannel multihop lightwave network under nonuniform traffic. *IEEE JSAC* 1988;16:1063–1077.
- [5] Sivarajan K, Ramaswami R. Multihop lightwave networks based on De Bruijn Graphs. *Proc IEEE INFOCOM*, 1991, pp. 1001–1011.
- [6] Marsan MA, Bianco A, Leonardi E, Neri F. Comparison of regular topologies for all-optical networks. *Proc IEEE INFOCOM*, 1993, pp. 36–47.
- [7] Labourdette JFP, Acampora AS. Logically rearrangeable multihop lightwave networks. *IEEE Transactions on Communications* 1991;39:1223–1229.
- [8] Ersoy C, Panwar SS. Topological design of multihop lightwave networks. *Proc IEEE GLOBECOM*, 1993, pp. 1803–1807.
- [9] Zhang Z, Acampora AS. A heuristic wavelength assignment algorithm for multihop WDM networks with wavelength routing and wavelength re-use. *IEEE/ACM Transactions on Networking*, June 1995, Vol. 3, pp. 281–288.
- [10] Ramaswami R, Sivarajan KN. Routing and wavelength assignment in all-optical networks. *IEEE/ACM Transactions on Networking* 1995;3:489–500.
- [11] Bertsekas D, Gallager R. *Data networks*. 2. Englewood Cliffs, NJ: Prentice-Hall, 1992.

- [12] Holland JH. *Adaptation in neural and artificial systems*. Ann Arbor: University of Michigan Press, 1975.
- [13] Davis L. *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold, 1991.
- [14] Goldberg DE. *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley, 1989.
- [15] Holland JH. Genetic algorithms. *Scientific American* 1991;265.
- [16] Androulakis IP, Venkatasubramanian V. A genetic algorithmic framework for process design and optimization. *Computers and Chemical Engineering* 1991;15(4).
- [17] Baluja S. An empirical comparison of seven iterative and evolutionary function optimization techniques. Technical Report CMU-CS-95-193, School of Computer Science, Carnegie Mellon University, 1995.
- [18] Mitchell M. *An introduction to genetic algorithms*. Cambridge, MA: MIT Press, 1996.
- [19] Tang KS, Man MF, Kwong S, He Q. Genetic algorithms and their applications. *IEEE Signal Processing Magazine* 1996;13:22–37.
- [20] Akarun L, Tasdizen T, Ersoy C. Color quantization with genetic algorithms. *Signal Processing: Image Communications*, 1998; 12:49–57.
- [21] Fratta L, Gerla M, Kleinrock L. The flow deviation method: an approach to store-and-forward communication network design. *Networks* 1973;3:97–133.
- [22] Zangwill WI. *Nonlinear programming, a unified approach*. Englewood Cliffs, NJ: Prentice Hall, 1969.
- [23] Lakshmi V. Genetic algorithms for uncapacitated network design, MS Thesis in EE and CS Department, MIT, 1995.
- [24] Fetterolf PC, Anandalingam G. Optimizing interconnection of local area networks: an approach using simulated annealing. *ORSA Journal on Computing* 1991:275–287.