Successive Survivable Routing for Node Failures

Yu Liu
OPNET Technologies, Inc.
200 Regency Forest Drive Suite 150, Cary, NC 27511

David Tipper
Dept. of Information Science and Telecommunications
University of Pittsburgh, Pittsburgh, PA 15260

***Abstract* — This paper addresses the spare capacity allocation (SCA) problem considering any single node failure in mesh networks. The SCA node failure problem aims at finding backup routes and providing sufficient spare capacity to protect traffic when any single node fails in communication network. Here, we introduce our novel matrix formulation of the arc-flow SCA node failure model. In this model, working paths are given before pre-planned backup paths are routed and reserved. Because backup paths can not be guaranteed if general shortest path routing of working paths is used, we give a graph algorithm to find working path which has at least one node-disjoint backup path. We extend our recent approximation algorithm, successive survivable routing (SSR), to solve the above SCA model. Numerical comparison shows that SSR has the best trade-off between solution optimality and computation speed.**

***Keywords*— spare capacity allocation, protection and restoration, network planning and optimization, network survivability**

I. Introduction

NETWORK survivability techniques have been proposed to guarantee seamless communication services in the face of network failures. Traditionally they include two phases, survivable network design and restoration scheme. They are complementary to each other and cooperate to achieve seamless service upon failures. The *spare capacity allocation* (SCA) problem is to decide how much spare capacity should be reserved on network links for given traffic flows and their working paths on two-connected mesh networks. It is part of survivable network design and is NP-complete [1]. Many research efforts on this issue has been done on SONET/SDH [2], [3], [4], [5], ATM [6], [7], [8], [9], WDM networks [10], [11], [12]. Recent issue on IP and MPLS networks are also given in [13], [14], [15]. A detailed literature review on different spare capacity allocation algorithms and restoration schemes is given in [16], [1].

In this paper, we first formulate an arc-flow spare capacity allocation problem to consider node failures on directed mesh networks. In this model, each flow has to find a node-disjoint backup path in order to protect any single intermediate node failure along its working path. A graph algorithm is used to find a working path which has at least one node-disjoint backup path. Such a protectable working path can not be guaranteed when it is found by shortest path routing due to trap topologies. Next, we briefly introduce the successive survivable routing (SSR) and use it to solve the SCA model. The numerical results comparing several SCA algorithms shows that SSR can find near optimal solution in real time.

Supported by grants: NSF ANIR 9980516 and DARPA F30602-97-1-0257

II. Arc-flow model for node failures

In this paper, the spare capacity allocation (SCA) problem is formulated to protect against any single node failure. In this section, we consider failure-independent path restoration (FID). All traffic flows require a 100% restoration for any given failure scenario. It requires that all affected flows can be detoured to their backup paths upon any given failure scenario. Provisioning enough spare capacity on links is the prerequisite condition to such traffic protection and restoration.

Given a network topology and the working paths, the objective of SCA is to minimize the total cost of spare capacity on network links. The decision variables are the backup paths and the spare capacity reservation on links. A way to reduce the total cost is sharing spare capacity on common links of backup paths whose working paths are link-disjoint.

A network is represented by a directed graph of N nodes and L links with R flows. The link capacity is unlimited for the simplicity of introduction and a treatment to capacitated networks can be generalized [1].

A flow $r, 1 \leq r \leq R$ is specified by its origin/destination node pair $(o(r), d(r))$ and traffic demand m_r. Working and backup paths of flow r are represented by two $1 \times L$ binary row vectors $\boldsymbol{p}_r = \{p_{rl}\}$ and $\boldsymbol{q}_r = \{q_{rl}\}$. The l-th element in these vectors equals to one if and only if the corresponding path passes link l. Path-link incidence matrices for working and backup paths of all flows are the collections of these row vectors, forming $R \times L$ matrices $\boldsymbol{P} = \{p_{rl}\}$ and $\boldsymbol{Q} = \{q_{rl}\}$ respectively. Let $\boldsymbol{M} = \text{Diag}(\{m_r\}_{R\times 1})$ denote the diagonal matrix representing the demands of flows. Note that if the protection level is under 100%, the elements in $\boldsymbol{M}$ can be adjusted to reserve partial spare capacities on back paths.

We characterize K failure scenarios in by a binary matrix $\boldsymbol{F} = \{\boldsymbol{f}_k\}_{K\times 1} = \{f_{kl}\}_{K\times L}$. The row vector $\boldsymbol{f}_k$ in $\boldsymbol{F}$ is for failure scenario k and its element f_{kl} equals one if and only if link l fails in scenario k. In this way, we can consider scenarios with multiple simultaneously failed links. We also define a flow failure matrix $\boldsymbol{U} = \{\boldsymbol{u}_r\}_{R\times 1} = \{u_{rk}\}_{R\times K}$, where $u_{rk} = 1$ if flow r will be affected by failure k, and $u_{rk} = 0$ otherwise. The single node failure is, hence, captured by an equivalent failure of all its adjacent links. Considering node failures, each flow has to avoid its source and destination nodes. This modification is to avoid unwanted distractions from unrestorable flows during the algorithm. In this way, we introduce two matrices $\boldsymbol{D}^o$ and $\boldsymbol{D}^d$ to indicate the source and destination nodes of a flow. These two nodes are excluded from the failures considered for the corresponding flows.

TABLE I

NOTATION

N, L, R, K	Numbers of node, link, flow & failure
n, l, r, k	Indices of node, link, flow, and failure
$\boldsymbol{P} = \{\boldsymbol{p}_r\} = \{p_{rl}\}$	Working path-link incidence matrix
$\boldsymbol{Q} = \{\boldsymbol{q}_r\} = \{q_{rl}\}$	Backup path-link incidence matrix
$\boldsymbol{M} = \text{Diag}(\{m_r\})$	Diagonal matrix of flow demands
$\boldsymbol{G} = \{g_{lk}\}$	Spare provision matrix
$\boldsymbol{G}^r = \{g^r_{lk}\}$	Contribution of flow r to $\boldsymbol{G}$
$\boldsymbol{s} = \{s_l\}_{L\times 1}$	Vector of spare capacity on links
$\boldsymbol{\phi}(\boldsymbol{s}) = \{\phi_l(s_l)\}_{L\times 1}$	Unit cost function of spare capacity
W, S	Total working, spare capacity
$\eta = S/W$	Network redundancy
$o(r), d(r)$	Origin/destination nodes of flow r
$\boldsymbol{v}_r = \{v_{rl}\}$	Link metrics for flow r
$\boldsymbol{B}\{b_{nl}\}_{N\times L}$	Node-link incidence matrix
$\boldsymbol{D} = \{d_{rn}\}_{R\times N}$	Flow-node incidence matrix
$\boldsymbol{D}^o$, $\boldsymbol{D}^d$	Binary incidence matrixes between flow and source node; or destination node, $\boldsymbol{D}^o - \boldsymbol{D}^d = \boldsymbol{D}$
$\boldsymbol{F} = \{f_{kl}\}_{K\times L}$	Binary failure link incidence matrix, $f_{kl} = 1$ iff link l fails in failure k
$\boldsymbol{U} = \{u_{rk}\}_{R\times K}$	Binary flow-failure incidence matrix, $u_{rk} = 1$ iff failure k will affect flow r's working path
$\boldsymbol{T} = \{t_{rl}\}_{R\times L}$	Binary flow tabu-link matrix, $t_{rl} = 1$ iff link l should not be used on flow r's backup path

A flow tabu-link matrix $\boldsymbol{T} = \{\boldsymbol{t}_r\}_{R\times 1} = \{u_{rl}\}_{R\times L}$ gives $t_{rl} = 1$ when the backup path of flow r should not use link l, and $t_{rl} = 0$ otherwise. Then we can find $\boldsymbol{U}$ and $\boldsymbol{T}$ in (1) and (2) respectively. In order to capture logical relations in above two equations, a binary matrix multiplication operation "$\odot$" which modifies general addition ($1 + 1 = 2$) to boolean addition ($1 + 1 = 1$) is introduced.

$$\boldsymbol{U} = \boldsymbol{A} \odot \boldsymbol{F}^T - \boldsymbol{D}^o - \boldsymbol{D}^d \tag{1}$$

$$\boldsymbol{T} = \boldsymbol{U} \odot \boldsymbol{F} \tag{2}$$

The *spare provision matrix* $\boldsymbol{G} = \{g_{lk}\}_{L\times K}$ is given in (5). Its element g_{lk} gives the minimum spare capacity required on link l when failure k happens. Moreover, the minimum spare capacities required on links are given by the column vector $\boldsymbol{s} = \{s_l\}_{L\times 1}$ in (4). The "max" operation on a matrix returns a column vector, with each entry being the maximum of the corresponding row. In this way, the minimum spare capacities on links are always enough to protect any failure.

Let $\phi_l(s_l)$ denote the link cost function of spare capacity on link l. $\boldsymbol{\phi}(\boldsymbol{s}) = \{\phi_l(s_l)\}_{L\times 1}$ is a column vector of link costs. The total link cost on network is $\boldsymbol{e}^T\boldsymbol{\phi}(\boldsymbol{s})$, where $\boldsymbol{e}$ is the unit column vector of length L.

Using the above notation in Table I, we formulate an *arc-flow* integer programming model for SCA in (3)-(8).

$$\min_{\boldsymbol{Q},\boldsymbol{s}} \quad \boldsymbol{e}^T\boldsymbol{\phi}(\boldsymbol{s}) \tag{3}$$

$$\text{s.t.} \quad \boldsymbol{s} = \max \boldsymbol{G} \tag{4}$$

$$\boldsymbol{G} = \boldsymbol{Q}^T\boldsymbol{M}\boldsymbol{U} \tag{5}$$

$$\boldsymbol{T} + \boldsymbol{Q} \leq 1 \tag{6}$$

$$\boldsymbol{Q}\boldsymbol{B}^T = \boldsymbol{D} \tag{7}$$

$$\boldsymbol{Q} : binary \tag{8}$$

The objective function in (3) is to minimize the total cost of spare capacity on networks. Constraint (4) and (5) give the method to calculate $\boldsymbol{s}$ from $\boldsymbol{Q}$. Note that constraint (5) can be replaced by (9) and (10).

Constraint (6) guarantees that backup paths will not use any link which might fail simultaneously with their working paths. For any single node failure, it assures backup paths are node-disjoint from their working paths.

Flow conservation constraint (7) guarantees that backup paths given in $\boldsymbol{Q}$ are feasible paths. It is given in a matrix representation and is also called the mass balance constraint in [17]. Only source and destination nodes have non-zero traffic accumulation while all the intermediate nodes have zero traffic accumulation. They are based on the properties of the path-link incidence matrix in [18]. The topology is given by a node-link incidence matrix $\boldsymbol{B} = (b_{nl})_{N\times L}$ where $b_{nl} = 1$ *or* -1 if and only if node n is the origin *or* destination of link l. $\boldsymbol{D} = (d_{rn})_{R\times N}$ is the flow node incidence matrix where $d_{rn} = 1$ or -1 iff $o(r) = n$ or $d(r) = n$. In directed network, both $\boldsymbol{B}$ and $\boldsymbol{D}$ are not binary matrices as those in undirected network.

Another way to compute $\boldsymbol{G}$ is through aggregating per-flow based information of working and backup paths. This is the key step on building our successive survivable routing algorithm in Section IV. First, the contribution of a single traffic flow r to $\boldsymbol{G}$ is given by $\boldsymbol{G}^r = \{g^r_{lk}\}_{L\times K}$ in (9), where $\boldsymbol{u}_r$ and $\boldsymbol{q}_r$ are the row vectors in $\boldsymbol{U}$ and $\boldsymbol{Q}$. The spare provision matrix $\boldsymbol{G}$, thus, is also given in (10).

$$\boldsymbol{G}^r = m_r(\boldsymbol{q}_r^T\boldsymbol{u}_r), \quad r = 1, \ldots, R \tag{9}$$

$$\boldsymbol{G} = \sum_{r=1}^{R} \boldsymbol{G}^r \tag{10}$$

From above equations, per-flow based information in $\boldsymbol{P}$, $\boldsymbol{Q}$ is replaced by $\boldsymbol{G}$ as the stored network state information. The space complexity is reduced from $O(RL)$ to $O(LK)$ and it is independent of the number of flows. This improves the scalability of the spare capacity sharing operation and makes it possible to be implemented distributively.

III. FIND A WORKING PATH WITH NODE-DISJOINT BACKUP

In above model, working paths are given before backup paths are decided. The reason is because working paths are used almost all the time and are much more important that backup paths. In order to provide node-disjoint backup paths, we have to guarantee that each working path has at least one node-disjoint backup path on the given 2-node-connected topology. This task is not trivial since working path found by general shortest path algorithms can not guarantee this property and may be *infeasible* for SCA node failure problem. Since the path costs of the backup and working paths are different, the problem to find both optimal paths is an NP-complete problem [19]. Hence, a graph algorithm to find a *feasible* working path is given here.

An example of infeasible working path is shown in Fig. 1. The working path from node 8 to node 11 is 8-13-1-23-18-19-4-11, where all the numbers between 8 and 11 are intermediate nodes on the path. This path has shortest hop but it does not have a node-disjoint backup path! Hence, before we start solving the spare capacity allocation problem, we have to make sure every working path has at least one node-disjoint backup path.

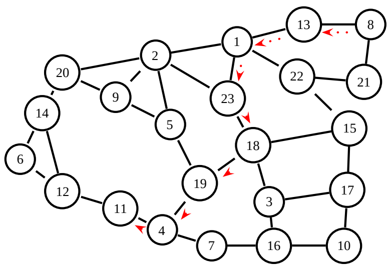

Fig. 1. Network 6 with 23 nodes and 33 links, a path is from node 8 to 11

A similar problem, called trap topology, has been discussed by Dunn, Grover and MacGregor [20]. In a trap topology, a working path may block all the possible link-disjoint backup paths although the network topology is two-link-connected. The problem to find a working path with link-disjoint backup path can be solved by using *augmenting path algorithm* from [17] as we discussed in [16]. Unfortunately, this algorithm can not be directly used to find working path with node-disjoint backup path.

The flow chart of our graph algorithm is given in Fig. 2. The algorithm finds a working path which has at least one node-disjoint backup path for flow r on network G which include N nodes and L links. Step 2 finds a working path $\boldsymbol{p}_r$ on G. Step 3 removes all links which are adjacent to any intermediate nodes of $\boldsymbol{p}_r$ to get a new network G_2. Step 4 tries to find a path on G_2. If such path is available, the original working path $\boldsymbol{p}_r$ has a node-disjoint backup path and the algorithm exits. Otherwise, the algorithms continues on step 5 where a residual network G_3 is generated by removing only directed links used by $\boldsymbol{p}_r$. If we skip Step 6 and Step 8, Step 7 finds a secondary path on G_3. Step 9 removes all the “trap links” from G and returns to Step 2. A *trap link* is defined as a link l which is on the working path $\boldsymbol{p}_r$ and its reversed direction link l' is on path $\boldsymbol{p}_*$ found

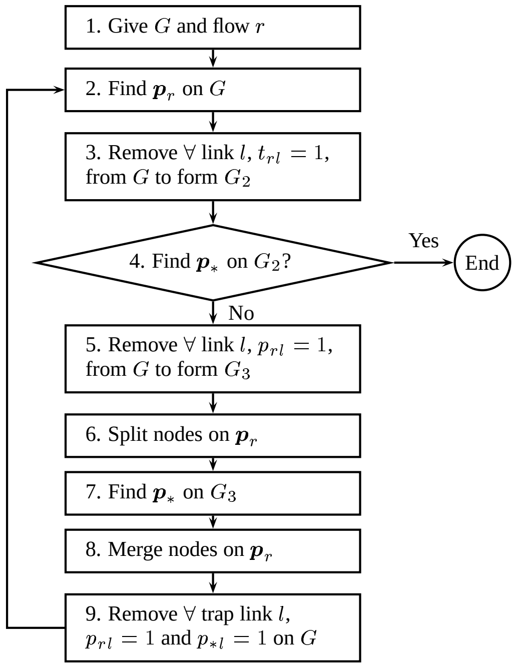
1. Give G and flow r
2. Find $\boldsymbol{p}_r$ on G
3. Remove $\forall$ link l, $t_{rl} = 1$, from G to form G_2
4. Find $\boldsymbol{p}_*$ on G_2?
Yes
End
No
5. Remove $\forall$ link l, $p_{rl} = 1$, from G to form G_3
6. Split nodes on $\boldsymbol{p}_r$
7. Find $\boldsymbol{p}_*$ on G_3
8. Merge nodes on $\boldsymbol{p}_r$
9. Remove $\forall$ trap link l, $p_{rl} = 1$ and $p_{*l} = 1$ on G

Fig. 2. Flow chart of the SSR algorithm at the source node of flow r

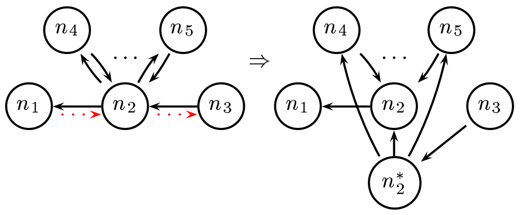

Fig. 3. Split node n_2 on path $\boldsymbol{a}_r$ which includes $n_1 - n_2 - n_3$

in Step 7. Without Step 6 and 8, this algorithm degrades to the algorithm we used to find working path with link-disjoint backup path based on the augmenting path algorithm [16].

Next, we concentrate on Step 6 and Step 8 to see how to split/merge intermediate nodes on the working path and why such operations can help us found working path with node-disjoint backups. Fig. 3 shows how to split an intermediate node n_2 on a working path $\boldsymbol{p}_r$. First, all links on $\boldsymbol{p}_r$ have been removed in G_3 as marked in dotted links on the left half of the figure. A mirror node n_2^* is added into G_3. It takes all outbound links from n_2 except the one in the reverse direction of $\boldsymbol{p}_r$. It also takes an inbound link in the reverse direction of $\boldsymbol{p}_r$ and adds a directed link to n_2. The new network G_3 is shown on the right half of Fig. 3. After all intermediate nodes on $\boldsymbol{p}_r$ have been divided, Step 7 routes a path represented by a path link vector $\boldsymbol{p}_*$ in the new network G_3. Step 8 merges all mirror nodes back to their original nodes with their links attachment restored. The resulted path vector $\boldsymbol{p}_*$ will also be modified following the contraction of G_3. After such node split/merge, the new path $\boldsymbol{p}_*$ either (a) passes some tabu links which is on the reversed direction of $\boldsymbol{p}_r$, or (b) does not crossing any intermediate nodes on $\boldsymbol{p}_r$. In case (a), the tabu links passed by $\boldsymbol{p}_*$ will be marked as trap links in Step 9 and be excluded from G in further working path routing. Case (b) should never happens since otherwise a path $\boldsymbol{p}_*$ should be found earlier in Step 4.

After Step 9, one or more trap links will be removed from G and the algorithm return to Step 2 to find a new working path. The iterations will be repeated until all trap links are removed and a working path with at least one backup path is found. The algorithm exits when Step 4 find a node-disjoint backup path.

IV. Successive survivable routing

The successive survivable routing (SSR) algorithm has been introduced in [16], [21]. Each traffic flow first routes its working path then routes its backup path on its source (or destination) node. Here, we briefly provide formulas and give the SSR flow chart at the source node of flow r in Fig. 4.

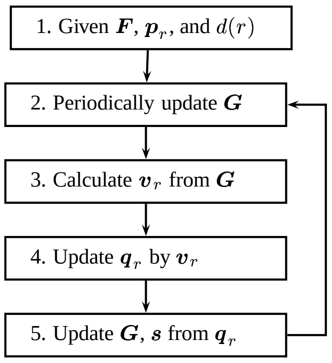
1. Given $\boldsymbol{F}$, $\boldsymbol{p}_r$, and $d(r)$
2. Periodically update $\boldsymbol{G}$
3. Calculate $\boldsymbol{v}_r$ from $\boldsymbol{G}$
4. Update $\boldsymbol{q}_r$ by $\boldsymbol{v}_r$
5. Update $\boldsymbol{G}$, $\boldsymbol{s}$ from $\boldsymbol{q}_r$

Fig. 4. Flow chart of the SSR algorithm at the source node of flow r

Step 1 initiates SSR on flow r with its working path $\boldsymbol{p}_r$, destination node $d(r)$ and the failure matrix $\boldsymbol{F}$. Then $\boldsymbol{u}_r$ and $\boldsymbol{t}_r$, which are part of $\boldsymbol{U}$ and $\boldsymbol{T}$ in (1) and (2), are calculated.

Step 2 periodically collects current network state information and $\boldsymbol{G}$. The per-flow based information is not required for backup path routing and reservation. This makes SSR scalable and suitable for a distributive implementation.

In Step 3, a shortest path algorithm is used to find a backup path. The vector of link metrics $\boldsymbol{v}_r$ are found based on the following notation. Given $\boldsymbol{G}$, $\boldsymbol{q}_r$ and $\boldsymbol{G}^r$ for current flow r, let $\boldsymbol{G}^- = \boldsymbol{G} - \boldsymbol{G}^r$ and $\boldsymbol{s}^- = \max(\boldsymbol{G}^-)$ be the spare provision matrix and the link spare capacity vector after $\boldsymbol{q}_r$ is removed. Let $\boldsymbol{q}_r^+$ denote an alternative backup path for flow r, and $\boldsymbol{G}^{r+}(\boldsymbol{q}_r^+) = m_r \boldsymbol{q}_r^{+T} \boldsymbol{u}_r$. Then, this new path $\boldsymbol{q}_r^+$ produces a new spare capacity reservation vector $\boldsymbol{s}^+(\boldsymbol{q}_r^+) = \max(\boldsymbol{G}^- + \boldsymbol{G}^{r+}(\boldsymbol{q}_r^+))$. Let $\boldsymbol{q}_r^+ = \boldsymbol{e} - \boldsymbol{t}_r$, which means the backup path uses all possible links, then we can find a vector of *link metrics* as

$$\begin{aligned} \boldsymbol{v}_r &= \{v_{rl}\}_{L\times 1} \\ &= \phi(\boldsymbol{s}^+(\boldsymbol{e}-\boldsymbol{t}_r)) - \phi(\boldsymbol{s}^-), 1 \le r \le R, \end{aligned} \quad (11)$$

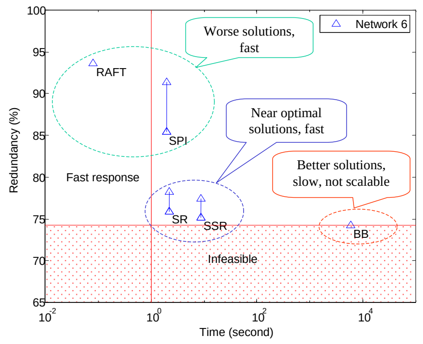

Fig. 5. Comparison of redundancy versus CPU time of different SCA algorithms for node failures

where $\boldsymbol{t}_r$ is the binary flow tabu-link vector of flow r. In the node failure case, its elemetns for the links adjacent to the intermediate nodes of the working path are marked by one.

Step 4 improves the backup path by using a shortest path algorithm with link metrics $\boldsymbol{v}_r$. In Step 5, if the backup path is changed in Step 4, the spare capacity reservations along the path will be updated accordingly. After this step, the algorithm returns to Step 2 to start next backup path update.

The algorithm terminates when there is no backup path update and it reaches a local optimum.

V. Numerical results

Eight networks in [16] are summarized in the beginning of Table II and used for numerical experiments. Symmetrical traffic flows are loaded between all node pairs. All flows have one unit bandwidth demand. This traffic demand pattern is given for the ease of comparison.

Several algorithms are compared on different networks. For network 6 shown in Fig. 1, we give the comparison of network redundancies versus CPU times of these algorithms in Fig. 5. It shows that SSR is near optimal comparing to Branch and Bound (BB). The ranges of redundancies SSR found is within 4% from the optimal solution found by BB. Moreover, SSR is very fast comparing to other algorithms like SR, SPI and RAFT as introduced in [16]. Since the CPU times for SSR, SR and SPI are the summation of 64 independent running cases, the time for a single case is lower than a second. Hence, these three algorithms also belong to fast algorithms as RAFT.

All numerical results on eight networks are summarized in Table II and their network redundancies are drawn on Fig. 6. The total spare capacities found by these algorithms can be consluded as: Optimal $=$ BB $<$ SSR $<$ SR $\ll$ SPI $\approx$ RAFT $\ll$ NS. SSR is a fast algorithm and achieves near optimal solutions.

VI. Summary

This paper extends our recent matrix representation and successive survivable routing (SSR) algorithm for node failures on mesh networks. First, node failures are modeled by the matrix-

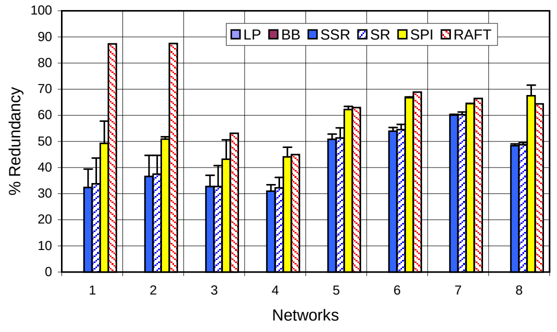

Fig. 6. Comparison of redundancy $\eta = S/W$ in SCA considering node failures.

based arc-flow SCA model. Second, a graph algorithm to find a shortest working path with existence of node-disjoint backup paths are provided. The objective here is to find shortest working paths which have node-disjoint backup paths. This problem is a NP-complete problem due to different path costs [19]. Third, based on the working paths prepared by the graph algorithm for node failures, we use the SSR algorithm to find near optimal SCA solutions. Comparing with several other algorithms for SCA problems considering node failures, SSR has the best trade-off between computation speed and solution optimality for node failures.

TABLE II

NUMERICAL RESULTS

Network	1	2	3	4	5	6	7	8
N	10	12	13	17	18	23	26	50
L	22	25	23	31	27	33	30	82
R	90	132	156	272	306	506	650	2450
W	142	224	324	640	826	1686	2732	11104
Total spare capacity S								
[1]BB	38	99	113	252	539	1252	1812	-
[2]SSRmin	42	108	124	272	552	1268	1812	5720
SSRmax	50	120	144	294	574	1306	1826	5800
SRmin	46	110	126	280	556	1280	1832	5764
SRmax	58	130	148	308	586	1320	1872	5894
SPImin	66	130	170	362	654	1440	1958	7394
SPImax	84	154	202	416	710	1540	2008	7780
RAFT	82	142	194	408	688	1578	2010	7690
NS	198	326	456	910	1324	2736	5652	16278
Total CPU time (in second)								
BB	60	130	720	1700	130	5900	41	-
SSRsum	3.25	3.63	3.84	6.51	5.94	8.6	14.73	293.43
SRsum	0.59	0.63	0.71	1.08	1.17	2.25	2.81	38.18
SPIsum	0.58	0.64	0.66	0.98	1.03	1.96	2.36	28.96
RAFT	0.02	0.02	0.02	0.04	0.04	0.08	0.11	0.92
NS	0.02	0.02	0.02	0.04	0.04	0.08	0.1	0.89

The experiments are run for the eight networks. The flows are full meshed in the network with one unit traffic load. Working paths are given with their total capacity reservation "W" above. The following rows provide the total spare capacity found from different algorithms. [1] Branch and bound (BB) algorithm use CPLEX [22] on a SUN Ultra Enterprise server with 4GB memory and 250MHz UltraSparc CPU. SSR, SR, SPI, RAFT and NS are coded in C++ on a Pentium III 533MHz PC. [2] For SSR, SR and SPI, 64 random number seeds are used for generating flow sequences to update backup paths. For these 64 results, their maximum and minimum total spare capacities and the sum of their CPU times are given.

REFERENCES

[1] Y. Liu, *Spare capacity allocation method, analysis and algorithms*, Ph.D. dissertation, School of Information Sciences, University of Pittsburgh, 2001.

[2] Tsong-Ho Wu, *Fiber Network Service Survivability*, Artech House, 1992.

[3] M. Herzberg, S. J. Bye, and U. Utano, "The hop-limit approach for spare-capacity assignment in survivable networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp. 775–784, Dec. 1995.

[4] T.-H. Wu, "Emerging technologies for fiber network survivability," *IEEE Communications Magazine*, vol. 33, no. 2, pp. 58–74, Feb. 1995.

[5] W.D. Grover, "Distributed restoration of the transport network," in *Telecommunications Network Management into the 21st Century, Techniques, Standards, Technologies and Applications*, Salah Aidarous and Thomas Plevyak, Eds., chapter 11, pp. 337–417. IEEE Press, 1994.

[6] Byung Han Ryu, Masayuki Murata, and Hideo Miyahara, "Design method for highly reliable virtual path based ATM networks," *IEICE Transactions on Communications*, vol. E79-B, no. 10, pp. 1500–1514, 10 1996.

[7] R. Iraschko, M. MacGregor, and W. Grover, "Optimal capacity placement for path restoration in STM or ATM mesh survivable networks," *IEEE/ACM Transactions on Networking*, vol. 6, no. 3, pp. 325–336, June 1998.

[8] W. D. Grover, R. R.Iraschko, and Y. Zheng, "Comparative methods and issues in design of mesh-restorable STM and ATM networks," in *Telecommunication Network Planning*, P. Soriano B.Sanso, Ed., pp. 169–200. Kluwer Academic Publishers, 1999.

[9] Y. Xiong and L. G. Mason, "Restoration strategies and spare capacity requirements in self-healing ATM networks," *IEEE/ACM Transactions on Networking*, vol. 7, no. 1, pp. 98–110, Feb. 1999.

[10] B. Van Caenegem, W. Van Parys, F. De Turck, and P. M. Demeester, "Dimensioning of survivable WDM networks," *IEEE Journal on Selected Areas of Communications*, vol. 16, no. 7, pp. 1146–1157, Sept. 1998.

[11] S. Ramamurthy and B. Mukherjee, "Survivable WDM mesh networks, part I - protection," in *Proceeding of IEEE INFOCOM*, New York, USA, March 21-25 1999, IEEE, IEEE Press.

[12] X. Su and C.-F. Su, "An online distributed protection algorithm in WDM networks," in *Proceeding of IEEE International Conference on Communications*, 2001, pp. 1571–1575.

[13] C. Dovrolis and P. Ramanathan, "Resource aggregation for fault tolerance in integrated service networks," *ACM Computer Communication Review*, vol. 28, no. 2, pp. 39–53, 1998.

[14] M. Kodialam and T.V. Lakshman, "Dynamic routing of bandwidth guaranteed tunnels with restoration," in *Proceeding of IEEE INFOCOM*, Mar. 2000.

[15] T. H. Oh, T. M. Chen, and J. L. Kennington, "Fault restoration and spare capacity allocation with QoS constraints for MPLS networks," in *Proceeding of IEEE Global Communications Conference*, Nov. 2000, vol. III, pp. 1731–1735.

[16] Y. Liu, D. Tipper, and P. Siripongwutikorn, "Approximating optimal spare capacity allocation by successive survivable routing," in *Proceeding of IEEE INFOCOM*, Anchorage, AL, April 24–28 2001, pp. 699–708.

[17] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, New York, 1993.

[18] L. R. Foulds, *Graph Theory Applications*, Universitext. Springer-Verlag, 1992.

[19] C. Li, S. T. McCormick, and D. Simchi-Levi, "Finding disjoint paths with different path costs: Complexity and algorithms," *Networks*, vol. 22, pp. 653–667, 1992.

[20] D. A. Dunn, W. D. Grover, and M. H. MacGregor, "Comparison of k-shortest paths and maximum flow routing for network facility restoration," *IEEE Journal on Selected Areas of Communications*, vol. 2, no. 1, pp. 88–99, 1 1994.

[21] Y. Liu and D. Tipper, "Spare capacity allocation for non-linear cost and failure-dependent path restoration," to appear Third International Workshop on Design of Reliable Communication Networks (DRCN), Budapest, Hungary, October 7–10 2001.

[22] ILOG, *Using the CPLEX Callable Library*, ILOG, Inc., 1998.