Dynamic Path Restoration Algorithms For On-Demand Survivable Network Connections

William Lau and Sanjay Jha

School of Computer Science and Engineering,

The University of NSW, Kensington Sydney 2052 Australia

wlau,sjha@cse.unsw.edu.au (Telephone: 0293854354, 0293856471)

UNSW-CSE-TR-0409

February 2004

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

Abstract

Restoration strategies that use offline pre-planning assume that the traffic matrix is static and is known prior to the network capacity assignment. These strategies require recalculation of the capacity requirement for any changes made to the traffic matrix, thus are not suitable for use in highly dynamic traffic environments.

This paper addresses the dynamic online capacity assignment problem where each source-destination restoration request is computed sequentially with no prior knowledge of requests that have not been calculated. We focus on the state-dependent approach that allows different backup paths to be used when the network suffers different failure scenarios.

The problem is first formulated into a new Integer Programming (IP) problem, and new heuristics are proposed that make trade-offs between bandwidth efficiency and computation time. Results show that the proposed polynomial-time algorithm performs competitively with the IP solution in terms of bandwidth efficiency, and performs better than existing heuristic algorithms of the same restoration strategy. The computation time for the polynomial-time algorithm is significantly shorter than the IP solution and thus suitable for large scale on-demand applications.

Further, a comprehensive performance comparison is made between online algorithms of state-dependent and state-independent strategies. Results from this analysis can be used as a guide for choosing between the two strategies. State-dependent algorithms are shown to provide better trade-offs between network efficiency and computation-time, as compared with state-independent algorithms.

I. Introduction

Traditional high quality service provisioning has normally been directed towards leased line or frame relay-based services that are normally very static and have long lifetime durations. These characteristics allow the network providers to take planning offline and calculate resource allocations that best utilize the network for the complete set of service demands. However, the trend is now moving towards integration of services into one network infrastructure, hence putting new types of requirements on networks. Next generation service applications, such as Extranet On Demand, Video On Demand, and Video Conferencing, are examples of services that are dynamic and have relatively short lifetimes. The question is: how can we support these services efficiently? This paper focuses on restoration strategies and algorithms that dynamically establish on-demand Quality of Service (QoS) connections with bandwidth guaranteed resilience in mesh-based networks.

Connection-oriented technologies such as MPLS [1], [2], [3] and GMPLS [4], [5], [6] are fast becoming the technologies of choice to provision such services. Traffic engineering [7] can improve network utilization and allow more sophisticated restoration strategies to be deployed. The most basic type of strategy in MPLS to ensure high availability is 1+1 path protection [8] where a backup path is pre-established along with the service path. The backup path is necessarily disjoint from the service path and must have the same bandwidth capacity if full traffic restoration is desired. The path can be link or node disjoint depending on the decision of the network provider.

In 1+1 path protection, both the service path and the backup path are assumed to be used simultaneously for traffic. A common technique is to send redundant traffic on the backup path to reduce the time required for restoring normal traffic operation [9]. An alternative is to use 1:1 protection where the backup path is not used unless the service path is known to have failed [8]. Upon detecting failure of the service path, the traffic is switched to the backup path and then switched back to the service path once recovery is completed.

These two restoration strategies are commonly used to protect from single link or node failure. However, the backup path bandwidth is not used in the 1:1 strategy, and many proposals [10], [11], [12], [13], [14], [15], [16] have aimed to exploit this property to maximize network utilization. Both 1+1 and 1:1 strategies are state-independent approaches, where one backup path is used to cover all possible single link (or node) failures affecting the service path. State-dependent approaches [17], [18], [19] allow multiple backup paths per service path and the backup path used depends on the state of the network after the failure. That is, the failure scenario dictates which backup path to use.

It is only recently that focus has moved toward dynamic online restoration strategies [12], [15], [14], [19], where connection requests are computed sequentially. Each request is optimized based on only the request and the current network resource state. The path computation does not affect previously calculated requests and it is not dependent on future requests. Online computation better fits the behavior of on-demand network services, especially where there is high connection turnover.

This paper addresses the dynamic online state-dependent capacity assignment problem. We concentrate mainly on online restoration strategies for single link failure and single node failure.

The contributions in this paper are grouped into the following four parts.

A. *New Online State-Dependent Algorithms*

We present a new Integer Programming (SD-IP) formulation for the online state-dependent capacity assignment problem that aims to minimize the spare capacity requirement. The term “spare capacity” refers to the total bandwidth allocated for backup paths in the network over a set of requests. We also propose a new approximation algorithm that runs in polynomial-time called Shortest Path Local Optimization (SPLO). Results show that SPLO performs better than the existing online heuristic algorithm [19], and performs competitively with the SD-IP solution, especially under single node failure.

B. *Comparing Online and Offline State-Dependent Algorithms*

A new offline heuristic algorithm is formed by combining SPLO with a non-polynomial-time post-processing component. Results show that the offline-SPLO algorithm reaches a better local optimum than the existing offline heuristic algorithm [18]. For analyzing the performance differences when using online algorithms, we compare the SD-IP and SPLO with the offline heuristic algorithms. Results indicate that the online SD-IP reaches a better local optimum than the offline heuristic algorithms. SPLO on the other hand, performs within 13% of the offline heuristic algorithms for link failure and 8% for node failure.

The practical running time for the SD-IP solution was found to be much shorter than that for the offline heuristic algorithms, but still significantly longer than the polynomial-time SPLO algorithm. Thus the online algorithms proposed in this paper are attractive alternatives to offline heuristics, which are inflexible and have high run-time cost.

C. *A New Service Path Algorithm*

Both SD-IP and SPLO calculate the backup paths only and use a shortest path algorithm (Dijkstra [20]) to calculate the service path first. This separate computation approach has been used in other existing state-dependent algorithms [19], [18]. We propose a new approach for calculating service paths for state-dependent strategies called Backup Paths Cost Estimation (BPCE). BPCE estimates the cost for protecting a link if this link is used in the service path. This estimated cost is added to the normal link cost and a shortest path algorithm is used to find the service path that has the least total cost. A backup algorithm like SPLO is then used to calculate the backup paths.

The aim of this approach is to influence the decision when calculating the service path such that the total bandwidth requirement could be reduced. BPCE is a polynomial-time algorithm for calculating the service path, and thus BPCE with SPLO is still a polynomial-time solution. Results show that using BPCE improves performance when combined with SPLO. The total bandwidth requirement is reduced slightly and the number of blocked requests is reduced significantly. However, BPCE only has minor impact when used with the SD-IP solution.

D. *Comparing State-Dependent and State-Independent Online Algorithms*

A detailed comparison is made between online algorithms that are based on the state-dependent strategies , and algorithms that are based on state-independent strategies. The state-independent Integer Programming solution [12] (CIS) is based on a joint computation approach where the service path and backup path are optimized together.

The total bandwidth required by SD-IP-BPCE and CIS are very close to each other for both single link failure and single node failure. CIS has slightly lower number of blocked requests in limited capacity scenarios. However, in terms of computation time, CIS requires orders of magnitude longer running time than SD-IP-BPCE. This is because of the high level of combinatorial constraints used for joint computation.

SPLO is also compared with an existing state-independent polynomial-time algorithm [15] (FIR) and results show that the two approaches have very similar performance. When combining BPCE with SPLO, the performance results are better than the state-independent FIR algorithm in both total bandwidth requirements and the number of blocked requests.

E. Organization of paper

The rest of this paper is organized as follows. Section II briefly describes related work in the field. We then formally define the online state-dependent capacity assignment problem, and formulate a new Integer Programming algorithm to solve it in Section III. A new polynomial-time algorithm that approximates the optimal solution is presented in Section IV, while the new offline heuristic algorithm is proposed in Section V. Section VI introduces the new BPCE approach for service path calculation and Section **??** explains the experimental setup and metrics used for the simulations. The simulation results are examined in four parts. The first part is presented in Section VII, which compares the SD-IP solution and SPLO, with an existing state-dependent online algorithm. The second part is in Section VIII, which compares the online algorithms with the offline heuristic algorithms. The third part in Section IX, investigates the advantages of using the BPCE service path calculation approach with SD-IP and SPLO. The final part in Section X compares and analyzes the state-dependent algorithms with the existing state-independent algorithms. A simple run-time benchmark in Section XI shows the computation time difference between the algorithms analyzed in this paper. We conclude this paper in Section XII.

II. Related Work

Kodialam et al. [12] proposed an online restoration strategy based on the state-independent approach. They proposed a new Integer Programming solution for the state-independent online problem based on joint-path computation. They refer this solution as: *Complete Information Scenario* (CIS). A heuristic algorithm was also proposed that finds a suboptimal solution based on partial information. The objective was to devise an algorithm that uses only aggregate information rather than per-path information, to reduce the amount of information that must be exchanged in a distributed computation system. This heuristic algorithm iteratively searches for a lower upper bound and a higher lower bound. The algorithm stops when neither bound can be further improved. This iterative process does not have a polynomial-time complexity bound.

We identified two flaws in their IP formulation and propose the correction to the formulation (Appendix). The corrected version of CIS is used for the simulations in this paper.

Li et al. [15] proposed a polynomial time algorithm for Kodialam's online model based on separate path computation. They called this algorithm: *Full Information Restoration* (FIR). Results in this paper show that FIR performs on a par with SPLO. However, when combining the new BPCE service path computation approach with SPLO, the results outperform Li's algorithm in terms of total bandwidth requirement. In limited capacity networks, BPCE with SPLO has significantly less blocked requests than FIR.

Liu et al. [14], [19] proposed a progressive heuristic algorithm for both the state-dependent and state-independent strategies. The idea was to compute the backup paths that protect the service paths using the standard shortest-hop path algorithm and then incrementally recompute all the backup paths at periodic time intervals for better solutions. They call this algorithm: *Successive Survivable Routing* (SSR). Although the time complexity for each iteration is polynomial, the algorithm as a whole is non-polynomial and may take an exponential amount of time to converge to a local optimum. The algorithm is designed to run in a distributed manner and its iterative nature pushes the network close to the local optimal state over time. However, the distributed algorithm may not converge to a stable state because some decisions can cause oscillations in path updates. Proof of convergence and stability was not provided in the paper.

We compare their SSR algorithm with SPLO in Section VII; the results show that SPLO out-performs Liu's algorithm by making better path selection decisions on the first (and only) computation of the request. This shows that the initial path selection decisions affect the quality of the local optima that the algorithms reach.

Xiong et al. [18] proposed an IP formulation for a state-dependent restoration strategy. They assumed separate path computation and computed all the requests offline together. An offline heuristic algorithm was proposed that has no polynomial-time bound. In this paper we refer this algorithm as *Xiong's Failure-Oriented Re-Configuration* (X-FORC) algorithm. X-FORC uses a post-processing component that we have combined with SPLO to form a new offline heuristic algorithm (nSPLO). The results in Section VIII show that nSPLO performs better than X-FORC. We also compared the offline heuristic algorithms with the online algorithms (SD-IP and SPLO). Results show that the SD-IP solution performs better than both offline heuristic algorithms. In our experiments, the running time for SD-IP is substantially faster than both heuristic algorithms. SPLO performs competitively with these heuristic algorithms and SD-IP, while having substantially shorter computational time.

Many other studies [11], [10], [21], [17] have been based on offline state-dependent strategies but differ in the linear/integer programming optimization formulation and the assumptions made. None of these studies provide polynomial-time or heuristic algorithms.

The IP formulation (SD-IP) proposed in this paper is based on a state-dependent strategy and assumes that the service path is computed separately using a shortest-path algorithm. Results show that the SD-IP solution's total bandwidth requirement is very similar to the CIS solution, and has substantially lower computation run-time. However, in limited network capacity scenarios, the CIS solution has slightly less blocked requests. By combining BPCE with the SD-IP solution, we are able to reduce the number of blocked requests. A new heuristic algorithm called Shortest Path Local Optimization (SPLO) is proposed in this paper and it has a polynomial-time bound.

III. Online State-Dependent Integer Programming Formulation

TABLE I

Mathematical Notations

Symbol	Definition
V	The set of vertexes.
E	The set of edges.
$G(V,E)$	The network graph
(i,j)	The directional link that connects node i to node j.
A_{ij}	Bandwidth available for allocation in link (i,j).
C_{ij}	The backup bandwidth in link (i,j).
FS	The set of failure scenarios.
s	A failure scenario in FS.
C_{ij}^s	Backup bandwidth required in (i,j) for s.
h	The source node of the request.
g	The destination node of the request.
b	The requested bandwidth
p	The service path in context.
FS_p	The set of failure scenarios that affects service path p.
τ_{ij}^s	The additional cost for backup path using (i,j) in s.
E_s	The set of links that are affected by failure scenario s.

A network graph is represented by $G(V,E)$. It consist of a set of nodes V and these nodes are connected by a set of links E. Each link (i,j), from node i to node j, maintains two types of information: the available un-allocated bandwidth (A_{ij}), and the total bandwidth allocated to backup paths (spare capacity, C_{ij}). A state-dependent strategy is used for our approach, so a set of all possible failure scenarios in the network is defined and is denoted by FS. We assume only single link failure (or single node failure), so each failure scenario only consists of one link failure (or one node failure that may pull down multiple outgoing links).

We also have to keep track of the total backup bandwidth required for each link by backup paths protecting against the failure scenarios (C_{ij}^s). The assumption of single link or node failure only means that no two failure scenarios can occur simultaneously, and the failure scenarios are independent of each other. If two backup paths protect their service path(s) from different failure scenarios then the backup paths are considered independent. Using this knowledge, the total backup bandwidth allocated in each failure scenario can be shared across failure scenarios. Therefore, we derive the following relationship where C_{ij} is the spare capacity allocated on link (i,j).

$$C_{ij} = \max C_{ij}^s, \forall s \in FS$$

Suppose a path with protection between source h and destination g is required and the bandwidth demand is b units. The objective for the path protection system is to produce a service path, and a set of backup paths that cover the service path from all defined failure scenarios. If we further assume that the service path is computed separately from the backup paths using a shortest-path algorithm (Dijkstra), then the reduced objective is to find the set of backup paths that protect the given service path and require minimum *additional* bandwidth allocation. Additional bandwidth allocation refers to the case where the current failure scenario bandwidth requirement plus the new request demand exceeds the spare capacity allocated on the link so that additional bandwidth must be allocated to the spare capacity of the link.

With the problem formally defined, a new Integer Programming (IP) formulation is devised to find the optimal solution for a given request based on the given service path p and the current network resource state. The IP formulation is as follows.

$$\min\left(\sum_{(i,j)\in E} z_{ij}\right) \tag{1}$$

$$\sum_j x_{ij}^s - \sum_j x_{ji}^s = 0, \text{ for } i \neq h, g\ , \forall s \in FS_p \tag{2}$$

$$\sum_j x_{hj}^s - \sum_j x_{jh}^s = 1, \forall s \in FS_p \tag{3}$$

$$\sum_j x_{gj}^s - \sum_j x_{jg}^s = -1, \forall s \in FS_p \tag{4}$$

$$z_{ij} \geq \tau_{ij}^s(x_{ij}^s), \forall s \in FS_p\ , \forall (i,j) \in E \tag{5}$$

$$x_{ij} \in \{0,1\} \tag{6}$$

$$z_{ij} \geq 0 \tag{7}$$

The objective function (Equation 1) minimizes the sum of the additional spare capacity required for each link (z_{ij}) in the network. Given the service path p, we find the set of failure scenarios that affect p and this set is denoted by FS_p. For each affected failure scenario ($s \in FS_p$), a backup path must be established from source h to destination g. The binary decision variable x_{ij}^s indicates whether link (i,j) is used for the backup path that protects from failure scenario s. Equations 2, 3, 4 are constraints for path construction. Equation 5 defines the additional cost for each link as the maximum of the additional bandwidth required for each of the backup paths on this link. The additional cost for using link (i,j) for failure scenario s is represented by τ_{ij}^s, which is defined below. The binary constraints for the decision variables are given in Equation 6. The additional cost for each link must be positive, which is represented by the constraints in Equation 7.

The additional cost for using link (i, j) in failure scenario s is given by τ_{ij}^s, which is defined as follows.

$$\tau_{ij}^s = \begin{cases} 0, & \text{if } (i,j) \notin E_s \text{ and } C_{ij}^s + b \leq C_{ij} \\ b - C_{ij} + C_{ij}^s, & \text{if } C_{ij}^s + b > C_{ij} \\ & \quad \text{and } A_{ij} \geq b - C_{ij} + C_{ij}^s \\ & \quad \text{and } (i,j) \notin E_s \\ \infty, & \text{otherwise.} \end{cases}$$

The first case occurs when a link can be used without additional cost because there is enough spare capacity already allocated on the link to accommodate the backup load in the failure scenario. The second case occurs when there is not enough spare capacity on the link to accommodate the current request fully, but there might be some residual capacity available that can be used. Thus the cost for using this link is the difference between the current request bandwidth and the residual spare capacity. The final case occurs if the link being considered is down (i.e., $(i, j) \in E_s$) or there is not enough available bandwidth on this link to satisfy the request. The Big M method [22] is used to represent infinity and discourages infeasible links from being used.

The IP formulation is not yet complete, as two factors may possibly cause the optimal solution to consist of backup paths that contain unnecessary loops (an example is illustrated in the Appendix):

1) Allowing sharing of spare capacity results in potential zero-cost partial segments. Zero-cost segments allow loops to form without increasing the objective function.
2) Backup paths calculated by this IP can also share additional spare capacity with each other, thus effectively allowing some links to be used without any cost. This allows loops to form without increasing the objective function.

The backup path is still correct but the loops cause inefficiency by consuming backup bandwidth that can otherwise be used by other backup paths.

We also encountered this looping issue in the state-independent IP formulation by Kodialam et al. [12]. One way to deal with this looping issue is to use the IP formulation to find the backup path(s) and then use a loop elimination post-process to remove looping components. A more complete approach, which is taken in this paper, is to formulate the problem with loop prevention constraints. This creates loop-free feasible (or optimal) solutions but with the trade-off of $O|FS_p||V|$ additional constraints in the IP formulation. The loop prevention constraints are as follows.

$$\sum_j x_{ji}^s \leq 1, \forall i \in V - \{h\} \ , \forall s \in FS_p \tag{8}$$

$$\sum_j x_{jh}^s = 0, \forall s \in FS_p \tag{9}$$

Equation 8 prevents more than one incoming link of each node from being used. A special case is the source node where no incoming links can be used (Equation 9). Together with the previous path construction constraints, the possibility of loop formation is eliminated.

The number of constraints in the complete IP formulation is in the order of $O(|FS_p|(|E| + |V|))$. The worst case is when $|FS_p| = |E|$, but in a mesh-based network this is very unlikely and the average number of hops for the service path is significantly smaller than the number of links in the network.

IV. Polynomial-time Heuristic Algorithm

Integer Programming solutions do not have polynomial-time bounds, and run in exponential time in the worst case. Thus in this paper, we propose a new heuristic algorithm called Shortest Path Local Optimization (SPLO) that runs in polynomial time. SPLO is a greedy algorithm that tries to find optimal backup paths on a sequential basis. This results in suboptimal solutions because each backup path's calculation is only based on the previous decision and the current network resource state. The IP solution computes all backup paths together at the same time thus potentially finding a better solution.

SPLO is described in Algorithm 1. For easier notation, we use l to represent a link. The inputs to the algorithm are the network graph (including the resource state information), the defined failure scenarios (including the spare capacity requirements C_l^s), and the service path request (including the bandwidth demand b). The output of the algorithm is the set of backup paths, the new network resource state, and the new failure scenario spare capacity requirements.

```
Data    : Network Graph: $G(V,E)$; Failure scenario set: $FS$; Service Path: $p$
Result  : New Network Graph: $G(V,E)$; Backup Path set: $BP$; Failure scenario set: $FS$
Initialize $BP = \emptyset$;
Find the set of failures $FS_p$ affecting the Service Path;
for each $s \in FS_p$ do
    Find the set of failed links $E_s$ in the current failure scenario;
    Remove all failed links $l \in E_s$ from $G(V,E)$;
    Recompute the network links cost $\tau_l^s$;
    Find the shortest path using the new network graph;
    Update $G(V,E)$ and $C_l^s$ to use this shortest path as the backup path;
    Add the backup path to $BP$;
    Insert failed links back into $G(V,E)$;
end
```

Algorithm 1: Shortest path local optimization

The logic behind Algorithm 1 is to find all the failure scenarios that will affect the service path. The next step is to compute a backup path for each affected failure scenario sequentially. To calculate the backup path, any failed links are removed from the network graph, and the cost for using the network links is then recomputed. The cost for each link is presented by τ_l^s:

$$\tau_l^s = \begin{cases} 0, & \text{if } C_l^s + b \leq C_l \text{ and } l \notin E_s \\ b - C_l + C_l^s, & \text{if } C_l^s + b > C_l \\ & \text{ and } A_l \geq b - C_l + C_l^s \\ & \text{ and } l \notin E_s \\ \infty, & \text{otherwise.} \end{cases}$$

This is essentially the same as the link cost (τ_{ij}^s) described in Section III.

Once we have the new network cost graph, a standard shortest path algorithm (Dijkstra) is used to find the best path for backup. The algorithm exits if no feasible solution is found. Otherwise, the shortest path is used as the backup path for this failure scenario and the network resource state is updated according to the equations below. The edges used by the currently computed backup path are denoted by $|E_{bp}|$.

$$C_l^s = C_l^s + b, \forall l \in E_{bp} \tag{10}$$

$$C_l = C_l + \tau_l^s, \forall l \in E_{bp} \tag{11}$$

The failed links are inserted back into the network graph and the process continues until a backup path is found for each failure scenario that affects the given service path. The time complexity for SPLO is $O(|FS|(|E|+|V|\log|V|))$ per request.

V. A New Offline Heuristic Algorithm

```
Data      : Network Graph: $G(V,E)$; Failure scenario set: $FS$; Service Path set: $P$
; Result  : New Network Graph: $G(V,E)$; Backup Path set: $BP$
; Initialize $BP = \emptyset$;
for each $p \in P$ do
    Find the set of failures $FS_p$ affecting the Service Path;
    for each $s \in FS_p$ do
        Find the set of failed links $E_s$ in the current failure scenario;
        Remove all failed links $l \in E_s$ from $G(V,E)$;
        Recompute the network links cost $\tau_l^s$;
        Find the shortest path using the new network graph;
        Update $G(V,E)$ and $C_l^s$ to use this shortest path as the backup path;
        Add the backup path to $BP$;
        Insert failed links back into $G(V,E)$;
    end
end
Initialize $link\_improve = true$ and $path\_improve = true$;
while $link\_improve$ do
    $link\_improve = false$;
    for each $l \in E$ do
        Remove $l$ from $G(V,E)$;
        while $path\_improve$ do
            $path\_improve = false$;
            Find $s \in FS$ that causes the worst-case state;
            Find the set of backup paths used in $s$: $BP_s$;
            Remove all failed links ($l \in E_s$) for $s$ from $G(V,E)$;
            for each $bp \in BP_s$ that uses link $l$ do
                Remove $bp$ from failure scenario (reduce $C_l^s$);
                Recompute the network links cost $\tau_l^s$;
                Find the shortest path $bp_{new}$ using the new graph;
                if total spare capacity decreases then
                    Update $G(V,E)$ and $C_l^s$ to use $bp_{new}$ as the backup path;
                    $BP = BP - \{bp\} + \{bp_{new}\}$;
                    $link\_improve = true$;
                    $path\_improve = true$;
                else
                    Put $bp$ back into the failure scenario (increase $C_l^s$);
                end
            end
            Insert all failed links ($l \in E_s$) back into $G(V,E)$;
        end
        Insert $l$ back into $G(V,E)$;
    end
end
```

Algorithm 2: Offline nSPLO algorithm

The offline heuristic algorithm (X-FORC) proposed by Xiong et al. [18] loops through all the links and for each link finds the set of service paths that will fail if this link goes down. For each affected service path, a backup path is calculated to protect against the link failure. After going through all the links, a post-processing component recomputes the backup paths iteratively for improvements. This iterative component does not have a polynomial-time boundary.

We combine this post-processing component with SPLO to form a new offline heuristic algorithm (nSPLO), which is described in Algorithm 2. The main difference between nSPLO and X-FORC is the way the backup paths are calculated initially. Results from our simulations (in Section VIII) show that nSPLO reaches a better local optima than X-FORC because of the better approach in the initial backup path calculation.

Algorithm 2 initially uses SPLO to find all the backup paths for a set of requests. The backup paths are then supplied to the post-processing iterative loop that recomputes the backup paths in an attempt to reduce the spare capacity requirement.

VI. Backup Path Cost Estimation

The SD-IP and SPLO both assume that the service path is pre-computed (normally using the shortest path algorithm) and is given as an input to the algorithms. However, the service path chosen can have an impact on the performance of the algorithm. In this paper, a new approach to service path computation is proposed based on Backup Path Cost Estimation (BPCE). The idea is to estimate the backup path's cost and include this cost when calculating the final link cost ($cost_l^{new}$).

```
Data   : Network Graph: $G(V,E)$, Failure scenario set: $FS$, Request: $r$
Result : New Network Graph: $G(V,E)$, Service Path: $p$, Backup Path set: $BP$, Failure scenario set: $FS$
Initialize $cost_l^{new} = cost_l$ for each link $l \in E$;
for each $s \in FS$ do
    Find the set of failed links $E_s$ in the current failure scenario;
    Remove all failed links $l \in E_s$ from $G(V,E)$;
    Recompute the network link cost $\tau_l^s$;
    Find the shortest path $bp$ using the new network graph;
    for each $l \in E_s$ do
        if $cost_l^{new} < cost_{bp} + cost_l$ then
            $cost_l^{new} = cost_{bp} + cost_l$;
        end
    end
    Insert failed links back into $G(V,E)$;
end
Find the shortest path $p$ using graph $G(V,E)$ with $cost_l^{new}$ as the link cost;
Update the network for using $p$ as the service path.
```

Algorithm 3: BPCE service path algorithm

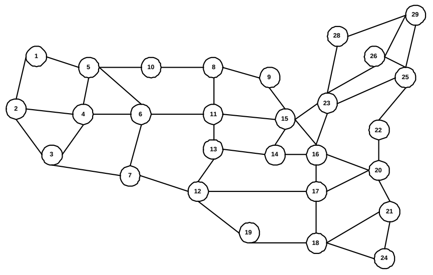
Fig. 1. Network1: US Long-Distance

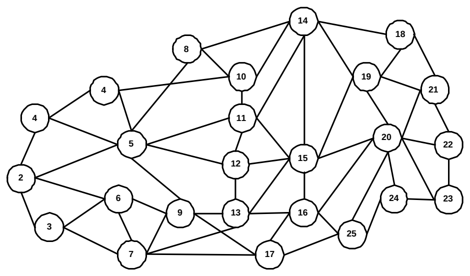
Fig. 2. Network2: Toronto Metropolitan

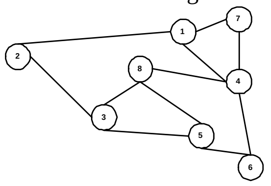
Fig. 3. Network3: Random Sparse Network

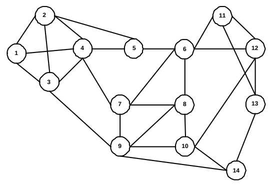
Fig. 4. Network4: Random Dense Network

The objective is to find a service path that minimizes total bandwidth requirement, i.e., the decision for the service path must take into account the service bandwidth allocation and the potential backup paths' bandwidth allocations. Therefore, $cost_l^{new}$ is equal to the original cost ($cost_l$) of using this link plus the estimated backup path cost to protect this link.

Algorithm 3 illustrates the BPCE service path algorithm. The inputs to the algorithm are: the network graph, the set of failure scenarios, and the connection request. The outputs are: the service path, updated state of the network graph.

For every failure scenario defined ($s \in FS$), the failed links are removed and a backup path is calculated using the current network resource state ($G(V, E)$) and link costs τ_l^s. τ_l^s is calculated using the same equation as defined in section IV. This backup path provides the estimated cost ($cost_{bp}$) to protect the failed links if the service path wants to use one of these links. The new cost ($cost_l^{new}$) for using link l in the service path is then the original cost $cost_l$ plus the backup cost ($cost_{bp}$). A shortest-path algorithm (e.g., Dijkstra with complexity $O(|V| \log |V|)$) is used to compute the service path. The time complexity for calculating the service path is $O(|FS|(|E| + |V| \log |V||))$.

Once the service path is calculated, a separate algorithm such as SPLO or SD-IP is then applied to compute the set of backup paths. Note that the estimated backup path cost provides an upper bound of the cost for the real backup paths, because the real backup paths can also share capacity with each other, thus potentially lowering the total backup cost further.

A potential variation is to bias the calculation towards minimizing the spare capacity as opposed to the total bandwidth. Therefore, the same algorithm is used except that the new link cost is equal to the estimated backup path cost ($cost_{bp}$) and $cost_l^{new}$ is initialized to zero at the beginning. In Section VI the results from these two variants in combination with SPLO are compared. The results show that minimizing the total bandwidth requirement is the better approach.

TABLE II

LINK CAPACITY LIMITS

Network Topology	**Link Capacity Limit (units)**
Network 1	100
Network 2	30
Network 3	140
Network 4	100

We use four network topologies for the simulations. The networks are shown in Fig. 1–4. Network 1 and Network 3 are sparse networks of different sizes. Network 2 and Network 4 are dense networks of different sizes. Network 1 and Network 2 are taken from the network topologies used by Xiong et al. [18] to allow comparisons of performance of the algorithms under similar conditions. The denser networks resemble metropolitan style networks while the sparse networks resemble backbone networks.

Without loss of generality, in all networks, all links are bi-directional and all the weights are set to 1. Each link is also given unlimited capacity unless explicitly stated otherwise in limited capacity scenarios. In all limited capacity experiments, each link is assumed to have the same capacity for simplicity and the link capacity for each network topology is shown in Table II. The link capacities are chosen such that there are significant number of blocked requests for all algorithms. Higher link capacity is required for smaller networks to support the same number of requests as in larger networks. This also applies to sparse networks, where more link capacity is required than in a dense network to support the same number of requests.

Request pairs are randomly generated and have uniform bandwidth demand of one unit each. For all experiments, the result is taken from the average of results from 10 different request sets. Each request set is fixed at 1000 requests.

The simulation considers only 100% recovery from single link failure and single node failure separately. Other types of failures are not part of the problem definition and are not considered in the simulations. For single node failure, the failure of the source node and the destination node are not considered. The request sets include considerable amount of direct connection between source and destination node, especially for the smaller network topologies. This causes some distortion when comparing between the link failure and node failure results.

The main metrics used for simulation results are as follows:

1) Total bandwidth refers to the sum of the bandwidths required by all the service paths and all the backup paths. This reflects the network utilization performance of the algorithms under normal network conditions.
2) Total spare capacity refers to the bandwidth used only by the backup paths.
3) Average backup path length is the sum of all backup path lengths divided by the number of backup paths. The length is measured by the number of hops. This shows an important attribute of the backup paths computed and answers questions about the results gained in other metrics.
4) Average service path length is the sum of all service path lengths divided by the number of service paths. The length is measured by the number of hops.

5) Number of Blocked Requests is the number of requests blocked in the simulation run. A request can be blocked if the algorithm fails to find a service path or backup path solution. This shows how the algorithms perform under stressed network conditions.

The simulation results are presented and examined in four parts that correspond to the list of contributions in Section I.

The first part compares the different state-dependent online algorithms. The two algorithms: SD-IP and SPLO, proposed in this paper are compared against the only other existing state-dependent online algorithm known to us: SSR [19].

SSR initially computes the backup paths using normal shortest paths that will protect the service path from the failure scenarios. At periodic intervals the algorithm recomputes all the backup paths using the latest network resource state. For the simulations, we assume a centralized version of the algorithm and one iteration of re-computations are initiated after every 100 requests. After 1000 requests, the iterative incremental process was allowed to continue until no more improvements were made in the iteration or the iteration count reaches 30. Fortunately, in the centralized computation case, the algorithm converges to a local optimum within 10 iterations.

Another algorithm that is used for comparison is the non-sharing shortest path algorithm. The non-sharing algorithm reflects the 1+1 restoration strategy and is used as a base performance metric. This algorithm uses the shortest path for the service path and then finds a shortest path that is disjoint from the service path as the backup path. The bandwidth used in the backup path is not shared with other backup paths.

The second part compares our new offline heuristic algorithm with the one proposed by Xiong et al. [18] (X-FORC). The results of the offline algorithms are further compared with the proposed online algorithms to compare the performance differences for using online computations.

All the state-dependent algorithms in the first part and second part of the simulations use the shortest path between the source and destination as the service path.

The third part of the simulation results examines the performance advantage of using BPCE for the service path computation while using SD-IP or SPLO as the backup path algorithm. We will also compare the performance using the biased variant of BPCE discussed in Section IX.

The final part of the simulation results compares online algorithms under different strategies: state-dependent and state-independent. For the state-independent online capacity assignment problem, the corrected version of Kodialam et al.'s [12] Integer Programming formulation (CIS) is compared to the SD-IP algorithm proposed in this paper. Li et al. [15] proposed a polynomial-time heuristic algorithm (FIR) for the state-independent problem. We compare FIR with our proposed polynomial-time algorithm (SPLO). The performance differences are analyzed and different characteristics of the algorithms are discussed.

The algorithms used in the simulations are summarized in Table III. The abbreviations are: SI for State-Independent, SD for State-Dependent.

TABLE III

ALGORITHM COMPUTATION TYPES

Algorithm	Computation	Time	Strategy
Non-sharing	Online	Polynomial	SI
X-FORC	Offline	Non-Polynomial	SD
SSR	Online	Non-Polynomial	SD
SPLO	Online	Polynomial	SD
nSPLO	Offline	Non-Polynomial	SD
SD-IP	Online	Non-Polynomial	SD
FIR	Online	Polynomial	SI
CIS	Online	Non-Polynomial	SI

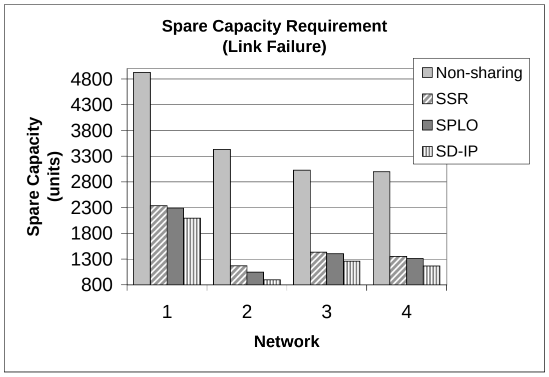

Fig. 5. Total spare capacity vs network (single-link failure)

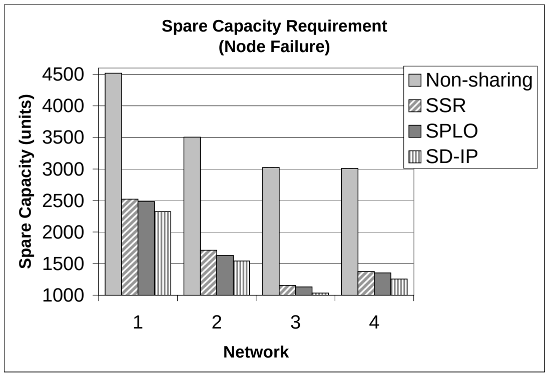

Fig. 6. Total spare capacity vs network (single-node failure)

VII. SIMULATION RESULTS PART I

The first experimental results illustrate the difference in performance between the state-dependent online algorithms. In particular, they show the significance of using a restoration algorithm in reducing the total spare capacity requirement. The heuristic algorithms are compared with the non-sharing algorithm in single-link (Fig. 5) and single-node failure scenarios (Fig. 6). The results show that for single link failure, the sharing based algorithms reduce the spare capacity requirement by 55–74%. The spare capacity requirement for the non-sharing algorithm for single node failure is not significantly larger than that for single link failure. The extra capacity is derived from the slightly longer backup paths required to communicate around the node failure. The sharing based algorithms suffer a larger performance reduction because of the need to protect from multiple link failure caused by a single node failure. However, the spare capacity reduction, in the range of 44–51%, is still substantial.

Note that the results for the non-sharing algorithm for Network 1 are distorted because of blocked requests in the single-node failure scenario. Blocking occurs when the algorithm fails to find a path that is node disjoint from the service path. In Network 1, the blocking is because of traps [23], where the decision made for the service path traps the network into the blocked state. The approximation algorithms do not have this problem because they are based on the state-dependent strategy, which uses one backup path per link or node failure. This highlights the

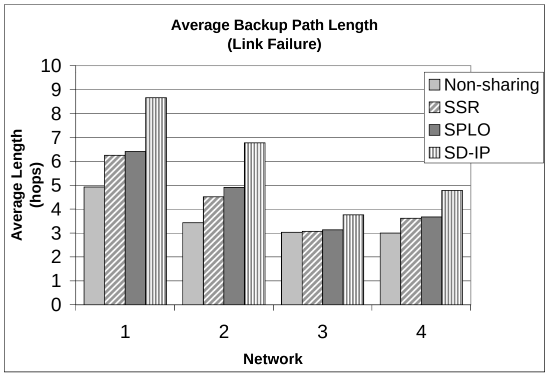

Fig. 7. Average backup path length vs network (single-link failure)

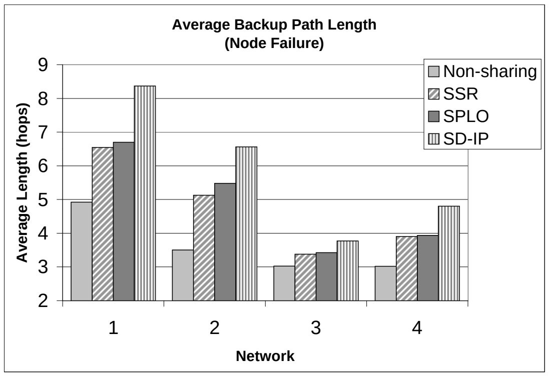

Fig. 8. Average backup path length vs network (single-node failure)

flexibility of the state-dependent restoration strategy to avoid traps in network topologies. We are not interested in blocked requests in this part of the results thus the blocking requests are removed from the experiment request set in Network 1 for single-node failure.

We now focus more closely on the results for the approximation algorithms. SSR requires 3–11% more spare capacity than SPLO in link failure and up to 5% more in node failure. By making better decisions on the first computation of the backup paths, SPLO shows that a better local optimum can be reached by the algorithm. Even without further re-computation of the backup paths, SPLO is able to reduce spare capacity more than SSR.

SPLO can be seen as a greedy algorithm that approximates the SD-IP solution. In the SD-IP algorithm, all the backup paths for the request are computed at the same time, hence optimizing the backup paths together. SPLO computes each backup path sequentially and makes the best decision for one backup path at a time. Results show that SD-IP is able to gain up to 15% advantage over SPLO in link failure and up to 6% for node failure.

To analyze these performance differences between the algorithms, some additional results are required and are shown in Fig. 7 and Fig. 8. The graphs show the average backup path lengths for different failure scenarios. These graphs, which should be compared with the corresponding graphs in Fig. 5 and Fig. 6, show the relationship between backup path length (in hop counts) and spare capacity. The algorithms that perform better have longer backup paths. This is because of the need to move away from the service path to find opportunities for finding spare capacity to utilize. This indicates that the more distributed the backup paths are across links, the better the performance. The non-sharing algorithm produces the shortest backup paths, and they are significantly shorter than the backup paths from the approximation algorithms.

Using this relationship, we can go deeper into the algorithms and find the logic that differentiates their performance. The SSR algorithm takes a fast computation approach for computing the initial backup paths, that is, it uses a normal shortest-path algorithm to find the backup paths. This pushes the backup paths closer to the service path, so the spare capacity is distributed only in a narrow range of links. After a given interval (100 requests in the simulations), the backup paths are recalculated in the hope of finding better sharing between the computed backup paths. This will re-organize the backup paths and spread out the spare capacity. SPLO makes the best decision for

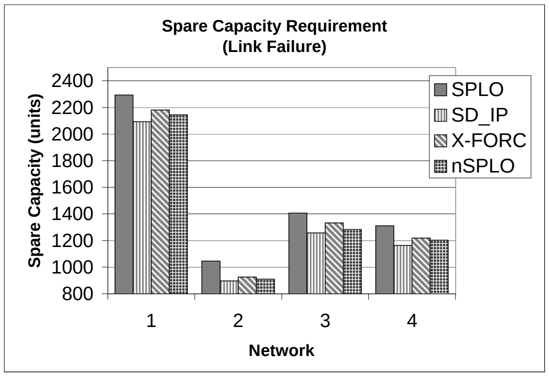

Fig. 9. Total spare capacity vs network (single-link failure)

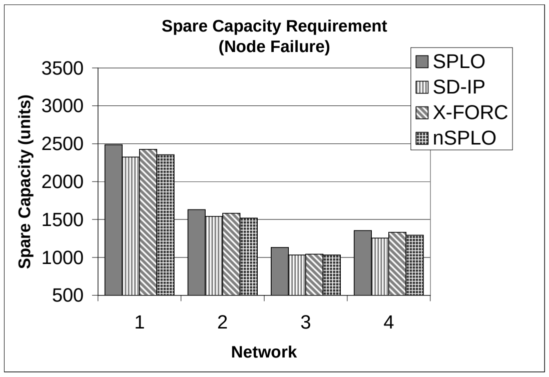

Fig. 10. Total spare capacity vs network (single-node failure)

each backup path on the first and only computation of the backup path. Thus it looks into a wider range of links to search for free spare capacity it can use. This spreads the spare capacity out in a wider range of links over each request. As a result, it spreads the backup paths further away from the main path and increases the backup paths' lengths. The initial decisions made by SSR push the solutions to an inferior local optima compared to those that SPLO can reach.

The average backup path length for SD-IP is significantly longer than the one for SPLO and this corresponds to the significantly lower spare capacity requirement shown earlier. SD-IP is able to find backup paths that not only utilize available spare capacity but also spare capacity used by the backup paths being computed in the current request. This comes from optimizing the backup paths together.

VIII. Simulation Results Part II

The second part of the simulation results compares the new offline heuristic (nSPLO) with another offline heuristic (X-FORC). We also include the best-performing online algorithms (SD-IP and SPLO) from Section VII to compare with the offline algorithms.

Fig. 9 and Fig. 10 show the spare capacity differences between the online and offline algorithms for link failure and node failure respectively. First we compare the offline algorithms. nSPLO performs better than X-FORC by up to 4% for both link failure and node failure. Comparing the best-performing offline algorithm with the best online algorithm, we observe that SD-IP does better than nSPLO by up to 3% for both type of failures.

For single link failure, we can observe that SPLO performs within 11% of X-FORC and 13% of nSPLO. The difference in performance is larger in dense networks where SPLO requires 7% to 11% more spare capacity than X-FORC. In sparse networks, however, the difference is lower (5%). For single node failure, SPLO performs comparatively better: spare capacity requirements are within 8% of X-FORC and nSPLO.

Fig. 11 and Fig. 12 show the average backup path length for different failure scenarios. Comparing with the corresponding graphs in Fig. 9 and Fig. 10, these graphs show the relationship between backup path length and spare capacity. Again, we observe that the algorithms that perform better have longer backup paths.

The X-FORC algorithm uses the failure scenario loop where it computes all the backup paths in a given scenario first. This puts priority on better utilization between backup paths in the same failure scenario. However, all the

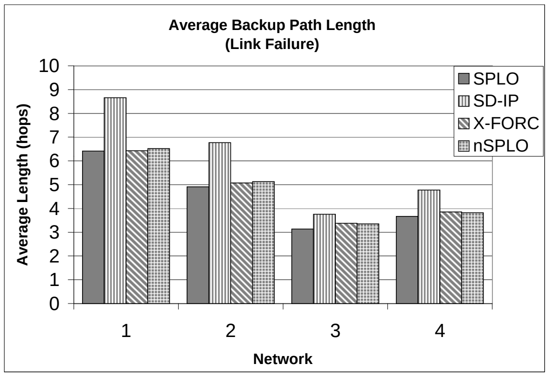

Fig. 11. Average backup path length vs network (single-link failure)

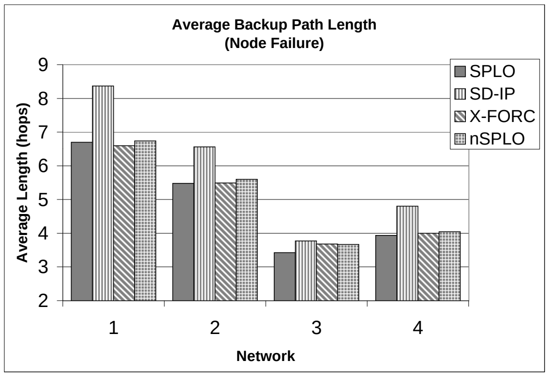

Fig. 12. Average backup path length vs network (single-node failure)

backup paths must recover from the same failed link so the backup paths will tend to be packed around the area near the failed link. Therefore, maximum sharing occurs within that small area. The SPLO algorithm uses the service path link loop where it computes all the backup paths for a service path only. In this case, priority is given to best utilizing spare capacity between backup paths belonging to the same service path. The spare capacity is thus spread over a large area surrounding the service path. Therefore, SPLO naturally distributes spare capacity in a wider area of the network than X-FORC. The iterative post-processing loop allows X-FORC to redo decisions made initially. The result is that it spreads the backup paths further out and increases the backup path length. However, the decisions made initially force X-FORC into inferior local optima compared to nSPLO. This is reflected by the consistently better performance of nSPLO. Although SPLO does not reach the local optimum, it comes very close (nSPLO), especially for single node failure.

The SPLO results are closer to those for X-FORC/nSPLO for sparse network topologies because in dense networks, there are many more alternative backup paths that the post-processing step can utilize and the chance of finding improved backup paths is much higher than in sparse networks. For single node failure, the choice for alternative backup paths is greatly reduced, hence the probability of finding improved backup paths is considerably lower, even for dense networks. The result is that post-processing has less impact for single-node failure.

IX. Simulation Results Part III

To investigate the advantage of using the BPCE service path calculation approach, we combine BPCE with SD-IP and SPLO. BPCE attempts to modify the service path based on the estimated backup cost. Thus, the benchmark metric used for performance is the total bandwidth requirement. This includes the bandwidth allocated to the service paths and the backup paths. The results are shown in Fig. 13 and Fig. 14 for link failure and node failure respectively.

The results show that the reduction in total bandwidth is within 2–3% for both SPLO and SP-ID. The important point to make is that BPCE makes a trade-off between using a longer service path and reducing the total bandwidth. Although the trade-off may have an impact for that request, the reduced backup cost will be insignificant as more requests are computed. This is because service bandwidth cannot be shared whereas the bandwidth used for backup

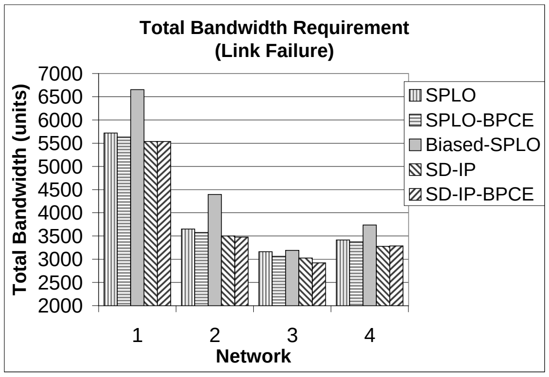

Fig. 13. Total bandwidth vs network (single-link failure)

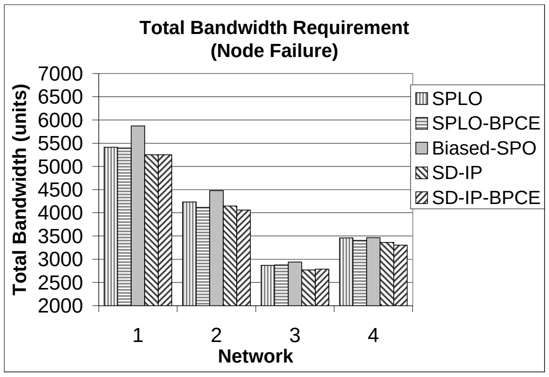

Fig. 14. Total bandwidth vs network (single-node failure)

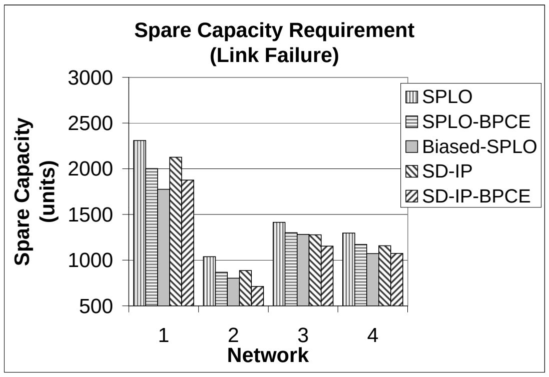

Fig. 15. Total spare capacity vs network (single-link failure)

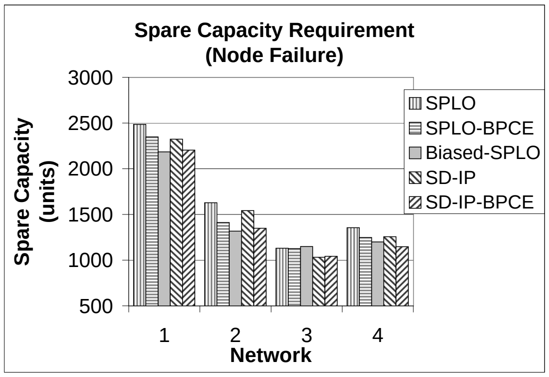

Fig. 16. Total spare capacity vs network (single-node failure)

can be shared between different backup paths. The biased variant of BPCE (Biased-SPLO) aggressively tries to reduce backup cost only.

Fig. 15 and Fig. 16 show that Biased-SPLO produces the best result in reducing total spare capacity. This implies that Biased-SPLO tries to minimize spare capacity at the expense of significantly increasing the service capacity. Total bandwidth performance results show that this is the wrong strategy to use when the objective is to minimize overall network bandwidth consumption. The significant reduction of the total spare capacity made by Biased-SPLO is offset by the increased service path length. This also shows that using the normal shortest-path algorithm for calculating the service path is very effective when the objective is to minimize overall network bandwidth consumption.

Fig. 17 and Fig. 18 show the number of blocked requests under stressed network conditions where link capacity is limited. The results for SSR are also included in the figures.

Results show that SSR performs better than SPLO in Network1 and Network4. SPLO performs better than SSR in Network2 and Network3. When SPLO is enhanced with BPCE, the number of blocked requests is significantly reduced and is lower than that of SSR in all the network topologies. This is because BPCE estimates the cost of protecting a link if used by the service path. If the estimation algorithm cannot find a feasible backup path to protect this link, the estimated cost will be set to infinity. Thus, this link will not be used in the service path computation. This reduces the chance of network traps as a result of the network topology or network conditions.

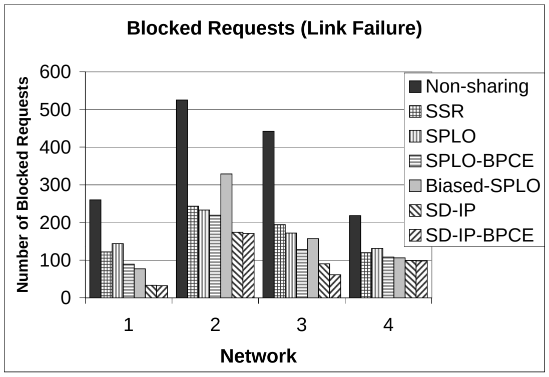

Fig. 17. Number of blocked requests vs network (single-link failure)

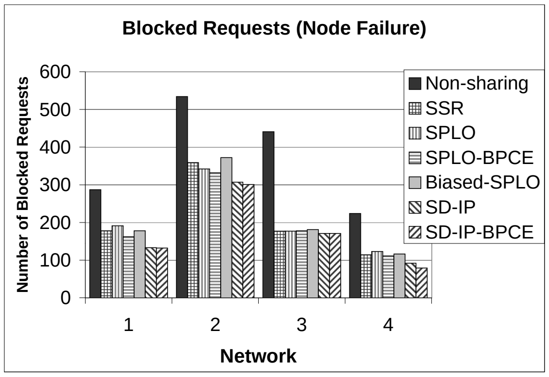

Fig. 18. Number of blocked requests vs network (single-node failure)

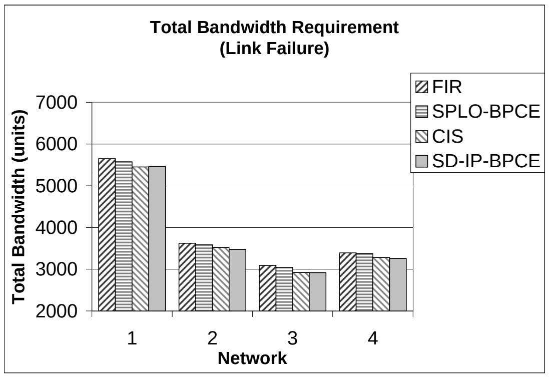

Fig. 19. Total bandwidth vs network (single-link failure)

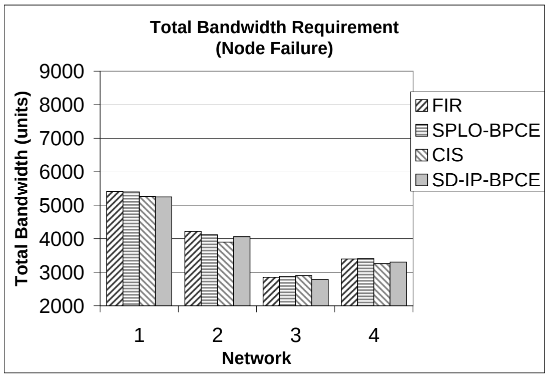

Fig. 20. Total bandwidth vs network (single-node failure)

SPLO-Biased performs the worst out of the restoration algorithms. This is because the higher total bandwidth requirement offsets the flexibility of BPCE to avoid network traps. This results in less requests being accepted in the network.

BPCE makes only a minor difference (up-to 2% better) when combined with SD-IP. SD-IP is already very efficient and flexible because it computes all backup paths for the request simultaneously. The total bandwidth efficiency of SD-IP-BPCE results in significantly less blocked requests than SPLO-BPCE.

We also observed that the significance of BPCE is reduced for node failure. Node failure have more impact on the network, increases the spare capacity dependencies, and also reducing the choice of alternative paths. BPCE also seems to have more impact on sparse networks. This is because the chance of network traps occurring in a dense network is much lower than in sparse networks.

X. Simulation Results Part IV

The final part of the simulation results compares the online algorithms using state-dependent and state-independent strategies. The results for the total bandwidth requirement are shown in Fig. 19 and Fig. 20 for single-link failure and single-node failure respectively.

Results show that the SD-IP-BPCE solution and the CIS solution require very similar total bandwidth in the network (within 2%). Comparing the polynomial-time algorithms, SPLO-BPCE and FIR, the results are very similar

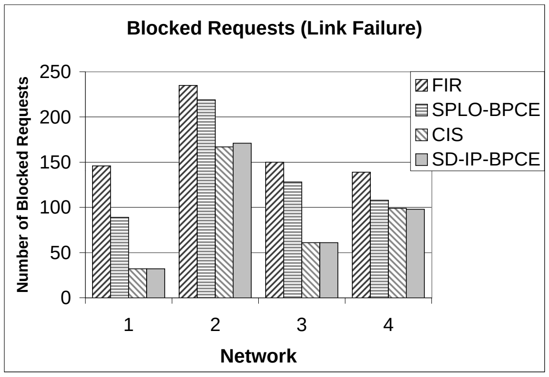

Fig. 21. Number of blocked requests vs network (single-link failure)

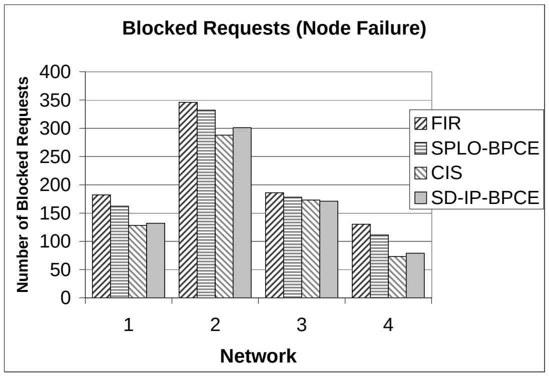

Fig. 22. Number of blocked requests vs network (single-node failure)

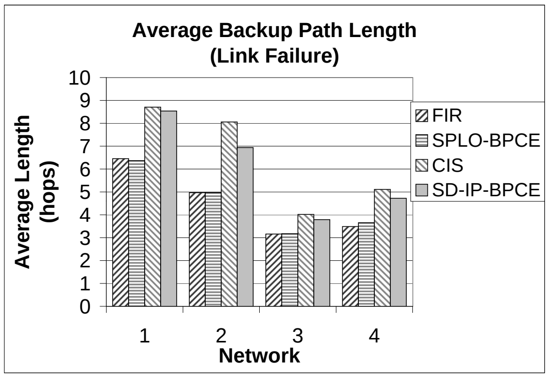

Fig. 23. Average backup path length vs network (single-link failure)

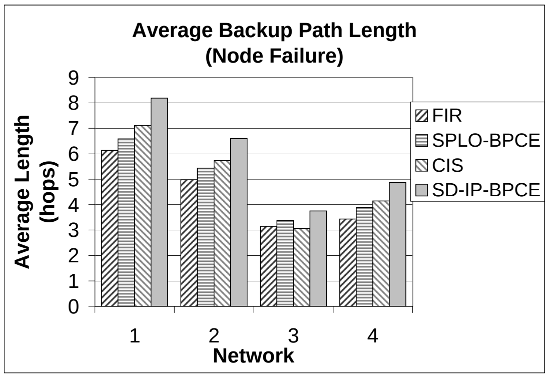

Fig. 24. Average backup path length vs network (single-node failure)

with SPLO-BPCE performing up to 2% better than FIR for most of the simulation runs.

The main difference is in the number of blocked requests as shown in Fig. 21 and Fig. 22. SPLO-BPCE uses a more complex service path computation algorithm while FIR uses only the standard shortest path algorithm.

The results show that SPLO-BPCE has significantly less blocked requests than FIR in link and node failure. CIS performs very similar SD-IP-BPCE in link failure and does slightly better than SD-IP-BPCE in node failure. The integer programming based solutions perform significantly better than the polynomial-time heuristics.

Fig. 23 and Fig. 24 show the average backup path lengths calculated by the algorithms. The polynomial-time algorithm for the state-independent strategy (FIR) shows a similar trend of long backup paths as a result of aggressive spare capacity reduction. The better performing integer programming based solutions (CIS and SD-IP-BPCE) use significantly longer backup paths. The longer backup paths reach further out and find more backup bandwidth sharing opportunities.

Fig. 25 and Fig. 26 show the average service path lengths of the algorithms. Although CIS clearly uses the longest service paths, it is close to those for SD-IP-BPCE and SD-SPLO. FIR obviously has the shortest service path because it uses the shortest path algorithm for calculating the service paths.

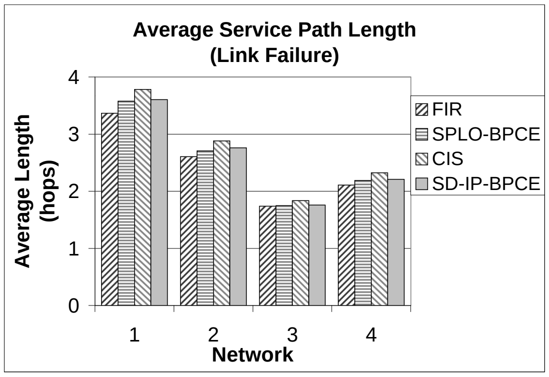

Fig. 25. Average service path length vs network (single-link failure)

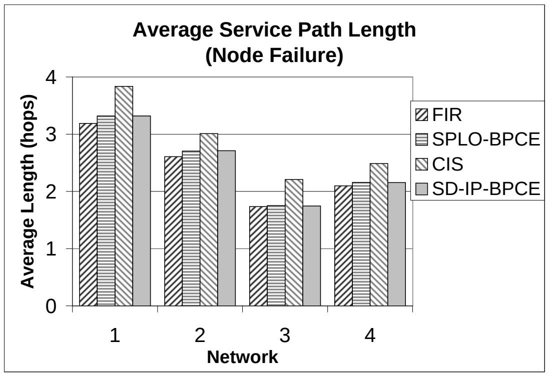

Fig. 26. Average service path length vs network (single-node failure)

TABLE IV

AVERAGE ALGORITHM RUN-TIMES FOR 1000 REQUESTS (IN SECONDS)

Network	SSR	X-FORC	nSPLO	SPLO	SPLO-BPCE	SD-IP	SD-IP-BPCE	FIR	CIS
1	113	323	297	3	47	45	85	2	257
2	90	477	471	2	48	46	83	1	688
3	4	6	6	0.5	3	15	22	0.3	31
4	19	53	50	0.9	13	20	33	0.6	82

XI. ALGORITHM RUN-TIME BENCHMARK

To illustrate the important differences in computation-time of the heuristic algorithms, a simple run-time benchmark experiment is demonstrated. Heuristic algorithms: SSR, X-FORC, nSPLO, SPLO, SPLO-BPCE, and FIR, are based on non-optimized implementation using C++. The Integer Programming algorithms: SD-IP, SD-IP-BPCE, and CIS are executed using CPLEX 8.1[1] [24]. Note that, caution should be placed when comparing between the heuristics and the CPLEX based computation time.

The run-times are in seconds and include the service path computation as well as the backup algorithm computation. The experiments were based on the single link failure simulation runs (1000 requests) for the unlimited link capacity scenario. All computations were made on a PIII 1 Ghz Linux Server with 1 GB of RAM. The processing times for the algorithms are shown in Table IV.

The slowest algorithm is clearly CIS, which takes an order of magnitude longer than all other algorithms. The difference is much larger when the network size increases and also when the density increases. CIS is definitely not a scalable algorithm and is not suitable for on-demand applications where response-time is just as critical as network efficiency.

Our proposed algorithm SD-IP-BPCE, runs up-to 8 times faster. The speed of the computation of SD-IP-BPCE seems reasonable for middle-scale restoration systems even for relatively large networks. The difference between the two IP algorithms is because of the number of constraints used in the formulation. The joint computation approach

[1]CPLEX is a commercial software that has been optimized.

of CIS requires constraints in the order of $O(|E|^2+|V|)$ while SD-IP-BPCE is in the order of $O(|FS_p|(|E|+|V|))$. $|FS_p|$ denotes the number of failure scenarios affecting the service path. In the worst case possible, the number of failures affecting the service path is equal to the number of links in the network (i.e., $|FS_p| = |E|$). However, in a mesh-based network this is very unlikely, especially if the network is of relatively large size. In practice, $|FS_p|$ is much smaller than $|E|$ ($|FS_p| \ll |E|$) and results from simulations (Section X) show that the average service path length is relatively short.

The run-times of SD-IP-BPCE and CIS differ more under node failure scenarios, especially in limited link capacity simulation runs. This is because of the exhaustive branching and cut-off CPLEX uses to search for a feasible solution in CIS. Over one request set in the larger network, CIS in limited capacity and node failure scenarios requires up-to 20 times longer computation time compared with SD-IP-BPCE.

The offline heuristic algorithms X-FORC and nSPLO are both computationally expensive and this is because of the non-polynomial time post-processing component. The results show that the iterative step is extremely expensive and is not suitable for online applications. The progressive computation component of SSR is also expensive since it requires re-calculating all the backup paths each time.

The run-times of the offline algorithms is much higher than that of SD-IP-BPCE. Compared with SD-IP-BPCE, the polynomial-time algorithm SPLO runs orders of magnitude faster. BPCE is a costly component that requires a substantially longer computation time for SPLO-BPCE. The fastest algorithm in our experiments is FIR, which performs better than SPLO because it only computes one backup path.

XII. Conclusion

The first contribution of this paper defines the online state-dependent capacity assignment problem and formulates it into a new Integer Programming problem (SD-IP). A new polynomial-time heuristic algorithm (SPLO) is developed to provide an approximate solution to the problem. Our simulations show that SPLO is better than the existing heuristic online state-dependent algorithm (SSR) and performs competitively with SD-IP, especially for single node failure.

The second contribution is a new offline heuristic that combines SPLO with a non-polynomial time bound component. Results show an improvement over the existing offline heuristic algorithm (X-FORC). In addition, we compared the offline algorithms with the proposed online algorithms and found that SD-IP performs better than the offline algorithms in link failure while performing comparably in node failure. Similarly, SPLO performs competitively with the offline algorithms. This shows that the online algorithms proposed are viable alternatives to offline applications.

The third contribution is a new service path computation algorithm for the state-dependent backup algorithms SD-IP and SPLO. Both SD-IP and SPLO compute the backup paths based on a given service path. We proposed a new service path computation algorithm that estimates the backup path cost when calculating the service path (BPCE). Results show that this approach only reduces the total bandwidth by a small amount over SPLO, but reduces the number of blocked requests significantly.

The final contribution of this paper is a detailed comparison and analysis between online algorithms of state-independent and state-dependent strategies. Results show that SD-IP performs similarly to the state-independent Integer Programming solution (CIS) in terms of total bandwidth requirements. CIS has slightly less blocked requests in limited capacity scenarios. However, the run-time cost for CIS is shown to be orders of magnitude longer than SD-IP, and is not scalable for on-demand applications. The performance of SD-IP and SD-IP-BPCE makes it an attractive restoration solution for reasonably large networks.

Polynomial-time algorithms are more suitable for large networks. The state-independent polynomial-time algorithm (FIR) has the fastest computation time and performs comparably with the state-dependent polynomial-time solutions: SPLO and SPLO-BPCE. However, SPLO-BPCE produces significantly less blocked requests than FIR. This makes SPLO-BPCE better for use in practical network situations where link capacity is limited.

Therefore, state-dependent algorithms provide better trade-offs between network efficiency and computation-time, as compared with state-independent algorithms.

REFERENCES

[1] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," *IETF RFC 3031*, 2001.

[2] J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, and W. Lai, "Applicability Statement for Traffic Engineering with MPLS," *IETF RFC 3346*, 2002.

[3] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for Traffic Engineering Over MPLS," *IETF RFC 2702*, 1999.

[4] L. Berger, "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description," *IETF RFC 3471*, 2003.

[5] P. Ashwood-Smith and L. Berger, "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Constraint-based Routed Label Distribution Protocol (CR-LDP) Extensions," *IETF RFC 3472*, 2003.

[6] L. Berger, "Generalized Multi-Protocol Label Switching (GMPLS) Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions," *IETF RFC 3473*, 2003.

[7] G. Swallow, "MPLS advantages for traffic engineering," *IEEE Communications*, vol. 37, pp. 54–57, December 1999.

[8] V. Sharma and F. Hellstrand, "Framework for Multi-Protocol Label Switching MPLS-based Recovery," *IETF RFC 3469*, 2003.

[9] Bellcore, "SONET Dual-Fed Unidirectional Path Switched Ring (UPSR) Equipment Generic Criteria," *GR-1400-Core*, 1995.

[10] K. Murakami and H. Kim, "Joint Optimization of Capacity and Flow Assignment for Self-Healing ATM Networks," in *Proceedings of IEEE ICC*, Seattle, USA, June 1995.

[11] R. Iraschko, M. MacGregor, and W. Grover, "Optimal Capacity Placement for Path Restoration in Mesh Survivable Networks," in *Proceedings of IEEE ICC*, Dallas, USA, June 1996.

[12] M. Kodialam and T. Lakshman, "Dynamic routing of restorable bandwidth-guaranteed tunnels using aggregated network resource usage information," *IEEE/ACM Transactions on Networking*, vol. 11(3), June 2003.

[13] T. Oh, T. Chen, and J. Kennington, "Fault Restoration and Spare Capacity Allocation with QoS Constraints for MPLS Networks," in *Proceedings of IEEE GLOBECOM*, San Francisco, USA, November 2000.

[14] Y. Liu, D. Tipper, and P. Siripingwutikorn, "Approximating Optimal Spare Capacity Allocation by Successive Survivable Routing," in *Proceedings of IEEE INFOCOM*, Anchorage, USA, April 2001.

[15] G. Li, D. Wang, C. Kalmanek, and R. Doverspike, "Efficient Distributed Restoration Path Selection for Shared Mesh Restoration," *IEEE/ACM Transactions on Networking*, vol. 11(5), October 2003.

[16] W. Grover and J. Doucette, "Design of a Meta-Mesh of Chain Subnetworks: Enhancing the Attractiveness of Mesh Restorable WDM Networking on Low Connectivity Graphs," *IEEE Journal on Selected Areas of Communication*, vol. 20, 2002.

[17] H. Sakauchi, Y. Nishimura, and S. Hasegawa, “A Self-Healing Network with an Economical Spare-Channel Assignment,” in *Proceedings of IEEE GLOBECOM*, San Diego, USA, December 1990.

[18] Y. Xiong and L. Mason, “Restoration Strategies and Spare Capacity Requirements in Self-Healing ATM Networks,” *IEEE/ACM Transactions on Networking*, vol. 7(1), February 1999.

[19] Y. Liu and D. Tipper, “Spare Capacity Allocation for Non-Linear Link Cost and Failure-Dependent Path Restoration,” in *Proceedings of the 14th International Workshop on the Design of Reliable Communication Networks*, Budapest, Hungary, October 2001.

[20] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, “Introduction to Algorithms,” *MIT Press*, 1990.

[21] M. Herzberg, S. Bye, and A. Utano, “The Hop-Limit Approach for Spare-Capacity Assignment in Survivable Networks,” *IEEE/ACM Transactions on Networking*, vol. 3(6), December 1995.

[22] Hillier and Lieberman, “Introduction To Operations Research, Seventh Edition,” *McGraw-Hill*, 2001.

[23] D. Dunn, W. Grover, and M. MacGregor, “Comparison of k-shortest paths and maximum flow routing for network facility restoration,” *IEEE Journal on Selected Areas of Communication*, vol. 2, 1994.

[24] ILOG, “CPLEX 8.1,” *http://www.ilog.com*, 2003.

Appendix

TABLE V

Mathematical Notations

Symbol	**Definition**
V	The set of vertexes.
E	The set of edges.
(i,j)	The directional link that connects node i to node j.
R_{ij}	Bandwidth available for allocation in link (i,j).
G_{ij}	The backup bandwidth in link (i,j).
s	The source node of the demand.
d	The destination node of the demand.
b	The bandwidth required by the demand.

Kodialam et al. [12] propose online dynamic routing algorithms for establishing paths with bandwidth guaranteed restoration. We identify two flaws in their Complete Information Scenario (CIS) Integer Programming formulation. The first flaw can produce incorrect results and is also found in the Aggregate Information Scenario algorithm (AIS). The second flaw applies to CIS only and involves loops that cause network inefficiency over a set of demands. This paper proposes corrections for these flaws.

The overall objective of online dynamic routing algorithms is to optimize the network resource utilization so as to increase the number of accepted potential future demands. The objective of CIS is therefore to minimize the additional amount of bandwidth needed to satisfy each demand. To satisfy a demand, a bandwidth guaranteed service path has to be established between the source node s and the destination node d, along with a disjoint backup path used for traffic restoration during a single link or node failure in the service path.

The original CIS algorithm is described below. For a more detailed explanation of the formulation, refer to the original literature [12].

Suppose x_{ij} is the decision variable for using link (i,j) in the service path, and y_{ij} is the decision variable for using link (i,j) in the backup path. The additional bandwidth required for the service path is defined as:

$$\sum_{(i,j)\in E} b \times x_{ij}$$

Here, b is the required bandwidth.

For the backup path, it is possible to share bandwidth with other backup paths thus the additional bandwidth required is calculated using a more complex method. For now, let z_{ij} denote the additional bandwidth required by the backup path on link (i,j). Thus the additional bandwidth required for the backup path is defined as:

$$\sum_{(i,j)\in E} z_{ij}$$

The total additional bandwidth required to satisfy the current demand is thus the sum of the service path's and backup path's additional costs. The objective of CIS is to minimize the total additional costs.

Let G_{uv} denote the backup bandwidth allocated in link (u,v). If the service path uses link (i,j) and the backup path protecting this service path uses link (u,v), then a dependency exists between the links. If (i,j) fails then (u,v) is required to be used for backup. Let δ_{ij}^{uv} denote the backup bandwidth required in (u,v) when (i,j) fails.

When routing a new demand, the cost for using link (u,v) in the backup path if link (i,j) is used in the service path is defined as θ_{ij}^{uv}:

$$\theta_{ij}^{uv} = \begin{cases} 0, & \text{if } \delta_{ij}^{uv} + b \le G_{uv} \text{ and } (i,j) \neq (u,v) \\ b + \delta_{ij}^{uv} - G_{uv}, & \text{if } \delta_{ij}^{uv} + b > G_{uv} \\ & \text{ and } R_{uv} \ge \delta_{ij}^{uv} + b - G_{uv} \\ & \text{ and } (i,j) \neq (u,v) \\ \infty, & \text{otherwise.} \end{cases}$$

The first case corresponds to when the current demand's required bandwidth added with δ_{ij}^{uv} is less than the backup bandwidth allocated in the link. Thus, using link (u,v) for the backup path when (i,j) is used in the service path does not require additional backup bandwidth allocation in link (u,v). The second case is when the reserved backup bandwidth in link (u,v) is not enough to cover the current dependency δ_{ij}^{uv} plus the requested bandwidth. Thus additional backup bandwidth needs to be allocated in link (u,v). The final case is when there is not enough available bandwidth R_{uv} in the link for the required additional bandwidth allocation. The final case also covers when (i,j) and (u,v) refers to the same link, which is not allowed since the service path and backup path need to be disjoint.

To find the final additional cost z_{uv} for using link (u,v) in the backup path for a given service path, we have to take the maximum of the additional costs that each link (i,j) on the service path demands in link (u,v).

The original CIS formulation is as follows:

$$\min\left(\sum_{(i,j)\in E} b \times x_{ij} + z_{ij}\right) \tag{12}$$

$$\sum_j x_{ij} - \sum_j x_{ji} = 0, \text{ for } i \neq s, d \tag{13}$$

$$\sum_j x_{sj} - \sum_j x_{js} = 1 \tag{14}$$

$$\sum_j x_{dj} - \sum_j x_{jd} = -1 \tag{15}$$

$$\sum_j y_{ij} - \sum_j y_{ji} = 0, \text{ for } i \neq s, d \tag{16}$$

$$\sum_j y_{sj} - \sum_j y_{js} = 1 \tag{17}$$

$$\sum_j y_{dj} - \sum_j y_{jd} = -1 \tag{18}$$

$$z_{uv} \geq \theta_{ij}^{uv}(x_{ij} + y_{uv} - 1), \forall (i,j), (u,v) \in E \tag{19}$$

$$x_{ij}, y_{ij} \in \{0,1\} \tag{20}$$

$$z_{ij} \geq 0 \tag{21}$$

The objective function is defined in Equation 12. Equations 13, 14, and 15 specify the flow conservation for the service path. Equations 16, 17, and 18 specify the flow conservation for the backup path. Equation 19 ensures that the maximum additional cost is used in each link. The binary constraints are given in Equation 20 and non-negative additional cost constraint is given in Equation 21.

The first flaw in this formulation is in the way the service path is calculated. The formulation does not take the bandwidth availability of links in the service path into account.

The objective function (Equation 12) assumes that all the links can be used by the service path with cost b. However, if a link does not have b bandwidth available then the formulation becomes incorrect and thus produces a non-feasible solution to the real problem. Therefore, the formulation needs to be modified to take bandwidth availability of the links in the service path into account. We propose the following modification to the objective function:

$$\min\left(\sum_{(i,j)\in E} \alpha_{ij} \times x_{ij} + z_{ij}\right)$$

Here, α_{ij} represents the cost for the service path for using link (i,j) and is defined as:

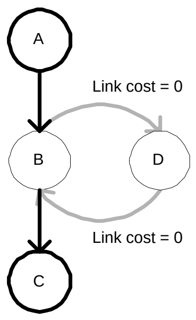

Fig. 27. Example: Backup Path Loop

$$\alpha_{ij} = \begin{cases} b, & \text{If } R_{ij} \geq b \\ \infty, & \text{otherwise.} \end{cases}$$

The first case is when there is enough bandwidth available for allocation and the cost will be the requested bandwidth. The second case is when there is not enough bandwidth available and the cost is set to infinity.

The same flaw exists in the Aggregate Information Scenario algorithm presented in the same paper [12]. Similarly, the algorithm can be corrected by checking the case for bandwidth availability on the service path.

The second flaw results in unnecessary loops in the backup path solution that reduces network efficiency over a set of demands.

An example of a backup path loop that can be formed using the original formulation is illustrated in Fig. 27. In the example, the backup path is correct but contains an unnecessary loop that consumes additional network resources. Although the loop component has cost of zero due to bandwidth sharing, it increases the bandwidth dependencies between links and indirectly reduces backup bandwidth available for future demands. Hence, the produced solution does not fully comply with the objective of the problem.

The formulation allows the loop because it satisfies the flow conservation constraints (Equation 13, 14, and 15) and does not increase the objective function (zero link cost). To prevent loops from forming, we propose the following additional constraints:

$$\sum_j y_{ji} \leq 1, \forall i \in V - \{s\} \tag{22}$$

$$\sum_j y_{js} = 0 \tag{23}$$

Equation 22 prevents using more than one incoming link for each node, except for the source node where no incoming links are allowed (Equation 23). Together with the previous path construction constraints, the possibility of loop formation is eliminated.

Similarly, the following constraints are proposed to prevent loops in the service path:

$$\sum_{j} x_{ji} \leq 1, \forall i \in V - \{s\} \tag{24}$$

$$\sum_{j} x_{js} = 0 \tag{25}$$

However, these constraints do not need to be applied to the service path (variable x) unless the demand has a zero bandwidth requirement (if $\alpha = 0$).