

# Minimum failure explanations for path vector routing changes <sup>\*</sup>

Mohit Lad<sup>1</sup>, Dan Massey<sup>2</sup>, Adam Meyerson<sup>1</sup>, Akash Nanavati<sup>3</sup>, and Lixia Zhang<sup>1</sup>

<sup>1</sup> Computer Science Department, University of California, Los Angeles, CA 90095, USA.  
{mohit, awm, lixia}@cs.ucla.edu

<sup>2</sup> Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA.  
massey@cs.colostate.edu

<sup>3</sup> Google Inc., Mountain View, CA 94043, USA. akash@google.com

**Abstract.** Path vector protocols in routing networks convey entire path information to each destination. When links fail, affected paths are replaced by new paths, and by observing the entire path information, one might hope to infer the failed links that caused these changes. However, inferring the exact topological changes behind observed routing changes may not be possible due to limited information, and the same changes may be explained by more than one set of candidate failures. In this paper, using a simple path vector routing model, we present the problem of finding the candidate set with minimum number of failures to explain observed route changes. We call this problem the minimum e-set problem and present algorithms for solving it optimally for certain cases. We also show that the minimum e-set problem is NP-complete in the general case.

## 1 Introduction

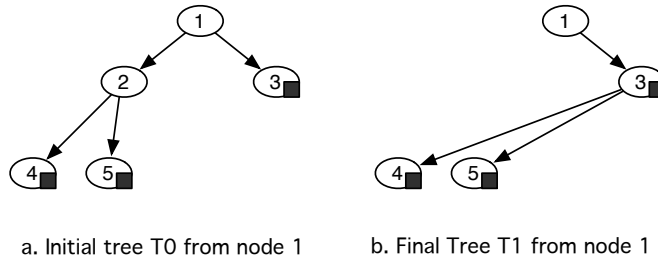
Path vector routing protocols convey complete path information to reach a destination node. These protocols can adapt dynamically to topological changes like link failures but usually do not convey any explicit notification of which links failed. Without any explicit failure notification, one can only hope to infer failed links based on the complete path information before and after failures. For example, assume a node 1 can reach a destination node 4 using the the path  $1 \rightarrow 2 \rightarrow 4$ . Now due to some link failure, this path changes to  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ . One can see that link  $(2, 4)$  must have failed to cause A to switch to path  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ . However, if the path to C changes from  $1 \rightarrow 2 \rightarrow 4$  to  $1 \rightarrow 3 \rightarrow 4$ , one cannot tell whether the link that failed was  $(1, 2)$  or  $(2, 4)$ . In cases where more than one failure scenario can explain the observed routing changes, simply analyzing the path to a single destination is not enough.

Even after looking at how paths to all destinations are affected, one might face multiple failures scenarios. For example, assume that the path to destination 5 also

---

<sup>\*</sup> This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No N66001-04-1-8926 and by National Science Foundation(NSF) under Contract No ANI-0221453. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DARPA or NSF. Part of the work was done when Akash Nanavati was at DA-IICT, India

changes from  $1 \rightarrow 2 \rightarrow 5$  to  $1 \rightarrow 3 \rightarrow 5$  at the same time as the path to destination 4 changes from  $1 \rightarrow 2 \rightarrow 4$  to  $1 \rightarrow 3 \rightarrow 4$  as shown in Figure 1. Given this information of path changes to destinations 4 and 5, three candidate failure scenarios are a) failure of link  $(1, 2)$ , b) failures of links  $(2, 4)$  and  $(2, 5)$  and c) failures of links  $(1, 2)$ ,  $(2, 4)$  and  $(2, 5)$ . We call each of these scenarios as an explanation set or e-set. Even in this simple case with one source and two destinations, we have three e-sets and cannot say for sure which one is the cause. In situations where multiple failure scenarios are possible, the minimum failure e-set problem involves identifying the minimum number of links whose failure can explain the observed route changes. Identifying minimum number of failed edges can give us a lower bound on the number of failures causing route changes. For example, in Figure 1, we can say for sure that at least one edge must have failed. In addition, if all links have the same failure probability, then one can see that the failure of link  $(1, 2)$ , i.e. the minimum e-set, is also the most likely solution.



**Fig. 1.** Possible e-sets are  $\{(1, 2)\}$  and  $\{(2, 4), (2, 5), (2, 6)\}$

In this work, formalize the minimum e-set problem and show the problem is NP-complete in the general case. We discuss conditions under which the minimum e-set can be optimally found and present simple algorithms for these cases. The remainder of the paper is organized as follows. Section 2 explains the routing and model used and formalizes a tree version of the minimum e-set problem. Section 3 presents an algorithm for the optimally solving the minimum e-set problem with tree inputs. Section 4 extends the problem of finding minimum failures to arbitrary graph inputs and presents a greedy algorithm to optimally solve the case where each node is a destination. This section also proves the general case of minimum e-set is NP-complete. Finally, Section 5 contains related work and discussion, and Section 6 concludes the paper.

## 2 Model and Problem Definition

We now provide details about the network and routing model used in this paper. We also formally define the problem of minimum e-set in this section.

## 2.1 Network and Routing Model

We model the network as a simple directed connected graph  $G = (V, E)$ , where  $V = V_D \cup V_N$  and  $E = E_D \cup E_N$ .  $V_D$  represents the set of destinations and the nodes in  $V_N$  are transit nodes. We are only interested in routes to  $V_D$ . Nodes in  $V_N$  are connected by links in  $E_N$  and each edge in  $E_N$  has the form  $(a, b)$  where  $a, b \in V_N$ . The destinations are attached to the nodes in  $V_N$  through edges in  $E_D$  and each edge in  $E_D$  has the form  $[d, n]$  where  $d \in V_D$  and  $n \in V_N$ . In all figures, nodes in  $V_D$  (destinations) are represented by the presence of small solid rectangles while nodes in  $V_N$  (transit nodes) do not have these solid rectangles. For example, in Figure 1, nodes 4, 5 and 3 belong to  $V_D$ , while nodes 1, 2 belong to  $V_N$ .

We use a Simple Path Vector Protocol (SPVP) [1]. In SPVP each node advertises *only* its best path to reach the destinations. A path from node  $v$  to destination  $d$  is a sequence of nodes  $path_v(d) = (v_k v_{k-1} \dots v_0 d)$  where  $v_k = v$ ,  $(v_i, v_{i-1}) \in E_N$  for all  $0 \leq i \leq k$ , and  $(v_0, d) \in E_D$ . We define  $pathLength(path_v(d)) = k + 1$ . After receiving and storing a route learned from its neighbors, node  $v$  selects its best path to destination  $d$  according to some routing policy.

The routing policy could be any policy that maintains the tree property. In other words, the routing table containing routes to all destinations at any stage can be graphically represented in the form of a tree. Thus, at any node  $u$ , there has to be only one path to any other node  $v$ , even if  $v$  is used as a transit node to reach some other destination. In the remainder of the paper, when we refer to ‘best’ path, we mean a path that is deemed best given the routing policy. We abstract out the notion of ‘best’ path and separate it from its physical interpretation. For illustration purposes, we use the shortest path routing policy where a node picks the path with the lowest number of hops as its best path. In case of a tie for more than one paths with the lowest number of hops, the node picks the path from the neighbor with the lowest numeric ID. While the shortest path policy is used for illustration purposes, our algorithms and proofs in the Section 3 work with any policy that can maintain the tree property.

## 2.2 Failure Model

We allow any number of links to fail in the network. All links in the network have the same failure probability. If a link fails, the nodes adjacent to the link detect the failure and all nodes using the link must switch to alternate path (or declare the destination unreachable). The link failures are not directly reported to any central database or monitoring site. However, a node whose path is affected by this failure, will see a new path implying that something went wrong on the initial tree. Different link failures may impact different observation points and hence we are concerned with the minimum failures as seen from a particular observation point only. Link failures are atomic, which means if a link fails, no node can use it. We look at route changes observed from a node to infer failed links. The route changes seen must have happened only due to link failures and cannot happen due to the addition of new links.

### 2.3 Input Requirements

We do not make any assumption about the the topology of the entire graph being known to any single node in the graph. We define an observation point as a node for which we can see the complete routing table at any time. The input we have constitutes of two routing table snapshots. We assume that the initial as well as final routing tables reflect the steady state tables after the routes for all the nodes in the network have converged. In other words, there are no more routing updates propagating in the network at the time of the two routing table snapshots.

### 2.4 The Minimum e-set problem: Tree version

We now formalize the problem of finding minimum failures given routing tree changes. At an observation point  $M$ , Let  $T_0$  and  $T_1$  indicate the routing trees used by node  $M$  at two different times  $t_0$  and  $t_1$  respectively.

Let  $\text{RouteCompute}(M, G)$  be a black box routine that computes the best path routing tree from  $M$  given an input graph  $G$ . Our goal is to determine a plausible set of links  $F \subseteq T_0 - T_1$ , that may have failed. We define such a set of plausible failures as an e-set. If many such e-sets are possible, we are interested in the set with minimum number of failures, or the minimum e-set. Formally,

$$\text{RouteCompute}(M, T_0 \cup T_1) = T_0 \quad (\text{A-1})$$

A set of links is called *explanation set* (e-set) iff:

$$\text{RouteCompute}(M, (T_0 \cup T_1) - F) = T_1 \quad (\text{A-2})$$

Note, we do not have information about the complete graph, but we can combine  $T_0$  and  $T_1$  to give us some information about the graph. Recall, Figure 1 shows routing trees  $T_0$  and  $T_1$  used by node 1. In this example, the routing policy used by the nodes is shortest path routing policy. It can be seen that  $\text{RouteCompute}(1, (T_0 \cup T_1) - \{(1, 2)\}) = T_1$  and hence  $\{(1, 2)\}$  is an e-set. Similarly  $\{(2, 4), (2, 5)\}$  is also an e-set. Our goal is to find the minimum e-set from the observed changes.

**Minimum e-set problem:** Given two routing trees  $T_0$  and  $T_1$  from a node  $M$ , and a known routing policy, find the minimum  $F$ , such that  $\text{RouteCompute}(M, (T_0 \cup T_1) - F) = T_1$ .

## 3 Minimum e-set for tree inputs

In this section, we present a heuristic to solve the tree version of the minimum e-set problem optimally. The results in this section apply to any routing policy that always produces a tree.

This heuristic uses the notion of *nearest-descendant* defined below. For any node  $u$  in  $T_0 \cap T_1$ , where  $M$  is the root of the trees, call node  $v$  a *nearest-descendant* of  $u$  if

1.  $v \in T_0$  and  $u$  is closer to  $M$  than  $v$  in  $T_0$
2.  $v$  is also in  $T_1$  or  $v \in V_D$ , where  $V_D$  is set of destinations and

3. for any  $x$  in  $T_0$  on the path from  $u$  to  $v$ ,  $x \notin T_1$

To find the nearest-descendant pairs, we use Algorithm 1. The nearest-descendant algorithm uses depth first search (DFS) over  $T_0$  to find the nearest descendants of a node  $u \in T_0$ . Along any DFS path from  $u$ , if a node  $v$  is present in  $T_1$ , then no descendant of  $v$  needs to be explored, and  $v$  is the nearest descendant of  $u$ . Figure 2 shows trees  $T_0$  and  $T_1$  observed from node 1. Node 1 has three children 2, 3 and 6. Node 2 is not present in  $T_1$  nor is a destination, hence 2 cannot be a nearest descendant of 1. Recursing from node 2, we can find 4 and 5 to be in  $T_1$ , and thus are nearest descendants of 2 and hence of 1. Similarly, 3 is also a nearest descendant of 1. Finally, notice that 8 is not in  $T_1$ . This is possible if there is no valid path to reach 8 in  $T_1$ . But, since 8 is a destination, it is also a nearest-descendant of node 1. Summarizing, node 1 has 4 nearest descendants in 3, 4, 5 and 8. Among, the other nodes, 2 has nearest descendants 4 and 5, while nodes 3, 4 and 5 do not have any nearest descendants.

---

**Algorithm 1:** nearest\_descendant(Node  $u$ , Path  $\rho$ )

---

**Input:** Node  $u$  in  $T_0$ , Path  $\rho$

**Output:**  $P$ : Set of paths to nearest-descendants of  $u$

**while** there is some  $v$ , such that  $(u, v) \in T_0$  AND  $v$  is not visited **do**

    Mark  $v$  as visited ;

    Add  $(u, v)$  to path  $\rho$ ;

**if**  $v \in T_1$  or  $v \in V_D$  **then**

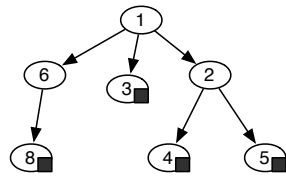
        └ Add path  $\rho$  to  $P$ ;

**else**

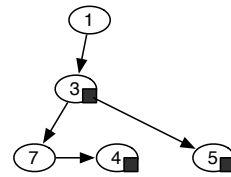
        └ nearest\_descendant( $v, \rho$ );

    Remove  $(u, v)$  from path  $\rho$ ;

---



a. Initial tree  $T_0$  from node 1



b. Final Tree  $T_1$  from node 1

**Fig. 2.** Example for nearest descendant and minimum e-set from Algorithm 2

Once, we find the set of all  $(u, v)$  pairs, such that  $v$  is a nearest-descendant of  $u$ , we use Algorithm 2 to compute the minimum e-set. For each  $(u, v)$  pair, we check if  $T_1$  would still be the computed tree if edges on the path  $(u, v)$  are added to  $T_1$ . For

example, consider the pair  $(1, 8)$  with path  $1 \rightarrow 6 \rightarrow 8$ , we have

$$\text{RouteCompute}(1, T_1 \cup \text{path}(1, 8)) \neq T_1$$

Thus, we fail the first link on the path  $1 \rightarrow 6 \rightarrow 8$ . Similarly, edge  $(1, 2)$  is marked failed for path between  $(1, 4)$  as well as  $(1, 5)$ . On the other hand consider nearest descendant pair  $(2, 4)$ . For this pair we have,

$$\text{RouteCompute}(1, T_1 \cup \text{path}(2, 4)) = T_1$$

and hence we do not fail any edge on this nearest descendant pair. After considering all the  $(u, v)$  pairs, the minimum e-set in this case is  $\{(1, 2), (1, 6)\}$ . We now prove correctness and optimality for Algorithm 2.

---

**Algorithm 2:** FindFault()

---

**Input:**  $T_0$  and  $T_1$ , the routing trees from vantage point  $M$   
**Output:**  $F$ : set of edges marked failed;  
**for** each pair nearest-descendant pair  $(u, v)$  with path  $P$  **do**  
    **if**  $(\text{RouteCompute}(M, T_1 \cup P)) \neq T_1$  **then**  
        /\* One of the edges on the  $(u, v)$  path  $P$  must have failed. \*/  
        Mark the first edge on the  $(u, v)$  path in  $T_0$  as failed.

---

**Lemma 1.** *The set of edges  $F$  thus determined comprises an e-set.*

*Proof.* We prove this theorem using shortest path routing, but the proof can easily apply to other tree based routing policies. Assume the contrapositive, in other words, when these edges fail the resulting tree  $T_f$  on  $(T_0 \cup T_1) - F$  is not the final tree  $T_1$ . It follows that there's some  $(a, b)$  in  $T_1$  such that a shorter path exists in  $T_f$ . Since we do not fail edges of  $T_1$ , the path  $(a, b)$  can only be shorter. Let  $(a, b)$  be such a pair whose real shortest path in  $T_0 + T_1 - F$  is as short as possible. Suppose the shortest path between  $(a, b)$  includes some node  $x$  in  $T_1$ . Then we must have either  $(a, x)$  or  $(x, b)$  closer in  $T_0 + T_1 - F$  than they are in  $T_1$ , thus contradicting the definition of  $(a, b)$ . It follows that the path between  $(a, b)$  does not include any other vertices of  $T_1$ , and therefore travels entirely through  $T_0$ . In particular,  $(a, b)$  are nodes of  $T_0$  (and  $T_1$ ) and the path between them does not contain any nodes of  $T_1$ , so one is a nearest-descendant of the other. It follows that we would have included an edge on the  $(a, b)$  path in  $F$ . □

**Theorem 1.** *The e-set  $F$  thus determined is minimum.*

*Proof.* Consider all pairs  $(u, v)$  where  $v$  is a nearest-descendant of  $u$ . We know that we must delete some edge on the path  $(u, v)$  in  $T_0$ , otherwise  $T_1$  could not be a shortest path tree. Note that if  $v$  is a nearest-descendant of  $u$ , then  $v$  cannot also be a nearest-descendant of  $u'$  for any  $u'$  not equal to  $u$ . We can therefore imagine chopping the trees

into edge-disjoint subtrees, where each subtree consists of a node in both  $T_0$  and  $T_1$  along with its nearest-descendants and the paths to them. For each subtree, we consider breaking it up into further edge-disjoint subtrees via the edges from the root ( $u$ ). For each edge outgoing from  $u$ , if we cut that edge then any algorithm must cut one of the edges in the relevant subtree. Since all subtrees are disjoint, it follows that any algorithm must cut at least as many edges as ours cuts.

□

## 4 Minimum e-set for general graphs

So far, we discussed how to optimally solve the minimum e-set problem with the nearest-descendent approach. In Section 3, we restricted the input to a node be the routing trees from that node before and after the failures. However, nodes might have additional information about the underlying topology besides its routing tree. One source of information is the routes received from other nodes. For example, lets assume a node  $A$  receives a route  $B \rightarrow C \rightarrow D$  to reach the destination  $D$  from neighbor  $B$ . Node  $A$  also receives the route  $F \rightarrow D$  to reach destination  $D$  from neighbor  $F$ . Assume node  $A$  selects route  $F \rightarrow D$  via neighbor  $F$  (assuming shortest path), and does not use  $B$  to reach any other destination. Even though links  $B \rightarrow C$  and  $C \rightarrow D$  may not be in  $A$ 's routing table, it knows these two links are up, else  $B$  would have sent a new route to reach  $D$ . To account for this extra information that a node might have, we generalize the problem to find minimum e-set when the inputs before and after failures can be general graphs instead of a trees.

### 4.1 Problem Definition: Graph Version

At an observation point  $M$ , let  $T_0$  and  $T_1$  indicate the routing trees used by node  $M$  at two different times  $t_0$  and  $t_1$  respectively. In addition, let  $G_0$  and  $G_1$  indicate the set of links known to be up at  $t_0$  and  $t_1$  respectively. The set  $G_0$  could be just  $T_0$ , but can contain additional arbitrary edge information, and  $T_0 \subseteq G_0$ . Similarly,  $T_1 \subseteq G_1$ . We have,

$$\text{RouteCompute}(M, G_0 \cup G_1) = T_0 \quad (\text{A-3})$$

A set of links is called *explanation set* (e-set) iff:

$$\text{RouteCompute}(M, (G_0 \cup G_1) - F) = T_1 \quad (\text{A-4})$$

**Minimum e-set problem:** Given two graphs  $G_0$  and  $G_1$  from a node  $M$  containing routing table links as well as other links, and a known routing policy, find the minimal set of edges  $F$ , such that  $\text{RouteCompute}(M, (G_0 \cup G_1) - F) = T_1$ , where  $T_1$  represents the routing table  $\text{RouteCompute}(M, G_1)$ .

While, we show the general case of this problem is NP-complete, a special case of this problem with graph inputs can be solved optimally if each node is a destination.

#### 4.2 The case of no purely transit nodes, $V_N = \phi$ or $V = V_D$

We present a greedy Algorithm 3 called `greedy_Fault()` to find optimal solution for the case where each node is a destination, or equivalently there are no transit nodes. This algorithm removes one edge at a time with edges ordered in breadth first order.

---

**Algorithm 3:** `greedy_Fault( $M, G_0, G_1$ )`


---

**Input:**  $M$ : the vantage point ;  
 $G_0$ : up edges at time  $t_0$ ;  
 $G_1$ : up edges at time  $t_1$ ;

**Output:** e-set of candidate failures  $F$

1. Initialize  $G_f = G_0 \cup G_1$ ;
2. Compute  $T_f = \text{RouteCompute}(M, G_f)$ ;
3. **for** edges  $(X, Y)$  of  $T_f$  in *BFS order* **do**
  - if**  $(X, Y) \notin G_1$  **then**
    - $F = F \cup \{(X, Y)\}$ ;
    - remove  $(X, Y)$  from  $G_f$ ;
    - goto step 2;
4. return  $F$ ;

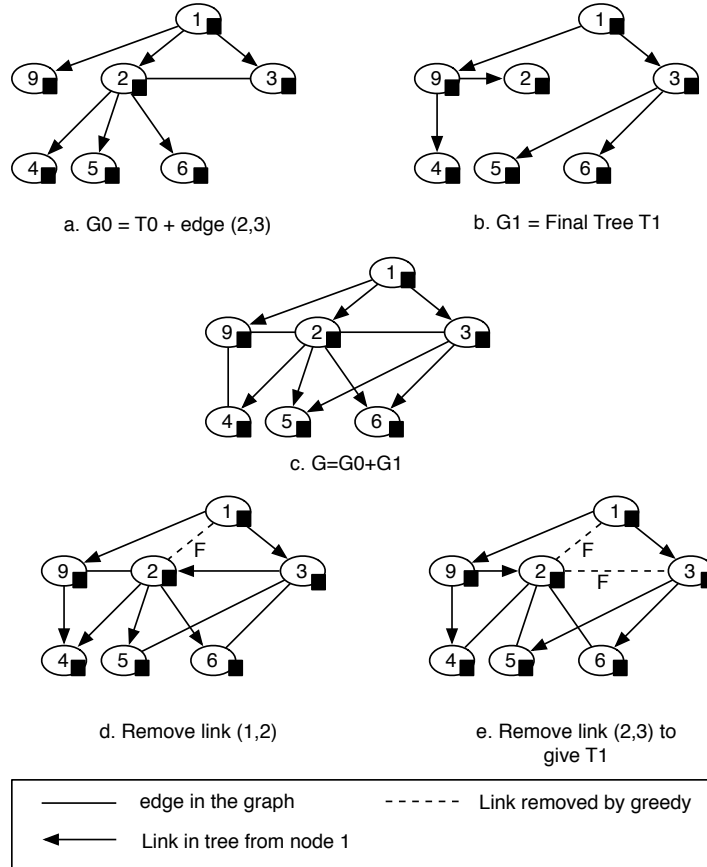
---

Figure 3 provides an example showing the output of the greedy algorithm on a simple graph. For this example, we assume the shortest path routing policy with lowest ID tie-breaking. In this example, each node is a destination. Figure 3a and Figure 3b represent the initial and final graphs. Note, the link  $(2, 3)$  is not in  $T_0$  but is present as additional information in the initial graph  $G_0$ . In contrast the final graph  $G_1$  does not contain any new information and hence is the same as  $T_1$ . At each stage, we superimpose the tree edges shown by directed edges over edges in the graph. After removal of link  $(1, 2)$  from  $G$ , the link  $(3, 2)$  is now on the tree from node 1 as shown in Figure 3d. It can be seen that link  $(3, 2)$  is not in the final tree and without its failure, the path cannot change to use link  $(9, 2)$ . Hence,  $(3, 2)$  is failed next to result in the final tree  $T_1$ .

We define ‘sufficient’ failure set as the links whose failure is enough to change the tree from  $T_0$  to  $T_1$ , and ‘necessary’ failure set as the links which must fail in order to change the tree from  $T_0$  to  $T_1$ . With each node as a destination our algorithm will always output sufficient as well as necessary failures.

**Theorem 2.** *Let  $F$  be the set of edges returned by the algorithm `greedy_Fault( $M, G_0, G_1$ )`. Then  $F$  is both necessary and sufficient set of edges to explain the change from  $T_0$  to  $T_1$ .*

*Proof.* To show that  $F$  is sufficient, we will show that  $T_f = T_1$  when the algorithm terminates. In the `for loop`, we remove edges only from  $G_0 - G_1$ . Thus at the end,  $T_f = \text{RouteCompute}(M, G')$  where  $G_1 \subseteq G'$ . Also, the `for loop` removes edges from  $T_f$  that are in  $G_0 - G_1$ . Thus at the end,  $T_f$  does not include any edges from  $G_0 - G_1$ . Thus  $T_f = \text{RouteCompute}(M, G') = \text{RouteCompute}(M, G_1) = T_1$ .



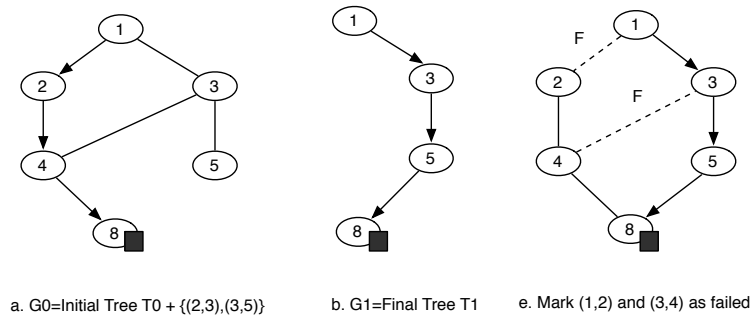
**Fig. 3.** `greedy_fault()` with each node as a destination

Now to show that  $F$  is necessary, let  $F'$  be any set of edges such that  $\text{RouteCompute}((G_0 \cup G_1) - F') = T_1$ . We will show that  $F \subseteq F'$ , by induction on the iteration of the `for` loop. Suppose by induction, at the end of the  $i$ -th iteration,  $e_1, \dots, e_i$  are the edges selected by the algorithm, and by induction,  $F'$  includes these edges. We will show that  $F'$  must include  $e_{i+1} = (u, v)$  selected by the algorithm in the  $(i + 1)$ -st iteration. Note that the algorithm chooses the *first* edge in the BFS order of  $T_f$  from  $M$  that is not in  $G_1$ . Consider the path  $P$  from  $M$  to  $u$  in  $T_f$ . All edges of  $P$  are available at time  $t_1$  (they are in  $G_1$ , so the algorithm did not select them). Hence if  $(u, v)$  did not fail during  $[t_0, t_1]$ , and since  $v \in V_D$ ,  $T_1$  would include  $(u, v)$ . It follows that any e-set  $F'$  must also include  $(u, v)$  in order to be a valid explanation-set.

□

### 4.3 The case of purely transit nodes, $V_N \neq \phi$

The Algorithm 3 from the previous section does not produce an optimal solution if the network contains purely transit nodes. Consider Figure 4 as an example where we see routes from node 1 to a single destination node 8, and nodes 2, 3, 4, 5 are just transit nodes. Parts 4a and 4b show the initial and final graphs. The edges with arrows represent the routing table, while the others represent information about edges through other means. We assume the network uses shortest path routing with lowest ID tie breaking for this example. The greedy Algorithm 3 would first mark edge (1, 2) failed, which would shift the route to  $1 \rightarrow 3 \rightarrow 4 \rightarrow 8$ , since 4 has a lower ID than 5 and both the path via 4 and 5 are of length 3. The greedy algorithm would then mark edge (3, 4) failed resulting in the final tree. But clearly in this case, the e-set of  $\{(4, 8)\}$  can explain the above change and is also minimal. We now prove that this general version is NP-complete.



**Fig. 4.** Example showing greedy not optimal in general case

To show that finding minimum e-set in the general case is NP-complete, we will consider a restriction of it, Directed Path Change (DPC), and prove it is NP-complete. In DPC, we restrict the number of destinations to one, so that each routing graph is a single path. But we allow arbitrary  $G_0$  and  $G_1$ . Since DPC involves change of a single path instead of multiple paths in our general problem, it is thus a subset of the general case and proving hardness for DPC will imply hardness for the general problem of multiple destinations. We use shortest path policy for this proof.

*Directed Path Change (DPC):* Given directed graphs  $G_0, G_1$ , two special nodes  $s, t$ , and an integer  $k \geq 0$ , let  $P_i$  denote the minimum hop  $s \rightarrow t$  path in  $G_i$  such that  $P_0$  is also the minimum hop  $s \rightarrow t$  path in  $G_0 + G_1$ . Does there exist a set of edges  $F \subseteq G_0 - G_1$  such that  $|F| \leq k$  and  $P_1$  is also a minimum hop  $s \rightarrow t$  path in  $G_0 + G_1 - F$ ?

We reduce following problem, UMC, proven to be NP-complete by [2].

**Undirected Multiway Cut (UMC):** Given undirected graph  $G$ , three special nodes  $x, y, z$  and an integer  $k \geq 0$ , is there a set of edges  $F$ ,  $|F| \leq k$  such that in  $G - F$ ,  $x, y, z$  are disconnected from each other?

**Reduction:** Given an instance of  $UMC(G, x, y, z, k)$ , we create a new directed graph  $G'$  as follows. The vertices of  $G'$  are the vertices of  $G$  plus many additional vertices. The edges of  $G'$  are defined as follows: for every (undirected) edge  $(u, v)$  of  $G$ , we put following gadget:<sup>4</sup> put two new vertices  $w_1, w_2$  (new for each different edge) and add edges  $(u \rightarrow w_1), (w_2 \rightarrow u), (v \rightarrow w_1), (w_2 \rightarrow v)$  and  $(w_1 \rightarrow w_2)$ . Now any path in  $G$  that goes  $u \rightarrow v$ , can be simulated in  $G'$  as  $u \rightarrow w_1 \rightarrow w_2 \rightarrow v$  and any path in  $G$  that goes  $v \rightarrow u$  can be simulated in  $G'$  as  $v \rightarrow w_1 \rightarrow w_2 \rightarrow u$ . We also add two special vertices  $s, t$  in  $G'$  and add edges  $(s \rightarrow x), (z \rightarrow t)$ . Let  $n$  denote the number of vertices in  $G$ . Take two long paths  $Q_1, Q_2$  of length  $D \gg 4n$  consisting of new vertices, where  $Q_1$  is  $x \rightarrow y$  and  $Q_2$  is  $y \rightarrow z$ . Let  $P_0$  be the minimum hop path  $s \rightarrow t$  in  $G'$  and let  $P_1$  be an  $s \rightarrow t$  path by following  $s \rightarrow x \xrightarrow{Q_1} y \xrightarrow{Q_2} z \rightarrow t$ . This  $(G_0 = G', G_1 = P_1, s, t, P_0, P_1, k)$  is our instance of DPC.

We shall show that there exists a UMC solution  $F$  iff it is also a DPC solution. Let  $F$  be a solution to UMC instance  $(G, x, y, z, k)$ . Let  $F'$  be the set of edges in  $G'$  where for each edge  $(u, v)$  in  $F$ , we put the corresponding edge  $(w_1, w_2)$  in  $F'$ . If we remove  $F'$  and  $P_1$  from  $G_0 = G'$ , then  $x, y, z$  are disconnected. So the minimum hop  $s \rightarrow t$  path in  $G_0 - F'$  is  $P_1$ , and hence  $F'$  is a solution to DPC  $(G_0 = G', G_1 = P_1, s, t, P_0, P_1, k)$ .

Conversely, let  $F'$  be a solution to  $(G_0 = G', G_1 = P_1, s, t, P_0, P_1, k)$ . For all the edges picked up from a single  $(u, v)$  gadget, we can replace them by  $(w_1, w_2)$  and get a new solution  $F''$ ,  $|F''| \leq |F'|$ . We claim that  $G' - F'' - P_1$  has  $x, y, z$  disconnected. If there was a path between any pair, say  $Q : z \rightarrow y$ , then by symmetry<sup>5</sup> there is also a path  $\bar{Q} : y \rightarrow z$ . Now any  $y \rightarrow z$  path in  $G' - F'' - P_1$  has a corresponding path in  $G$  of length at most  $n - 1$ , and by our gadget the pathlength multiplies by 3 in  $G'$ , so  $|Q| = |\bar{Q}| \leq 3(n - 1)$ . Then the path  $s \rightarrow x \xrightarrow{Q_1} y \xrightarrow{\bar{Q}} z \rightarrow t$  is available in  $G' - F''$ , and is of length at most  $1 + D + 3(n - 1) + 1$  which is less than the length of  $P_1 = 2 + 2D$  (recall  $D \gg 4n$ ). This contradicts the claim that minimum hop path in  $G' - F''$  is  $P_1$ . Therefore  $F''$  disconnects  $x, y, z$  and by choosing corresponding  $(u, v)$  edges, we get  $F$ ,  $|F| \leq k$ , such that deleting  $F$  disconnects  $x, y, z$  in  $G$ .

Thus we have shown,

**Lemma 2.** *DPC is NP-hard.*

Since DPC is a special case of finding minimum e-set, and given any candidate solution e-set, it is can clearly be verified in polynomial time, we get,

**Theorem 3.** *Finding minimum e-set is NP-complete.*

<sup>4</sup> This gadget is also used by [3].

<sup>5</sup> If the original path traverses  $v \rightarrow w_1 \rightarrow w_2 \rightarrow u$ , then we reverse it to  $u \rightarrow w_1 \rightarrow w_2 \rightarrow v$  since our gadget has all these edges.

## 5 Related Work

The problem of isolating faults has been discussed in the context of multicast trees in [4]. In their approach, session watchers are used to identify faulty segments of multicast trees based on whether the tree changed at the common ancestor or not. Their technique localizes the faults to a particular section of the multicast tree. Active probing techniques using a Time to Live (TTL) based approach for delay monitoring and fault identification was proposed by [5]. To check if a link  $(u, v)$  has failed, a monitoring station sends probe messages for the same destination  $v$ , but with different TTL values, and makes the inference based on reply. [6] proposes a scheme to produce a ranked list of most probable failed links. These most likely faulty links then have to be tested to see which ones have failed.

In path vector routing, the single fault case is analyzed in [7] which combines the views from multiple observation points to increase accuracy. Various works have looked at identifying the root causes of Internet routing changes. Using the publicly available data of route changes to identify root cause is the focus of work in [8], [9], [10]. Each of these works uses some heuristics to relate changes across different destinations and identify the problem areas. Though these works have talked about identification of failures, to the best of our knowledge there is no prior work on identifying minimum failures.

On a related front, [11] talks about the problem of placement of monitoring points to best detect failures. A similar problem is also discussed in [5], where they show computing the minimal number of monitoring stations is NP-hard, and present an approximation algorithm.

Finally, another way to view the cause of changes is *tree-edit distance* when the operations allowed are delete and insert and trees are restricted to be Shortest Path Trees. But in general finding how to convert one tree into the other is NP-hard and many have considered approximate solutions [12].

## 6 Conclusions

In this paper, we discussed the problem of finding minimum number of failures given observed routing changes. We considered two variants of the problem, one with tree inputs and the other with graph inputs. We presented the nearest-descendent algorithm to optimally solve the tree version. We showed this problem is NP-complete in the general case where the input information represent arbitrary graphs and we may have additional information about arbitrary edges. We proved that the special case where there are no purely transit nodes can be solved optimally. An obvious future work is to find an approximation algorithm. We also plan to extend the current work by combining route changes from multiple nodes and finding the minimum e-set on this combined view.

## References

1. Griffin, T., Wilfong, G.T.: A safe path vector protocol. In: INFOCOM (2). (2000) 490–499

2. Dahlhause, E., Johnson, D.S., Papadimitriou, C.H., Seymore, P.D., Yannakakis, M.: The complexity of multiway cuts. In: 24th Annual ACM Symposium on Theory of Computing. (1992)
3. Garg, N., Vazirani, V., Yannakakis, M.: Multiway cuts in node weighted graphs. In: Journal of Algorithms. (2004) 49–61
4. Reddy, A., Govindan, R., Estrin, D.: Fault isolation in multicast trees. In: SIGCOMM. (2000) 29–40
5. Bejerano, Y., Rastogi, R.: Robust Monitoring of Link Delays and Faults in IP Networks. In: INFOCOMM. (2003)
6. Wang, C., Schwartz, M.: Identification of Faulty Links in Dynamic-Routed Networks. IEEE Journal on Selected Areas in Communication **11** (1993)
7. Lad, M., Nanavati, A., Massey, D., Zhang, L.: An Algorithmic Approach to Identifying Link Failures. In: 10th Pacific Rim International Symposium on Dependable Computing (PRDC). (2004)
8. Wu, J., Mao, Z.M., Rexford, J.: Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network. In: Proceedings of 2nd symposium on Networked Systems Design and Implementation (NSDI). (2005)
9. Feldmann, A., Maennel, O., Mao, Z.M., Berger, A., Maggs, B.: Locating Internet routing instabilities. In: Proceedings of Sigcomm. (2004)
10. Chang, D., Govindan, R., Hiedemann, J.: The temporal and topological characteristics of BGP path changes. In: ICNP. (2003)
11. Kleinberg, J., Sandler, M., Slivkins, A.: Network Failure Detection and Graph Connectivity. Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (2004)
12. Garofalakis, M., Kumar, A.: Correlating xml data streams using tree-edit distance embeddings. In: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM Press (2003) 143–154