Heuristic Algorithms for Designing Minimum Cost FSO Networks

Shanshan Chen, Samuel Cheng, Pramode Verma, and Robert Huck
School of Electrical and Computer Engineering
University of Oklahoma, Tulsa, OK, 74135
Email: {shanshan.chen, samuel.cheng, pverma, rchuck}@ou.edu

***Abstract* — This paper presents six heuristic algorithms for designing minimum cost Free Spaces Optics (FSO) networks. The criterion used for minimizing cost is the same as minimizing the total number of links in the network or, equivalently, minimizing the number of transceiver pairs.**

Keywords-FSO; topology; minimum cost; minimum number of links

I. Introduction

FSO networking is emerging as a viable networking option in several situations. Generally, the focus of the research is on ways to create a topology that provides reliable connectivity [1], [2]. However, the major cost of an FSO network lies in the transmitters and receivers, and the total cost increases as the number of links needed increases. This is because unlike a wireless network that could support omnidirectional antennas, FSO links are highly focused point-to-point links. Consequently, in an FSO network, for each two nodes that need to directly communicate, a transceiver pair is mandatory. Therefore, the task of minimizing the cost of an FSO network is actually to generate a topology having the fewest number of links, or equivalently, fewest number of transceiver pairs, required to meet the necessary communication demand.

This brings up an unaddressed problem, which is how to minimize the number of links in the topological design for FSO networks. While design of other networks has been widely studied (for example, [3], [4], [5]), there is not much research on FSO network design. In this paper, we develop heuristic algorithms to minimize the cost of a FSO network and analyze the capabilities of the algorithms.

FSO nodes are installed to satisfy the communication demand of buildings located close together having a direct line of sight. The traffic between any two buildings may not be the same considering the different number of people in each building and the characteristics of traffic demand. Therefore, the location of sources and destinations of traffic and the amount of traffic between any pair of nodes is a pre-requisite for designing a cost-effective FSO network [6].

This paper is organized as follows. In Section II, we give the rationale and an overview of our proposed algorithms. In Section III, we present the detailed implementation. In Section IV, we present the findings of the analysis and the results. Section V captures our conclusions.

II. Framework of our algorithms

A conventional method to generate a topology with a minimum number of links in a graph usually starts by constructing a minimum spanning tree. However, this method would not apply to our situation because the minimum spanning tree might not be able to carry all the traffic specified.

To limit the *internal traffic* within the network, it is reasonable to always establish a direct link among nodes that have the maximum traffic first. This is because a direct link between nodes with high traffic will likely minimize the number of links needed. This will also avoid the consequence of a large amount of traffic occupying more than the necessary number of links and thus become a burden to the whole network. On the contrary, a new link should not be added blindly. In order to satisfy all the demand requests with as few links as possible, it is desirable to utilize the established links before adding new links to the network.

Occasionally, the traffic demand between two nodes can only be partially satisfied by the currently established links. In this case, we have to decide if we want to split the demand traffic and allocate some of it to the establish links or simply add a new link.

In a nutshell, our approach designs a FSO network by sequentially assigning a path for the curently largest traffic demand in which the path maximally utilizes the already established links. To simplify the implementation, we consider three different graphs: 1) a graph containing all possible links; 2) a graph containing only links that have already been established; 3) a graph containing only links that have already been established and have sufficient remaining capacities for the target traffic demand. Note that any path belonging to the second graph will only utilize the already established links. Moreover, any path belonging to the third graph will only utilize the already established links and a newly assigned path will be able to accomodate the entire desired traffic. Therefore, to effectively use the already established links, one should try to assign a path belonging to the second or third graph first. If no such path exists, we can then resort to paths that belong to the first graph.

For the sake of minimizing the traffic load within the network, it is evident that the selected path should have the least number of hops. To achieve this, we can simply apply the Bellman-Ford algorithm to find the path with minimum number of hops [6], [7].

III. Implementation of our Algorithms

Before we describe the detailed implementation of our algorithms, we clarify the several assumptions we have made for implementation as follows:

1) All the FSO nodes are visible to each other, and all the links have enough margin of transmission.
2) The graphs we deal with are undirectional graphs, i.e., the demand request from Node A to Node B is equal to those from Node B to Node A. This also means the matrices we use in our algorithms are symmetric.
3) The link with a saturated load, i.e., the link with traffic load equal to capacity, is considered not usable, which eliminates it from carrying any additional traffic.

For convenience, we define the common components used in these algorithms. Because the most intuitive way to present graphs is by using adjacency matrices [8], [9], we adopt matrices to generate and record the variables we use.

Demand Matrix: This matrix, Demand(i,j), records the (residual) demand request between each pair of nodes, Node i and Node j. The initial value of the demand matrix is the original traffic demand specified.

Load Matrix: This matrix, Load(i,j), records the traffic load between each two nodes, Node i and Node j, that has been currently assigned between these nodes.

Capacity Matrix: This matrix, Capacity(i,j), records the capacity between each two nodes, Node i and Node j.

Update Demand Matrix: This process updates the demand between the two nodes to zero when the demand is completely satisfied; otherwise, this process updates this demand to the previous demand minus the satisfied demand.

Update Load Matrix: This process is run at the end of each iteration, because the Load Matrix is changed after the demand is loaded between one or more node pairs. It updates the load between the node pairs by adding the traffic loaded in the last iteration. This process is the basis for the graphs to be updated in each iteration.

We also generate three different graphs represented by adjacency matrices to store the links created and occupied for each solution. G_0 is the graph including all links that are not saturated (excluding those with no remaining capacity). G_1 is the graph that includes all established links that are not saturated. And G_2 is the graph that includes all the established links that can both satisfy the current demand request and are not saturated.

Let V be the vertex set containing all potential FSO nodes. We can write the three graphs more precisely as $G_0 = (V_0, E_0)$, $G_1 = (V_1, E_1)$, and $G_2 = (V_2, E_2)$, where their edge sets, E_0, E_1, and E_2, are respectively defined as
$E_0 = \{(i, j) \mid \text{Load}(i, j) < \text{Capacity}(i, j),\ i, j \in V\}$;
$E_1 = \{(i, j) \mid 0 < \text{Load}(i, j) < \text{Capacity}(i, j),\ i, j \in V\}$;
$E_2 = \{(i, j) \mid 0 < \text{Load}(i, j) < \text{Capacity}(i, j) - \text{Request demand},\ i, j \in V\}$.

And their corresponding vertex sets, V_0, V_1, V_2, are defined by $V_l = \{i | (i, j) \in E_l\}$, $l = 0, 1, 2$. Note that $G_0 \supset G_1 \supset G_2$.

Specifically, there are two ways to select the demand request to be satisfied in each iteration. One way to select the demand request to be satisfied is to find the maximum requested demand in the demand matrix and identify the two nodes needing a path to satisfy the demand. We classify algorithms using this node selection method into category A. The other way to select a demand request is to identify the node with maximum demand. This can be done by summing each row of the demand matrix. The row with the maximum sum represents the node with the maximum demand. We classify algorithms using this node selection method into category B. For each category, we can further order the path searching steps according to three different combinations as follows.

1) G_1-G_0 Scheme: In this scheme, we search for the possible paths in G_1 first. Note that the links along a path in G_1 may or may not have sufficient remaining capacity to accommodate the entire request demand. If a path is found and has sufficient capacity for the demand, the current demand will be assigned completely to the path. However, if a path exists but does not have sufficient capacity, the maximum available capacity will be subtracted from the current request and assigned to the path. If no path can be found in G_1, the algorithm will search graph G_0, which includes all possible links.

2) G_2-G_1-G_0 Scheme: In this scheme, we search for the possible paths in G_2 first, which includes all the links that will satisfy the current demand request completely. If no such paths can be found in G_2, we then search for the possible paths in G_1. If the path cannot even be found in G_1, the algorithm will search graph G_0, which includes all possible links.

3) G_2-G_0 Scheme: In this scheme, we search for the possible paths in G_2 first, which includes all the links that will satisfy the current demand request completely. If there is no such path found in G_2, the algorithm will search G_0.

IV. Simulations and Results

Two test cases have been conducted sharing the following common assumptions:

1) The capacity of each link is equal.
2) The demand between two nodes is symmetric.

A. Demand Matrix with Equal Demand

In this case, we restrict the demand between any two nodes to be equal. The number of nodes is set to 10 and the demand between each pair of nodes is set to 10 also.

Fig. 1 illustrates how the number of links needed changes with the link capacity. Interestingly, we can see from Fig. 1 that the required number of links (thus the cost of the network) can significantly decrease as the link capacity increases.

Moreover, it is apparent that the minimum link capacity required to satisfy the demand is equal to 10 (i.e., the demand between each pair of nodes). At this minimum capacity, all six algorithms successfully found the fully connected graph as the solution. However, not all algorithms perform equally

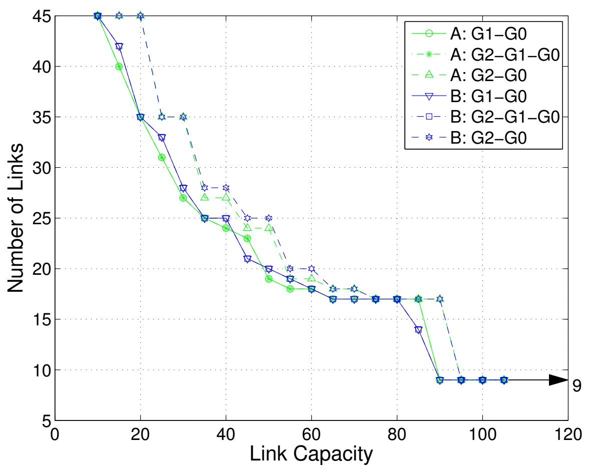

Fig. 1. Number of links versus link capacity for equal demand case.

well as we increases the link capacity. For example, when link capacity increases to 20, schemes G_1-G_0 and G_2-G_1-G_0 in both categories A and B manage to decrease the number of links to 35. On the other hand, schemes G_2-G_0 in both categories can only find the fully connected network as the their best solution. Actually, schemes G_2-G_0 perform consistently poorer than the other schemes. Since schemes G_2-G_0 will only assign path that can accomodate the entire current demand request, our result suggests that to allow splitting traffic is beneficial in this case.

Note that while the number of links decreases as the capacity increases, after the capacity exceeds a certain level, the number of links stops decreasing. It is expected since the topology now becomes a minimum spanning tree with number of links equal to just one less than the number of nodes (i.e., 9). Since each node has traffice demand of 10 to the 9 other nodes, the minimum link capacity that allows a minimum spanning tree solution is 90. At this minimum link capacity, all schemes except schemes G_2-G_0 of both categories manage to find a minimum spanning tree solution.

B. Demand Matrix with Gaussian distribution

To make our testing closer to a realistic situation, we also apply our algorithms to Gaussian distributed demand matrix. We experiment with the following setting: number of nodes = 20, capacity = 15, mean demand = 10, variance of gaussian distribution = 4. And the result is tabulated below.

	Average Number of Links	Variance of Number of Links	Failure	Number of Links in Fully connected Topology = 190
A: G_1-G_0	179.1540	21.3476	371/1000	
A: G_2-G_1-G_0	179.1840	21.0992	367/1000	
A: G_2-G_0	163.2030	12.2140	0/1000	
B: G_1-G_0	181.1150	19.1529	482/1000	
B: G_2-G_1-G_0	181.0960	18.9738	483/1000	
B: G_2-G_0	162.6900	12.1480	0/1000	

We then lower the variance by half to 2 and conduct our experiment again. The result is tabulated as follows.

	Average Number of Links	Variance of Number of Links	Failure	Number of Links in Fully connected Topology = 190
A: G_1-G_0	181.1470	3.4709	27/1000	
A: G_2-G_1-G_0	181.1470	3.4709	27/1000	
A: G_2-G_0	180.1790	6.2072	0/1000	
B: G_1-G_0	183.7650	3.1009	39/1000	
B: G_2-G_1-G_0	183.7650	3.1009	39/1000	
B: G_2-G_0	180.4910	5.9599	0/1000	

From our experiment, we found that for all schemes, there exists least capacities to guarantee that the schemes provide successful solutions for any demand matrix, and the topology generated is very near a fully connected topology. Moreover, the results generated by the G_1-G_0 scheme are quite similar to those generated by the G_2-G_1-G_0 scheme. As a matter of fact, in some situations, the output results of both schemes were even identical.

On the other hand, the G_2-G_0 scheme tends to give the best performance while dealing with deficient capacity as it provides the least number of failures. Surprisingly, while the schemes are shown to be worst for the equal demand case, for Gaussian distributed demand, it provided no failure while other algorithms failed frequently. The G_2-G_0 scheme also gave the least average number of links compared with other algorithms.

V. Conclusion

The paper has presented six heuristic algorithms for designing a topology for minimizing the number of links in FSO networks. The simulation results have shown that the best strategy depends significantly on the distribution. While schemes G_2-G_0 of both categories perform poorly for equal demand case, they perform the best among all schemes when the demand is Gaussian distributed.

References

[1] P. C. Gurumohan and J. Hui, "Topology design for free space optical networks," in *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, 2003, pp. 576–579.

[2] Z. Hu, P. Verma, and J. Sluss, "Improved reliability of free-space optical mesh networks through topology design," *Journal of Optical Networking*, vol. 7, no. 5, pp. 436–448, 2008.

[3] C. Charnsripinyo and D. Tipper, "Topological design of survivable wireless access networks," in *Design of Reliable Communication Networks, 2003.(DRCN 2003). Proceedings. Fourth International Workshop on*, 2003, pp. 371–378.

[4] L. A. Cox and J. R. Sanchez, "Designing least-cost survivable wireless backhaul networks," *Journal of Heuristics*, vol. 6, no. 4, pp. 525–540, 2000.

[5] S. Singhal and G. L. Thompson, "A method for maximizing the reliability coefficient of a communications network," *Annals of Operations Research*, vol. 4, no. 1, pp. 307–326, 1985.

[6] L. Ford and D. Fulkerson, "Flows in networks," 1962.

[7] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Q. Appl. Math.*, vol. 27, pp. 526–530, 1970.

[8] C. D. Godsil, G. Royle, and C. D. Godsil, *Algebraic graph theory*. Springer New York, 2001.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT Press, Cambridge, Ma, 2001.