

# Backup Path Restoration Design using Path Generation Technique

Balaji Krithikaivasan\*, Shekhar Srivastava\*, Michał Pióro<sup>†</sup> and Deep Medhi\*

\* School of Interdisciplinary Computing and Engineering  
University of Missouri-Kansas City, Missouri-64110  
Email:dmedhi@umkc.edu

<sup>†</sup>Warsaw University of Technology, Poland, Lund University, Sweden

**Abstract**—In this work, we present a flow restoration based network design problem where we assume the knowledge of possible failure situations. We use the idea of a situation disjoint path couple (nominal path, backup path) which are constructed in such a way that atleast one of them is operational in any given failure situation for a flow. We present a linear programming formulation of the problem and use path generation technique (based on column generation) to solve the same for large networks. The path generation approach is an iterative method that allows us to add path couples with greater discretion, based on dual Lagrangean multipliers in every iteration. We consider four different scenarios that differ in the way we compute and add new path couples at every iteration. We present the results of our approach for an example network. From our observations, we conclude that path generation approach is an effective method to solve the backup path restoration design problem.

**Index Terms**—Path couple, Column generation, Backup path restoration

## I. INTRODUCTION

In the last decade, survivability has become a crucial issue in telecommunications network design. There are many kinds of events like thunderstorm, floods, equipment failures, fibre cuts, etc., that can disrupt the network operation. A network becomes more reliable, if the disrupted traffic can be restored either partially or completely during such failures. While link based restoration mechanisms are not robust towards multiple link failures, flow based restoration mechanisms can very well be designed to address one to many simultaneous link failures. We use the term "flow" here to refer to the traffic demand in terms of link capacity units. A unified approach towards flow based restoration in the event of multiple node and link failures has been presented in [4]. Moreover, flow based restoration mechanisms allow significant spare capacity sharing as compared to link based mechanisms.

Flow level survivability can be provided in many ways. For example, we can enforce path diversity (say, 0.25) at the time of flow allocation. In this way, only one-fourth of the traffic volume will be lost during a single link failure. Note that, this approach is very general in the sense that there is no assumed knowledge of link failures. However, if we have a fairly good understanding of possible link failures against which a network needs to be protected, better restoration mechanisms can be provided. In fact, knowledge of failures can be derived to an extent based on the geographical location, past catastrophic events, etc.

One approach (with assumed knowledge of failures) could be to maintain (failure) situation dependent backup paths for all the flows. So, when a link fails, the affected flows will be redirected to its backup path corresponding to the occurred failure situation. It is evident here that the number of paths need to be maintained at every node could be fairly large in order to address all the failure situations. Moreover, the flow restoration here involves identifying the situation corresponding to a link failure before it could be rerouted to the desired path. In [8], efficient ways to design a network with situation dependent backup paths have been presented recently.

Another approach could be to maintain a pair of paths, say nominal and backup path, that can address all the failure situations [5]. This translates to the requirement that either the nominal or the backup path is operational at any given situation. In other words, the nominal, backup pair should be "situation disjoint" (explained later). Thus, we need to maintain only one backup path for every flow irrespective of the number of given failure situations. Moreover, the restoration task becomes much simpler. In our work, we use such an approach towards flow restoration in the design of a survivable backbone network which we refer to as "backup path restoration".

We present a linear programming formulation to solve backup path restoration problem. The direct solution method relies on choosing the right set of path pairs in order to have a solution of desired quality. But, it is extremely difficult to identify the right ones. Hence, we apply a column generation based technique called path generation, to iteratively add path pairs to the problem based on the dual Lagrangean multipliers. In this way, we effectively control the number and the choice of path pairs that are added to the problem while leading to a solution of desirable quality.

The rest of the paper is organized as follows. In section II, we present the formal definition of backup path restoration design problem. In section III, we demonstrate the suitability and application of Path Generation technique to the design problem. We then discuss two heuristics to identify the desired path pairs in section IV. In section V, we present the convergence results for an example network. Finally, we conclude with a summary in section VI.

## II. BACKUP PATH RESTORATION - PROBLEM DEFINITION

In this section, we define the problem of Backup Path Restoration (BPR) and present a link-path formulation to solve it. We first describe the notation used further in this paper.

$\mathcal{N}$	Set of nodes
$\mathcal{E}$	Set of links
$\mathcal{D}$	Set of demands
$\mathcal{S}$	Set of link failure situations
$\mathcal{E}_s$	Set of links that fail in failure situation $s \in \mathcal{S}$
$\mathcal{P}_d$	Set of candidate path pairs for $d \in \mathcal{D}$
$\mathcal{P}$	Set of $\mathcal{P}_d$ 's of all the demands
$h_d$	Volume of demand $d \in \mathcal{D}$
$\delta_{edj}$	Entries for link-path incidence matrix (nominal); 1 if nominal path of pair $j \in \mathcal{P}_d$ for demand $d \in \mathcal{D}$ uses link $e \in \mathcal{E}$ , 0 otherwise
$\beta_{edj}$	Entries for link-path incidence matrix (backup); 1 if backup path of pair $j \in \mathcal{P}_d$ for demand $d \in \mathcal{D}$ uses link $e \in \mathcal{E}$ , 0 otherwise
$\alpha_{es}$	1 if link $e \in \mathcal{E}$ is operational in failure situation $s \in \mathcal{S}$
$\theta_{djs}$	1 if nominal path of pair $j \in \mathcal{P}_d$ for demand $d \in \mathcal{D}$ is operational in failure situation $s \in \mathcal{S}$ , 0 otherwise
$\xi_e$	Cost of unit capacity of link $e \in \mathcal{E}$
$x_{dj}$	flow allocated to path pair $j \in \mathcal{P}_d$ for demand $d \in \mathcal{D}$ (variable)
$y_e$	capacity of link $e \in \mathcal{E}$ (variable)

We consider the set  $\mathcal{S} = \{0, 1, 2, \dots, S\}$  (with  $S+1$  failure situations). Without loss of generality, we consider  $s = 0$  (in the set  $\mathcal{S}$ ) to represent the "nominal situation" in which all the links are considered to be operational. In other words,  $\mathcal{E}_0$  is an empty set. We consider single link as well as multiple link failures here by not restricting the cardinality of set  $\mathcal{E}_s$  for any given failure situation  $s \in \mathcal{S}$  to just *one*. In a network, some links might never fail at all or their failures might not be critical. Thus, the failure situations considered in the problem are the ones that are critical.

Then, we assume that the set of path pairs ( $\mathcal{P}_d$ ) is somehow precomputed (to be explained later) for every demand  $d \in \mathcal{D}$  and provided to the formulation. Each path pair consists of a nominal path and a backup path that are *situation disjoint*. This property is much more general than the link-disjointness which we illustrate through the following example.

Consider the 4-node network shown in Figure 1. Say, we have only one demand pair  $(s, t)$  and only two critical failure situations given by  $\mathcal{E}_1 = \{1, 3\}$  and  $\mathcal{E}_2 = \{2, 5\}$ . The only possible link disjoint configuration is given in Figure 1(a) and a possible situation disjoint configuration is shown in Figure 1(b). We use the term "path couples" hereafter to refer to path pairs similar to the one shown in Figures 1(a) and 1(b). In failure situation  $s = 2$ , neither among the path  $\{2, 4\}$  and the path  $\{1, 5\}$  is operational in the link disjoint configuration. On the other hand, with the given situation disjoint configuration, the path  $\{2, 4\}$  is operational in situation  $s = 1$  and the path

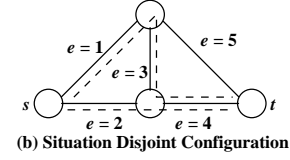
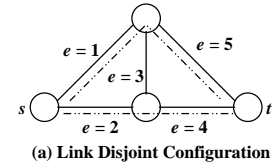


Fig. 1. An example

$\{1, 3, 4\}$  is operational in situation  $s = 2$ . Thus, we have one operational path with in the path couple for either of the failure situations in the case of situation-disjoint configuration. Note here that in Figure 1(b), link  $e = 4$  is shared by the path couple and thus, situation disjoint path pair need not be link-disjoint. In fact, we can say that if there exists a feasible link disjoint path couple for a flow, we can always find a feasible situation disjoint path couple for the same. But, the converse is not true.

For notational convenience, we use  $(n_{dj}, b_{dj})$  to refer to a path couple  $j \in \mathcal{P}_d$  further in this paper where  $n_{dj}$  and  $b_{dj}$  refers to nominal and backup path respectively. It is obvious that "nominal path" is the one that is always operational in the nominal situation ( $s = 0$ ). It may or may not work in other situations. Mathematically, if  $\theta_{djs} = 0$  in failure situation  $s$ , we have  $\alpha_{es} = 1$  for all the links  $e$  on the path  $b_{dj}$  of path couple  $j \in \mathcal{P}_d$ . Note that, we can have more than one path couple for a demand  $d \in \mathcal{D}$  that have the same nominal (backup) path but, with a different backup (nominal) path (i.e., for  $j, j' \in \mathcal{P}_d$ , we can have  $n_{dj} = n_{dj'}$ ,  $b_{dj} \neq b_{dj'}$  or vice versa). In other words, the uniqueness of a path couple is defined in terms of the pair as a whole and not individually.

Next, we present the formulation (P) for the problem BPR:

$$\min F(\mathbf{y}) = \sum_{e \in \mathcal{E}} \xi_e y_e \quad (1a)$$

subject to

$$\sum_{j \in \mathcal{P}_d} x_{dj} = h_d \quad d \in \mathcal{D} \quad (1b)$$

$$\sum_{d \in \mathcal{D}} \sum_{j \in \mathcal{P}_d} (\delta_{edj} \theta_{djs} + \beta_{edj} (1 - \theta_{djs})) x_{dj} \leq \alpha_{es} y_e \quad e \in \mathcal{E}; s \in \mathcal{S} \quad (1c)$$

In the formulation (P), constraint (1b) ensures that all the demand requirements in the network will be satisfied by the flow variables. Constraint (1c) forces the required capacity on a link to satisfy the maximum volume of flow that will be routed on it among all the failure situations. Observe that when  $\theta_{djs}$  is one (i.e.,  $n_{dj}$  of path couple  $j \in \mathcal{P}_d$  for demand  $d \in \mathcal{D}$  is operational in failure situation  $s \in \mathcal{S}$ ), the second term in the constraint (1c) vanishes and similarly, when  $\theta_{djs}$  is zero (i.e.,  $b_{dj}$  of path couple  $j \in \mathcal{P}_d$  for demand  $d \in \mathcal{D}$  is operational

in failure situation  $s \in \mathcal{S}$ ), the first term in the constraint (1c) vanishes. In this way, situation disjointness property of path couple is taken into consideration in the formulation (P). Finally, we have the objective function (1a) that minimizes the sum of weighted capacity required in the network to satisfy the demand requirements under all the failure situations.

Next, we comment briefly on solving the problem BPR. We observe that the problem BPR is a linear programming problem with the variables  $\mathbf{x}$  and  $\mathbf{y}$  taking continuous values. For networks of smaller size, we can use simplex algorithm to solve the formulation (P) directly within an acceptable time bound. However, for large scale problems, direct method might not be very helpful unless we have sufficiently large path couple sets which in turn, inflates the number of flow variables involved in the problem. Hence, we resort to a technique called Path Generation (PG) technique [8] which we discuss in detail in the next section.

### III. PATH GENERATION APPROACH

The path generation approach is one of the applications of the well-known Column Generation [1], [6] technique when applied to the link-path formulation of multicommodity flow problems. The basic premise of PG is to generate smaller routing lists to start with; then, the total number of flow variables are still manageable by the simplex method to solve. Proceeding iteratively, this technique uses Lagrangean multipliers to compute new paths at every iteration, that are included in the routing lists of the demands. The advantage of this method is that it can stay with a small subset of candidate routing lists as the iteration progresses towards arriving at the optimal solution. Since the paths being added at every iteration are based on the dual Lagrangean multipliers, PG technique is not biased towards the choice of initial routing lists. Suitability of PG technique for the design of robust networks has been presented recently in [8].

Now, we discuss in detail about why and how, we use PG technique to solve the formulation (P). From the formulation (P), we know that the number of variables grows linearly with respect to the number of candidate path couples. Such an observation encourages us to have a smaller candidate path couple sets in order to have a reasonable bound on the computation. However, the choice of path couples influences the optimal solution of the formulation (P). In fact, if we start with a smaller candidate set, at the optimal solution, path couple flow variables that have non-zero flow on them might be the best ones among all the candidate couples but, not necessarily the best ones among all the possible couples that can be found in a network. Thus, in order to obtain the optimal solution (for the maximal list of allowable path couples) candidate path couple set must be either fairly large or must be chosen with greater discretion. We call such a solution as *optimal in the wider sense*.

One of the well-known and extensively used method to generate candidate set of path couples is to follow a two-step approach. For the simplicity of explanation, let us consider a single source-destination pair, say  $(a, b)$ . As a first step, a

set of  $k$  nominal paths is computed between  $a$  and  $b$  (for nominal situation) using a  $k$ -shortest path algorithm. As the next step, with respect to each nominal path from  $a$  to  $b$ , we construct a surviving network by deleting all those links in the nominal path that fail under any given failure situation and use the Dijkstra's shortest path algorithm to find the backup path in the surviving network. In this way, fairly large number of candidate path couples can be computed for all the demands and it can be included in the problem. However, it is true that very few flow variables will actually be non-zero at the optimal solution besides the fact that a very high computational time is involved in solving such an inflated problem. Such a key observation justifies the use of PG technique where we start with a smaller path couple set and iteratively add path couples to it with greater discretion until we reach the optimal solution.

In order to see how the PG technique can be applied in finding the optimal solution (in the wider sense) for the formulation (P), we first need to look at its Lagrangean, the associated dual variables and the optimality conditions. Let  $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_D)$  be the vector of dual variables with unconstrained signs corresponding to the constraint (1b) where  $D = |\mathcal{D}|$ . Let  $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_E)$  be the matrix of non-negative dual variables corresponding to the constraint (1c) where  $\pi_e$  is a vector of dimension equal to  $|\mathcal{S}|$  for  $e \in \mathcal{E}$  and  $E = |\mathcal{E}|$ . For convenience, we introduce  $\omega_{djs}$  to represent the term  $(1 - \theta_{djs})$  in constraint (1c) (i.e.,  $\omega_{djs} = 1$  when  $\theta_{djs} = 0$  and vice versa). The Lagrangean function  $L(\mathbf{x}, \mathbf{y}; \boldsymbol{\lambda}, \boldsymbol{\pi})$  can be constructed as follows:

$$\begin{aligned} & \sum_{e \in \mathcal{E}} \xi_e y_e + \sum_{d \in \mathcal{D}} \lambda_d (h_d - \sum_{j \in \mathcal{P}_d} x_{dj}) - \sum_{e \in \mathcal{E}} \sum_{s \in \mathcal{S}} \pi_{es} \alpha_{es} y_e \\ & + \sum_{e \in \mathcal{E}} \sum_{s \in \mathcal{S}} \pi_{es} \left( \sum_{d \in \mathcal{D}} \sum_{j \in \mathcal{P}_d} (\delta_{edj} \theta_{djs} + \beta_{edj} \omega_{djs}) x_{dj} \right) \end{aligned}$$

which can be rewritten as

$$\begin{aligned} & \sum_{d \in \mathcal{D}} \lambda_d h_d + \sum_{e \in \mathcal{E}} \left( \xi_e - \sum_{s \in \mathcal{S}} \alpha_{es} \pi_{es} \right) y_e \\ & + \sum_{d \in \mathcal{D}} \sum_{j \in \mathcal{P}_d} \left( \sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} \pi_{es} (\delta_{edj} \theta_{djs} + \beta_{edj} \omega_{djs}) - \lambda_d \right) x_{dj}. \end{aligned} \quad (2)$$

Then, the dual function can be stated as follows:

$$W(\boldsymbol{\lambda}, \boldsymbol{\pi}) = \min_{\mathbf{x}, \mathbf{y} \geq 0} L(\mathbf{x}, \mathbf{y}; \boldsymbol{\lambda}, \boldsymbol{\pi}) \quad (3a)$$

It follows that for any given  $(\boldsymbol{\lambda}', \boldsymbol{\pi}')$ ,  $W(\boldsymbol{\lambda}', \boldsymbol{\pi}')$  is equal to  $-\infty$  unless  $\boldsymbol{\lambda}'$  and  $\boldsymbol{\pi}'$  satisfies

$$\sum_{s \in \mathcal{S}} \alpha_{es} \pi'_{es} \leq \xi_e \quad e \in \mathcal{E} \quad (3b)$$

and

$$\lambda'_d \leq \sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} \pi'_{es} (\delta_{edj} \theta_{djs} + \beta_{edj} \omega_{djs}) \quad d \in \mathcal{D}; j \in \mathcal{P}_d \quad (3c)$$

in which case,  $\mathbf{x}$  and  $\mathbf{y}$  variables will be forced to zero and  $W(\boldsymbol{\lambda}', \boldsymbol{\pi}')$  will be given by

$$W(\boldsymbol{\lambda}', \boldsymbol{\pi}') = \sum_{d \in \mathcal{D}} \lambda'_d h_d \quad (3d)$$

If  $F(\mathbf{y})$  represents the primal objective function, from the weak duality condition, we know that for any given  $(\boldsymbol{\lambda}', \boldsymbol{\pi}')$  that satisfy (3b) and (3c), we have

$$W(\boldsymbol{\lambda}', \boldsymbol{\pi}') \leq F(\mathbf{y}) \quad (4)$$

Hence the dual problem (DP) can be stated as follows:

$$W(\boldsymbol{\lambda}^0, \boldsymbol{\pi}^0) = \max_{\boldsymbol{\pi} \geq 0, \boldsymbol{\lambda}} \sum_{d \in \mathcal{D}} \lambda_d h_d \quad (5a)$$

subject to constraints (3b) and (3c). Note that

$$\alpha_{es} = 0 \text{ implies } \pi_{es} = \infty \quad e \in \mathcal{E}; s \in \mathcal{S}$$

The solution of (DP) yields the optimal multipliers  $(\boldsymbol{\lambda}^0, \boldsymbol{\pi}^0)$ . At the optimum solution, the equalities

$$\sum_{s \in \mathcal{S}} \alpha_{es} \pi_{es}^0 = \xi_e \quad e \in \mathcal{E} \quad (6a)$$

$$\lambda_d^0 = \min \left\{ \sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} \pi_{es}^0 (\delta_{edj} \theta_{djs} + \beta_{edj} \omega_{djs}) : j \in \mathcal{P}_d \right\} \quad d \in \mathcal{D} \quad (6b)$$

hold since all optimal  $\pi_{es}^0$  and  $\lambda_d^0$  must be as large as possible. We define the *generalized path length* of a path couple  $j \in \mathcal{P}_d$  for  $d \in \mathcal{D}$  as the sum of the length of the nominal path ( $n_{dj}$ ) with respect to the link costs  $\pi_{es}$  for all the situations  $s$  in which  $n_{dj}$  is operational ( $\{s : \theta_{djs} = 1\}$ ) and of the length of the backup path ( $b_{dj}$ ) with respect to the link costs  $\pi_{es}$  for all the situations  $s$  in which  $b_{dj}$  is operational and  $n_{dj}$  is not operational ( $\{s : \omega_{djs} = 1\}$ ). As an example, consider the situation disjoint configuration shown in Figure 1(b) with  $\pi_e$  for all  $e \in \mathcal{E}$  given as follows:  $\pi_1 = \{1, 0, 1\}$ ,  $\pi_2 = \{1, 1, 0\}$ ,  $\pi_3 = \{1, 0, 2\}$ ,  $\pi_4 = \{1, 1, 1\}$ ,  $\pi_5 = \{1, 2, 0\}$ . In the path couple  $(\{2, 4\}, \{1, 3, 4\})$ , path  $\{2, 4\}$  works in nominal situation as well as situation  $s = 1$  and path  $\{1, 3, 4\}$  works in situation  $s = 2$ . Thus the generalized length of the couple is  $2 (s = 0) + 2 (s = 1) + 4 (s = 2) = 8$ . Now, each  $\lambda_d^0$  can be interpreted as the generalized length of the *shortest* situation disjoint path couple (with respect to the link costs  $\boldsymbol{\pi}^0$ ) for demand  $d \in \mathcal{D}$  among all the candidate path couples.

It can then be shown that if there does not exist a demand  $d \in \mathcal{D}$  for which there can be found a situation disjoint path couple  $(n_{dj'}, b_{dj'})$  shorter (with respect to  $\boldsymbol{\pi}^0$ ) than  $\lambda_d^0$  in the sense of generalized path length measure

$$\sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} \pi_{es}^0 (\delta_{edj'} \theta_{djs} + \beta_{edj'} \omega_{djs}) \quad (7)$$

(i.e. shorter than any path couple  $j \in \mathcal{P}_d$  in the current candidate path couple sets), then including one or more path couples for one or more demands in their current candidate path couple sets will not improve the current optimal solution. On the other hand, if there exists a demand  $d \in \mathcal{D}$  for which there can be found a path couple with its generalized length (with respect to  $\boldsymbol{\pi}^0$ ) less than  $\lambda_d^0$ , then including such a path couple in the current candidate set will possibly improve the current optimal solution. In fact, the new optimal solution can never be inferior than the current optimal solution since we do not remove any existing path couple from the candidate set. In

other words, adding path couples to the candidate sets using PG technique always guarantee a *non-increasing* behaviour between the optimal values of successive instances of (P).

Such an observation provides the basis for PG approach to solve the formulation (P) effectively. In PG approach, we consider two different rules while adding new path couples to the existing candidate set. For convenience, we introduce  $\zeta_d(\boldsymbol{\pi}^0)$  to refer to the generalized length of *shortest* path couple among with respect to  $\boldsymbol{\pi}^0$  costs for demand  $d \in \mathcal{D}$ . We would like to recall here that  $\lambda_d^0$  refers to the generalized length of shortest path couple among the candidate path couples. The rules are:

- AA (Add-All) rule: for every demand  $d \in \mathcal{D}$ , we find the shortest path couple, say  $p'_d$ , with respect to  $\boldsymbol{\pi}^0$  and compute  $\zeta_d(\boldsymbol{\pi}^0)$ . We add  $p'_d$  to the set  $\mathcal{P}_d$  of those demands  $d \in \mathcal{D}$  for which  $l_d = \lambda_d^0 - \zeta_d(\boldsymbol{\pi}^0)$  is strictly positive.
- AB (Add-Best) rule: for every demand  $d \in \mathcal{D}$ , we find the shortest path couple, say  $p'_d$ , with respect to  $\boldsymbol{\pi}^0$  and compute the difference  $l_d = \lambda_d^0 - \zeta_d(\boldsymbol{\pi}^0)$ . We add  $p'_d$  to the set  $\mathcal{P}_d$  of the demand  $d$  for which  $l_d$  is *maximum*.

Now, we present Algorithm (1) based on the PG approach. The algorithm takes two arguments: the set  $\mathcal{P}$  (set of  $\mathcal{P}_d$ 's); the rule considered in adding path couples (AA/AB). The function  $SolveBPR(\mathcal{P}, \mathbf{x}^{i-1}, \mathbf{y}^{i-1})$  calls the simplex method to solve the  $i^{th}$  instance of problem (P). We consider  $\mathbf{x}^{i-1}$  and  $\mathbf{y}^{i-1}$  (values from previous iteration) as the initial basis for the simplex method in the  $i^{th}$  iteration where  $i > 0$ . We use  $(\mathbf{0}, \mathbf{0})$  as the initial basis (for the first iteration). The function  $ShortestCouple(\boldsymbol{\pi}^i, o_d, t_d, Q_d)$  is discussed in the next section. We use  $o_d$  and  $t_d$  to refer to the source and sink respectively for a demand  $d \in \mathcal{D}$  and  $Q_d$  to refer to the master set of path couples (discussed later).

#### IV. SHORTEST COUPLE COMPUTATION

It is evident that the shortest path couple computation with respect to the current dual  $\boldsymbol{\pi}$  variables at every iteration actually drives the PG algorithm. But, it is not a trivial task to compute the shortest path couple. Rather, it is a computationally intensive operation in addition to solving the LP problem itself. Hence, in our approach, we maintain two sets of path couples for every demand  $d \in \mathcal{D}$ ; a master path couple set and a candidate path couple set. The master set is invisible to the problem. In fact, it is used only to derive the candidate set used in the problem. Before we discuss on how we derive the candidate set from the master set, we present here Algorithm (2) that computes the master set of couples.

We define the set  $Q_d$  very similar to the set  $\mathcal{P}_d$  except that we denote the couple  $j$  in the master set for demand  $d \in \mathcal{D}$  using  $(n'_{dj}, b'_{dj})$ . Let us denote the cardinality of set  $Q_d$  using  $k$ . First, we compute the nominal paths  $(n'_{dj})$ 's using a  $k$  shortest path algorithm for all the demands. In particular, the  $k$  shortest path algorithm that we use here is designed around Dijkstra's shortest path algorithm. Then, we compute the corresponding backup paths based on the Dijkstra's algorithm using updated link weights. For notational

---

**Algorithm 1** *PathGeneration*( $\mathcal{P}$ , *rule*)

---

!t  
 $\mathbf{x}^0 \leftarrow \mathbf{0}$ ;  $\mathbf{y}^0 \leftarrow \mathbf{0}$   
 $i \leftarrow 1$   
**repeat**  
   $updated \leftarrow \text{false}$   
   $l_{max}^i \leftarrow 0$   
   $(\mathbf{x}^i, \mathbf{y}^i, \boldsymbol{\lambda}^i, \boldsymbol{\pi}^i) \leftarrow \text{SolveBPR}(\mathcal{P}, \mathbf{x}^{i-1}, \mathbf{y}^{i-1})$   
  **for all**  $d \in \mathcal{D}$  **do**  
     $(p_d^i, \zeta_d(\boldsymbol{\pi}^i)) \leftarrow \text{ShortestCouple}(\boldsymbol{\pi}^i, o_d, t_d, \mathcal{Q}_d)$   
    **if**  $\zeta_d(\boldsymbol{\pi}^i) < \lambda_d^i$  **then**  
      **if** *rule* is AA **then**  
         $\mathcal{P}_d \leftarrow \mathcal{P}_d \cup \{p_d^i\}$   
         $updated \leftarrow \text{true}$   
      **else**  
         $l_d^i \leftarrow \lambda_d^i - \zeta_d(\boldsymbol{\pi}^i)$   
        **if**  $l_d^i > l_{max}^i$  **then**  
           $l_{max}^i \leftarrow l_d^i$   
           $d^* \leftarrow d$   
           $p_{d^*} \leftarrow p_d^i$   
        **end if**  
      **end if**  
    **end if**  
  **end for**  
  **if** *rule* is AB **and**  $l_{max}^i > 0$  **then**  
     $\mathcal{P}_{d^*} \leftarrow \mathcal{P}_{d^*} \cup \{p_{d^*}\}$   
     $updated \leftarrow \text{true}$   
  **end if**  
   $i \leftarrow i + 1$   
**until**  $updated$  is false

---

convenience, we use  $\mathcal{Q}_d^{n'}$  and  $\mathcal{Q}_d^{b'}$  to refer to set of nominal paths and backup paths in the master set respectively. Also, we introduce a vector  $\mathbf{w} = (w_1, w_2, \dots, w_E)$  in Algorithm (2) where  $w_e$  represents the link weight of an edge  $e \in \mathcal{E}$ .

Once the master set is computed, for the very first iteration of PG algorithm, we remove the shortest path couple (in terms of its generalized length) from the master set and include it in the candidate set. Thus, we start the PG algorithm with  $|\mathcal{P}_d|$  equal to 1 for every demand  $d \in \mathcal{D}$ . Now, we discuss two heuristics that differ in the way we update the master set further and hence the candidate set of path couples. For clarity, henceforth we use  $k_1$  and  $k_2$  to denote the cardinality of the master set (rather  $\mathcal{Q}_d^{n'}$ ) with heuristic I and II respectively.

#### A. Heuristic I

Here, in every iteration  $i$  ( $i > 1$ ), inside the subroutine *ShortestCouple*, we disregard the current backup paths in the master set ( $\mathcal{Q}_d^{b'}$ ) and recompute it by taking into account the present value of dual  $\boldsymbol{\pi}$  variables ( $\boldsymbol{\pi}^i$ ). In other words, *ComputePathCouple* is invoked inside the *ShortestCouple* routine with  $\boldsymbol{\pi}^i$ . Once the backup paths are recomputed, a shortest couple is chosen based on the current generalized length. If we choose to add the shortest couple to the candidate set, we remove it from the master set.

---

**Algorithm 2** *ComputePathCouple*( $o_d, t_d, \mathcal{Q}_d, k, \boldsymbol{\pi}$ )

---

!t  
**for all**  $j \in \mathcal{Q}_d^{n'}$  **do**  
   $\mathbf{w} \leftarrow \mathbf{0}$   
  **for all**  $s \in \mathcal{S}$  **do**  
    **if**  $n_{dj}^{n'} \cap \mathcal{E}_s = \{\phi\}$  **then**  
       $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\pi}_s$   
    **else**  
      **for all**  $e \in \{n_{dj}^{n'} \cap \mathcal{E}_s\}$  **do**  
         $w_e \leftarrow \infty$   
      **end for**  
    **end if**  
  **end for**  
   $b_{dj}^{n'} \leftarrow \text{Dijkstra}(o_d, t_d, \mathbf{w})$   
**end for**

---

Note that  $\mathcal{Q}_d^{n'}$  is computed *only once* for each demand  $d \in \mathcal{D}$  in this approach. Also, the cardinality of the set  $\mathcal{Q}_d^{n'}$  decreases (not necessarily at every iteration) as Algorithm (1) proceeds. Thus, for a very small value of  $k_1$ , it is possible that Algorithm (1) might terminate due to  $\mathcal{Q}_d^{n'}$  becoming empty. Hence, it is important that we have high enough value of  $k_1$  such that the result obtained is of acceptable quality. On the other hand, for higher values of  $k_1$ , the complexity of *ComputePathCouple* routine increases due to the fact that backup paths are recomputed for all the nominal paths in the master set at every iteration of Algorithm (1). Moreover, the complexity of  $k$  shortest path algorithm (in terms of number of shortest path computations) invoked initially is also high.

#### B. Heuristic II

Here, at every iteration of Algorithm (1), we disregard the current master set completely and compute it again. In other words, inside the *ShortestCouple* routine, first we recompute  $\mathcal{Q}_d^{n'}$  using a  $k$  shortest path algorithm with  $\boldsymbol{\pi}_0^i$  as weights where  $\boldsymbol{\pi}_0^i$  corresponds to the  $\boldsymbol{\pi}$  variables for nominal situation in the PG iteration  $i$ . Then, we invoke the *ComputePathCouple* routine to compute the backup paths for the new set of nominal paths. Once we compute  $k_2$  couples, the shortest couple is chosen based on the generalized length. If we choose to add the shortest couple to the candidate set, it will be removed from the master set. Finally, the master set  $\mathcal{Q}_d$  is deleted only to be recomputed in the next iteration.

With heuristic I, only the backup paths are recomputed at every iteration and not the entire couple. Thus, the nominal paths that are already there in the master set might not be helpful in finding the shortest couple in the true sense. In other words, we are not taking advantage of the information from the  $\boldsymbol{\pi}_0$  variables at every iteration. Hence, in this heuristic, at each iteration  $i$ , we recompute the complete set of path couples based on the new information and thereby, compute the shortest couple. Note that, it is sufficient to have a smaller value of  $k_2$  here since we recompute  $\mathcal{Q}_d^{n'}$  at every iteration.

In the next section, we present numerical results for the network shown in Figure 2. We consider four different scenarios

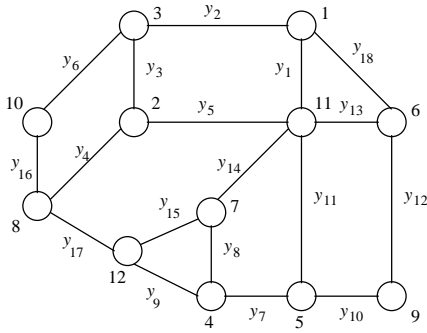


Fig. 2. 12 node, 18 link Polish Network

based on the rules AA and AB and the heuristics I and II.

## V. NUMERICAL RESULTS

We have implemented all the scenarios in  $C^{++}$  using CPLEX callable libraries [10]. The four scenarios can be explicitly understood as Heuristic I with AA rule, Heuristic I with AB rule, Heuristic II with AA rule and Heuristic II with AB rule respectively. The objective of this section are three-fold: (a) to show the convergence behavior of the four scenarios, (b) to show their computational requirement and (c) to evaluate the quality of the final solution. Due to space constraints, we show the results for only one network which is shown in Figure 2. This network is derived from the Polish backbone network. The network is fairly general and no special consideration has been made so as to avoid any biases in the results. The demand volumes (the unordered pairs of demands' and nodes are listed lexicographically) are as follows:  $h = (10, 11, 13, 13, 16, 7, 19, 10, 6, 13, 14, 3, 21, 19, 21, 14, 32, 14, 5, 68, 32, 27, 21, 25, 17, 35, 14, 3, 107, 24, 11, 21, 15, 51, 21, 19, 84, 40, 13, 20, 35, 5, 18, 74, 28, 16, 34, 4, 9, 30, 18, 21, 14, 12, 4, 15, 28, 47, 129, 14, 7, 61, 13, 24, 19, 97)$ . The failure situations considered are as follows:  $\mathcal{E}_1 = \{3, 5\}$ ,  $\mathcal{E}_2 = \{8\}$ ,  $\mathcal{E}_3 = \{7, 1\}$ ,  $\mathcal{E}_4 = \{6, 5, 2\}$ ,  $\mathcal{E}_5 = \{12\}$ . Note here that failure situations  $\mathcal{E}_1$ ,  $\mathcal{E}_3$  and  $\mathcal{E}_4$  contain multiple link failures. We would add here that the *optimal* objective value (in the wider sense) of the formulation (P) for the example network is 5520.

### A. Scenario I - Heu. I, AA rule

In Figure 3, we present convergence results for  $k_1 = 5, 10, 12$ . We see that as the value of  $k_1$  increases, optimal objective value decreases and it reaches the global optimum (in this case) for  $k_1 = 12$ . However, such a higher value of  $k_1$  leads to a very high computational complexity for computing the shortest couple at every iteration. In order to quantitatively assess this complexity, we consider the cumulative number of shortest path computations observed due to Algorithm (1). We show the same in Figure 4.

We observe that the cumulative number of shortest path computations is least for  $k_1 = 5$ . However, we compromise on the quality of the optimal solution at this value of  $k_1$ . Thus, we have a tradeoff here between the wider sense optimality

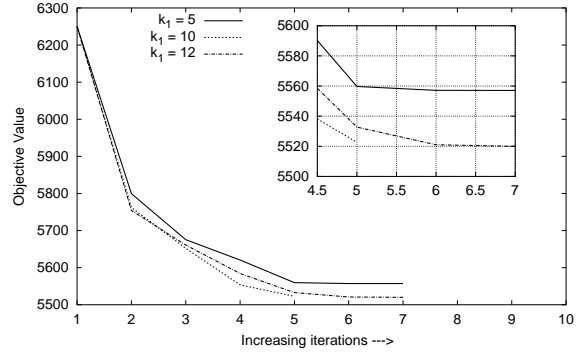


Fig. 3. Convergence results for Scenario I

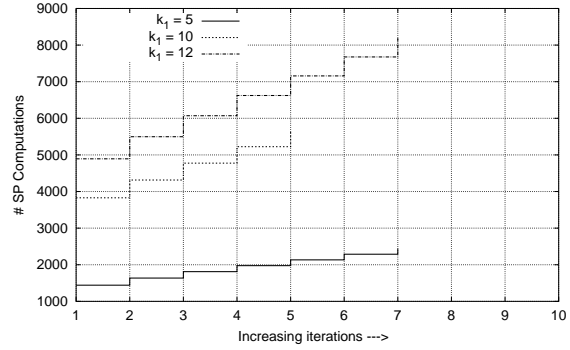


Fig. 4. Complexity results for Scenario I

of the solution and the cumulative number of shortest path computations. In other words, the value of  $k_1$  has the direct impact on the solution quality. Finally, the number of iterations observed before convergence to the global optimal solution is 7 (for  $k_1 = 12$ ). Since, we add all the path couples with a generalized length less than the current  $\lambda_d$  for a demand  $d \in \mathcal{D}$  in every iteration, the PG algorithm converges faster.

### B. Scenario II - Heu. I, AB rule

Similar to scenario I, we report the convergence results for  $k_1 = 5, 10, 12$  and the cumulative shortest path computations observed for scenario II in Figures 5 and 6 respectively. First, we observe that the number of iterations required before convergence is far greater than the AA rule (scenario I) which in turn increases the cumulative number of shortest path computations observed in general. Our next observation is that we still require the value of  $k_1$  to be 12 in order to obtain the global optimal solution. But, the cumulative shortest path computations observed for  $k_1 = 12$  is around 45000 as compared to 8000 observed in scenario I. This shows that AA rule has a lower computational complexity than AB rule. However, with AB rule, we gain in terms of the number of path couples added to the candidate set (remember that the cardinality of candidate set directly affects the flow variables involved). We discuss on this issue further in subsection V-E.

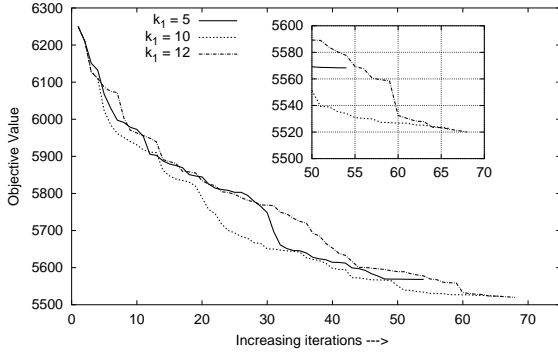


Fig. 5. Convergence results for Scenario II

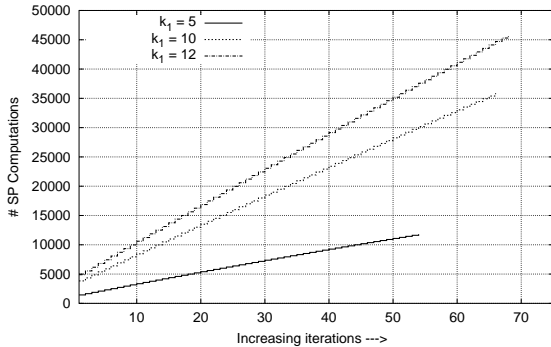


Fig. 6. Complexity results for Scenario II

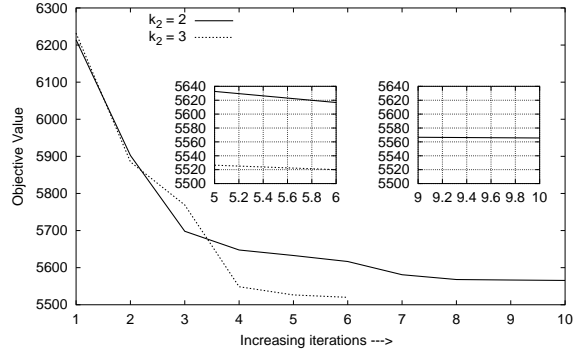


Fig. 7. Convergence results for Scenario III

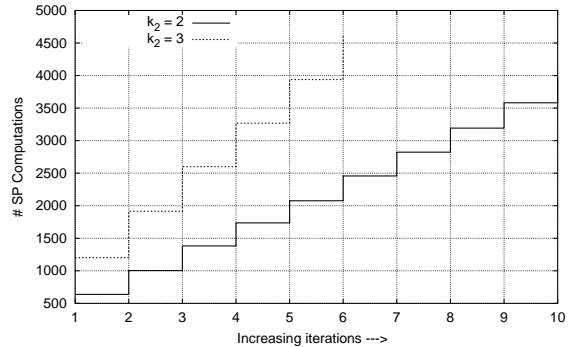


Fig. 8. Complexity results for Scenario III

### C. Scenario III - Heu. II, AA rule

We report the convergence and complexity results for scenario III in Figures 7 and 8. Note that we used small values of  $k_2$  here. In fact, for  $k_2 = 3$ , with in 6 iterations, we reached the global optimum solution (5520). Moreover, the cumulative number of shortest path computations observed for  $k_2 = 3$  is around 4500 (far less than that of scenario I). This result is counter intuitive in the sense that we require to do  $k$  shortest path computations at every iteration of PG algorithm as against just once with heuristic I. But, we should also consider that the value of  $k_2$  ( $= 3$ ) at the global optimal solution is far less than that of  $k_1$  ( $= 12$ ). It can also be accredited to the superlinear complexity of  $k$  shortest path computation.

### D. Scenario IV - Heu. II, AB rule

Figures 9 and 10 show the convergence and complexity results for Heuristic II with AB rule. We observe that the global optimum objective value is reached again at  $k_2 = 3$  but the number of iterations is nearly ten times that of AA rule to achieve the same. This behaviour is due to the addition of only one path couple (overall) at every iteration. The most interesting observation here is that the cumulative number of shortest path computations at the optimal  $k_2$  value ( $= 3$ ) is 44471 whereas it is 45883 for heuristic I with AB rule at the optimal  $k_1$  value ( $= 12$ ). Thus, we can claim that heuristic II outperforms heuristic I in terms of both the quality of optimal

solution and the computational complexity involved (in terms of SP computations) for the Polish network.

### E. AA/AB rules - Comparison

In this section, we comment on the significance of AB rule. In Figures 11 and 12, we show the total number of path couples considered (flow variables) along with the active path couples (non-zero flow variables) at the global optimal solution for AA and AB rule respectively. We observe that the total number of flow variables involved in the final iteration for AB rule is 133 (131) out of which 84 (88) of them are non-zero at the optimal solution with heuristic I (II). On the other hand, for AA rule, the total number of flow variables involved is 208 (194) out of which 90 (84) of them are non-zero at the optimal solution with heuristic I (II). Thus, with AB rule, we gain significantly in terms of the number of flow variables added to the problem as compared to AA rule.

## VI. SUMMARY

In this work, we have presented the backup path restoration problem using an link-path linear programming (LP) formulation. We have considered the notion of path couple (a pair of paths) to effectively address all the failure situations including the nominal situation. We discussed the difficulty involved in solving the LP formulation through the direct approach even for moderate networks due to the proportional

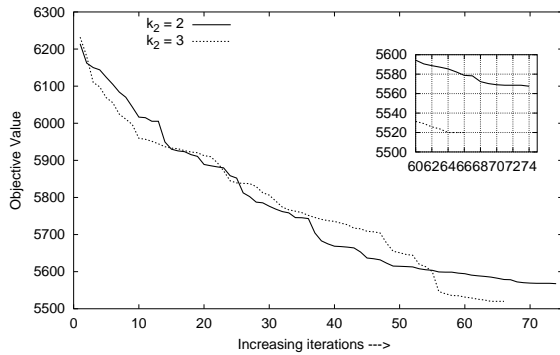


Fig. 9. Convergence results for Scenario IV

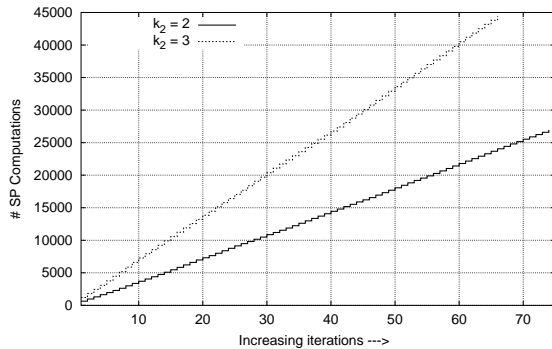


Fig. 10. Complexity results for Scenario IV

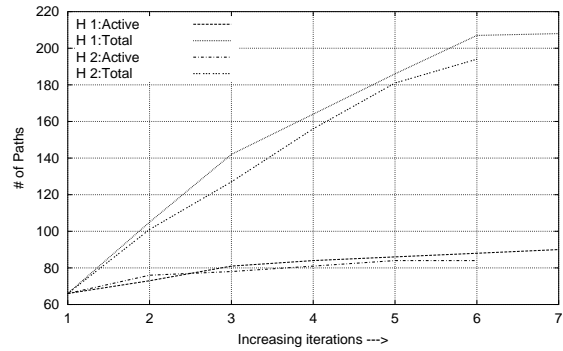


Fig. 11. AA rule - Heuristics I and II

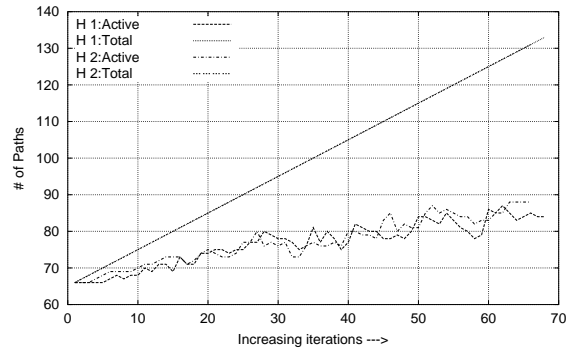


Fig. 12. AB rule - Heuristics I and II

growth of path couple flow variables. In fact, the number of candidate path couples used directly affects the quality of the optimal solution. Moreover, there is not an easy way to determine the number of candidate couples needed in advance. Hence, we applied the PG technique where we iteratively solve multiple instances of the problem and at each iteration, using Lagrangean multipliers, we update the current candidate set of path couples. We presented a PG algorithm and discussed two heuristic approaches to compute the shortest couple at every iteration. We also considered two different rules (AA and AB) to determine the number of new couples added in each iteration. Thus, with two heuristics and two different rules, we evaluated four different scenarios on an example network. We have considered the cumulative number of shortest path computations observed as a measure of computational complexity to compare all the four scenarios. For the example network, we observed that heuristic II outperforms heuristic I in terms of the computational complexity with both AA and AB rules. The other observation is that the computational complexity with AA rule is significantly lesser than that of AB rule. However, AA rule involves much more flow variables as compared to AB rule. As an ongoing work, we are currently exploring the effectiveness of path generation technique with these heuristics for other networks.

## REFERENCES

- [1] A. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows*, Prentice Hall, 1993.
- [2] B. Cotter, "Design of multi layered survivable networks", Ph.D. dissertation, School of Interdisciplinary Computing and Engineering, University of Missouri-Kansas City, 1999.
- [3] J. Geffard, H. Kerivin, D. Nace, T. Pham, "Design of survivable networks with a single facility", *Proc. 2<sup>nd</sup> European Conference on Universal Multiservice Networks*, pp. 208-218, 2001.
- [4] D. Medhi, "A unified approach to network survivability for teletraffic networks: models, algorithms and analysis", *IEEE Transactions on Communications*, vol. 42, pp 534-548, 1994.
- [5] D. Medhi, "Diverse routing for survivability in a fiber-based sparse network", *Proc. of IEEE International Conference on Communication (ICC'91)*, pp. 672-676, June 1991.
- [6] L.S. Lasdon, *Optimization Theory for Large Systems*, MacMillan, 1970.
- [7] M. Minoux, J. Y. Serrault, "Synthese optimale d'un reseau de telecommunication avec contraintes des securite", *Annales des Telecommunications*, vol. 36, no. 3-4, 1981.
- [8] M. Pióro, "Some remarks on the path generation approach in multi-commodity flow networks", Technical Report, School of Interdisciplinary Computing and Engineering, University of Missouri-Kansas City, January 2003.
- [9] T. Szymanski, "Application of the Benders decomposition to the design of robust telecommunication networks with constrained reconfiguration", *Proc. 2<sup>nd</sup> Polish-German Teletraffic Symposium*, Gdansk, pp. 259-270, 2002.
- [10] *CPLEX callable libraries manual*, ILOG Corporation, USA.