Shortest Node-Disjoint Path using Dual-Network Topology in Optical Switched Networks

Mostafa Dahshan
College of Computer and Information Sciences, King Saud University, Saudi Arabia

Abstract: *This paper presents a new method for finding shortest node-disjoint paths in optical-switched networks with no wavelength conversion. The proposed method is based on a modified version of Dijkstra algorithm that works on an expanded so-called dual-network topology with $n\times n$- nodes and $2\times m\times n$ links, where n is the number of nodes and m is the number of links in the original network. Despite the larger network size, the execution time of the algorithm is in polynomial order ($mn+n^2 \log n$). Considering that the problem is NP-complete, the presented algorithm takes much less time than using Integer Linear Programming (ILP), which takes exponential time. Yet, it is able to find all available disjoint paths obtainable by ILP.*

Keywords: *disjoint paths, optical switched networks, network survivability, optimization algorithms.*

Received March 17, 2011; accepted May 24, 2011; published online August 5, 2012

1. Introduction

Optical networks constitute the core infrastructure of metropolitan and wide area networks. This is due to their enormous transmission capacities and their ability to span large geographical areas. A single optical fibre link can carry thousands of phone calls per second across a desert or undersea. Due to the nature of these environments, risks of optical link or node failures are common. When they happen, they can cause huge service outages or data losses. For this reason, survivability is an indispensable part of optical network design and planning. Survivability is classified into two main categories: protection and restoration. Protection schemes provide alternative paths between endpoints prior to connection establishment. Restoration schemes attempt to find alternative path after the failure occurs in the primary path. Protection schemes generally have faster recovery time and thus are preferred over restoration schemes [6]. Protection involves finding a backup path along with the primary path when a connection request arrives. For the backup path to be effective, the primary and backup paths should either be link-disjoint (no common links) or node-disjoint (no common nodes).

To improve network performance, it is required that the two paths (primary and backup) are the shortest (i.e., least cost) paths. Lightpaths in optical networks require using the same wavelength along the path, unless wavelength conversion is used. Combining the two constraints (disjoint and shortest) with the wavelength-continuity constraint, the problem of finding the two shortest disjoint paths becomes an NP-Complete problem [5].

This paper focuses on the case of node-disjoint paths, which provides more fault-tolerance and load sharing [1]. We also, assume no wavelength conversion. The goal of this paper is to present an algorithm for finding the two shortest node-disjoint paths in optical switched network. The algorithm presented in this paper takes a new approach for addressing this problem by making it similar to the single shortest path problem on an auxiliary network topology called dual-network topology. This allows using ordinary shortest path algorithms to solve this problem. There are some constraints that must be considered when applying the shortest path algorithm to the dual-network topology to ensure that the two paths are actually node-disjoint. In this paper we base our method on Dijkstra algorithm and show the modifications required for it to work on the dual-network topology.

The remainder of this paper is organized as follows: section 2 provides literature review and related work on path recovery algorithms for optical networks. In section 3, we describe how to construct the dual-network topology. Following that, the details of using modified shortest path algorithm are presented in section 4. To demonstrate the effectiveness of our method, we compare it with two well-known heuristics and with the optimal solution using Integer Linear Programming (ILP). To do this, we develop the ILP formulation of the problem and apply the three methods on the STC optical backbone network and the ARPANET network, using random network usage scenarios. This is done in section 5. Section 6 provides an analytical study on the performance of the presented method. Finally, section 7 concludes the paper by providing a summary and future work.

2. Related Work

The topic of optical network survivability has been addressed extensively in the literature. Techniques for both protection and restoration have been published and analysed. In this paper, we focus on protection techniques, which involve finding two optimal disjoint paths prior to connection setup. The simplest method for finding the two shortest paths is known as the two-step algorithm. In the first step of this algorithm, the shortest path is calculated using Dijkstra algorithm. Then, the links used in the shortest path are eliminated from the network. The second step applies Dijkstra to the modified network to find the second shortest path. Despite its simplicity, this algorithm doesn't work for many network topologies. In some "trap topologies", the first shortest path splits the network in such a way that no other paths between the two end points can be found, although, they were available in the original network [3]. A more efficient algorithm for finding optimal disjoint paths was developed by Suurballe [8] and later improved by Suurballe and Trajan [9]. This algorithm also, works in two steps. In the first step, Dijkstra is applied to the original network. In the second step, the links which belong to the shortest path are reversed and assigned negative costs. Then, Dijkstra is applied to the modified network. The links common between the paths of step 1 and step 2 are eliminated, yielding the two shortest paths between the two end nodes. The Suurballe algorithm is guaranteed to find in polynomial time the two shortest disjoint paths if they exist, provided that the same wavelength is used in both paths [2, 11]. In the cases where the two disjoint paths have to use different wavelengths, Suurballe algorithm may not always work.

Although, the problem of finding the two shortest disjoint paths with the wavelength-continuity constraint has long been believed to be NP complete, the first formal proof was provided in [2]. Moreover, another study has shown that the problem is NP complete even without length constraint [5]. The optimal solution for this problem can be obtained using ILP. ILP Problem formulations for the problem have been developed in [2, 5, 11]. However, the execution time for ILP is exponentially proportional to network size, which prohibits using such a method for moderate to large scale networks. Therefore, most of the studies that addressed this problem have focused on developing heuristic algorithms. In [2], two simple algorithms are proposed. The first algorithm, Active Path First (APF), is essentially the two-step algorithm mentioned earlier. It attempts to find the first shortest path. It then removes the channels used in that path before attempting to find the second path on the modified network. If either attempt fails, the call request is blocked. The second algorithm, APF Enhanced (APFE), improves the APF by reducing the number of shared links between the two paths. In [11], two heuristic solutions are proposed. The first one, Route-First, scans all fibre links and increases the cost of each link linearly with the number of wavelengths already in use. It then runs Suurballe algorithm and checks the two returned routes. If each route has at least one wavelength available in its links, the algorithm will succeed. Otherwise, the algorithm will fail. The second algorithm, Wavelength-Scan, scans each wavelength for two link-disjoint paths using Suurballe. If this fails, it searches on a different wavelength for two link-disjoint paths using the simple two-step procedure mentioned above. Because of their good performance, we compare our method with Route-First and Wavelength-Scan algorithms in section 7. In [4], a heuristic algorithm is proposed which is a hybrid combination of ant-based mobile agents and genetic algorithms. The hybrid algorithm aims to combine speed of finding first population of disjoint paths set using Ant Colony Optimization (ACO) with the efficiency of finding better routes from the first set using genetic algorithms.

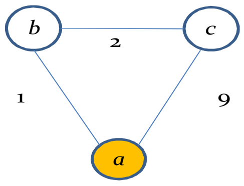
The aforementioned studies aimed to provide survivability through dedicated protection, which means that a single link cannot be used for more than one backup path. Other studies consider sharing links among multiple backup paths to reduce blocking probability and increase network utilization. In [10], the authors introduce heuristic iterative algorithm to compute two disjoint paths under wavelength continuity constraint and dependent cost structure. The algorithm computes k shortest paths and uses one path per iteration as a seed along with the best solution so far to find a better one. The algorithm uses a layered graph which is a transformation of the original graph in which finding the shortest path is equivalent to finding the shortest path and wavelength assignment in the original graph. After removing trap links of the seed path, the algorithm computes two disjoint paths p and q. It then uses path p as a seed path to compute cost dependent protection path. The iterations are stopped when the k-th shortest path is reached or when the value of the best solution is less than half the value of the current seed path.

3. Constructing Dual-Network Topology

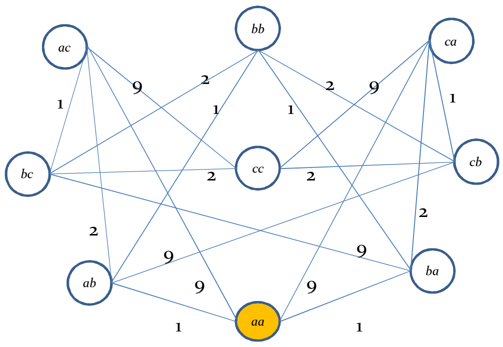
The main idea behind the method proposed in this paper is to construct the disjoint paths in parallel, rather than in sequence. This approach is equivalent to constructing another network, which we'll refer to as dual-network, and find the shortest path on that network. If such path is found, then the two shortest disjoint paths are found for the original network. In this section, we describe how to construct the dual-network. In the following section, we describe the algorithm to find the shortest path on that network.

Figure 1 shows a simple example constructing dual-network topology for a 3-node network. Although, such a network is too small to represent a real network, it is instructive to use it as an example.

Let *n* denote the number of nodes and *m* denote the number of links in the original network. The dual-network topology consists of $n×n$ nodes and $2×m×n$ links. Each node in the dual topology is denoted by two letters which represent the two current nodes in the dual path. i.e., node (*ba*) indicates a step in disjoint path where path 1 currently, ends at node (*b*) and path 2 currently, ends at node (*a*). Note that the order matters. (*ba*) is different from (*ab*).



a) 3-node network topology.

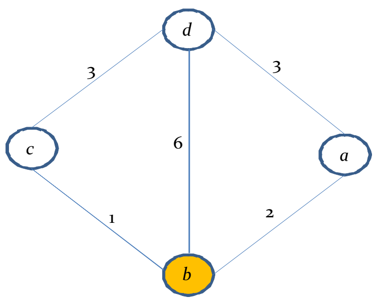


b) Dual-network representation.

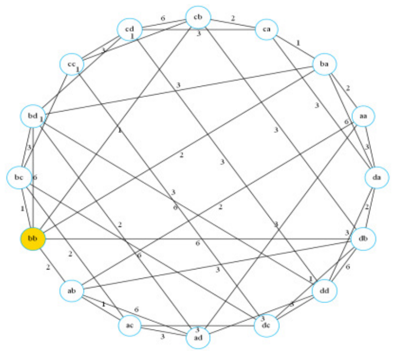
Figure 1. Example constructing dual-network topology for a 3-node network.

Looking for two shortest disjoint paths from source node (*a*) to destination node (*c*) in the original network is equivalent to looking for a single shortest path from node (*aa*) to node (*cc*) in the dual-network. This explains the number of nodes in the dual-network.

Each link in the dual-network represents a single step in one of the two paths using the two nodes from which the link originates. For example, link from (*ab*) to (*ac*) indicates a step in path 2 from (*b*) to (*c*) while path 1 is still at (*a*). Therefore, the number of links originating from each node in the dual-network equals the sum of the number of links originating from both of its constituents. The number of links in the dual-network is explained in section 6.



a) 4-node network topology.



b) dual-network representation.

Figure 2. Example of constructing dual-network topology for a 4-node network.

Figure 2 shows an example of constructing dual-network topology for a 4-node network. The dual-network has 16 nodes and 40 links.

A formal description of the algorithm for constructing the dual-network is shown in Algorithm 1.

```
Algorithm 1 TransformGraph(G)
#dist(L) is the cost of the link L
n=number_of_nodes(G)
#construct nodes
dnode=array[n*n]
#construct links
for(i=1 to n)
  for(j=1 to n)
  {
    foreach (x in link[i, x])
    {
      create dlink[ij, xj]
      dist(dlink[ij, xj])=dist(link[i, x])
    }
    foreach (y in link[j, y])
    {
      create dlink[ij, iy]
      dist(dlink[ij, iy])=dist(link[j, y])
    }
  }
return DG=graph(dnode, dlink)
```

4. Disjoint Shortest Path Algorithm

The main idea behind the method proposed in this paper is to build on the well-established Dijkstra's algorithm. The algorithm Disjoint-WaveLength-Paths (DWLP) makes few modifications on Dijkstra. The objective is to not affect the proven optimality of Dijkstras' algorithm. The detailed algorithm is shown in Algorithm 2, with changes from the original Dijkstra are emphasized with bold text. The DWLP algorithm works on the dual-network topology obtained by running the TransformGraph algorithm on the original network topology. Each node in the dual-network is referred to as dnode.

The algorithm starts by initializing the costs of all dnodes to infinity and label them as tentative (lines 11-18). The source dnode [ss] is marked as PERMANENT and assigned a cost of zero.

```
 1 Algorithm 2 DWLP(s, λ1, λ2)
 2 #s is the source node in G
 3 #λ1, λ2 are the two wavelengths to be
 4 #   used in the disjoint paths 1, 2
 5 #ss is source node in DG
 6 #n is the number of nodes in G
 7 #dist(L) is the cost of link L
 8 #wl_available(λ, i, j) returns true if the
 9 #   wavelength λ is available on link
10 #   between nodes i, j
11 for i = 1 to n
12   for j = 1 to n
13   {
14     cost(dnode[ij]) = INFINITY
15     visited(dnode[ij]) = new list(empty)
17     label(dnode[ij]) = TENTATIVE
18   }
19 label (dnode[ss]) = PERMANENT
20 cost (dnode[ss]) = 0
21 s_adjacent = NULL
22 (x, y) = (s, s)  #current working node
23 while exists dnode[ii]
24          with label(dnode[ii]) == TENTATIVE
25 {
26   #loop until reach all the destinations
27   #with two paths
28   if previous(dnode[xy]) == dnode[ss]
29   {
30     #exceptional case
31     if (s==x)
32       s_adjacent = y
33     elseif(s==y)
34       s_adjacent = x
35   }
36   foreach neighbor dnode[vu] of dnode[xy]
37   {
38     if (u==y) #only x changed
39     {
40       λ=λ1; node2=v; prev1=x
41     }
42
43     if(v==x) #only y changed
44     {
45       λ=λ2; node2=u; prev1= y
46     }
47     if  label(dnode[vu)] != PERMANENT
48         and node2 not in visited(node[xy])
49         and node2 != s_adjacent
50         and wl_available(λ, prev1, node2)
51     {
52       if cost(dnode[xy])+dist(dlink[xy, vu])
53           < cost(dnode[vu])
54       {
55         previous(dnode[vy]) = dnode[xy]
56
57         cost(dnode[vu]) = cost(dnode[xy])
58                              + dist(dlink[xy, vu])
59         visited(dnode[vu])= visited(dnode[xy])
60                                + node(prev1)
61       }
62   }
63   first = NULL
64
65   do
66   {
67   #get min cost node; make it PERMANENT
68   (x,y)= (i,j) with min(cost(dnode[ij]))
69      and label(dnode[ij])!= PERMANENT
70
71   if cost(dnode[xy])==INFINITY
72     return
73   else
74     label(dnode[xy]) = PERMANENT
75   } while (x==y)
76 }# end while
```

The main loop of the algorithm is then started (line 23), which continues until no more dnodes [*ii*] are left with TENATIVE label. Recall from the previous section that reaching a dual-network node (dnode) which has identical two-letter label [*ii*] means that the destination node with that letter [*i*] in the original network has been reached with two disjoint paths.

Lines 36-62 loop on all neighbouring dnodes [*uv*] for the current working dnode [*xy*], as done in the regular Dijkstra (relaxation phase). If the cost of dnode [*xy*] + cost of link between [*xy*] and [*uv*] is less than the original cost of dnode [*uv*], then the cost of [*uv*] is updated and the previous dnode for [*uv*] is changed to [*xy*].

Three precautions need to be considered here. First, to ensure that the path is node-disjoint, the algorithm maintains an array called *visited* for each dnode. This array contains the list of nodes in the original network which have been traversed in the current dual path. This is handled in line 48. Because the visited array does not contain the constituent nodes of the current dnode, an exception needs to be added to split the path after the initial step. For example, in Figure 2, if we start from dnode (*bb*), the first neighbour can be (*cb*). From (*cb*), we can't allow the next step to be (*cc*), otherwise the path won't be disjoint. Therefore, a condition is added in lines 28-35 and checked in line 49 to prevent the paths from initially colliding.

Finally, to satisfy the wavelength continuity constraint, the algorithm must check that the wavelength to be used in the corresponding side of the path is available. This is done in line 50.

The do-while loop in line 65 performs the search for minimum-cost dnode with cost less than infinity, if such dnode is found, it will be marked. If the dnode [*ij*] has two identical constituents (*i* equals *j*), one destination is reached and so the algorithm doesn't examine the neighbour of that dnode, and continue to search for the next minimum dnode. The loop will end when a dnode [*ij*] with $i \neq j$ is found.

In a wavelength-switched network with *w* possible wavelengths per link, the DWLP algorithm needs to be executed once for each possible λ1 and λ2 combinations, including the case where (λ1=λ2) but not considering the order. The number of executions is thus equal to $w(w+1)/2$. The values of (λ1, λ2) that yield the lowest cost disjoint paths are selected.

At the end, the disjoint paths from the source node (*s*) to any destination node (*i*, *i*=1 to *n*, *i*≠*s*) are obtained by following the previous of each dnode [*ii*] marked as PERMANENT back to dnode [*ss*].

5. Performance Study

As mentioned in section 2, the problem of finding shortest disjoint path under wavelength continuity constraint is considered NP-complete. As such, the optimal solution of the problem can only be obtained using ILP. Execution time of ILP is exponentially proportional to the number of nodes in the network but is guaranteed to find all possible disjoint paths. Heuristic solutions, on the other hand, require much less time, but may not find all possible disjoint paths and the shortest path obtained by heuristic algorithms is not necessarily the ultimate shortest path. The purpose of this section is to compare the performance of the DWLP algorithm proposed in this paper with both ILP and heuristic algorithms. The comparison is based on both the execution time and the number of obtainable disjoint paths.

5.1. ILP Formulation of the Problem

The ILP formulation of the problem of finding two disjoint paths has been developed in several related works as mentioned in section 2. We use the formulation developed in [2]. Some modifications are made to the formulation to make it applicable for node-disjoint instead of link-disjoint paths.

Let *V* denote the set of nodes and *E* denote the set of links in the network. Links belonging to *E* are defined as pairs (*u*, *v*) that represent the nodes they are connecting. The source and destination nodes are denoted as *s* and *t*, respectively. The two wavelengths to be used in the two paths are denoted as *λR* and *λB*, which refer to the red and blue paths, respectively. *R*(*u*, *v*) is set to 1 if wavelength *λR* is available on the link (*u*, *v*) and 0 if it is unavailable. Similarly, *B*(*u*, *v*) is set to 1 if wavelength *λB* is available on the link (*u*, *v*) and 0 if it is unavailable. For each link (*u*, *v*) that belongs to *E*, four variables are defined: *r*(*u*, *v*), *r*(*v*, *u*), *b*(*u*, *v*) and *b*(*v*, *u*). These variables can take value 0 or 1. If the link from *u* to *v* belongs to the red path, *r*(*u*, *v*)=1, else, *r*(*u*, *v*)=0. Note that order of *u* of *v* is important, because it defines the direction of the path. Same can be said about *b*(*u*, *v*) and *b*(*v*, *u*). The function *δ* () is defined as follows:

$$s(x) = \begin{cases} 1 & x = s \\ -1 & x = t \\ 0 & otherwise \end{cases} \tag{1}$$

With the previous definitions, the ILP formulation can be stated as follows:

$$min \sum_{[u,v] \in E} r(u,v)+r(v,u)+b(u,v)+b(v,u) \tag{2}$$

Subject to the following equations:

$$\sum_{v \in V} r(x,v) - \sum_{v \in V} r(u,x) = \delta(x), \forall x \in V \tag{3}$$

$$\sum_{u \in V} b(x,u) - \sum_{v \in V} b(u,x) = \delta(x), \forall x \in V \tag{4}$$

$$\sum_{v \in V} r(v,x) + b(v,x) = 2, for\ x = t \tag{5}$$

$$\sum_{v \in V} r(v,x) + b(v,x) = 0,\ for\ x = s \tag{6}$$

$$\sum_{v \in V} r(v,x) + b(v,x) \leq 1,\ \forall x \in V,\ x \notin \{s,t\} \tag{7}$$

$$r(u,v)+r(v,u) \leq R(u,v),\ \forall (u,v) \in E \tag{8}$$

$$b(u,v)+b(v,u) \leq B(u,v),\ \forall (u,v) \in E \tag{9}$$

$$r(u,v)+r(v,u)+\ b(u,v)+b(v,u) \leq 1,\ \forall x(u,v) \in E \tag{10}$$

$$r(u,v),\ b(u,v) \in \{0,1\} \tag{11}$$

The requirement in equation 2 is to find the minimum total number of red and blue links. Note that all links have equal cost of 1. Hence, minimizing the number of links means minimizing the total cost. The equation 3 ensures that the red path is connected. Each node in the path is connected to two nodes, with the exception of *s* and *t* nodes, which have only connected to one node each. Equation 4 ensures that same for the blue path. The equations 5, 6 and 7 ensure that, excluding the source and destination nodes, if any node is found in the red path, it is not in the blue path and vice versa. i.e., the path is node disjoint. The equation 8 ensures that the red path can only have links with *λR*. Similarly, equation 9 ensures that the blue path can only have links with *λB*. The equation 10 ensures that the two paths are also, link disjoint. Finally, equation 11 ensures that *r*(*u*, *v*) and *b*(*u*, *v*) can only take values 0, 1.

5.2. Network Topologies and Setup

We study the performance of our DWLP algorithm compared to ILP and two heuristic algorithms developed in [2]. The comparative studies have been done on two network topologies: STC backbone network, shown in Figure 3, and ARPANET network, shown in Figure 4.

For both topologies, we examined networks with 5, 10 and 20 available wavelengths. For each case, we tested network loads of 25%, 50% and 75%. The network load refers to the average number of wavelengths per link that are already in use before starting the simulation. The STC backbone network has 35 nodes and 45 links, while the Arpanet network had 20 nodes and 32 links.

The algorithms which have been compared include ILP, as formulated in this section, Route-First and Wavelength-Scan heuristics described in section 2, and our DWLP algorithm. Recall that Route-First and Wavelength-Scan as described in [2] aim to find link-disjoint rather than node-disjoint paths. However, Suurballe's algorithm, which is used in these heuristics, can easily be adapted to find node-disjoint paths using a transformation called node splitting [8]. We have used this transformation when we programmed Route-First and Wavelength-Scan in order to provide a fair comparison. We have used C# programming language for coding our DWLP algorithm, as well as the Route-First and Wavelength-Scan algorithms. ILP implementation was programmed using Mathematica software. Simulations were done on Core 2 Duo computers with about 2GHz speed.

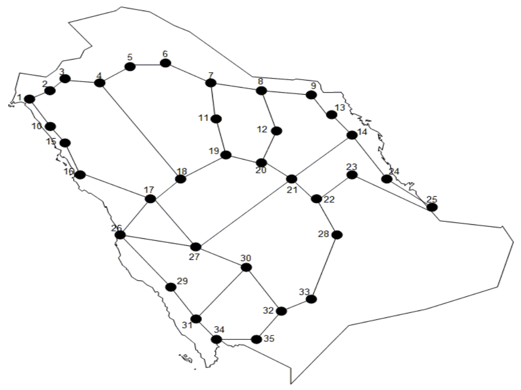


Figure 3. STC backbone network.

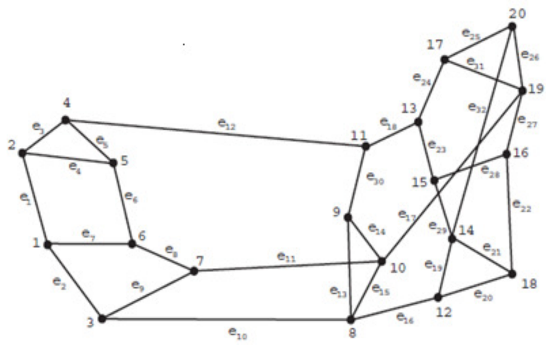


Figure 4. ARPANET network.

5.3. Results and Discussion

Simulation results for STC backbone network and the ARPANET network are shown in Tables 1 and 2, respectively. In both tables, the left half shows the execution time in seconds and the right half shows the number of disjoint paths. The rightmost column shows the total number of cases considered, which equals the number of source/destination pairs.

Several observations can be made on Tables 1 and 2. In almost all cases, the execution time is lowest for Route-First, then Wavelength-Scan, then our DWLP, while the ILP always comes last with at least three orders of magnitude more than other three algorithms. On the other hand, the ILP always find all possible disjoint paths, which can be used to benchmark other algorithms. We note that both STC backbone network and ARPANET are well planned so that for each source/destination pair, a node-disjoint path always exists. The number of disjoint paths found decreases with increasing network load, whereas it increases with more number of wavelengths per link.

The most significant result in both tables is that our DWLP links always finds all possible paths obtainable by ILP, while the execution time required for DWLP is only a fraction of that required for ILP. We also note that the advantage of DWLP over Route-First and Wavelength-Scan is more apparent in larger networks, than in smaller networks. The results in Table 1 are graphically illustrated in Figures 5 and 6. It should be noted that our algorithm and the algorithms we have studied for comparison can use link-state routing, which means that information about available wavelengths in each link in the network is readily available at the time of algorithm execution. Link state information is advertised using extended OSPF protocol [7].

Table 1. Simulation results for STC network.

Wavelengths/ Link	Network Load	Execution Time (seconds)				Number of Disjoint Paths Found				Total Number of Cases
		R-First	W-Scan	DWLP	ILP	R-First	W-Scan	DWLP	ILP	
5	25%	1.4976	2.844163	7.9092	8455.99	369	409	487	487	595
5	50%	1.56	3.104178	2.4336	14950.8	55	57	58	58	595
5	75%	1.5444	2.490142	1.6848	16730.7	2	2	2	2	595
10	25%	1.482	3.176182	22.0116	9668.33	448	510	560	560	595
10	50%	1.5912	4.408252	5.6472	59662.02	53	59	84	84	595
10	75%	1.5288	4.035231	3.6348	66606.2	2	2	2	2	595
20	25%	1.6068	4.827276	20.7324	14020	439	510	595	595	595
20	50%	1.6536	7.612435	20.0928	184975	81	92	228	228	595
20	75%	1.5288	7.286417	10.3272	252400	5	5	10	10	595

Table 2. Simulation results for ARPANET network.

Wavelengths/ Link	Network Load	Execution Time (seconds)								Total Number of Cases
		R-First	W-Scan	DWLP	ILP	R-First	W-Scan	DWLP	ILP	
5	25%	0.394023	0.423024	0.947054	44.086	190	190	190	190	190
5	50%	0.450026	0.501029	0.757043	126.906	84	111	117	117	190
5	75%	0.669038	0.494028	0.731042	146.532	7	7	7	7	190
10	25%	0.417024	0.418024	1.091062	58.548	189	190	190	190	190
10	50%	0.431025	0.650037	1.252072	267.558	86	127	163	163	190
10	75%	0.732042	0.718041	1.57809	520.404	14	17	18	18	190
20	25%	0.585034	0.398023	1.711098	133.46	184	190	190	190	190
20	50%	0.507029	1.026059	2.012115	359.863	76	154	188	188	190
20	75%	0.562032	1.181068	1.851106	1964.8	12	12	34	34	190

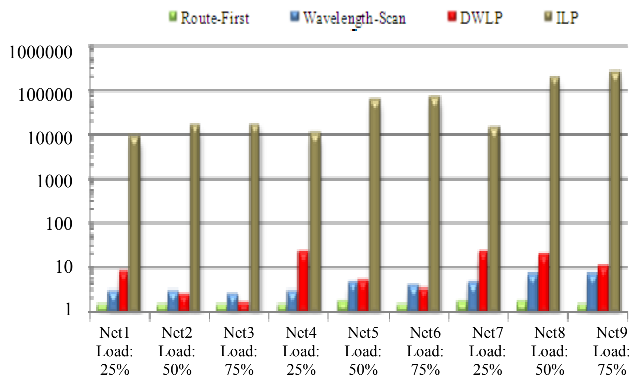


Figure 5. Execution time in seconds (log scale)-STC.

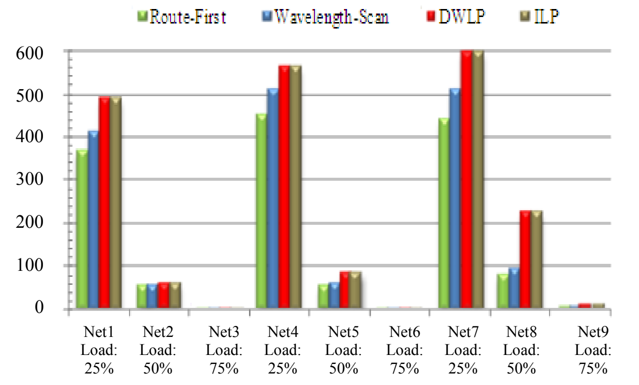


Figure 6. Number of disjoint paths found-STC.

6. Analytical Study

From the previous section, we can observe that the execution time of our DWLP algorithm is in polynomial order. To provide a more accurate estimation of the execution time of the algorithm, note that Dijkstra algorithm can find the shortest path in $O(m+n \log n)$, where n is the number of nodes and m is the number of links. The DWLP algorithm works on an $n\times n$ dual-network topology with $2nm$ links.

To show that the number of links in the dual-network topology is $2mn$, recall from section 3 that the number of links originating from each node in the dual-network equals the sum of the number of links originating from both of its constituents. Let the number of links coming from nodes i and j equal l_i and l_j, respectively. The number of links originating from dnode $[ij]$ in the dual-network equals l_i+l_j. To calculate the total number of links, note that each link is connected to two nodes. If we add the number of links originating from each node, every link will be counted twice. Thus, the summation of the number of links should be divided by 2, i.e.

$$\begin{aligned} Number\ of\ links &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(l_i+l_j) \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} l_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} l_j \end{aligned} \tag{12}$$

We can see that $\sum_{j=1}^{n} l_j = 2m$. Similarly $\sum_{i=1}^{n} l_i = 2m$.

Substituting in equation 12:

$$\begin{aligned} &\frac{1}{2}\sum_{i=1}^{n} n\cdot l_i + \frac{1}{2}\sum_{i=1}^{n} 2m \\ &= \left(\frac{1}{2}\right)(2m) + (2m)\left(\frac{1}{2}\right)(n) = 2mn \end{aligned} \tag{13}$$

A single run of the DWLP algorithm yielding a run time of approximately:

$$DWLP\ single\ run \approx O(2mn + n^2 \log n^2) \tag{14}$$

We have mentioned in section 4 that, in order to attempt all possible combinations between the two wavelengths, the number of DWLP executions is equal to $w(w+1)/2$, where w is the number of wavelengths per link. Adding this result to c14 yields:

$$DWLP\ full\ run \approx O(w^2(mn + n^2 \log n)) \tag{15}$$

7. Summary and Conclusions

In this paper, we have developed a new method for finding node-disjoint paths in optical-switched networks under wavelength continuity constraint. The new method transforms the original network topology into an auxiliary graph called dual-network topology. After that, it applies a modified version of the proven-optimal Dijkstra's algorithm to find the shortest path on the dual-network topology. The shortest path in the dual-network topology becomes equivalent to a node-disjoint path in the original network. The modifications to the original Dijkstra algorithm ensure that no common nodes between the two paths exist and that the wavelength-continuity constrained on each path is satisfied. We have provided a detailed description of the algorithm based on our new method, DWLP, and compared its performance with the ILP solution and two famous heuristic algorithms in terms of the execution time and the number of node-disjoint paths found on two different networks. We found that our algorithm was able to find all node-disjoint paths obtainable by ILP algorithm in a fraction of the time required by ILP.

We argue that the method developed in this paper provides a solution that is practically optimal for the problem of finding node-disjoint optical paths with wavelength-continuity constraint and challenges the commonly held belief that such optimal solution can only be obtained by ILP.

Future work should consider the overhead of advertising the link state information on the performance of the algorithm in terms of speed and blocking probability.

Acknowledgements

This work is supported by the Research Center of the College of Computer and Information Sciences, King Saud University. The author is grateful for this support.

References

[1] Abbas A., “A Hybrid Protocol for Identification of a Maximal Set of Node Disjoint Paths in Mobile Ad hoc Networks,” *The International Arab Journal of Information Technology*, vol. 6, no. 4, pp. 344-358, 2009.

[2] Andersen R., Chung F., Sen A., and Xue G., “On Disjoint Path Pairs with Wavelength Continuity Constraint in WDM Networks,” *in Proceedings of the 23rd AnnualJoint Conference of the IEEE Computer and Communications Societies*, USA, vol. 1, pp. 524-535, 2004.

[3] German M., Castro A., Masip-Bruin X., Yannuzzi M., Martinez R., Casellas R., and Munoz R., “On the Challenges of Finding Two Link-Disjoint Lightpaths of Minimum Total Weight Across an Optical Network,” *in Proceedings of the 14th European Conference on Networks and Optical Communications*, Spain, pp. 217-224, 2009.

[4] Le V., Ngo S., Jiang X., Horiguchi S., and Inoguchi Y., “A Hybrid Algorithm for Dynamic Lightpath Protection in Survivable WDM Optical Networks,” *in Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks*, Ispain, 2005.

[5] Liang W., “Robust Routing in Wide-Area WDM Networks,” *in Proceedings of the 15th International Parallel and Distributed Processing Symposium*, CA, pp. 8, 2001.

[6] Sahasrabuddhe L., Ramamurthy S., and Mukherjee B., “Fault Management in IP-Over-WDM Networks: WDM Protection Versus IP Restoration,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no.1, pp. 21-33, 2002.

[7] Shen S., Xiao G., and Cheng T., “A Novel Method of Link-State Update in Wavelength-Routed Networks,” *Journal of Lightwave Technology*, vol. 24, no. 3, pp. 1112-1120, 2006.

[8] Suurballe J., “Disjoint Paths in a Network,” *An International Journal Networks*, vol. 4, no. 2, pp. 125-145, 1974.

[9] Suurballe J. and Tarjan R., “A Quick Method for Finding Shortest Pairs of Disjoint Paths,” *An International Journal Networks*, vol. 14, no. 2, pp. 325-336, 1984.

[10] Todimala A. and Ramamurthy B., “Least-Cost Disjoint Paths With Dependent Cost Structure in Wavelength Continuous Optical WDM Networks,” *in Proceedings of Computer Communications and Networks*, USA, pp. 311-316, 2005.

[11] Yuan S. and Jue J., “Dynamic Lightpath Protection in WDM Mesh Networks under Wavelength Continuity Constraint,” *in Proceedings of Global Telecommunications Conference*, vol. 3, pp. 2019-2023, 2004.

Mostafa Dahshan received his BSc degree in computer engineering from Cairo University, Egypt in 1999 and his MSc in telecomm systems and PhD in electrical and computer engineering from the University of Oklahoma, USA in 2002 and 2006, respectively. Currently, he is an assistant professor of computer engineering at the college of computer and information sciences, king Saud University, Saudi Arabia. His current research interests include network protocols, performance, reliability and security.