A Tabu Search Based Routing Optimization Algorithm for Packet Switching Networks

DANIELE CASALI[1], GIOVANNI COSTANTINI[1,2], MASSIMO CAROTA[1]
[1]Department of Electronic Engineering
University of Rome "Tor Vergata"
Via del Politecnico, 1 – 00133 ROMA
ITALY
[2]Institute of acoustics "O. M. Corbino"
Via del Fosso del Cavaliere, 100 – 00133 ROMA
ITALY

Abstract: - In this paper, we present a tabu-search based algorithm that optimizes routing for packet switching networks. The problem of routing optimization can be seen as the search of the shortest path in a graph, where the bandwidths of connections, together with their traffic, can be considered as weights. This kind of optimization is usually carried out by means of the well-known Dijkstra algorithm or its various implementations. However, an exhaustive research tends to be very heavy, from a computational point of view, when the number of nodes gets high. For this reason, we opt for a meta-heuristic algorithm, particularly tabu search, capable of finding a non-optimal solution, that can be considered quite good, even without the need of an exhaustive research.

Key-Words: - Tabu search, optimization, routing, communication networks

1 Introduction

The routing problem for packed switching networks is a crucial problem in telecommunications, and, as long as Internet is spreading more and more all over the world, it involves a number of nodes that is difficult to manage. The most obvious way to find the optimal solution is the Dijkstra algorithm [1]: the network is considered as a graph, whose weights are bandwidths and traffic. Hence, the routing problem is simply a shortest path problem. Alternate approaches, such as neural networks [2,3], have been studied as well.

The application of a meta-heuristic algorithm could be a good alternative to the exhaustive search of Dijkstra algorithm, because it does not need to explore the whole graph. Hence, it can afford an improvement in time performances. Ant-colonization has been adopted in various works [4,5]. Tabu search [6] is a well-known meta-heuristic algorithm, and it has been applied by different authors to the routing problem, coupled with the wavelength assignment problem [7,8]. In this paper, we present a tabu search based algorithm, optimized by means of some variants that can speed up the search.

The paper is organized as follows: in the second section, we make a description of the algorithm, while the third section is dedicated to the study of the cost function. Section 4 describes our implementation, with experiments and results. Finally, conclusions are described in Section 5.

2 Description of the Algorithm

As in the case of Dijkstra algorithm, the network that we must optimize can be represented as a graph. As a consequence, the routing problem can be seen as the search of a path between two given nodes. The first difference with the Dijkstra algorithm is that, in our case, we are not going to find the shortest path, but just a "good" path, which is not granted to be the optimum. We will start from a random path, and then we will transform it, according to the tabu search rules.

The only restriction on the initial path is that it must start and end on the given nodes, as well as it cannot contain any closed loop. The path is then transformed applying iteratively operations taken from a set of three possible elementary operations, that we call OP1, OP2, and OP3.

OP1 regards two consecutive nodes in the path, A and B, connected directly by arc A-B, as well as indirectly through a third node α and arcs A-α and B-α (Fig. 1). So, according to rule OP1, we can exclude arc A-B from the path and include node α and the two arcs A-α and B-α. This way, we transform a set of two nodes and one arc into a set of three nodes and two arcs.

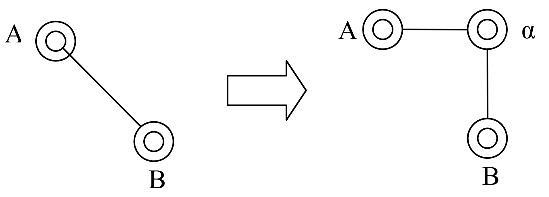

Figure 1. Operation OP1: A new node (α) is inserted in the path.

OP2 regards a set of three nodes and is the inverse operation of OP1. With this operation, shown in Fig. 2, we exclude a given node a between two nodes *A* and *B*, and include the arc *A*-*B* in the path, provided that this arc already exists in the graph. As an obvious consequence, if a node is the start node or the end node of the path, it cannot be deleted, because it is connected to one node of the path only.

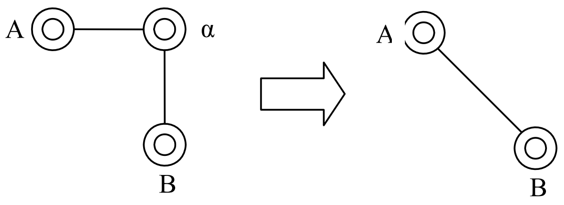

Figure 2. Operation OP2: node α is deleted from the path.

Operation OP3 is the last kind of basic operation, according to which we exclude a node α from the path and include another node β, together with arcs *A*-β and *B*-β, provided that node β already exists in the graph and it is connected with nodes *A* and *B*.

Depending on the number of arcs and their weights, we can define a "cost" of the path, which will change whenever we apply a basic operation to the path. Our aim is to minimize this cost by means of the tabu search based algorithm described in the following.

Start generating a random path from the start node to the end node that does not contain loops.

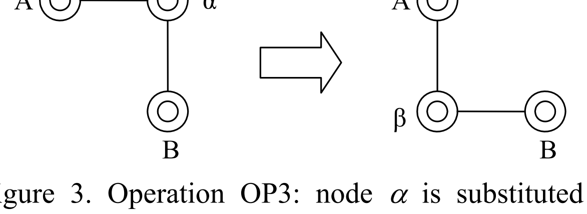

Figure 3. Operation OP3: node α is substituted with node β.

1. Call N_1 , …, N_m the nodes in the path
2. Calculate the cost of the path, according to a cost function that takes into account the weights of all arcs in the path.
3. Starting from first node, check all nodes couples (N_i, N_{i+1}) to see if OP1 is possible, and triples (N_i, N_{i+1}, N_{i+2}) to see if OP2 or OP3 is possible. If one of these operation is possible, we store in a list the path obtained by applying it to the current path, and continue until $i=m$. All path in this list are called adjacent paths.
4. Calculate the weight of all adjacent paths which are not in the tabu list. Tabu list is initially empty.
5. Insert the current path and all adjacent paths to the tabu list, and select as new current path the adjacent path which has the lowest cost. Classic tabu search would insert only current path to the tabu list. Anyway, in our application, where there can be a lot of adjacent paths, increasing the tabu list with paths that will not be probably useful will help in lightening and speeding-up the search for good paths. Finally, if the new path has a cost lower then the current path, select it as best path.

Repeat steps 1 to 5 for a given number of times.

3 Example

In order to better explain the algorithm, we will apply it to the simple graph with 7 nodes, shown in Fig. 4. We want a path from *A* to *D*. We start by simply finding any path from *A* to *D*: in this case, we start from the path *ABCD*.

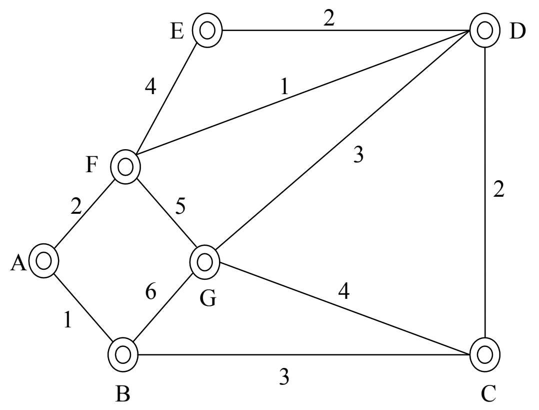

Figure 4. Example graph.

First execution of the loop

The cost of the path is 6, which is the sum of the weight of arcs *A*-*B*, *B*-*C*, and *C*-*D*. We start checking if OP1 is applicable on nodes (*A*,*B*): OP1 cannot be used, because no arc in the graph is connected with both A and B. Then we check for the applicability of OP2 or OP3 on nodes (*A*,*B*,*C*):

OP2 doesn't fit, because *A* and *C* are not connected, while OP3 cannot be used, because no node is connected with both *A* and *C*.

Next couple is (*B,C*). We see that OP1 is applicable, because node *G* is connected with both *B* and *C*. This operation yields to path *A-B-G-C-D*, which is added to the adjacent paths list. Now we continue the search for adjacent paths, by checking for OP2 and OP3 on the triple (*B,C,D*): OP2 can be used, while OP3 can't, because node *G* is connected both with *B* and *C*. So we obtain another adjacent path: *A-B-G-D*.

Finally, the last couple in the path is (*C,D*): OP1 is applicable, because node *G* is connected with both *C* and *D*. This operation yields to the path *A-B-C-G-D*.

The list of adjacent path is:

· *A-B-G-C-D*, whose weight is 13.
· *A-B-G-D*, whose weight is 10.
· *A-B-C-G-D*, whose weight is 11.

The adjacent path with lower cost is *A-B-G-D*. So, it will be the next path. Anyway its cost is 10, greater than 6. So, the current path *A-B-C-D* is stored as best path. Path *A-B-C-D*, together with *A-B-G-C-D*, *A-B-G-D* and *A-B-C-G-D*, are stored in the tabu list. We are now ready for step two.

Second execution of the loop

First couple is (*A,B*), and OP1 is not applicable. First triple is (*A,B,G*) and OP2 is applicable. So, we can substitute node *B* with *F*, yielding to path *A-F-G-D*, which is not in the tabu list. Next couple is (*B,C*) and OP1 is applicable, yielding to path *A-B-C-G-D*. Being this path already in the tabu list, it will not be considered. Next triple is (*B,G,D*) and OP2 is not applicable, while OP3 yields to *A-B-C-D* which is in the tabu list. Last couple is (*G,D*), and two different OP1s can be applied. In fact, both *F* and *C* are connected with *G* and *D*. So, two more paths can be generated: *A-B-G-F-D* and *A-B-G-C-D*, but *A-B-G-C-D* is in the tabu list. The list of adjacent paths, excluding the paths already present in the tabu list, is:

· *A-F-G-D*, whose weight is 10
· *A-B-G-F-D*, whose weight is 13

The adjacent path with lower cost is *A-F-G-D*, so we select this path as next path, and add *A-F-G-D* and *A-B-G-F-D* to the tabu list. Best path is still *A-B-C-D*.

Third execution of the loop

Current path is *A-F-G-D*. First couple is (*A,F*) and OP1 is not applicable, while OP2 is not applicable on triple (*A,F,G*). On the contrary, OP3 generates path *A-B-G-D*, which is in the tabu list. Second couple is (*F,G*) and OP1 is not applicable, because the only node connected to *F* and *G* is *D*, that is the end node. Last triple is (*F,G,D*), where we can apply OP2, yielding to path *A-F-D*, and OP3, yielding to path *A-F-E-D*. Finally, last couple is (*G,D*): OP1 is applicable, yielding to path *A-F-G-C-D*. The list of adjacent paths, excluding paths already in the tabu list is:

· *A-F-D*, whose weight is 3
· *A-F-E-D*, whose weight is 8
· *A-F-G-C-D*, whose weight is 13

The selected path is *A-F-D*, which is also stored as best path. Tabu list is incremented with *A-F-D*, *A-F-E-D*, and *A-F-G-C-D*.

Fourth execution of the loop

Current path is *A-F-D*, but none of the possible operations on any couple yields to a path that is not present in the tabu list. Hence, the loop is halted, resulting that best path is *A-F-D*, which in this case is exactly the shortest path.

4 Cost function

Let's consider a telecommunication network represented by a graph, in which every connection is represented by an arc. The weight of every arc depends on the bandwidth and saturation of the connection.

Every time a new request is associated with a path on the graph, the weight of every arc in the path is increased according to its traffic and the data flux that we want to route on it. If we assign paths without any rule, after having complied with a lot of requests, we can cause the saturation of some arcs, which will be unavailable for subsequent requests.

For any couple of nodes, there can be many possible paths, and we must define a rule to decide which is the best path to choose. Moreover, it is important for the solution of the problem to avoid saturating arcs with an excessive number of requests. Therefore, the choice for an arc that is in danger of saturation must be very costly. Hence, we can think to the problem in terms of the minimization of a cost function: we must define a function that increases when the residual capacity of the arc is decreasing. The residual capacity is defined as the difference between the bandwidth of the arc and the data flux that is currently present in the connection.

Another parameter must be taken into account in the cost function: consider, for example, graph in Fig. 4. Suppose that we must choose arc *C-G* or arc *C-B* and that they have the same residual capacity: the two choices would have the same cost. But, actually, the choice of *C-B* is better, because if we choose *B*, then we will have only two arcs that start from it, while if we choose *G* we will have more arcs, with more available paths. If we define the degree of a node as the number of arcs connected to a given node, the cost function will depend on three parameters:

- *f* : data flux (bit/s)
- *C*: capacity of the connection
- *GN*: degree of the destination node

The cost function of an arc *i* is $F_i\ (f_i,C_i,GN_m)$, where *m* is the destination node of arc *I*.

We take as cost function the product of two functions, which take into account, separately, the node degree, and the residual capacity: $F_i(f_i,C_i,GN_m) = F_{i1}\ (f_i,C_i)\ F_{i2}(GN_m)$.

F_{i1} is defined as:

$$F_i(f_i,C_i) = \left(\frac{1}{K_1 + C_i + f_i}\right)^2$$

while F_{i2} is defined as:

$$F_i(GN_i) = \left(\frac{K_2}{GN_i}\right)^2$$

with K_1 in [0,1] and $K_2 > 1$.

In Fig. 9 we show the graph of this function, when $C_i = 3$, $K_1 = 0.2$ and $K_2 = 5$.

Obviously, given the cost of every arc in a path, the cost of the latter equals the sum of the cost of its arcs.

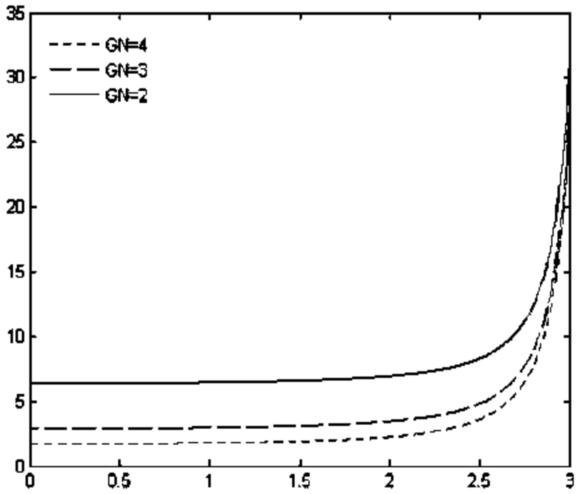

Figure 5. Graph of the cost function with three values of GN.

5 Experimental Results

We implemented the algorithm in Matlab, with a number of iteration fixed to 300 and a length of the tabu list fixed to 40. We performed two kinds of tests: in the first test, we generated a set of 1200 graphs with the same number of nodes N=50, and a number of arcs from 200 to 700, with random weights uniformly distributed from 0 to 100. For every graph, we calculated the path by means of our algorithm and the optimum path by means of the Dijkstra algorithm. Results are shown in Fig. 6, in terms of cost versus number of arcs.

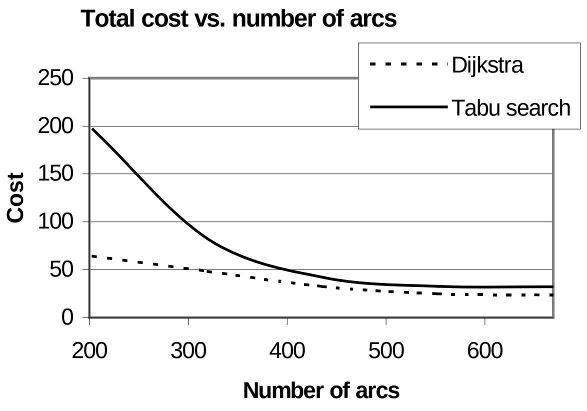

Figure 6. Graph of minimum path average cost versus number of arcs.

We can see that the performance of the algorithm tends to be nearer to the perfect one, when the number of arcs gets high: more specifically, when the number of arcs is greater than approximately 400, which is 8N, the cost remains about 1.3 times the optimal cost. In the second test, we used a variable number of nodes from 19 to 90, and 8N arcs. Results are shown in Fig. 6, where we report the cost of the minimum path versus the number of nodes N: we can see that, if the number of links increases linearly with the number of nodes, the average cost of the real minimum path, calculated by Dijkstra algorithm, tends to be constant. On the other hand, the paths obtained by the algorithm tend to have a higher cost than the optimum, when N increases. The cause is arguably due to the fact that the number of iterations is fixed to 300.

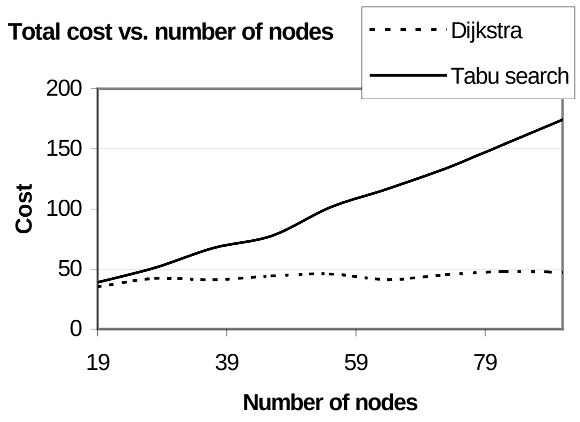

Figure 7. Graph of minimum path average cost versus number of nodes.

6 Conclusion

We designed and implemented an algorithm that, by means of a tabu-search, carries out a non-exhaustive search of best path on a given graph. Since there is no need to explore all paths, there could be a great saving of time. Of course, the found path is not granted to be always the shortest path, but the obtained results show that, if the graph is not very sparse, the probability that the result is one of the shortest paths is very high. Advantages in adopting our algorithm become evident, especially when the number of arcs is high. One way to improve the performances of the algorithm could consist in finding a better method to fix both the maximum number of iterations allowed, and the length of the tabu list, as functions of the parameters of the graph, namely the number of arcs and the number of nodes.

References:

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, pp. 269-271, 1959.

[2] N. Shaikh-Husin, M. K. Hani, Teoh Giap Seng, "Implementation of recurrent neural network algorithm for shortest path calculation in network routing", *Parallel Architectures, Algorithms and Networks, I-SPAN '02. Proceedings. International Symposium on,* pp. 313 – 317, 22-24 May 2002.

[3] Liu Rong, Liu Ze-Min, Zhou Zheng, "Neural network approach for communication network routing problem", *TENCON '93. Proceedings. Computer, Communication, Control and Power Engineering*, Issue 0, Part 30000, pp. 649 - 652 vol.3, 19-21 Oct. 1993.

[4] R. .Schoonderwoerd,, O. Holland, J. Bruten and L. Rothkrantz "Ant-based Load Balancing in Telecommunications Networks", *Adaptive Behavior*, 5(2), pp. 169-207, 1997.

[5] G. Di Caro G. and M. Dorigo, "AntNet: A Mobile Agents Approach to Adaptive Routing", *Tech. Rep. IRIDIA/97-12*, Université Libre de Bruxelles, Belgium, 1997.

[6] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence", *Computers and Operations Research,* 13, pp. 533-549, 1986.

[7] C. Dzongang, P. Galinier, S. Pierre, "A tabu search heuristic for the routing and wavelength assignment problem in optical networks", *Communications Letters, IEEE*, Volume 9, Issue 5, pp. 426 – 428, May 2005.

[8] Ying Wang, Tee Hiang Cheng, Meng Hiot Lim, "A Tabu search algorithm for static routing and wavelength assignment problem", *Communications Letters, IEEE*, Volume 9, Issue 9, pp. 841 – 843, Sep 2005.