Approximative Algorithms for Configuration of Multi-Layer Networks with Protection

Csaba Gáspár  Szabolcs Szentes  János Tapolcai  Tibor Cinkler
High-Speed Networks Laboratory, Department of Telecommunications and Telematics
Budapest University of Technology and Economics, Budapest, Hungary
{gaspár|szentes|tapolcai|cinkler}@ttt-atm.ttt.bme.hu

Abstract

A novel heuristic algorithm is proposed for wavelength routing (WR) with protection in DWDM (Dense Wavelength Division Multiplexing) optical networks. If the number of available wavelengths (WLs) over all links is sufficient, the algorithm finds a pair of direct wavelength-paths (WPs) for all demands (single-hop case). If not, paths consisting of multiple WPs are used (multi-hop case), where involvement of an upper, electrical layer is assumed, which performs traffic grooming. In this case the method iteratively optimises the wavelength system and the traffic routing at the electrical layer. Both phases search for shortest paths using modified versions of Dijkstra’s algorithm and the whole method relies on numerous heuristics - some of them borrowed from Simulated Annealing, Genetic Algorithm and Tabu Search. The method is useful for estimating (upper-bounding) the number of WLs needed for single-hop WR in networks of arbitrary topology. The performance of the algorithm is numerically compared to the case without protection paths on three realistic networks with various numbers of WLs.

1 Introduction

It has been shown in [1] that the required number of different wavelengths per fibre for networks with and without WL conversion capability is about the same. In general, for networks of practical size, the number of available wavelengths is lower by a few orders of magnitude than the number of connections to be established. The only solution here is to join some of the connections (referred to as traffic grooming), that can fit into the capacity of wavelength-links. This can be done at electrical layer only, since re-multiplexing is required. For this reason, taking not only the optical, but both, optical and electrical layer into account when configuring the system is demanded.

Many excellent papers deal with design, configuration and optimisation of WR-DWDM Networks. The widely accepted approach is to decompose the problem to the following sub-problems: First, determine the virtual topology (route the light-paths); second, assign a wavelength to each light-path (Wavelength Assignment WA); and third, route the traffic over the light-paths (see, e.g., [2],[3]). We will use decomposition as well, while trying to take interactions between these phases into account by iterations.

Our subject is to configure the WP system optimally by iteratively configuring network-layers instead of separating the problem completely into two subproblems. This improves the quality of results without increasing the complexity significantly.

As the result of optimisation we decrease the traffic to be processed and carried in the electrical domain over-bridging the speed limits of electronics. Since a considerable part of the load of electrical (e.g., SDH, ATM or MPLS) switches/cross connects is overtaken by the optical switches/cross connects much larger networks with higher loads can be realised by the current technology offering better granularity and using optimally any limited number of WLs. A similar problem has been solved in [4] with a greedy algorithm without protection. The drawback of that approach is that the complexity grows polynomially as the number of WLs is being increased.

In Section 2 we present the proposed model, in Section 3 formulate the problem, in Section 4 the proposed algorithm and problem of diverse routing are discussed, while in Section 5 the obtained results are presented.

2 The Model

The network is modeled as a “wavelength graph” which has topology as the physical network, while it consists of as many graph layers as many WLs are allowed. The graph layers are connected to each other in the network
nodes where re-multiplexing (traffic grooming) is performed or WL conversion is allowed. Only without loss of
generality the graphs are undirected and demands are assumed to be symmetrical to reduce the complexity. A
cost is assigned to each edge.

Figure 1: Illustration of the Model Used.

Figure 1.a shows the model of the routing of a traffic demand which includes several transit nodes. Some of
these nodes just simply pass the incoming optical signal to another link (for example Node 3), while some others
can make all optical WL conversion (Node 4). There is a third type of transit nodes which is able to perform
the opto-electrical transformation as well (Node 2). The signals can be multiplexed and de-multiplexed in these
nodes in the electrical layer. A path which only passes the first types of transit nodes are called wavelength path
(WP). This is an optical signal route, it has a fixed wavelength associated with, and can not be interrupted.
In Figure 1.b we can see the cost of a WP. The delay of the signal in a node where we make an opto-electrical
transformation is higher then in a node which only passes it to an other link. A traffic demand is routed over
several WPs, its cost is the sum of the cost of the used WPs. Therefore, the total cost has to be expressed as:

\[
\text{Total cost} = \sum_{p \in P} c_p \cdot b_p = \sum_{t \in T} c_t \cdot b_t
\]

where \( P \) is the set of the WPs, \( c_p \) is the cost of a \( WP_p \) and \( b_p \) is the traffic through the \( WP_p \). In the second
expression \( T \) is the set of the traffic demands, \( c_t \) is the cost of a traffic demand and \( b_t \) is the value of the traffic
over the demand. For details of the model see [5].

In the model the algorithm has been organised in two layers. The first one where WPs are routed is the
level of the physical network. The topology of the WPs is the second layer and the traffic demands are routed
over that.

3 Description of the Problem

According to the model the problem can be divided into two parts:

- Routing the wavelength-paths (WP) and associating them with a wavelength (first level).

- Routing the real traffic demands over the WPs, between two nodes of the network, using several WPs. In
other words after laying down the WPs we get a new graph of which edges are the WPs and nodes are
the end of the WPs and the traffic demands have to be routed over this new graph (second level).

The best solution would be routing all traffic demands by an own WP, but the number of the wavelengths
is insufficient to do this, therefore, we have to brake up the traffic demands into more WPs, and some of traffic
demands will be multiplexed and will share a common WP.

The topology of the physical layer, the type of each node (e.g., OADM, OXC or EXC [6]), the number of
available wavelengths per link and the capacity of each wavelength-channel and the aggregated traffic demand
of each node-pair are given. However, the WPs are not given we will have to find them. Our task is to find the
minimum cost solution, by configuring the network nodes, choosing the optimal WPs and routing optimally the
traffic demands over the WPs.

The output of the optimisation is the system of wavelength-paths (WPs) and the routed traffic demands.
For each demand both, working and protection paths are needed which are node-disjoint, i.e., they have no
common link or node.

Our task is not a simple optimisation for two levels separately, since the efficiency of the second level's
solution depends on the the first level's results, i.e., on the WP set. Therefore, finding the optimal solution
which is also the WP topology and the routing of the traffic demands is NP-complete, therefore, the problem
has been solved by heuristics using genetic algorithms. This approach allows better approaching the global
minimum in polynomial time.
4 The Proposed Algorithm

Our algorithm consists of this iteration:

Initialisation

While {  
  • Step 1: Choosing the end nodes of the WPs, i.e., choosing the WP set which is the set of the end node pairs (first level).
  • Step 2: Routing the WPs between these end nodes (first level).
  • Step 3: Routing the traffic demands over this new topology built of WPs (second level). }

where (first level) consists of Step 1 and Step 2, while (second level) of Step 3 only.

Genetic algorithm is used in Step 1. An entity of the population is a WP set. Before the iteration there is an initial step when an initial state of the solution state-space is chosen. The mutation of the entities means that we either brake a WP into two parts or we concatenate two WPs into one using the statistics of the previous generations. One of these operators (Cutting or Concatenating WPs) of mutation are used to generate a transition to a new neighbour-state. The state is accepted if it is better or "not much worse" than the previous one and it was not visited in a last few steps or else rejected according to simulated annealing and tabu search policies. Then a valid state is generated and the quality of results (fitness) for that state is evaluated. The fitness of an entity shows the chance of survival. This is equal to the cost of the network counted in the further two steps starting from this WP set. Note that, this algorithm does not use crossover, therefore it is called sometimes bacterial algorithm too.

Our task in Step 2 is associating the WPs with wavelengths, laying down the exact paths between the ends of the WPs. This is done by Dijkstra’s shortest path algorithm over the original topology of the network. The problem within this layer is that the result of the routing depends on the order of the WPs which we want to route. The earlier routed WPs reserve wavelengths, therefore those routed later have insufficient both, wavelengths and edges. It can happen that a routing sequence does not give solution at all, while another order is feasible. To handle this problem the algorithm tries several sequences for routing the WP set. To succeed in getting a solution in every state we use extra wavelength. Later, these extra wavelength will be extorted by Simulated Annealing which increases the cost of paths with extra wavelength.

In Step 3 the demands are routed over the virtual topology of WPs obtained in Step 2. This step is analogous to routing WPs between the end nodes of the WP set (Step 2), however, here the disjoint protection paths have to be determined as well. The algorithm we propose for this purpose is a heuristic one based on Suurballe’s polynomial time algorithm. Finding the shortest path by Dijkstra’s algorithm, deleting it temporarily and then searching for another shortest path often does not give the optimal solution, and sometimes does not give any solution, though there is one. That’s why we choose the algorithm of Suurballe.

4.1 Original Algorithm of Suurballe

The algorithm of Suurballe [7, 8] finds the pair of paths which has the lowest total cost (shortest pair of disjoint paths). Steps of the Suurballe’s algorithm are demonstrated on an example shown in Figure 2.

Figure 2: Steps of the original Suurballe’s algorithm.

Figure 2a shows the original network where the costs assigned to the arcs are shown. The task is to find the two cheapest node-disjoint paths from S to T. The first step is to find the shortest path P1 between these
nodes. We can get it for example by using Dijkstra’s shortest path algorithm. This directed path is also shown in Figure 2.a.

The second step starts with multiple transformations. First of all we have to transform the original graph into a new one (Figure 2.b). Each node has to be split into two nodes joined by an arc having cost 0. This arc is directed, e.g., from the upper node to the lower one. Let us label the upper node with 1 and the lower one with 2, therefore instead of node A there will be two nodes A1 and A2 and a new arc from A1 to A2. In the undirected case every edge means two new directed edges (arcs). These arcs have to start from the node labeled 2 and have to end in the node labeled 1. For example the edge between S and A will be substituted by two arcs in the new graph: One arc from S2 to A1 and the other one from A2 to S1. As it can be seen always that node is the beginning of the arc which is labeled by 2.

Along the P1 we have to make some changes. For example the arc from B to A in the original graph is in P1. Therefore, the arc from B2 to A1 is in P1 in the new graph as well. The direction of this arc has to be turned. The other arc between A and B is the arc from A2 to B1. It has to be deleted. These changes have to be done for each arc of P1. The costs also have to be transformed according to the following formula: 
\[d'(i, j) = c(i, j) + d(i) - d(j)\] where \(c(i, j)\) is the original cost and \(d(i)\) is the cost of the shortest path from the source node to the node \(i\). The result of this transformation is shown in Figure 2.c.

Over this new graph we have to run a second shortest path algorithm. The result can be seen in Figure 2.d. In Figure 2.e we can see both paths over the original graph. Here is the end of the second step.

In the third step after eliminating the common links we obtain the two desired paths having minimal total cost (Figure 2f).

4.2 Our Realization of Suurballe’s algorithm

Building a new graph can be memory and CPU time demanding. Going round this problem we have changed the original algorithm keeping the main principles. The first step is the same as in the original one - the first shortest path is obtained.

In the second step we delete temporarily the first edge of the first path from the graph and transform the weight of each edge as in step one of the original algorithm.

Over this graph a second shortest path algorithm has to be started which is a modified Dijkstra algorithm in our realization. When this path reaches a node of the first path it has to step back along the last link (or multiple links) of the first path. This principle is based on the idea to eliminate common links whenever we want to have node-disjoint paths.

If there are no common links and the second path simply crosses the first path and goes alone to an other node which is not in the first path, then there is no opportunity to make them node-disjoint. To avoid this problem the back stepping on the first path provides the common link which can be eliminated. In the original algorithm the back stepping is substituted by the node splitting and the direction of arcs. In our algorithm after this back stepping the second shortest path algorithm proceeds and can choose any node. When it reaches a node of the first path again it has to make a back step again, and so on.

This approach to the Suurballe’s algorithm arises some difficulties. We can reach a node several times through back steps. It is not possible in the original Dijkstra’s algorithm. In the original Suurballe’s algorithm the node splitting solves this problem. Going round the main principles we have two rules:

- Rule 1: If the second path reaches a node of the first path it has to step back along the first path by minimum one edge.
- Rule 2: We can reach a node two times. In this case the path has to enter over an unused link.

Figure 3.a shows the example network while Figure 3.b points to the problem of reaching a node two times. The dotted line is the first path (S-A-B-C-T) between nodes S and T while the thick solid one is the second one (S-B-A-C-B-T) between nodes S and T. Now, let us see how to build the second path. The first node can only be node B because the edge between S and A is deleted as this edge is reserved by the first path. In node B we have reached the first path so we have to make a step back into node A. This edge between A and B will be the common link which can be eliminated. Now the path can only go to node C where it has to make a back-step again to node B. In this case the greedy shortest path algorithm chooses node A again getting stuck in a loop.

The solution of this problem is that after the second path does a step back it can continue its way to any other except to a node of its previous path. To provide this we have to register the whole path from the source to a node. In the original algorithm of Dijkstra it was enough to register for every node that node from which the path comes. Now we have to register the whole path till this node. In the other words the spanning
Figure 3: Illustration of the diverse routing problem encountered.

system of paths from the source has to be registered. The last figure (Fig. 3.c) shows the two node-disjoint paths obtained using this modified Dijkstra algorithm.

The third step of our realization of Saurballe’s method is the same as that of the original one.

Otherwise, this figure shows why two simple shortest path searches (e.g., Dijkstra’s) one after the other do not work. As it can be seen the first path uses every node so the second can not use any of them.

4.3 Saurballe’s Algorithm in the Two Layer Model

After discussing Saurballe’s algorithm let us see how to use it in our two layer model and by the genetic algorithm.

At the beginning of the algorithm the first population has to be initialized as a valid solution, although it is not an optimal one it is needed for the genetic algorithm to avoid idle steps. If a new state generated by the genetic algorithm is invalid it has to step back, until it finds a valid one. A valid state means a possible feasible routing of the traffic demands. The simplest way to achieve this is to treat the traffic demands like a single WP. Then every traffic demand is routed over an own WP. This is a very expensive solution which uses extra wavelengths but it will be optimized in the further generations. Therefore, the diverse routing at the first population is equivalent to the node-disjoint routing of the WPs (which are now equivalent to the traffic demands) over the real network topology, where it is still possible to use the algorithm of Saurballe. In the further generations the WPs and the traffic demands are separated and the real two layer model is developed. It causes problems to the algorithm of Saurballe, so we have to make some changes.

As it has been already mentioned the diverse routing is carried out in the second level which works on the virtual topology of the WPs. The second level does not deal with the original topology of the network, but it routes the traffic demands over the WPs. If we want to have node-disjoint traffic demand pairs we have to use node-disjoint WPs. Therefore, it has to be registered which WPs use common nodes. This registration lets us know which WPs can not be used in the second light-path because of using a WP in the first path.

Due to this restriction in this two-layer model the algorithm of Saurballe does not work properly - it does not always give the optimal solution. Figure 4 illustrates this situation.

Figure 4: Example for showing the sub-optimality of Saurballe’s algorithm in the two-layer case.

In Figure 4.a we can see the example network with the weights of the edges. In Figure 4.b the topology of the WPs is shown with the weights of the WPs. It has to be noticed that node B does not exist in the topology of the WPs so the WP between S and T having weight 6 seems to be node-disjoint with the WP between A and C which has weight 2. Unfortunately, these WPs are not node-disjoint in reality and this will cause problems.

In Figure 4.c the first path is shown by a thick line between the nodes S-C-A-T and the WPs which can not be used in the second path due to the first path. In this case this is only one WP signed in the figure by a broken line.
Now the second path will be a suboptimal path (dotted line S-A-C-T) since the first path (solid thick line S-C-A-T) has locked out that one (S-T solid line) which would be the optimal (Figure 4.d).

Figure 4.e contains the final paths after the elimination and it contains a better solution too. The first path is S-A-T and the second path is S-C-T. The sum of their cost is 12. The better solution could be to have S-A-T as the first path and S-T as the second one. The sum of their cost is 11 only. The algorithm could not find this solution because the first path locked out the WP between S and T.

It has to be noticed that if the edge between S and A was deleted then the algorithm wouldn’t give solution. Although there exists one.

Finally, there is an interesting problem caused by the two-layer architecture. Routing the paths over the virtual topology can cause that a path is not “node-disjoint with itself”, i.e., there might be a physical level loop in the upper layer path. If there is no such a loop we will call it “self node-disjoint”.

As we have seen in the last step of the Suurballe’s algorithm we have two paths which are the results of the modified shortest path algorithms. The common links of these two paths have to be eliminated. Due to this elimination the final paths are combined from the results of the shortest path algorithms. If a path is not “self-node-disjoint” the combination of the two paths can put out of order the node-disjointness of the final path. This case can be most easily understood by looking to an example.

![Figure 5: Illustration of the “self-node-disjointness”](image)

In Figure 5.a the example network is shown with a solid thick line, and there are the WPs also with broken lines. We have five WPs: between S-B, S-B-A, B-A, B-A-T, B-T. Figure b shows the virtual topology. It has to be noticed that the role of node B and node A has changed. An other important remark that the WP between S and A contains node B in the physical level. There is a dotted circle next to the WP if it contains another node in the physical level. The circle contains the node which passed in the physical level. The figure c shows the results of the two shortest path algorithms. The path S-A-B-T isn’t “self-node-disjoint”. The last figure shows the final paths after the elimination. The WP between S and A contains node B therefore it is not node-disjoint with the WP between B and T. To avoid this problem the ”self-node-disjointness” should be checked when using the shortest path algorithms. Because of the reason of this case the Suurballe’s algorithm can be run without this option and if it does not give a solution we can try to find off this case. Then modified Dijkstra’s algorithm solves the problem with finding the shortest ”self-node-disjoint” path.

Note that ”self-node-disjointness” is a too strong criterion. We could allow paths which have adjacent WPs with more than one common node as well.

Summarizing the realizations and the problems we can conclude that the algorithm of Suurballe is a good means for diverse routing. It is fast and it minimizes the sum of costs of the links of the path-pair. Unfortunately, in the two layer model in some rare cases it is suboptimal.

5 Numerical Results

In this section we compare the performance of the method with protection to that one without protection [5] on three test networks N5, N9 and N20 (Figure 6) consisting of 5, 9 and 20 nodes respectively. N20 is analogous to the fictive European Optical Transport Network (EurOTN) with a few links added.

The numerical tests have been carried out on a 500 MHz Intel Pentium III processor, with 128 MB of RAM running operating system Linux. The programs have been written in C++.
Figures 7 - 9 show the obtained results. The left bar of Figure 7 shows the normalised cost obtained by running the algorithm for finding working paths only, i.e., without protection. This cost will be used for normalising the other cost values as well. The next three pairs of bars (from left to right) show the costs of all working and all protection paths relative to case without protection for three networks N5, N9 and N20. The working paths always have slightly higher costs than the case without protection. This is as expected because they have slightly longer paths through a doubled number of demands to be routed. The protection paths are always significantly longer than working paths (in some cases even twice as long). This strongly depends on the topology. In networks where many alternative paths are present between pairs of nodes the total cost is lower, i.e., mesh protection has more significant gains in larger networks having denser topology.

Figure 8 shows the hop-counts. From left to right the first pair of bars shows the average hop-counts for working and protection paths at the WL (λ) level for N5. This expresses the number of WPs crossed by an path in average. The next pair of bars shows the average hop-count for working and protection paths counted at the physical level, i.e., the number of physical links crossed by an end-to-end path in average. The remaining two groups of four bars show the same values for test networks N9 and N20. It can be seen that the working paths always consist of less hops than protection paths, and that WL (λ) hop-count is always lower than the physical one. The reason is that the more WPs are used, the longer the WPs will be, i.e., by increasing the number of WPs within the network the physical to WL hop-count ratio increases.

Figure 9 shows the running time expressed in milliseconds for methods without and with protection for three networks. It is very interesting that while increasing the size of the network the running time increases faster for the case without protection than for that one with protection! The ratio of the running time of the protected case to running time of unprotected case drops while the network size grows.

Figure 10 shows how the network cost decreases during the iterations. Each bar shows the average cost for 10 sequential generations. For each such group there are 3 bars, they show the average cost of working paths, of protection paths and of that of paths for the case without protection from left to right respectively. The value for the case without protection practically does not change at all, while in case with protection both working and protection values keep increasing and then drop. The reason is, that in the beginning our method allows a larger number of "virtual" WPs where practically all demands are routed over single-hop paths, and then the method starts cutting down the number of WPs until the given WL number is reached by increasing the cost assigned to these "virtual" additional WPs. Figure 10 shows clearly this transient due to the tradeoff between the change in cost and number of WPs.

5.1 Conclusion

A new iterative wavelength routing algorithm based on decomposition and numerous heuristics has been proposed which optimises both the electrical and the underlying optical layer simultaneously finding pairs of near-optimal single- and multi-hop wavelength paths in polynomial time. The problem of diverse routing in multilayer networks was investigated and heuristics were proposed for handling disjointness in lower layers while routing in an upper layer. The numerical results show that the proposed method performs particularly well for large sparse networks with many WPs. The increasing number of WPs allowed by the technology and the increasing need for protection of services requires this kind of algorithms.

5.2 Acknowledgments

This work has been done in the research cooperation framework between Ericsson and the High-Speed Networks Laboratory (HSN Lab) at the Department of Telecommunications and Telematics, Budapest University of Technology and Economics. A part of this work has been supported by COST 266. The authors are grateful to M. Boda (Ericsson), T. Henk (HSN Lab) and A. Kuciar (COST 266) for their support.
Figure 7: Costs of working and protection paths for 3 networks relative to the cost obtained for the case with working paths only (i.e., without protection).

Figure 8: Working and protection hop-count pairs for \( \lambda \) and physical levels for 3 networks.

Figure 9: Running times for 3 networks without and with protection.

Figure 10: Average cost for sequence of 10-generation groups for working and protection paths and for the case without protection for N20.

References


