Survivable Routing in IP-over-WDM Networks: An Efficient and Scalable Local Search Algorithm

Frederick Ducatelle and Luca M. Gambardella

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)

Galleria 2, CH-6928 Manno-Lugano, Switzerland

Abstract

In IP-over-WDM networks, a logical IP network is routed on top of a physical optical fiber network. An important challenge hereby is to make the routing survivable. We call a routing survivable if the connectivity of the logical network is guaranteed in case of a failure in the physical network. In this paper we describe FastSurv, a local search algorithm for survivable routing. The algorithm works in an iterative manner: after each iteration it learns more about the structure of the logical graph and in the next iteration it uses this information to improve its solution. The algorithm can take link capacity constraints into account and can be extended to deal with multiple simultaneous link failures and node failures. In a large series of tests we compare FastSurv with current state-of-the-art algorithms for this problem. We show that it can provide better solutions in much shorter time, and that it is more scalable with respect to the number of nodes, both in terms of solution quality and run time.

Key words: Survivable routing, IP-over-WDM, IP restoration, failure propagation

1 Introduction

Optical fiber connections are an increasingly popular technology for high-speed Wide Area Networks. This is because they offer enormous bandwidth and because their bandwidth can be shared among different channels through the use of Wavelength Division Multiplexing (WDM) [9,13]. In IP-over-WDM networks, an IP network is placed as a logical topology on top of the physical topology of the optical network. Each logical IP link needs to be routed on a path in the optical network. Thanks to the WDM technology, one physical link can carry several logical links, each on a different wavelength. The problem of setting up logical links by routing them on the optical network and assigning wavelengths to them is called the *routing and wavelength assignment* problem. See [20] for an overview. In what follows we will refer to logical IP links as *clear-channels* and to physical (optical) links simply as *links*.

Using high capacity links carrying multiple clear-channels is not without danger: in case of just a single link failure, a huge amount of data can get lost. Therefore a lot of attention is being paid to network protection. There are two main approaches to protection in IP-over-WDM networks: WDM level protection and IP level restoration [15]. In the case of *WDM protection*, a physically disjoint backup path is reserved for each clear-channel. This can only be provided at the cost of hardware redundancy. Therefore *IP restoration* can be a cheaper option. In this approach, no action is taken at the optical layer: failures are detected by the IP routers, which adapt their routing tables. So when one link breaks, data traffic which used to go over clear-channels using

Email address: {frederick,luca}@idsia.ch (Frederick Ducatelle and Luca M. Gambardella).

this link is rerouted over other clear-channels which do not use this link.

An important problem with the use of IP restoration in WDM networks is caused by the fact that each link usually carries several clear-channels. This means that a single link failure normally causes a number of clear-channels to go down at the same time. It is then possible that the logical network becomes disconnected due to these concurrent failures, and IP restoration becomes impossible. This phenomenon is called *failure propagation* [3]. In order to avoid this, the clear-channels should be routed in such a way that no single link failure can disconnect the logical network. This combinatorial problem is often called *survivable routing*, and is NP-complete [8].

An example of a survivable routing problem is given in figure 1. The clear-channels of the IP network of (B) need to be routed on the WDM network of (A). A first possible routing is given in (C). Clearly this routing is not survivable. When link $(4, 5)$ breaks, both clear-channels c and d get disconnected, leaving the logical network in two parts: one containing only node 4 and the other containing all other nodes. Routing (D) on the other hand is survivable: no matter which link breaks, the logical network always remains connected. In particular, a failure of link $(2, 5)$ disconnects the clear-channels d and f. However, their endpoints stay connected in what is left of the logical graph (2 and 5 stay connected using clear-channels a and e, whereas 4 and 5 stay connected using clear-channels c, b, a and e).

The authors of [7] observed that a routing is survivable if and only if no link carries a full cut set of the logical network. A *cut set* of a network is defined by a cut of the network: a cut is a partition of the set of nodes V in two sets S and $V - S$, and the cut set defined by this cut is the set of edges which

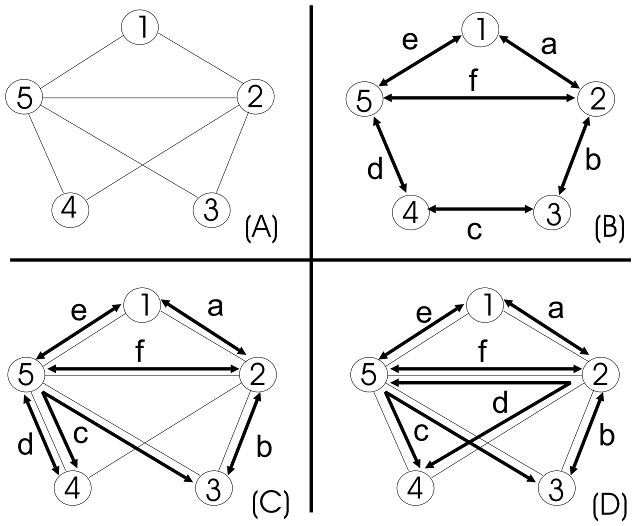


Fig. 1. Routing a logical topology onto a physical topology. (A) Physical topology. (B) Logical topology (C) An unsurvivable routing. (D) A survivable routing.

have one endpoint in S and one in $V - S$. In the example of figure 1, $\{c, d\}$ forms a cut set of the logical network of (B), and the fact that c and d share a link therefore leads to an unsurvivable routing in (C). $\{f, d\}$ on the other hand does not make up a full cut set ($\{f, d, a\}$ or $\{f, d, e\}$ would though), and placing f and d together on a link in (D) does not cause the routing to be unsurvivable. In [7,8], this observation is used to formulate an ILP for the survivable routing problem: for each clear-channel and for each cut set of the logical network, a constraint is added to the ILP. This leads to exact solutions, but also to very long run times, which makes this method impractical for large networks. In [8], the authors propose two relaxations to their ILP, in which they do not include all cut sets. This considerably speeds up the algorithm, but can easily lead to suboptimal solutions.

In this paper, we present a local search algorithm for survivable routing, called FastSurv. FastSurv uses the notion of cut sets in an approximative way: after each iteration, the algorithm learns more about the cut sets of the logical graph, and in the next iteration it uses this information to improve the previous solution. This approach allows the algorithm to consider important structural information about the problem in an efficient way. We describe a basic version of the algorithm which does not take link capacity constraints into account, and an extended version which does. We also explain how the algorithm can be extended to provide survivable routing with respect to node failures or multiple simultaneous link failures. In an extensive set of empirical tests, we show that our algorithm can provide better solutions than existing state-of-the-art algorithms while using running times which are several orders of magnitude lower. We also show that FastSurv scales well with respect to the number of nodes in the network, and that it can deal with sparse as well as dense physical and logical graphs.

The rest of the paper is organized as follows. In section 2, a detailed description of the problem is given and related work is discussed. In section 3 the working of the algorithm is explained and in section 4 test results are presented.

2 Problem definition and related literature

For this paper, we assume that a physical WDM topology and a logical IP topology are given. The clear-channels of the logical topology need to be placed on paths in the physical topology in such a way that the complete routing is survivable. This means that no failure in the physical network can leave the logical network disconnected. In this section we focus on survivability with respect to single link failures. We elaborate on node and multiple simultaneous link failure survivability later, in subsection 3.3, when we explain how the algorithm can be extended to deal with this. Note that in reality survivable routing is a subproblem of the overall logical topology design problem, in which the logical topology needs to be generated before it can be routed onto the given physical topology (see [2,11]). In the following we give a formal description of the problem, and an overview of existing related literature.

2.1 Formal problem definition

The problem input consists of two undirected [1] graphs: $G_{ph}(V, L)$ representing the physical topology, and $G_{lo}(V, C)$ representing the logical topology. Both graphs need to be at least two-connected for survivable routing to be possible. V is the set of nodes in the network, and N is the number of nodes $|V|$. L is the set of links of the physical network. C is the set of clear-channels of the logical network. To obtain a solution to the problem, each clear-channel c of C should be routed onto a path in the physical graph G_{ph}. A complete routing is represented as a $|C| \times |L|$ routing matrix $R = (r_l^c)$, for which $r_l^c = 1$ if link l is on the path of clear-channel c.

A routing is evaluated by considering a failure of each of the links of the physical graph G_{ph} separately. When considering a failure of link l, l is temporarily removed from G_{ph}, and all clear-channels c for which $r_l^c = 1$ are temporarily removed from G_{lo}, giving rise to a partial logical graph G_{lo}^l. Using the clear-channels of this partial logical graph, an attempt is made to find a path connecting the endpoints s_c and d_c of each of the removed clear-channels c. If no path is found (s_c and d_c are disconnected in G_{lo}^l), we say that c is unsurvivable on l, and we call $\{c, l\}$ an unsurvivable pair. By considering all links

[1] Optical links are inherently unidirectional, so each undirected link is made up of two different optical fibers.

of G_{ph} one by one in this way an unsurvivability matrix $U = (u_l^c)$ is obtained, in which u_l^c is 1 if $\{c, l\}$ is an unsurvivable pair and 0 otherwise. The function to minimize is the total number of unsurvivable pairs, given in equation 1.

$$F(R) = \sum_{l \in L} \sum_{c \in C} u_l^c \tag{1}$$

The problem as it is described above does not take into account wavelength assignment. If we consider wavelengths, there are two extra constraints: the distinct wavelength constraint and the wavelength continuity constraint [14]. The distinct wavelength constraint states that all clear-channels routed on the same link should use a different wavelength. Since an optical fiber can only carry a limited number of wavelengths, this comes down to a capacity constraint for each link. To reflect this, we introduce a vector *cap* of size $|L|$, where entry cap_l contains the capacity of link l. The wavelength continuity constraint states that a clear-channel should use the same wavelength on all the links along its path. This constraint can be relaxed if nodes in the network are capable of wavelength conversion.

In this paper we first present an algorithm for survivable routing without taking into account link capacity constraints, and then an extension for the case with link capacity constraints. We do not consider the wavelength continuity constraint, assuming that all nodes are capable of full wavelength conversion.

2.2 Related literature

There have been a number of publications on survivable WDM routing as it is defined in subsection 2.1. In [1], a tabu search approach to find survivable routing is presented. This solution method does not take into account capacity constraints. [2] is the follow-up to the previous paper. It presents a slightly different algorithm, which does take capacity constraints into account. The last paper from the same research group is [3]. It considers also other forms of failure propagation (not only connectivity problems). The authors of [7] point out that the logical topology can only get disconnected by a single link failure if that link carries a full cut set of the logical graph. They formulate an ILP using this idea, providing a method to find exact solutions. Even though they did not include the link capacity constraints in the ILP for their test runs, the algorithm still needed very long run times. In [8] the same authors provide some heuristic ILP methods, in which only a subset of the cut sets is taken into account. This leads to good and faster results. Finally, in [16] a simple heuristic method is proposed for the survivable routing problem, and in [5] an algorithm based on a reduction of the problem is presented. Both of these last two algorithms cannot deal with link capacity constraints.

Some other papers consider related problems, or special cases of the survivable WDM routing problem. The algorithm proposed in [11] looks at the complete logical topology design problem: the input for the problem is a physical topology, an average traffic matrix and a routing algorithm. A tabu search algorithm generates a logical topology as an intermediate result, which is routed on the physical network using a simple heuristic. To find a survivable routing the logical topology is optimized, rather than the mapping of the logical topology onto the physical topology. The authors of [17] proof that not only the problem of finding a survivable routing, but also determining whether a survivable routing is possible for a certain combination of logical and physical topology is NP-Complete. In [18] the same authors describe a survivable routing algorithm for the special case where the logical topology is a ring. This specification can be justified by the fact that many protection mechanisms in the higher layers use ring structures. Finally, [6] and other papers by the same author treat the special case where the physical topology is a ring. This specification allows to deduct theorems about survivable designs which can be used to make heuristic solution methods.

3 The FastSurv survivable routing algorithm

This section gives an overview of the FastSurv algorithm. First, the basic algorithm is presented: it tries to find a survivable routing solution without taking into account link capacity constraints. Next, this algorithm is extended so that it does observe link capacities. Finally, we show how the algorithm can be adapted to provide survivability in the presence of node failures and multiple simultaneous link failures.

3.1 The basic FastSurv survivable routing algorithm

An overview of the FastSurv algorithm is given in figure 2. It starts from an initial solution obtained with a simple heuristic method and tries to improve it in subsequent iterations by rerouting a subset of the clear-channels each time. Solutions are evaluated as explained in subsection 2.1, and the algorithm reroutes all clear-channels which were unsurvivable on at least one of the links in the solution of the previous iteration. While rerouting FastSurv tries to avoid placing clear-channels together on the same link if they were unsurvivable when they shared links in previous iterations. This approach allows

Create an initial solution $R(0)$;

Evaluate the survivability of $R(0)$;

Initialize matrix $P(0)$, which contains information about which combinations of clear-channels were unsurvivable when routed together on the same link;

Set the number of iterations t to 0;

While (($R(t)$ is unsurvivable) AND

($t <$ the maximum number of iterations)) **do**

Reroute the unsurvivable clear-channels of $R(t)$, using matrix $P(t)$;

Evaluate the survivability of the new solution $R(t+1)$;

Update $P(t)$ to $P(t+1)$ and increment t;

End While

Fig. 2. The basic FastSurv survivable routing algorithm

to approximate the notion of cut sets explained in the introduction. This will be made clearer at the end of this section. The algorithm ends when the full routing is survivable or when a maximum number of iterations is reached.

To obtain an initial solution, the clear-channels are routed on the physical graph one by one in random order. The routing mechanism uses shortest path routing with the cost of a path depending on the clear-channels which were already routed on the network before. Suppose we have routed k clear-channels, and are calculating a shortest path for the $(k+1)^{th}$ clear-channel. If we indicate by $C^k \subset C$ the subset of the k first routed clear-channels, then the cost $cost^{k+1}(l)$ for the $(k+1)^{th}$ clear-channel to use link l is equal to the number of clear-channels c from C^k which use l, as indicated in formula 2. This simple algorithm avoids that some links carry a lot more clear-channels than others, which would make them more vulnerable with respect to survivability (a similar strategy is used in [16]).

$$cost^{k+1}(l) = \sum_{c \in C^k} r_l^c \tag{2}$$

After the initial solution is constructed, FastSurv reroutes at every new iteration all clear-channels which were unsurvivable on at least one link in the solution of the previous iteration. In what follows we indicate by $R(t) = [r_l^c(t)]$ the routing matrix of the solution of iteration t and by $U(t) = [u_l^c(t)]$ the unsurvivability matrix of the solution of iteration t. The initial solution is referred to as iteration 0. Formally, the clear-channels to be rerouted in iteration $t+1$ are the ones for which $\sum_{l \in L} u_l^c(t) > 0$. They are taken away from the solution $R(t)$ of iteration t, while the other clear-channels are left routed as they were. A new solution $R(t+1)$ is obtained by placing the removed clear-channels in random order and routing them back onto the physical graph. While doing the rerouting, the algorithm tries to avoid routing together clear-channels which were in the previous iterations unsurvivable when routed together over the same link. So when the algorithm is rerouting a clear-channel c_i, the cost to use a link l depends on which other clear-channels use l. Consider a clear-channel c_j which uses l: if c_i and c_j have shared a link in a previous iteration, and they were both unsurvivable on this shared link, the cost for c_i to use l will be high. The rest of this subsection is dedicated to the explanation of how exactly this is done, and what the reasoning is behind this mechanism.

The information necessary for the rerouting is kept in a $|C| \times |C|$ matrix $P(t) = [p_{c_i c_j}(t)]$. The entries $p_{c_i c_j}(t)$ of this matrix are updated according to formulas 3-5. In formula 3, $a_{c_i c_j}(t)$ is defined as the number of times that clear-channels c_i and c_j shared a link in the solution of iteration t, and in formula 4, $b_{c_ic_j}(t)$ is defined as the number of times that clear-channels c_i and c_j were both unsurvivable on a shared link in iteration t. Dividing $b_{c_ic_j}(t)$ by $a_{c_ic_j}(t)$, one obtains a ratio which can be seen as an estimate (based on the experience of iteration t) of the probability that clear-channels c_i and c_j both become unsurvivable when they are routed together. $p_{c_ic_j}(t)$ is then defined in formula 5 as the exponential average of these probability estimates (with $\alpha \in [0,1]$). The entries of the matrix $P(0)$ are initialized using the average value $(\sum_{c_ic_j} b_{c_ic_j}(0))/(\sum_{c_ic_j} a_{c_ic_j}(0))$, obtained from the initial solution $R(0)$.

$$a_{c_ic_j}(t) = \sum_l r_l^{c_i}(t) r_l^{c_j}(t) \qquad \forall c_i, c_j \in C \tag{3}$$

$$b_{c_ic_j}(t) = \sum_l u_l^{c_i}(t) u_l^{c_j}(t) \qquad \forall c_i, c_j \in C \tag{4}$$

$$p_{c_ic_j}(t) = \begin{cases} \alpha p_{c_ic_j}(t-1) + (1-\alpha)\frac{b_{c_ic_j}(t)}{a_{c_ic_j}(t)} & if \quad a_{c_ic_j}(t) > 0 \\ p_{c_ic_j}(t-1) & if \quad a_{c_ic_j}(t) = 0 \end{cases} \tag{5}$$

The probability estimates of $P(t)$ are used to reroute clear-channels: a shortest path algorithm is applied in which the cost of a path for a clear-channel c_i is the probability that c_i will be unsurvivable along the path. The probability $Prob_{path}^{c_i}(t)$ that c_i will be unsurvivable on a path is the probability that it will be unsurvivable on at least one link of the path. The probability $Prob_l^{c_i}(t)$ that c_i will be unsurvivable on a link l of the path is the probability that c_i will be unsurvivable when routed together with any of the other clear-channels which use l (other clear-channels can already be using l either because they were not removed after the previous iteration or because they were rerouted before c_i). In the shortest path algorithm (which is an adaptation of the standard Dijkstra algorithm [12]), we use formula 6 to estimate $Prob_l^{c_i}(t)$ and formula 7 to estimate $Prob_{path}^{c_i}(t)$. In these formulas independence of probabilities is assumed, even though this is often not the case. For our heuristic solution method this rough approach is a good enough guideline towards better solutions.

$$Prob_l^{c_i}(t) = 1 - \prod_{c_j \text{ on } l} (1 - p_{c_i c_j}(t)) \tag{6}$$

$$Prob_{path}^{c_i}(t) = 1 - \prod_{l \text{ on } path} (1 - Prob_l^{c_i}(t)) \tag{7}$$

The algorithm described here is based on the observation mentioned in the introduction that a routing is survivable if and only if no link is shared by all clear-channels belonging to a cut set of the logical network. In FastSurv, we approximate the notion of the cut sets by the probability estimates $p_{c_i c_j}(t)$. A look at the example logical graph given in figure 1 (B) could shed a bit more light on the meaning of $p_{c_i c_j}(t)$. In this graph, the pair of clear-channels $\{b, d\}$ forms a cut set of size two. b and d will both be unsurvivable every time they are routed together, and therefore $p_{bd}(t)$ will always be 1. $p_{be}(t)$ on the other hand will normally be lower than 1, since b and e together do not form a cut set. $\{b, e\}$ is part of the larger cut set $\{b, e, f\}$ though, and therefore b and e will both be unsurvivable if they are routed together with the third clear-channel f. So, when two clear-channels c_i and c_j do not form a cut set of their own, the probability $p_{c_i c_j}(t)$ depends on which other clear-channels are also routed together with them. If, going back to the example, the situation is such that f is often routed together with b and e (e.g. because of the structure of the physical graph), this will be reflected in high values for $p_{be}(t)$, $p_{bf}(t)$ and $p_{ef}(t)$, and the cut set will be taken into account by the algorithm. If on the other hand f is hardly ever routed together with b and e, all three probabilities will be low: the cut set will be considered unimportant and the algorithm will not take it into account. Also for very large cut sets, it will hardly ever happen that all the elements of the cut set are routed together on the same link, and therefore the individual pairwise probabilities $p_{c_i c_j}(t)$ among the elements of the cut set will be very low. So the pairwise probabilities can be seen as a simple way to focus only on important cut sets. This way the algorithm can use information about the structure of the logical graph efficiently.

At this point, it can also be easy to understand why we need the discount factor α in formula 5: the routing of the whole set of clear-channels changes from iteration to iteration, and therefore also the probabilities. It could for example be the case that in early iterations, clear-channel f of our example problem is routed close to b and e, so that placing b and e together usually leads to unsurvivable solutions, and $p_{be}(t)$ is high. However, in later iterations this might no longer be true due to a different routing of f. It is then good that $p_{be}(t)$ is lowered again, since placing b and e on the same link is now less likely to cause a problem, due to the absence of f. Therefore we use a moving average to gradually lower the importance of older estimates.

3.2 FastSurv for survivable routing with link capacities constraints

The FastSurv algorithm as it is described in the previous subsection can find a survivable routing, but does not take into account link capacity constraints. In this subsection we present an extension for this purpose. The full algorithm is presented in figure 3. A full iteration of the algorithm consists of a number of iterations of the previously described survivable routing algorithm, which we will call *survivability iterations*, and then a number of iterations to decrease the number of link capacity constraint violations, which we will call *capacity iterations*. The survivability iterations are run until the routing is survivable or the maximum number of iterations (empirically set to 2) is reached, and the capacity iterations are run until there is no further reduction in the number of capacity constraint violations. The capacity constraint violations are evaluated by taking the sum of the overcapacity over all links, as indicated in formula 8. The full iterations (the combination of survivability and capacity iterations) are repeated until the routing is survivable and at the same time observes all link capacity constraints, or until a maximum number of iterations is reached.

$$F_{cap}(R) = \sum_{l \in L} max\left(\left(\sum_{c \in C} r_l^c\right) - cap_l, 0\right) \tag{8}$$

Like in the survivability iterations, FastSurv tries in each capacity iteration to improve the solution of the previous iteration by rerouting a number of clear-channels. The clear-channels to be rerouted are chosen randomly from among the clear-channels which are routed over at least one link with overcapacity. More formally, a clear-channel c is considered for rerouting if $\sum_{l \in L}(r_l^c * max((\sum_{c' \in C} r_l^{c'}) - cap_l, 0)) > 0$. The maximal number of clear-channels to be rerouted in one capacity iteration step is 10 percent of the total number of clear-channels in the graph. This number was set empirically because it gave good results over a wide range of different test problems. Too high a number makes a step in the local search too large, whereas too low a number does not allow the algorithm to escape from local optima.

Create an initial solution $R(0)$;

Evaluate $R(0)$ for survivability and link capacity constraint violations;

Initialize the matrix of pairwise probabilities $P(0)$;

Set the number of full iterations t_f to 0;

While (($R(t_f)$ is unsurvivable or violates capacity constraints) AND

($t_f <$ the maximum number of full iterations)) **do**

Set the number of survivability iterations t_s to 0, the initial routing matrix for the survivability iterations $R_s(0)$ to the current routing matrix of the full iterations $R(t_f)$, and $P_s(0)$ to $P(t_f)$;

While (($R_s(t_s)$ is unsurvivable) AND

($t_s <$ the maximum number of survivability iterations)) **do**

Reroute the unsurvivable clear-channels of $R_s(t_s)$, using $P_s(t_s)$;

Evaluate the new solution $R_s(t_s + 1)$ for survivability;

Update $P_s(t_s)$ to $P_s(t_s + 1)$ and increment t_s;

End While

Set $R(t_f)$ to $R_s(t_s)$ and $P(t_f)$ to $P_s(t_s)$;

Set the number of capacity iterations t_c to 0, and the initial routing matrix for the capacity iterations $R_c(0)$ to $R(t_f)$;

Do

Reroute clear-channels which are on overfull links in $R_c(t_c)$;

Evaluate $R_c(t_c)$ for capacity constraint violations and increment t_c;

Until (there is no more reduction in capacity constraint violations)

Set $R(t_f + 1)$ to $R_c(t_c)$ and increment t_f;

End While

Fig. 3. The FastSurv algorithm for survivable routing with link capacity constraints

All the chosen clear-channels are removed from the routing matrix of the previous iteration and they are rerouted one by one in random order. The rerouting is again done with shortest path routing. Like in the heuristic used to obtain the initial solution for the basic algorithm, the cost of a link is equal to the total number of clear-channels on the link. However, if the total number of clear-channels on the link is lower than the link's capacity, this cost is now divided by the link's capacity. In this way overfull links get a much higher cost, and hence they are avoided. Formally, if $C^k \subset C$ is the set of the k clear-channels which are already routed (either because they were not chosen for rerouting, or because they have already been rerouted), the cost of using link l for the $(k+1)^{th}$ clear-channel is given in formula 9. Also for the initial routing we now use cost formula 9 rather than 2. The capacity iterations have some similarities with a local search for routing and wavelength assignment described by Nagatsu et Al. in [10].

$$cost^{k+1}(l) = \begin{cases} \frac{\sum_{c \in C^k} r_l^c}{cap_l} & \text{if } \sum_{c \in C^k} r_l^c < cap_l \\ \sum_{c \in C^k} r_l^c & \text{if } \sum_{c \in C^k} r_l^c \geq cap_l \end{cases} \tag{9}$$

3.3 FastSurv for node failures and multiple simultaneous link failures

A routing is survivable with respect to node failures if no single node failure can leave the logical topology disconnected. The main difference with respect to the problem description in subsection 2.1 is that a solution is not evaluated by removing the links l one by one, but by removing the nodes n. Clear-channels which are incident on n can never be routed survivably with respect to a failure of n, and they should therefore be left out of consideration when removing n

during the evaluation. Adapting the algorithm description of subsection 3.1 is straightforward: the probability that a clear-channel is unsurvivable on a path is the probability that it is unsurvivable on a node in its path, and the probability that it is unsurvivable on a node is the probability that it is unsurvivable when routed together with any of the other clear-channels on the node. Clear-channels incident on a node are also not taken into account for the probability calculations.

For the case of multiple simultaneous link failures, one usually defines shared risk groups [4]. These are groups of links which are likely to fail together (e.g. because they share a conduit [19]). The problem description of subsection 2.1 can easily be adapted to take this into account. We define a number of shared risk groups srg_m, which are sets of links. A solution is evaluated by considering the shared risk groups srg_m one by one, and removing all the links $l \in srg_m$ simultaneously. Adapting the algorithm of subsection 3.1 is again straightforward: the probability that a clear-channel is unsurvivable on a link l in its path is now the probability that it is unsurvivable with any of the clear-channels routed over any of the links in the shared risk group(s) to which l belongs.

4 Test results

In this section we describe test results obtained with FastSurv. We compare it to current state-of-the-art algorithms on different physical and logical network topologies. Using test problems of increasing sizes, we show that FastSurv is much more scalable than existing algorithms. Subsection 4.1 contains results for the basic survivable routing algorithm, and subsection 4.2 for the algorithm with link capacities. Our programs are implemented in C++ and all tests were run on a PC with 2.4 GHz Intel Pentium 4 processor.

4.1 Test results for the basic FastSurv algorithm

For the case without link capacities, we made comparisons with the full and the relaxed ILP methods presented in [8], and with the tabu search algorithms presented in [1] and [2]. For every test problem, FastSurv is given 100 iterations. Since FastSurv is a local search algorithm, it can get stuck in local optima. The 100 iterations are therefore spread over 10 random restarts with 10 iterations each (these numbers were chosen empirically). The tabu search algorithms are run with the parameters given in their respective papers.

For the comparison with the ILP methods, we ran FastSurv on the same test problems which were used in [8]. The physical network used for these tests is the 14-node 21-link NSFNET. The logical networks were randomly generated by the authors of [8]. There are logical networks of degree 3, 4, and 5, where a network of degree n is defined as one in which all nodes have degree n. For each degree there are 100 logical networks. The algorithms are run once for each test problem. The results are summarized per network degree in table 1, where *ILP* refers to the full ILP solution method, and *relax-1* and *relax-2* to the two relaxations of the ILP, which use only a subset of the survivability constraints. *Unsurvivable* is the number of problems for which no survivable solution was found, and *Time* is the average run time per problem in CPU seconds. The results show that FastSurv gives good results in short times. The other approaches are much slower (especially ILP), and relax-1 is not able to solve all problems.

Table 1

Tests on the NSFNET physical network using 100 logical networks of degrees 3, 4 and 5. This table reports the number of problems which where not mapped survivably and the average run time per problem in CPU seconds.

Network Degree		3	4	5
FastSurv	Unsurvivable	0	0	0
	Time	0.0117	0.0155	0.0166
ILP	Unsurvivable	0	0	0
	Time	8.3	173	1157
Relax-1	Unsurvivable	10	0	0
	Time	1.3	1.5	2.0
Relax-2	Unsurvivable	0	0	0
	Time	1.3	1.5	2.0

For the comparison with the tabu search algorithms, we could not obtain the original test problems used by the authors, nor the source code, so the results presented here are obtained with our own implementation of the algorithms presented in [1] and [2]. Strictly speaking, only the tabu search of [1] (which we refer to as *TS97*) addresses exactly the same problem as FastSurv, since the tabu search of [2] (*TS98*) takes link capacities into account. Nevertheless, there are differences in TS98 which make it more efficient. Therefore we made an adaptation of TS98 which does not take link capacities into account (*TS*03*).

As a first series of tests we ran the algorithms on the same physical network

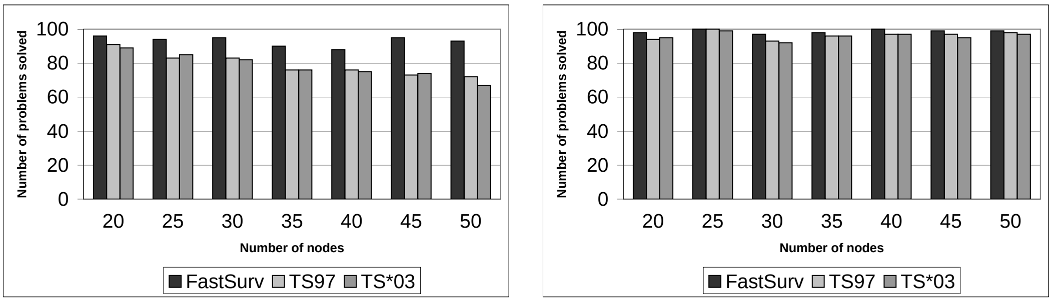
Table 2

Tests on the ARPA2 physical network using two logical networks of degree 3 and two of degree 4, with 10 runs per test problem. This table reports the average time per run in CPU seconds (*Time*), and the number of runs in which no survivable routing was found (*Uns*).

		TS97		TS*03		FastSurv	
		Time	Uns	Time	Uns	Time	Uns
Degree 3	1	4.34	0	0.31	0	0.07	0
	2	4.91	0	1.04	0	0.05	0
Degree 4	1	5.38	0	2.37	2	0.02	0
	2	4.75	0	0.33	0	0.02	0

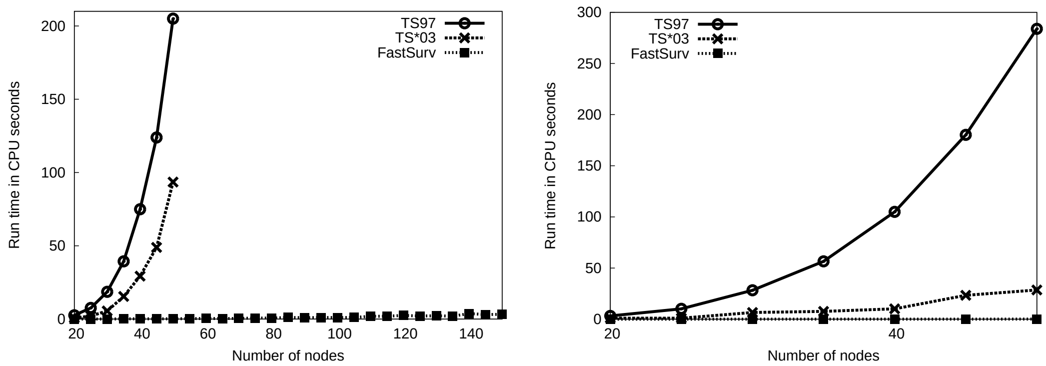
which was used in [1] and [2]: the 21-nodes, 26-links ARPA2 network. As logical networks we used 4 randomly generated graphs: 2 of average node degree 3 and 2 of average node degree 4. We ran each algorithm 10 times for each logical network. The results are given in table 2, where *Time* is the average run time per problem in CPU seconds, and *Uns* is the number of runs in which no solution was found. All three algorithms always find a survivable solution, except TS*03 in two cases. It is quite clear however that FastSurv has much shorter run times.

We then ran a much larger series of tests, with networks of increasing sizes, in order to compare the scalability of the algorithms. Physical networks were randomly generated with number of nodes ranging from 20 to 50 with a step size of 5, and with average node degree 3. Logical networks of the same number of nodes were generated with average degrees 3 and 4. For each size and



(a) (b)

Fig. 4. Number of problems routed survivably, using random physical graphs of degree 3 and logical graphs of (a) degree 3 and (b) degree 4, with increasing number of nodes, and 100 test problems per network size.



(a) (b)

Fig. 5. Average run time in CPU seconds, using random physical graphs of degree 3 and logical graphs of (a) degree 3 and (b) degree 4, with increasing number of nodes, and 100 test problems per network size.

each degree there are 10 different physical networks and 10 different logical networks. We ran each algorithm once for each combination of physical network and logical network, giving 100 results per network size and node degree combination. The results are summarized in figures 4 and 5. The bars in figure 4 indicate for each network size how many problems (out of 100) each algorithm managed to route survivably. Figure 4a contains the results for the logical graphs of degree 3, and figure 4b for logical graphs of degree 4. TS97 and TS*03 show comparable performance over this wide range of problems, while FastSurv performs consistently better. The graphs of figure 5 indicate the average run time per problem in CPU seconds needed by the different algorithms on the networks of degree 3 (figure 5a) and degree 4 (figure 5b). They confirm that FastSurv is much more scalable with respect to the network size. For degree 3, we ran FastSurv on more problems, up to 150 nodes. The run times of the other algorithms were prohibitively large for these problem sizes. FastSurv shows continued short run times for these larger problems.

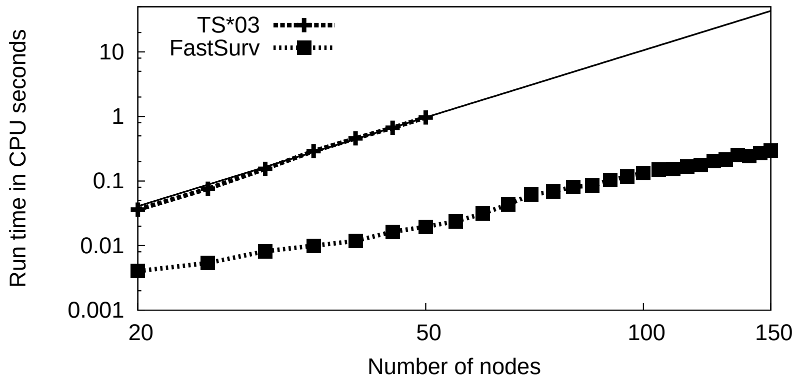
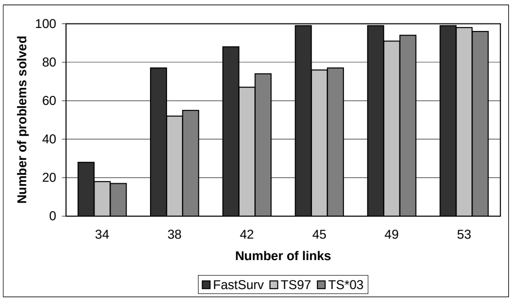
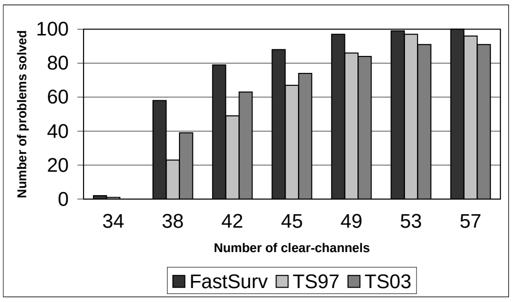


Fig. 6. Average run time *per iteration* in CPU seconds, using the random physical and logical networks of degree 3 with increasing number of nodes, in log-log scale.

An explanation for the faster run times of FastSurv can be found in the complexity of the iterations. The tabu search algorithms try in each iteration to reroute each clear-channel separately, and pick the rerouting which improves the solution most. Rerouting a clear-channel has a complexity of $O(Dijkstra)$, which is maximally $O(N^2)$. To pick the rerouting which improves the solution most, one has to evaluate the solution produced by each rerouting. Solution evaluation is done as described in subsection 2.1, removing each link in turn, and with it all clear-channels that use it, and trying to connect the endpoints of the removed clear-channels in the remaining partial logical graph. Therefore the whole evaluation has a complexity of $|L| \times |C| \times O(Dijkstra)$. Following [2], we consider the average node degrees of G_{lo} and G_{ph} as fixed, and express the number of clear-channels $|C|$ as αN, and the number of links $|L|$ as βN. The



(a)

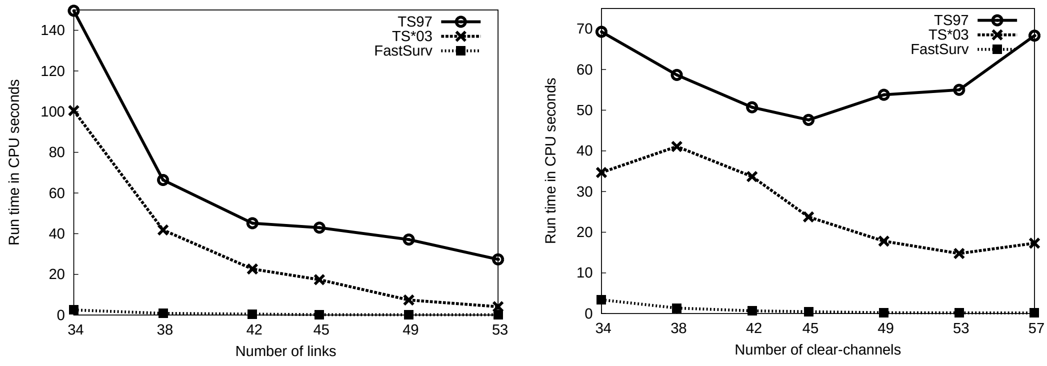


(b)

Fig. 7. Number of problems routed survivably on 30-node networks with (a) increasing number of links and (b) increasing number of clear-channels, using 100 test problems per network setup.

complexity of an evaluation is then of $O(N^4)$. Each iteration, consisting of a rerouting and an evaluation for each clear-channel, then has a complexity of $O(\alpha N \times (N^2 + N^4)) = O(N^5)$. In FastSurv a number of clear-channels are rerouted in each iteration using the probabilities, and the evaluation is done only once, at the end of the iteration, which means that one iteration has a maximal complexity of $O(\alpha N \times N^2 + N^4) = O(N^4)$. The fact that the complexities are polynomial is confirmed in figure 6, where we compare the average run time per iteration for FastSurv and TS*03: the run times per iteration plotted against the number of nodes in log-log scale follow more or less a straight line for both algorithms (for TS*03 values above 50 nodes were extrapolated). The gradients are lower than the theoretically predicted maxima though: 2.15 for FastSurv and 3.45 for TS*03. Other explanations for the better scalability of FastSurv are the fact that more than one clear-channel is rerouted in each iteration, and the fact that FastSurv uses information about the relationship between clear-channels, which is not done in the tabu search algorithms.

Looking at figure 4, it is clear that all three algorithms find better solutions for logical networks of degree 4 (figure 4b) than for logical networks of degree



(a) (b)

Fig. 8. Average run time in CPU seconds on 30 node networks with (a) increasing number of links and (b) increasing number of clear-channels.

3 (figure 4a). This is because networks with higher connectivity are easier to map survivably, since more alternative paths are available in the logical topology. Also better connectivity of the physical network makes the survivable routing problem easier, since there are more possibilities to spread the clear-channels of each cut set over different physical paths. We ran a series of tests in which we kept the number of nodes constant and varied the connectivity of the physical and the logical network. In all these tests, the number of nodes was 30. In the first set of tests, the number of clear-channels was kept to 45 (degree 3), while the number of links was varied from 34 (degree 2.25, slightly higher connectivity than a ring) to 53 (degree 3.5). In the second set of tests, the number of links was kept to 42 (degree 2.75), while the number of clear-channels was varied from 34 to 57 (degree 3.75). As before, 10 different logical and physical graphs were generated randomly for each combination of parameter values, giving 100 different test problems. As can be seen in figures 7a (for varying number of links) and 7b (for varying number of clear-channels), all three algorithms have an increasing performance with increasing connectivity of both physical and logical graphs, but for FastSurv this increase is faster. Also in terms of solution time, presented in figures 8a (for varying physical

connectivity) and 8b (for varying logical connectivity), FastSurv outperforms the other algorithms. For the tabu search algorithms the evolution in solution time with respect to the logical connectivity (figure 8b) is not monotonic. This is because the tabu search algorithms try to reroute all clear-channels at each iteration: while more clear-channels make the problem easier, they also lead to longer iterations.

4.2 Test results for the extended FastSurv algorithm

For the extended algorithm we only compare to TS98 [2], since it is the only one which takes link capacities into account. As physical networks, we again use the ARPA2 network and the randomly generated networks with increasing number of nodes of the previous subsection, and we give a maximum capacity for each link. The link capacities are the same for each link of the network, but they are set differently for different logical graphs. For the ARPA2 network, link capacities are set to 7 for degree 3 logical networks and to 8 for degree 4 logical networks. For the randomly generated networks, capacities are set to 6 for degree 3 logical networks up to 40 nodes, to 7 up to 55 nodes, and to 8 up to 150 nodes. For degree 4 logical networks, capacities are set to 7 up to 30 nodes and 8 up to 50 nodes. As logical networks, we use the same as in the previous subsection.

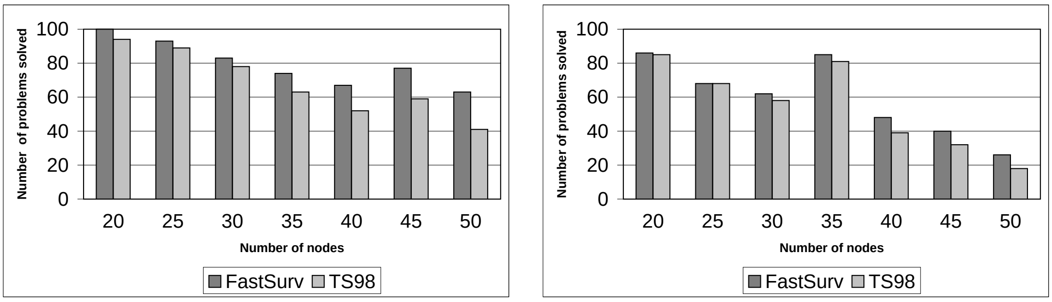
The results for the ARPA2 network are presented in table 3, where *Time* is the average run time per problem in CPU seconds, *Au* the average number of unsurvivable pairs (according to formula 1) and *Ac* the average overcapacity (according to formula 8). *Best* is the best solution over 10 runs, with on the left its number of unsurvivable pairs, and on the right its overcapacity. The

Table 3

Tests on the ARPA2 physical network with link capacities, using two logical networks of degree 3 and two of degree 4, with 10 runs per test problem. This table reports the average time per run in CPU seconds (*Time*), the average number of unsurvivable pairs (*Au*) and the average overcapacity (*Ac*) per run, and the number of unsurvivable pairs and overcapacity for the best run (*Best*).

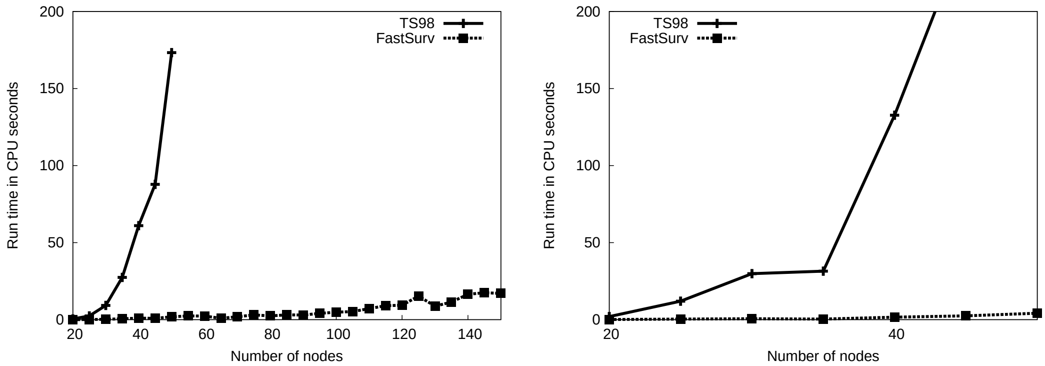
		TS98				FastSurv			
		Time	Au	Ac	Best	Time	Au	Ac	Best
Degree 3	1	2.69	0	0.3	0/0	0.13	0	0	0/0
	2	1.89	0	0.2	0/0	0.18	0.2	0	0/0
Degree 4	1	4.79	0.8	0	0/0	0.05	0	0	0/0
	2	4.90	0	0.4	0/0	0.31	0	0.3	0/0

best solution is the one with the lowest sum of these two values. FastSurv gives comparable to better results than TS98, and again its run times are much shorter. The results on the random graphs are shown in figures 9 and 10, where figure 9 presents the number of test problems solved to optimality (meaning survivable solutions without any overcapacity) for logical networks of degree 3 (figure 9a) and degree 4 (figure 9b), and figure 10 presents the average run time in CPU seconds per problem for logical networks of degree 3 (figure 10a) and degree 4 (figure 10b). These results confirm FastSurv's advantage both in terms of solution quality and run time, and show that also the extended FastSurv algorithm scales much better than TS98 with respect to the number of nodes. For the degree 3 problems we again ran FastSurv for networks up to 150 nodes to show the continued good performances. The discontinuities



(a) (b)

Fig. 9. Number of problems routed survivably under link capacity constraints, using random physical graphs of degree 3 and logical graphs of (a) degree 3 and (b) degree 4, with increasing number of nodes, and 100 test problems per network size.



(a) (b)

Fig. 10. Average run time in CPU seconds for problems with link capacity constraints, using random physical graphs of degree 3, and logical graphs of (a) degree 3 and (b) degree 4, with increasing number of nodes.

visible at some points in the figures (e.g., at 35 nodes in figures 9b and 10b) are due to changes in the link capacities at those points.

5 Conclusions

We have described FastSurv, a local search algorithm for survivable routing in WDM networks. FastSurv works in an iterative way: in each iteration it improves its previous solution using information learned from solutions of earlier iterations. In an extensive series of test runs, we have compared FastSurv to other algorithms for survivable WDM routing. It gave better results while using much shorter run times. Moreover, the advantages in terms of solution quality and run time became larger for increasing network sizes. Also when increasing the difficulty of the problem by decreasing the connectivity of the physical and logical graphs, FastSurv proved more effective and efficient.

The property of giving high quality results in very short time can be important. As was pointed out in section 2, routing a logical topology survivably on a physical network is normally part of a larger logical topology design algorithm, and a time-consuming algorithm for this subproblem could considerably slow down the larger algorithm. For example, in [11] the authors need to do survivable routing as part of their algorithm, but have to use a greedy heuristic since existing state of the art algorithms take too long.

6 Acknowledgements

We thank Christoph Ambühl, Gianni Di Caro and Roberto Montemanni for useful help and advice, and Aradhana Narula-Tam for sharing the test problems used in her work. This work was partially supported by the Swiss HASLER Foundation project DICS-1830.

References

[1] J. Armitage, O. Crochat, and J.-Y. Le Boudec. Design of a survivable WDM photonic network. In *Proceedings of IEEE INFOCOM*, 1997.

[2] O. Crochat and J.-Y. Le Boudec. Design protection for WDM optical networks. *IEEE Journal on Selected Areas in Communications, Special Issue on High-Capacity Optical Transport Networks*, 16(7), 1998.

[3] O. Crochat, J-Y. Le boudec, and O. Gerstel. Protection interoperability for WDM optical networks. *IEEE Transactions on Networking*, 8(3), 2000.

[4] I.P. Kaminow and T.L. Koch. *Optical Fiber Telecommunications IIIA*. Academic Press, 1997.

[5] M. Kurant and P. Thiran. Survivable mapping algorithm by ring trimming (SMART) for large IP-over-WDM networks. In *Proceedings of the First Annual International Conference on Broadband Networks (BroadNets)*, 2004.

[6] H. Lee, H. Choi, S. Subramaniam, and H.-A. Choi. Survivable embedding of logical topology in WDM ring networks. *International Journal of Information Sciences – Informatics and Computer Science*, 149(1), 2003.

[7] E. Modiano and A. Narula-Tam. Survivable routing of logical topologies in WDM networks. In *Proceedings of IEEE INFOCOM*, 2001.

[8] E. Modiano and A. Narula-Tam. Survivable lightpath routing: a new approach to the design of WDM-based networks. *IEEE Journal on Selected Areas in Communications*, 20(4), 2002.

[9] B. Mukherjee. *Optical Communication Networks*. McGraw-Hill, 1997.

[10] N. Nagatsu, Y. Hamazumi, and K.I. Sato. Number of wavelengths required for constructing large-scale optical path networks. *Electronics and Communications in Japan, Part I - Communications*, 78(9), 1995.

[11] A. Nucci, B. Sanso, T.G. Crainic, E. Leonardi, and M.A. Marsan. Design of fault-tolerant logical topologies in wavelength-routed optical IP networks. In *Proc. of the IEEE Global Telecommunications Conference (Globecom)*, 2001.

[12] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[13] R. Ramaswami and K.N. Sivarajan. *Optical Networks: a Practical Perspective.* Morgan Kaufmann Publishers Inc., 1998.

[14] G. Rouskas. *Wiley Encyclopedia of Telecommunications*, chapter Routing and Wavelength Assignment in Optical WDM Networks. Wiley and Sons, 2001.

[15] L. Sahasrabuddhe, S. Ramamurthy, and B. Mukherjee. Fault management in IP-over-WDM networks: WDM protection versus IP restoration. *IEEE Journal on Selected Areas in Communications*, 20(1), 2002.

[16] G.H. Sasaki, C.-F. Su, and D. Blight. Simple layout algorithms to maintain network connectivity under faults. In *Proceedings of the* 38^{th} *Annual Allerton Conference on Communication, Control, and Computing*, 2000.

[17] A. Sen, B. Hao, and B.H. Shen. Survivable routing in WDM networks. In *Proceedings of the* 7^{th} *International Symposium on Computers and Communications (ISCC)*, 2002.

[18] A. Sen, B. Hao, B.H. Shen, and G.H. Lin. Survivable routing in WDM networks – Logical ring in arbitrary physical topology. In *Proceedings of the International Communication Conference (ICC)*, 2002.

[19] J. Strand, A.L. Chiu, and R. Tkach. Issues for routing in the optical layer. *IEEE Communications Magazine*, 39(2), 2001.

[20] H. Zang, J.P. Jue, and B. Mukherjee. A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. *SPIE Optical Networks Magazine*, 1(1), 2000.