



Efficient computation of loop-free alternates

Haijun Geng^a, Han Zhang^b, Xingang Shi^{c,*}, Zhiliang Wang^c, Xia Yin^d

^a School of Software Engineering, ShanXi University, China

^b School of Cyber Science and Technology, Beihang University, China

^c Institute for Network Sciences and Cyberspace, Tsinghua University, China

^d Department of Computer Science & Technology, Tsinghua University, China

ARTICLE INFO

Index Terms:

Loop-free alternates
Fast reroute
Multipath routing
Network availability

ABSTRACT

Network failures may lead to serious packet loss and degrade network performance. Therefore, loop-free alternates (LFA) have been widely deployed by many Internet service providers to cope with single network component failure in large Internet backbones. However, the efficiency of LFA has not been sufficiently studied. Some existing methods have extremely large computational overhead, and their computational complexity is linear with the average node degree of the network. The current methods consume a large amount of central processing unit resources, thereby aggravating the burden on the router. To improve the routing resilience without introducing significant extra overhead, this study proposes an incremental alternate computation with negative augmentation algorithm (IAC), which is based on incremental shortest path first algorithm. First, IAC turns the problem of quick implementation of LFA into efficiently calculating the minimum cost of all its neighbors to all other network nodes on the shortest path tree rooted at the computing node. Then, several theorems for calculating the cost are presented and their correctness is validated. Finally, we evaluate IAC through simulations with real-world and generated topologies. Compared with TBFH, DMPA, and DMPA-e, which are algorithms optimized for limited scenarios, IAC finds approximately 50% more available alternates, is more than three times faster, and provides node protection capabilities that TBFH, DMPA, and DMPA-e cannot provide. These advantages make IAC a good candidate for traditional telecommunication networks and emerging complex networks that require failure repair and load balancing in a highly dynamic environment.

1. Introduction

At present, the Internet supports an increasing number of real-time and mission-critical applications, which are sensitive to packet loss, network delay, or available bandwidth. However, network failures are common on the Internet (Zheng et al., 2016) (Xu et al., 2017) because hardware and software face occasional failures. Typical scenarios include a damaged fiber, breakdown of a forwarding port, or malfunction of a router. Without effective failure mitigation methods, packets forwarded to the failed equipment cannot be correctly forwarded, thereby resulting in packet losses and poor network performance. Although routing protocols automatically detect topology changes, heal the network by globally computing new paths, and reroute, these protocols typically exhibit a slow response of hundreds of milliseconds or even longer due to the convergence delay in the distributed execution of the protocol. This slow response cannot meet the requirements of real-time applications, although improving the network availability has always been a high-priority task of Internet service providers (ISPs) (Krist, 2015)–(Stephens et al., 2016).

To support critical applications or guarantee a high quality of service, ISPs generally deploy a technique called loop-free alternate (LFA) (Shan, 2010), (Atlas and Zinin, 2008) for local fast rerouting. LFA proposes several criteria for selecting proper next hops, including loop-free criterion (LFC), node-protection condition (NPC), and downstream path criterion (DC). These criteria are designed to provide alternative paths bypassing the failed component and guarantee loop-freeness. With LFA, alternate next hops are stored alongside the default next hops in the forwarding table of a router and can be immediately activated to invoke a loop-free repair path when link or node failures occur. As long as routing loops are not introduced, these alternative next hops can also be used for multipath transmission if bandwidth or load balancing faces stringent demands. Most ISPs use a link-state routing protocol, such as OSPF or IS-IS, in their intra-domain systems. Large data center networks are also incorporating link-state routing into their Layer 2 network architecture (Perlman, 2011), (Touch and Perlman, 2009) to achieve benefits such as transparent node movement, shortest path routing, traffic engineering, and multipath propagation.

However, in such link-state networks, computing LFA typically requires

* Corresponding author.

E-mail address: shixg@cernet.edu.cn (X. Shi).

one or more rounds of full shortest path tree computation on a graph, thereby posing a heavy burden to the processor load and memory consumption of network equipment. This situation further brings challenges to networks facing complicated protocol processing (such as border routers handling a large amount of BGP route updates) or dynamic reconfiguration (such as frequent path computation and traffic rebalancing). Thus, an efficient implementation of the LFA mechanism is important. Although some algorithms (Meirindol et al., 2011)– (Geng et al., 2018) have been proposed to address the aforementioned issues, those algorithms reduce computational complexity at the cost of a fair portion of candidate alternates that can be utilized, thereby sacrificing their capability of failure repair and redundancy provision. In (Meirindol et al., 2011), P. Mrindol et al. propose an efficient algorithm, TBFH, which provides greater path diversity than equal-cost multipath routing (ECMP) with an extremely low overhead. In particular, TBFH efficiently computes the two best first-hop disjoint paths. The time complexity of TBFH does not depend on the degree of the calculating router. In (Geng et al., 2013), Haijun Geng et al. present the dynamic multipath algorithm (DMPA) for a router in a link-state network to compute multiple next hops for each destination. The algorithm constructs a single shortest path tree based on ordinary network link states and dynamically adjusts it in response to network state changes so that the sets of next hops can be efficiently computed and incrementally updated. At the same time, DMPA guarantees loop-freeness of the induced routing path by implicitly maintaining a partial order of the underpinning routers. To implement high protection coverage with DC, authors in (Geng et al., 2018) present an extension of DMPA (DMPA-e) by incorporating hop count, a topology-dependent metric. However, DMPA-e cannot achieve full failure coverage against single network component failures. To achieve full protection, G. Retvari et al. in (Retvari et al., 2011) revisit LFA to provide theoretical insights and practical hints for LFA failure coverage analysis; the study concludes that cleverly adding only a couple of new links can drastically improve the quality of LFA protection.

In this paper, we propose IAC, an algorithm that can compute the full set of alternates for a given network topology in a highly efficient manner. IAC performs incremental shortest path computation on specific link cost update, where the signs of some costs are simply reversed, i.e., from l to $-l$. We formally prove the property of IAC and use synthesized and real-world topologies to evaluate its performance. Compared with the naive implementation in typical commercial routers, IAC achieves the same coverage of alternates but with an order of magnitude reduction in computation time or $O(k)$ reduction in memory consumption, where k is the degree of a node. Being compatible with the proposed state of the art (Retvari et al., 2011), IAC can cope with any single network component failure in case (Retvari et al., 2011) is deployed in the network.

Our main contributions are summarized as follows:

- We propose IAC, which is the first algorithm that can solve LFC, DC and NPC in a unified framework. However, for other algorithms, the different algorithmic framework is required for each different rule.
- To the best of our knowledge, IAC is the first algorithm, which can efficiently solve the NPC problem. The algorithms TBFH, DMPA, and DMPA-e can only deal with two rules, i.e., LFC and DC, in LFA; however, NPC cannot be solved using these algorithms.
- We theoretically analyze the correctness and computation complexity of the IAC.
- iSPF has been widely deployed on the backbone of ISPs, and IAC is compatible with the currently deployed intra-domain routing protocols. Therefore, IAC can be easily deployed on the Internet.
- We evaluated IAC by simulations on a large number of network topologies. Experiments indicate that the computation complexity of IAC is less than that of constructing a shortest path tree and can provide the same network availability as LFA.

The rest of this paper is organized as follows. Section II presents the background of LFA, graph model, and necessary notations. Section III describes the details of our IAC algorithm with its theoretical analysis,

and Section IV reports our evaluation results. Finally, Section V concludes the paper.

2. Background

2.1. Failure repair and multipath routing

Network component failures have become routine events rather than exceptions. Thus, industry and academia have focused on improving network resilience and proposed various schemes from physical-level methods, such as optical routing protection, to IP-level approaches, such as IP fast rerouting (IPFRR) (Shan, 2010) that has been standardized by the Internet Engineering Task Force (IETF). The basic idea of IPFRR is that when a node detects the failure of a link directly connected to it, the node can immediately switch to backup paths that are specifically computed for this failure.

Based on this framework, the following two categories of rerouting mechanisms have been proposed. However, naively computing LFA with these rules require multiple rounds of shortest path tree (SPT) computation, and the cost increases proportionally with the degree of a node, i.e., one SPT for each neighbor. TBFH and DMPA achieve fast computation at the cost of tightening the selection rules and with few alternates found. The first category works in a local manner. Besides the primary next hop for forwarding packets to a given destination, a node computes and stores some alternative next hops and selects the alternate when the primary one fails to work. Owing to the absence of global cooperation, the alternates have to be selected carefully so that no routing loop is introduced. The IETF standard (Atlas and Zinin, 2008) defines specific rules for selecting such LFA, and we will introduce details in the following subsections. In (Nagy et al., 2018a), the authors show that improving LFA failure case coverage is feasible without touching the physical topology and the forwarding paths in any way or requiring new features from the IP data and control planes that are essentially fixed by the available commercial network gear today. FIR (Nelakuditi et al., 2007) improves the protecting capability against a single-link failure by computing different alternates for packets from different incoming ports at the cost of an increased computation overhead and a restricted scenario, i.e., only for protecting a link. FIFR++ approach (Nagy et al., 2018b) employs progressive link metric increments in conjunction with FIR and proves that FIFR++ is loop-free while protecting against single-link failures. The second category computes a multi-hop repair path and requires explicit cooperation or signaling between routers to ensure that packets are indeed routed along this path. Multi-topology routing (Apostolopoulos, 2006), (Gjessing, 2006) computes routes on different backup network topologies tailored for specific failures by removing the corresponding links or increasing their associated weights. Routers control which topology they activate by changing additional bits in the packet headers. FCP (Lakshminarayanan et al., 2007) carries link failure information in the IP Packet header to allow routers to diagnose problems and select alternate paths. Not-via (Bryant and Previdi, 2013) (Francois and Bonaventure, 2005) (Menth et al., 2010) uses special not-via addresses in establishing multi-hop protecting paths. In this scheme, each protected component has a special address, which is called the not-via address. To repair a failure, the repairing router encapsulates the packet to the not-via address of the router interface on the far side of the failure. The routers on the repair path then know the router to which they must deliver the packet and the network component that they must avoid. We present a simple example to explain the concept.

In Fig. 1, the shortest path from node A to destination node D is (A, B, C, D) (labeled by solid lines). When a packet is forwarded from A to D, the packet is forwarded to node B in the normal state. Source node A encapsulates the packet to a not-via address C_B when the node B fails, where C_B is the address to use to obtain a packet to C not-via B. The packets are carried from A not-via B to C (indicated by dotted lines), decapsulated at C, and from there the original packet is further forwarded to D.

F. Hao et al. in (Hao et al., 2016) propose the scheme to cope with single network component failure in segment-routing networks. W.

Braun et al. in (Braun and Menth, 2016) propose the deployment of LFA into software-defined networks, which involves building an augmented fat-tree topology that allows LFA to protect against all single-link and node failures. Tunneling on demand (Yang et al., 2018) entails handling one or dual-link failures but needs an additional signaling protocol to establish tunnels. M. Chiesa et al. in (Chiesa et al., 2017) presented the static-routing-resiliency problem and explored the power of static fast-fail over routing in various models. Furthermore, the schemes can provide resiliency against $k - 1$ failures with limited path stretch.

Besides failure repair, alternate next hops also find applications in multipath routing, where multiple next hops can increase the bandwidth utilization, achieve load balancing, or improve routing resilience and reliability, if LFA are guaranteed (Andersen et al., 2002). ECMP (Moy, 1998) allows packets to be forwarded along multiple paths of equal cost but relies on network operators to tune link costs. Routing deflection (Yang and Wetherall, 2006) extends rules of LFA to allow a large number of next hops at the cost of greater implementation complexity. Permutation routing (Vo et al., 2012), (Zheng et al., 2013) creates permutations of routers and offer forwarding alternatives for each permutation. Path splicing (Motiwala et al., 2008) creates a set of slices for the network based on random link-weight perturbations, and end systems control which slices the routers should use by embedding control bits in packet headers. Loops can also be avoided by computing directed acyclic graphs (Ohara et al., 2009)–(Cho et al., 2012), but the complexity also increases significantly.

Owing to the simplicity of its mechanism, LFA has been implemented by commercial router vendors such as Cisco, Juniper, and Huawei, and widely deployed. As LFA plays an important role in real-world networks and also has great potential in multipath routing, we study how to implement it efficiently so that it will not become a bottleneck when faced with more complex networks.

2.2. Network model and basic notations

A link state network is modeled as an undirected graph $G = (V^G, E^G)$, where V^G and E^G respectively denote the set of nodes (routers) and the set of edges (links) in the network. Every link $(u, v) \in E^G$ in the network has a non-negative link weight $L^G(u, v)$. We use $C_c^G(v)$ to denote the lowest cost from node c to node v , and use $N_c^G(v)$ to denote the set of next hops that c can forward to when receiving a packet destined to node v . For the cost between two neighboring nodes u of v , we further assume the triangle inequality that $C_u^G(v) = C_v^G(u) = L^G(u, v)$, which is a common practice followed by ISPs. In traditional link state routing protocols such as IS-IS or OSPF, a node c constructs such a graph G from states carried by link state advertisements. With a shortest path algorithm such as the Dijkstra, which builds a shortest path tree (SPT) T_c^G , where c is the root of the tree, and all other nodes are included. Let $D_c^G(x)$ denote the descendants of node x (including x itself) in T_c^G , then each direct child x of c is c 's next hop for packets destined to c 's descendants, i.e., " $N_c^G(v) = x$ iff $(c, x) \in E^G$ and $v \in D_c^G(x)$ ". For failure repair or multipath transmission, a multi-set $N_c^G(v)$ should be computed by c , for each v . To support the algorithm design, we also define a dual graph $G_{(u,v)}^G$ for the prime graph G , where the only difference between them is a simple sign reversion of the cost of link (u, v) , i.e., $L_{(u,v)}^G(u, v) = -L^G(u, v)$. These notations are summarized in Table 1, and when the graph is clear from context, we simply omit the superscript G from them.

2.3. LFA

In the following, we describe the LFA with regard to a node c receiving packets destined to node d , $c \neq d$, where c detects that the default next hop for d , i.e., b , cannot be used, and has to compute an.

Alternate next hop x among c 's neighboring nodes, and forward those packets to x . The corresponding graph is illustrated in Fig. 2.

LFC: x can be chosen when $C_x(d) < C_x(c) + C_c(d)$, which means that when packets are routed from x to d , they are not routed back to c ,

because $C_x(c) + C_c(d)$ is the lowest cost of any path from x to d that passes c . Thus the protection route bypasses c , thereby also bypassing (c, b) .

NPC: x can be selected when $C_x(d) < C_x(b) + C_b(d)$, which means that the protection route bypasses b , thereby also bypassing link (c, b) .

DC: x can be selected when $C_x(d) < C_c(d)$, which means that the protection route bypasses c (and link (c, b)), and the remaining cost to the destination strictly decreases.

From the preceding explanation, we can see that all of the three criteria can bypass a link failure, while NPC can also bypass a node failure. In addition, even when DC is used simultaneously by multiple nodes, no loop occurs due to the decreasing cost to the destination, while LFC and NPC can only be used by node c locally. This property makes DC useful in building multipath routing algorithms (Me'rindol et al., 2011), (Yang and Wetherall, 2006).

As shown in the concrete example in Fig. 2, let c be the computing node. For all packets destined to d , c selects node b as its default next hop. As $C_a(d) = 5$, $C_a(c) = 2$, and $C_c(d) = 4$, node a satisfies LFC, i.e., $C_a(d) < C_a(c) + C_c(d)$, and can be used as an alternate next hop if the link (c, b) fails. However, node a does not satisfy NPC because $C_a(d) < C_a(b) + C_b(d)$; thus, if the failed component is node b , then a cannot be used as an alternate. Similarly, because $C_a(d) > C_c(d)$, a cannot be used in DC based multipath routing schemes.

We consider another destination e , and the default next hop on c for destination e is now a . As $C_b(e) < C_b(c) + C_c(e)$, $C_b(e) < C_b(a) + C_a(e)$, and $C_b(e) < C_c(e)$, b is a valid next hop for destination e according to LFC, NPC, or DC; thus, b can be used for local route protection from link failure and node failure, and for DC based multipath routing.

3. Incremental alternate computation

3.1. Basic idea

On a router c , we focus on how to efficiently compute alternate next hops for each destination. We would like to adopt the criterion of LFC, or NPC, or DC, in Section II-C for different scenarios. By carefully investigating each criterion, we can see that the difficulty lies in computing, for each destination d , the lowest cost from each neighbor x to d (i.e., $C_x(d)$), with $C_x(b)$ being a special case. All the other costs, including $C_c(d)$, $C_b(d)$ and $C_x(c)$, are already available because c always maintains a SPT rooted at itself, while $C_b(d)$ is only $C_c(d) - C_c(b)$, and $C_x(c)$ is only $C_c(x)$. Typical LFA implementations of existing router vendors naively maintain an SPT for each neighbor x , thereby requiring $O(k)^*$ SPT time and space, where SPT is the complexity of constructing from scratch. This condition results in a bottleneck for nodes with a large number of neighbors.

Typical LFA implementations of nowadays router vendors naively maintain a SPT for each neighbor x , thus require $O(k)^*$ SPT time and space, where SPT is the complexity of constructing a SPT from scratch. For nodes with a large number of neighbors, this will become a bottleneck. TBFH (Me'rindol et al., 2011) and DMPA (Geng et al., 2013) accelerate the computation at the cost of failure repair capability and route availability, because they each utilize a criteria that is more restrictive than DC. For example, DMPA adopts $C_c(u) - C_c(x) + L(u, d) < C_c(d)$, where u is a descendant of x , and the left-hand equation is probably no less than $C_x(d)$. Thus, DMPA, as well as TBFH, computes fewer alternate next hops than our IAC algorithm, which strictly follows the criterion of DC. In addition, DMPA and TBFH cannot be easily adapted to follow NPC while the unified framework of IAC can also handle NPC.

Our algorithm IAC is embedded in a dynamic SPT algorithm, which takes a graph and the corresponding SPT as input, and performs incremental shortest path computation with the sign of some specific link cost simply reversed, i.e., from l to $-l$. During this specific dynamic update process, the cost from each neighbor x to each destination d can be efficiently computed, and the alternates can be computed exactly according to each LFA criterion, thereby achieving high speed, low space, and good quality simultaneously.

3.2. IAC algorithm

Algorithm 1: Incremental Alternates Computation**Input:** Network graph $G = (V, E)$, the corresponding SPT T_c , and the LFA criterion**Output:** $N_c(d)(d \in V \setminus \{c\})$

```

1 foreach neighbor  $x$  of  $c$  do
2   make a temporary copy  $T$  of  $T_c$ ;
3    $weight \leftarrow L(c, x)$ ;
4    $L(c, x) \leftarrow -L(c, x)$ ;
5    $ENQUEUE(x, (c, C_c(x) - 2 * L(c, x)), -2 * L(c, x))$ ;
6   while  $Q$  is not empty do
7      $(m, (p, n, \delta)) \leftarrow EXTRACTMIN(Q)$ ;
8     adjust  $T$  to make  $p$  the parent of  $m$ ;
9      $C(m) \leftarrow n$ ;
10    foreach descendant  $t$  of  $m$  do
11       $C(t) \leftarrow C(t) + \delta$ ;
12      if  $t \in Q$  and the parent of  $t$  in the  $Q$  is not equal to its parent in the  $T_c$  then
13         $REMOVE(n, Q)$ ;
14      foreach  $t \in D_c(m)$  do
15        foreach each neighbor  $q$  of  $t$  do
16          if  $C_c(q) > C(m) + L(m, q)$  then
17             $newdist \leftarrow C(m) + L(m, q)$ ;
18             $\delta \leftarrow newdist - C_c(q)$ ;
19             $ENQUEUE(q, (m, newdist, \delta))$ ;
20       $L(c, x) \leftarrow weight$ ;
21  foreach  $d \in V \setminus \{c\}$  do
22    if  $d \in D_c(x)$  then
23       $C_x(d) \leftarrow C_c(d) - C_c(x)$ ;
24    if  $d \notin D_c(x)$  and  $d$  is a descendant of  $x$  in  $T$  then
25       $C_x(d) \leftarrow C(d) + C_c(x)$ ;
26    if  $d \notin D_c(x)$  and  $d$  is not a descendant of  $x$  in  $T$  then
27       $C_x(d) \leftarrow C_c(d) + C_c(x)$ ;
28    if the LFA criterion is satisfied then
29      add  $x$  to  $N_c(d)$ ;
30 return  $N_c(d)(d \in V \setminus \{c\})$ 

```

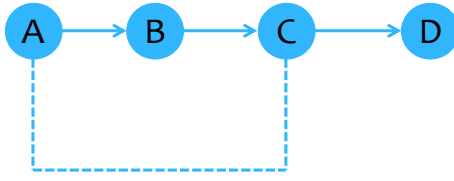


Fig. 1. Example for not-via method.

Table 1

Notations.

Full notation	When context is clear	Definition
$G = (V^G, E^G)$	$G = (V, E)$	An undirected graph with nodes and edges
$L^G(u, v)$	$L(u, v)$	The direct link cost between nodes u and v
T_c^G	T_c	The shortest path tree rooted at node c
$C_c^G(v)$	$C_c(v)$	The lowest cost from c to v in the graph
$D_c^G(v)$	$D_c(v)$	The descendants of node v (including v itself) in SPT T_c^G
$N_c^G(v)$	$N_c(v)$	The set of next hops computed by node c for destination v
$G'_{(u,v)}$	G'	A dual graph of G with the sign of the cost of link (u, v) reversed

Algorithm 1 presents the pseudo code of IAC, where $C_c(*)$ and $D_c(*)$ denote, for a given node, the lowest cost from c to it in graph G , and its descendants in the SPT T_c^G (superscript G is omitted). For each neighbor x of c , IAC adopts the ball-and-string model framework (Narvez et al., 2002) to perform an incremental shortest path tree construction for a specific link cost update, i.e., cost of link (c, x) is changed from $L(c, x)$ to $-L(c, x)$ (line 4). In the framework, a priority queue Q is maintained to determine the proper node whose position in the new tree should be fixed. Each element in Q is of the form $(u, (p, n, \delta))$, where p is the potential parent for node u , n denotes the potential lowest cost from c to u , and δ is the corresponding cost change. The incremental search begins by inserting element $(x, (c, C_c(x) - 2 * L(c, x), -2 * L(c, x)))$ into the queue (line 5). Then, we extract node m with the lowest cost in Q and fix its position as well as the positions of its descendants in the new tree (lines 7–8). For each descendant t of m , the cost from c to t is updated, and node t is removed from Q if the parent of t in Q is not equal to its parent in T_c (lines 10–13).

Here, we use notation $C_c(*)$ without any subscript to keep track of the potential lowest cost from c to a node in the update process. Then, for m

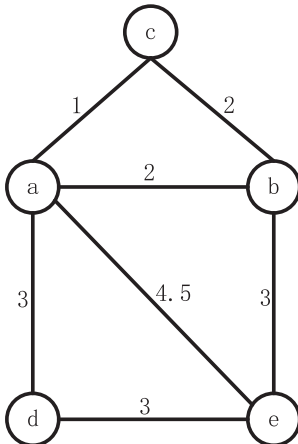


Fig. 2. Example for LFA

and its descendants, we search the potential lower costs of their neighbors and insert them into Q if the cost from root c can be reduced (lines 14–19) to allow further EXTRACTMIN operation. The descendants of m are not reinserted into Q because their costs and positions are already finalized. After this incremental tree construction, we compute the cost from x to each node and add the node to $N_c(d)$ if the LFA criterion is satisfied (lines 21–29). Finally, completing this process for each neighbor of c , we return the set of all alternate next hops for each destination d (line 30).

Later, we will prove that this incremental tree construction process constructs the SPT for $G_{(c,x)}$, where $G_{(c,x)}$ is the dual graph of G with the sign of the cost of (c, x) reversed, and $C_x(d)$ is correctly computed.

The proof has to be performed carefully because the triangle inequality property may no longer hold in the dual graph with negative link weight. Before presenting the formal proof, we use the following example to explain how ICA works. Fig. 3(a) depicts a network topology consisting of five nodes and six edges, while the corresponding SPT T_c is depicted in Fig. 3(b), with c being the root. For each neighbor or c , i.e., a (or b), we use Algorithm 1 to compute a shortest path tree $T_c^{G_{(c,a)}}$ (or $T_c^{G_{(c,b)}}$), where the sign of the cost of (c, a) or (c, b) is reversed. The corresponding SPTs are depicted in Fig. 3(c) and (d).

For the neighbor a , according to lines 22–27 in Algorithm 1, we compute the distance from each node to a as follows:

$$C_a(d) = C_c(d) - C_c(a) = 3, \text{ since } d \in D_c(a)$$

$$C_a(e) = C_c(e) + C_c(a) = 6, \text{ since } e \notin D_c(a) \wedge e \in D_c^{G_{(c,a)}}(a)$$

$$C_a(b) = C_c(b) + C_c(a) = 8, \text{ since } b \notin D_c(a) \wedge b \notin D_c^{G_{(c,a)}}(a)$$

The correctness of this result, as well as the case for the neighbor b , can be easily verified against the original topology.

3.3. Theoretical analysis

In this section, we use the abbreviated notation G' to represent the dual graph $G_{(c,x)}$ of G , where the link cost of (c, x) changes from $L(c, x)$ to $-L(c, x)$.

Theorem 1. For each neighbor x of node c , the dual graph G' contains no negative cycle, and Algorithm 1 constructs the corresponding SPT $T_c^{G'}$ and lowest distance from c to each node.

Proof: We prove by contradiction. We assume that a negative cycle exists in $G' = G_{(c,x)}$ and then the cycle must go through (c, x) because it is the only link with negative cost, i.e., $-L(c, x)$ where $L(c, x)$ is the link cost in the prime graph G . Let the cycle be $(c, x, y, \dots, c)$, and the lengths of the two paths (c, x) and $(c, x, y, \dots, c)$ in this cycle be $L(c, x)$ and $\text{Length}(x, y, \dots, c)$, respectively, and then $-L(c, x) + \text{Length}(x, y, \dots, c) < 0$. This condition means that $\text{Length}(x, y, \dots, c) < L(c, x)$, which contradicts our triangle inequality assumption in G that $L(c, x)$ is the lowest cost between c and x . Therefore, an SPT in G' must exist. The correctness of the tree construction procedure in IAC can be proven by following the proof of the ball-and-string algorithm because IAC is only a special case with specifically crafted link cost decrease in their incremental shortest tree algorithm (Narvez et al., 2002).

Theorem 2. For c 's neighbor x and any node d that is x 's descendant in T_c , $C_x(d) = C_c(d) - C_c(x)$.

Proof: As the path from x to d in the SPT T_c is also the shortest path from x to d in G , $C_c(d) = C_c(x) + C_x(d)$. Thus, we obtain $C_x(d) = C_c(d) - C_c(x)$.

Theorem 3. For c 's neighbor x and any node d (other than c and x) that is not x 's descendant in T_c , if d becomes x 's descendant in $T_c^{G'}$, then $C_x(d) = C_c^{G'}(d) + C_c(x)$.

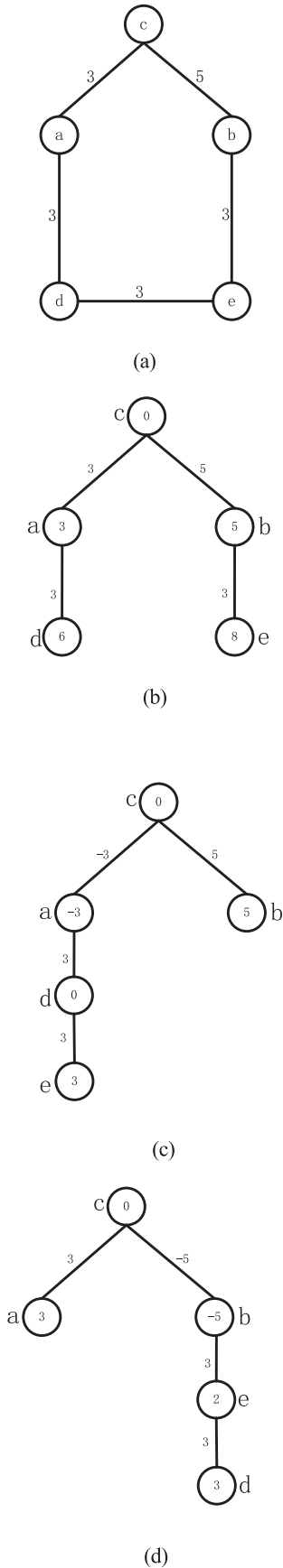


Fig. 3. An example to explain how ICA works.

Proof: Let P denote the path from x to d in $T_c^{G'}$, and let $\text{Length}(P)$ denote the cost of P . Let Q denote the shortest path from x to d in G , and its cost is $C_x(d)$. Evidently, $\text{Length}(P) \geq C_x(d)$ because P is also a path in G . As d is x 's descendant in SPT $T_c^{G'}$, we know that $(c, x) \notin P$, and $C_c^{G'}(d) = C_c^{G'}(x) + \text{Length}(P)$. Therefore $C_c^{G'}(d) \geq C_c^{G'}(x) + C_x(d)$.

We suppose that Q does not pass through c , and then we obtain a non-cycling path $(c, x) \oplus Q$ from c to d in G' , of which the path cost is $C_c^{G'}(x) + C_x(d)$. As $C_c^{G'}(d)$ is the lowest cost from c to d in G' , it cannot be larger than $C_c^{G'}(x) + C_x(d)$. Therefore, $C_c^{G'}(d) = C_c^{G'}(x) + C_x(d)$.

We suppose that Q passes through c , and then $C_x(d) = C_x(c) + C_c(d)$. Therefore we obtain $C_c^{G'}(d) \geq C_c^{G'}(x) + C_x(d) = C_c^{G'}(x) + C_x(c) + C_c(d) = -L(c, x) + L(c, x) + C_c(d) = C_c(d)$. As $C_c^{G'}(d)$ can be no larger than $C_c(d)$, equality has to be achieved in the last inequation, which means that $C_c^{G'}(d) = C_c^{G'}(x) + C_x(d)$.

As $C_c^{G'}(x) = -C_c(x)$, we obtain $C_x(d) = C_c^{G'}(d) + C_c(x)$ in both conditions.

Theorem 4. For c 's neighbor x and any node d (other than c and x) that is not x 's descendant in T_c , d becomes x 's descendant in $T_c^{G'}$ if and only if $C_x(d) < C_x(c) + C_c(d)$.

Proof: We first prove if $C_x(d) < C_x(c) + C_c(d)$, then $d \in D_c^{G'}(x)$. As $C_x(d) < C_x(c) + C_c(d)$, the shortest path in G from node x to node d does not pass through node c . Let the path be denoted by $P(x, d)$, then we can construct a non-cycling path $P(c, d) = (c, x) \oplus P(x, d)$ from c to d in $G_{(c,x)}$, where $\oplus$ indicates the concatenation of two paths. As the link cost of (c, x) in $G_{(c,x)}$ is $-L(c, x)$, the cost of $P(c, d)$ in $G_{(c,x)}$ becomes $-L(c, x) + C_x(d)$. Conversely, from the condition $C_x(d) < C_x(c) + C_c(d)$ we obtain $C_x(d) < L(c, x) + C_c(d)$, which means that $C_c(d) > C_x(d) - L(c, x)$. As d is not a descendant of x in T_c (i.e., $d \notin D_c(x)$), at a certain time, the condition of line 16 in Algorithm 1 is triggered, and d is inserted into Q by the ENQUEUE operation in line 19, and later positioned in the subtree rooted at x as x 's descendant (i.e., $d \in D_c^{G'}(x)$). From Theorem 1, we know that the subtree is part of $T_c^{G_{(c,x)}}$.

We then prove that if $d \in D_c^{G'}(x)$, then $C_x(d) < C_x(c) + C_c(d)$. According to Theorem 3, We know that $C_x(d) = C_c^{G'}(d) + L(c, x)$. As $d \notin D_c(x)$ and $d \in D_c^{G_{(c,x)}}(x)$, node d must be inserted into the Q by the ENQUEUE operation at some time, while the cost $C_c^{G'}(d)$ must be less than $C_c(d)$ (line 16 in Algorithm 1). Thus, $C_x(d) = C_c^{G'}(d) + L(c, x) < C_c(d) + L(c, x) = C_x(c) + C_c(d)$.

Theorem 5. For c 's neighbor x and any node d (other than c and x), that is, not x 's descendant in T_c , d does not become x 's descendant in $T_c^{G'}$, and then $C_x(d) = C_c(d) + C_c(x)$.

Proof: From Theorem 4, because $d \notin D_c(x)$ and $d \notin D_c^{G'}(x)$, we have $C_x(d) \geq C_x(c) + C_c(d)$. However, $C_x(d) \leq C_x(c) + C_c(d)$ because $C_x(d)$ is the lowest cost from node x to node d in G . Combining these two inequalities, we obtain $C_x(d) = C_x(c) + C_c(d) = C_c(d) + C_c(x)$. Incrementally constructing the SPT for the dual graph G' and efficiently computing $C_x(d)$ for each x and d are critical to the correctness, completeness, and efficiency of IAC. Theorems 2, 3, and 5 show concrete support for lines 22–27 in Algorithm 1, where $C_x(d)$ is computed in various ways for each situation.

To compute all the backup next-hop sets on node c for each destination in the network, IAC needs to run the incremental SPT construction (i.e., iSPF) algorithm k times, where k is the number of its neighbors. For one neighbor x , let δ_n denote the number of potentially affected nodes, and δ_e the number of edges attached to any of these nodes. According to the ball-and-string model (Narvez et al., 2002), the worst-case complexity of the iSPF procedure for this neighbor is $O(\delta_e * \lg(\delta_n))$ with a binary heap or $O(\delta_e + \delta_n * \lg(\delta_n))$ with a Fibonacci heap. The remaining

Table 2
Comparison of different algorithms.

Network topology	LFC	NPC	DC
Naive	<i>full</i>	<i>full</i>	<i>full</i>
DMPA	<i>partial</i>	–	<i>partial</i>
DMPA-e	<i>partial</i>	–	<i>partial</i>
TBFH	<i>partial</i>	–	<i>partial</i>
IAC	<i>full</i>	<i>full</i>	<i>full</i>

part, including duplicating the tree and computing $C_x(d)$ for all destinations, requires only $O(|E| + |V|)$ time. Thus, the worst-case complexity of IAC becomes no more than $O(k * |V| * |g| |V|)$ (or $O(k * |E| * |g| |V|)$) depending on the heap implementation. However, in practice, we find this bound to be quite loose, and IAC is always faster than computing a single SPT. The reason may be the special link cost decrement operation in IAC.

4. Performance evaluation

As our main motivation is to achieve good computational efficiency and LFA coverage, we compare IAC with TBFH, DMPA, DMPA-e, and the naive method that computes k SPT (denoted by “Naive” in the graphs). Table 2 lists the capabilities of computing alternate next hops with respect to each criterion, where “*full*” means that the corresponding algorithm can compute all alternate next hops that satisfy the LFA criterion in the column header, “*partial*” indicates that only a subset of alternate next hops for the criterion can be computed, while “–” means that the algorithm cannot be used for that criterion. IAC and the naive method support all the three criteria and compute all the next hops, whereas DMPA, DMPA-e and TBFH can compute a subset of next hops for LFC and DC only.

We evaluate the performance of these algorithms in the following types of topologies:

- (1) Topologies published by ISPs, including Cernet2, Abilene, Njlata, Toronto, and Usld.
- (2) Network topologies inferred by measurement studies (Spring et al., 2004), including AS1221, AS1239, AS1755, AS3257, AS3967, and AS6461.
- (3) Synthetic topologies generated by BRITE (Medina et al., 2001), a random graph generator.

The topology sizes of the published and inferred topologies vary from 11 to 315, and the average node degrees vary from 2.28 to 6.18. In the synthetic topologies, the number of nodes is from 100 to 1000 and the average node degree is from 4 to 22. We note that a much larger topology size or node degree exists in practice, i.e., networks with thousands of nodes or routes with hundreds of neighbors exist, but our evaluation on the topologies is enough to demonstrate the gap between IAC and the other algorithms. To provide more practical settings to validate our analysis, we have conducted all the algorithms on NS3 to test IAC and other algorithms. Additionally, the parameters of NS3 are set to the same value as OSPF. Because the metrics such as Alternate Next Hops, Protection ratio, and Network availability are independent of operation platform and experiment parameters, we did not test these indicators on NS3 platform. The computational time is relevant to the operating platform; therefore, computational time is conducted on NS3 platform. To avoid uncertain factors that may affect the performance of the algorithms, we define that the computational overhead of an algorithm is the relative ratio of the CPU time consumed by each algorithm and the standard SPF algorithm.

4.1. Computation overhead

First, we compare the computational time of the algorithms by

Table 3
Computation time for real topologies.

Network	Computing time(μ s)				
	Naive	TBFH	DMPA	DMPA-e	IAC
Cernet2	9.23	7.56	3.67	3.56	1.21
Abilene	10.37	9.54	4.94	4.96	1.45
AS1221	70.34	62.32	45.45	45.36	15.05
AS1239	540.34	350.3	172.56	173.56	53.42
AS1755	152.29	68.32	42.45	43.36	14.05
AS3257	200.34	159.45	84.56	83.45	20.34
AS3967	150.29	65.32	41.45	42.36	13.05
AS6461	193.23	132.45	63.45	62.36	17.34

recording the CPU time consumed by each for different topologies. The computation time results for NS3 are listed in Table 3, which shows that IAC outperforms all other algorithms. The naive method performs k SPT constructions, TBFH approximately constructs two SPTs, and DMPA and DMPA-e construct exactly one SPT.

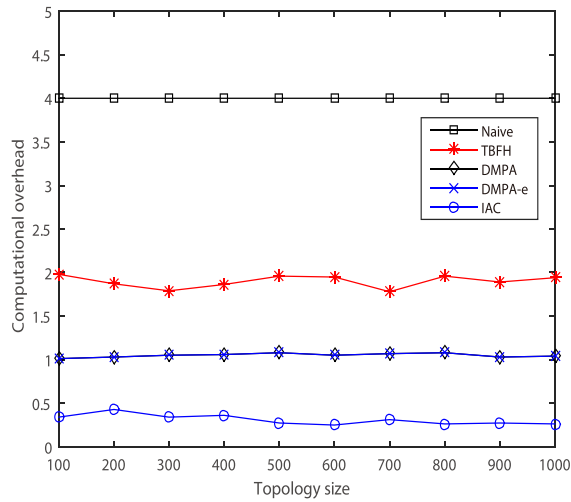
In Fig. 4(a), we increase the number of nodes in the synthetic topologies generated by BRITE from 100 to 1000, with an average node degree of 4. In Fig. 4(b), the average degree ranges from 4 to 22 when the network topology size is 1000. We can observe that IAC always runs faster than DMPA and DMPA-e for approximately three times. In Fig. 4(c), the performance of the algorithms (with respect to SPF) on real-world topologies is compared, where we see a similar effect. IAC often outperforms the naive method by an order of magnitude, and outperforms DMPA and DMPA-e by approximately three times.

4.2. Alternate next hops

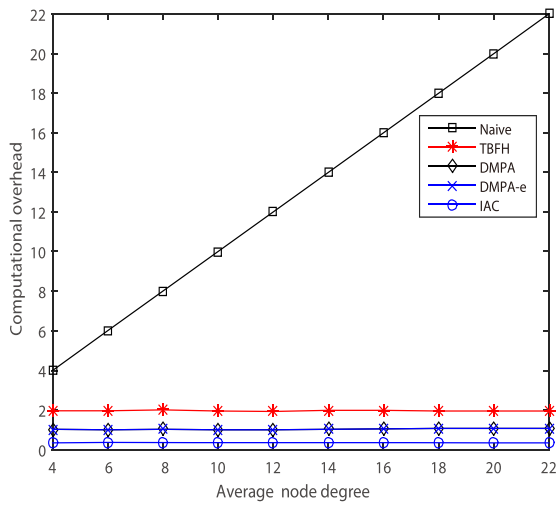
Then, we compare the set of alternate next hops that each algorithm can compute. As described in Table 2, NPC cannot be applied to DMPA, DMPA-e, and TBFH, whereas IAC and the naive method compute exactly the same results (this consistency is verified in every case we have evaluated). Thus, we plot the results of DMPA, DMPA-e, and TBFH for DC, as well as the results of IAC for DC and NPC, but simply omit those of the naive method. The results of IAC for DC and NPC are plotted individually, and then indicated by IAC(DC) and IAC(NPC), respectively. As LFC can only be used in a more limited scenario than either DC or NPC, we also omit the results for LFC. We will implement this convention in all the graphs in the subsequent sections.

Fig. 5(a) - Fig. 5(c) depicts, for different topologies, the average number of alternate next hops (excluding the default one) that each algorithm can compute for each destination in a topology. Similar to the complexity graph, the first subfigure shows the results for synthetic topologies of size from 100 to 1000 with an average node degree of 4, the second for synthetic topologies with 1000 nodes and an average degree from 4 to 22, and the last for real topologies. The results demonstrate the correctness of our conclusion in Table 2. We can observe that typically, IAC outperforms the other two algorithms by at least 30% in the synthetic and real topologies. For example, in a network with 1000 nodes and an average node degree of 22, the average number of alternates that IAC(NPC), IAC(DC), DMPA, DMPA-e and TBFH can find is 13.67, 12.43, 8.2, 8.4 and 7.04, respectively. DMPA and DMPA-e always find more next hops than TBFH, which also confirmed the statements and results reported by Geng et al. (2018). Most of the time, more next hops satisfy NPC than DC, while the opposite happens occasionally.

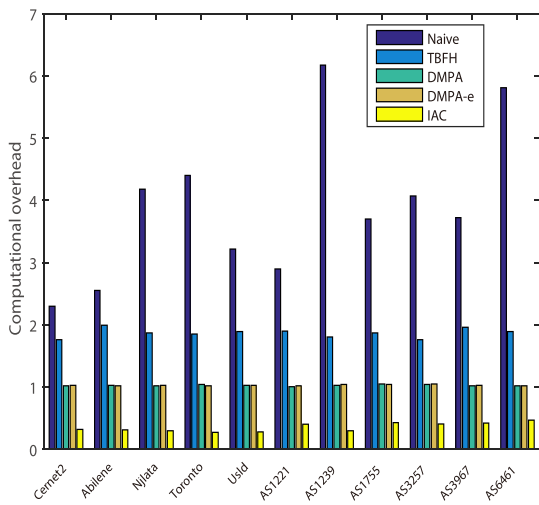
To further understand how great the difference will be on individual nodes (because they may have very different node degrees), we plot the cumulative distribution of the number of alternate next hops for some real topologies in Fig. 6(a) - Fig. 6(c). For example, in AS1239, when we use IAC(NPC) or IAC(DC), approximately 58% nodes can find at least one alternate next hop, and approximately 10% nodes can find at least five alternates. However, if DMPA or DMPA-e is used, only approximately 37% and 5% nodes can find at least one and five alternates, respectively. TBFH performs the worst at only approximately 27% and 2% nodes can



(a)

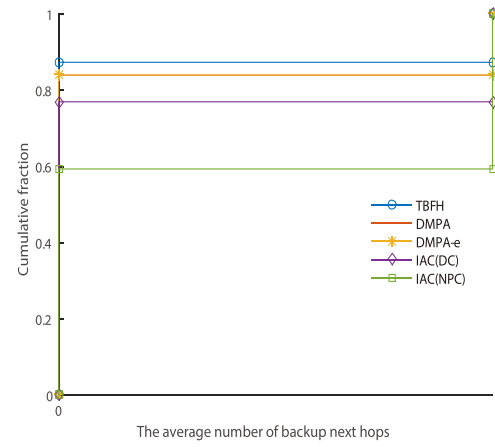


(b)

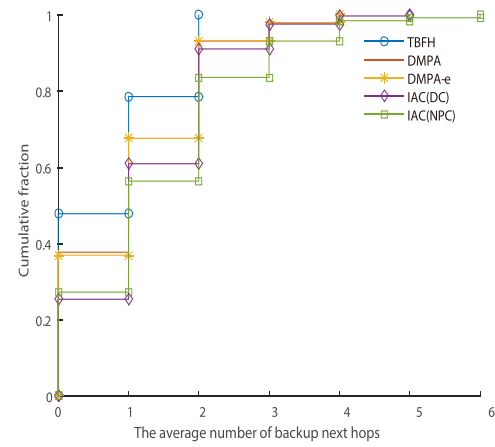


(c)

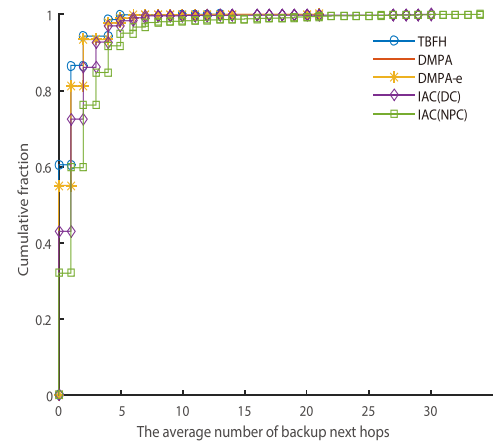
Fig. 4. Computation complexity in different topologies.



(a)



(b)



(c)

Fig. 5. Average number of backup next hops in different topologies.

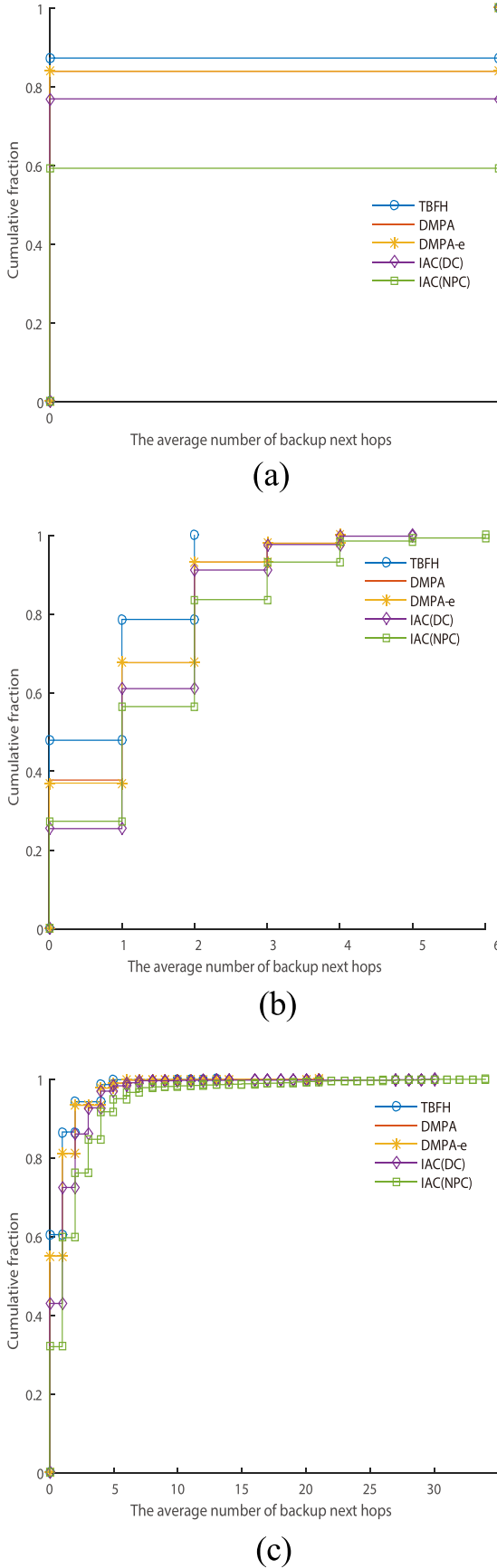


Fig. 6. Cumulative distribution of number of backup next hops on real topologies.

find at least one and five alternates.

4.3. Protection ratio

In this subsection, we study how well each algorithm can protect nodes from failures by employing the alternate next hops it can find. We define the protection ratio as

$$Pr = \sum_{c,d \in V, c \neq d} \frac{K(c,d)}{(|V| * (|V| - 1))}$$

where $K(c,d)$ indicates whether any alternate next hop can be found on node c for destination d , and is defined as

$$K(c,d) = \begin{cases} 1 & \text{if at least an alternate next hop is found} \\ 0 & \text{if no alternate next hop is found} \end{cases}$$

As IAC always finds additional alternates, it provides better protection than DMPA, DMPA-e, and TBFH. As shown in Fig. 7(a) - Fig. 7(c), for example, in the synthetic topology with 1000 nodes and an average node degree of 22, IAC (NPC) provides a protection ratio of 96.8%, while DMPA, DMPA-e, and TBFH can only respectively provide approximately 85%, 85%, and 80% protection ratio. In topologies of real networks, IAC typically provides a protection ratio that is 10%–20% greater than those of the other algorithms.

4.4. Network availability

Last, we evaluate how LFA can protect applications based on another metric, network availability (Terruggia, 2010), (Phani Krishna Penu-marthi and Pecora, 2017). We formally define the network availability $A(G)$ as follows, and use it as a main metric to evaluate the protection capability of different schemes.

The end-to-end availability of a source-destination ($s-d$) pair is defined as the probability that the packets can be correctly forwarded from s to d . We assume that n different forwarding paths exist from s to d , and the i -th is denoted by $p_i(s,d)$. We also use $P_i(s,d)$ to represent the set of links on $p_i(s,d)$. Furthermore, we let the event that $p_i(s,d)$ works be denoted by $A_i(s,d)$, which has a probability of $P(A_i) = \prod_{(m,n) \in P_i(s,d)} r(m,n)$

According to the inclusion-exclusion principle (Feller, 1968), the end-to-end availability of a source-destination pair can be expressed as:

$$A(s,d) = \sum_{k=1}^n (-1)^{(k-1)} S_k$$

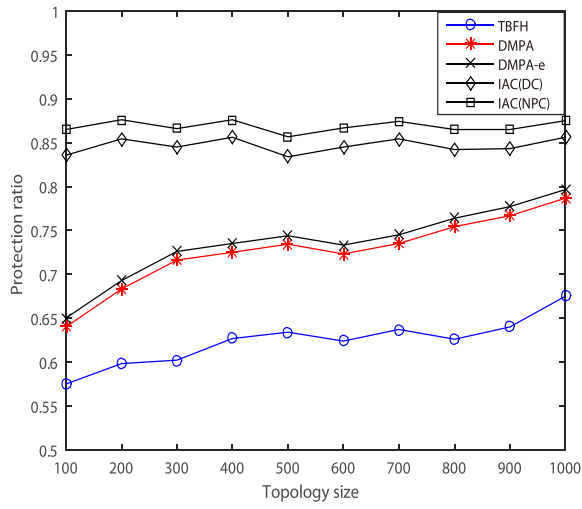
where S_k denotes the sum of the probabilities that a unique set of k paths from s to d is simultaneously working, which is further expressed as:

$$S_k = \sum_{i < j < \dots < k} P(A_i \cap A_j \dots A_k) = \sum_{i < j < \dots < k} \prod_{(m,n) \in P_i(s,d) \cup P_j(s,d) \dots P_k(s,d)} r(m,n)$$

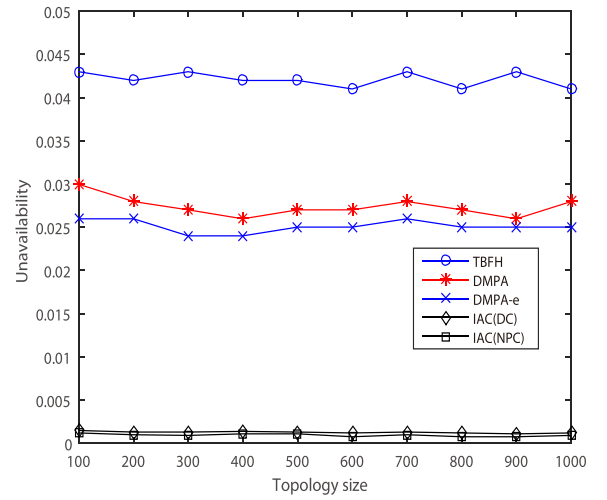
Then, the network availability can be computed as

$$A(G) = \sum_{(s,d) \in V, s \neq d} \frac{A(s,d)}{(|V| * (|V| - 1))}$$

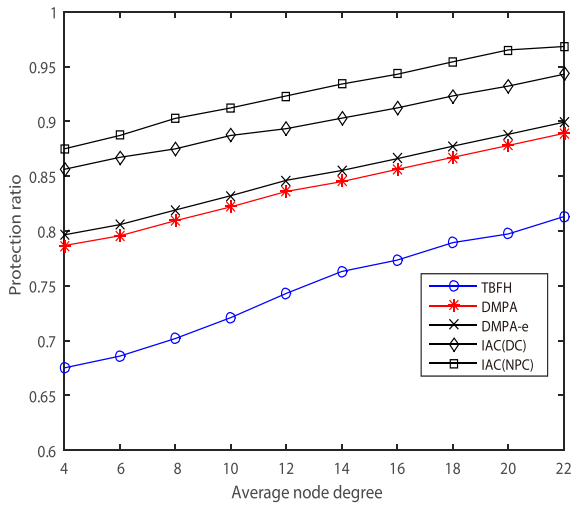
For each link in the network, we introduce a probability of failure that is generated randomly between 0 and 0.01, and compute the network availability as defined above. For ease of understanding, in Fig. 8(a) - Fig. 8(c), we plot the unavailability of the network defined by $1 - A(G)$ because it somehow indicates the possibility that the network is not functional. In the synthetic and real topologies, we find that IAC reduces the unavailability by more than an order of magnitude in every case. For example, for a network with 1000 nodes and an average degree of 4, the unavailability rates of IAC (NPC), IAC (DC), DMPA, DMPA-e, and TBFH are 0.0009, 0.0012, 0.028, 0.026, and 0.041. When the average node degree increases to 22, the unavailability changes to 0.00045, 0.0005,



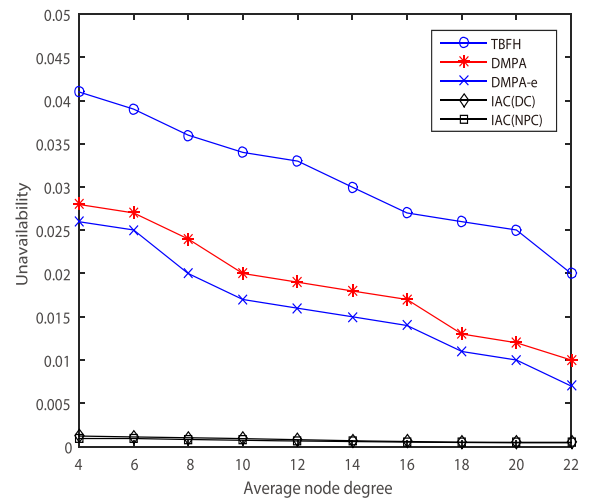
(a)



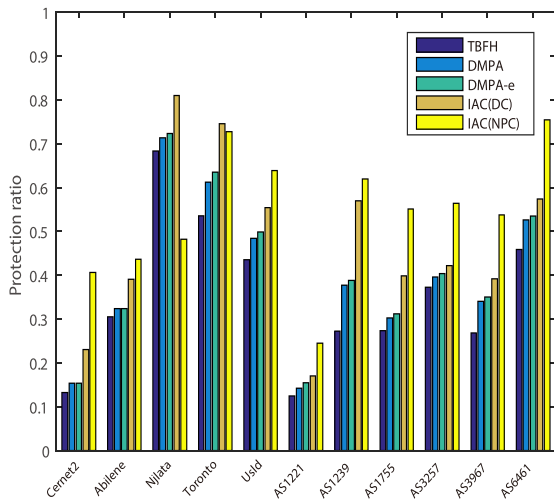
(a)



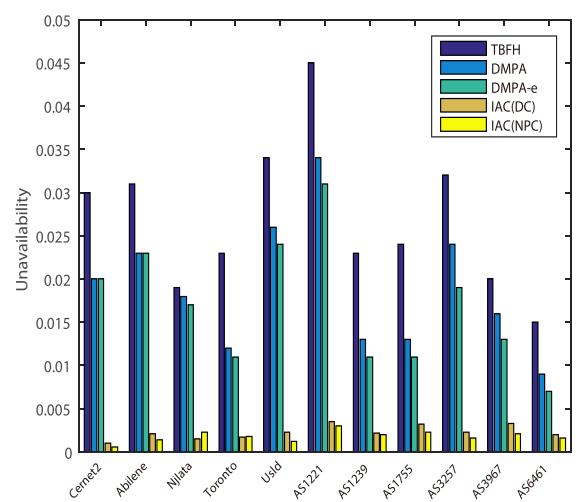
(b)



(b)



(c)



(c)

Fig. 7. Protection ratio in different topologies.

Fig. 8. Unavailability of network in different topologies.

0.01, 0.007, and 0.02.

5. Conclusions and future work

In this paper, we propose IAC, an algorithm that can efficiently compute LFA next hops in link-state networks. These alternates can effectively protect traffic and applications from network component failures, as well as increase the network availability. We carefully analyze different LFA criteria and find that the critical point is to compute the cost from any neighbor to each destination. We then propose a further understanding of shortest paths, which are fundamental properties of graphs, as shown in Theorems 1–5. We build IAC based on these results and the ball-and-string framework proposed by Narvez et al. for incremental shortest path computation. Although IAC seems to be only a special case of iSPF with a special link cost update, its correctness needs non-trivial proofs. Our evaluation with real and synthetic topologies shows that IAC always achieves less complexity than the other algorithms and provides better failure repair capability and network availability. In the future, we will study whether tighter bounds on the complexity of IAC can be formally established because in our experiments, we observe a constant pattern in which IAC runs faster than constructing a single SPT. We will also examine how IAC can be utilized in multipath schemes for improved utilization of bandwidth, load balancing, and other network features.

6 Acknowledgement

This work is partially supported by the National Natural Science Foundation of China (Grant No. 61702315), Open Foundation of State Key Laboratory of Networking and Switching Technology(Beijing University of Posts and Telecommunications)(SKLNT-2018-1-19).

References

- Andersen, D., Balakrishnan, H., Kaashoek, F., Morris, R., 2002. Resilient overlay networks. *Comput. Commun. Rev.* 32 (1), 66–66.
- Apostolopoulos, G., 2006. Using multiple topologies for ip-only protection against network failures: a routing performance perspective. ICSFORTH, Greece, pp. 1–19. Tech. Rep.
- Atlas, E.A., Zinin, E.A., 2008. Basic Specification for Ip Fast Reroute: Loop-free Alternates, Rfc 5286.
- Braun, W., Menth, M., 2016. Loop-free alternates with loop detection for fast reroute in software-defined carrier and data center networks. *J. Netw. Syst. Manag.* 24 (3), 1–21.
- Bryant, M.S.S., Previdi, S., 2013. In: “A Framework for Ip and Mpls Fast Reroute Using Not-Via Addresses, Rfc 6981.”.
- Chiesa, M., Nikolaevskiy, I., Mitrovic, S., Gurtov, A., Madry, A., Schapira, M., Shenker, S., 2017. On the resiliency of static forwarding tables. *IEEE/ACM Trans. Netw.* 25 (2), 1133–1146.
- Cho, S., Elhourani, T., Ramasubramanian, S., 2012. Independent directed acyclic graphs for resilient multipath routing. *IEEE/ACM Trans. Netw.* 20 (1), 153–162.
- Feller, W., 1968. *An Introduction to Probability Theory and its Applications*, i. Wiley.
- Francois, P., Bonaventure, O., 2005. “An evaluation of ip-based fast reroute techniques,”. In: CoNEXT 2005 - Proceedings of the 2005 ACM Conference on Emerging Network Experiment and Technology, pp. 244–245.
- Geng, H., Shi, X., Yin, X., Wang, Z., 2013. Dynamic distributed algorithm for computing multiple next-hops on a tree. In: 21th IEEE International Conference on Networking ICNP 2013. IEEE.
- Geng, H., Shi, X., Wang, Z., Yin, X., 2018. A hop-by-hop dynamic distributed multipath routing mechanism for link state network. *Comput. Commun.* 116, 225–239.
- Gjessing, S., 2006. “Implementation of two resilience mechanisms using multi topology routing and stub routers,” in *Telecommunications*. In: AICT-ICIW’06. International Conference on Internet and Web Applications and Services/Advanced International Conference on. IEEE, 2006, 29–29.
- Hao, F., Kodialam, M., Lakshman, T.V., 2016. Optimizing restoration with segment routing. In: IEEE INFOCOM 2016 - the 35th Annual IEEE International Conference on Computer Communications, pp. 1–9.
- Krist, P., 2015. Scalable and efficient multipath routing: complexity and algorithms. In: IEEE 23rd International Conference on Network Protocols (ICNP), pp. 376–385.
- Lakshminarayanan, K., Caesar, M., Rangan, M., Anderson, T., Shenker, S., Stoica, I., 2007. Achieving convergence-free routing using failure-carrying packets. In: ACM SIGCOMM 2007: Conference on Computer Communications. Kyoto, Japan, pp. 241–252.
- Medina, A., Lakhina, A., Matta, I., Byers, J., 2001. Brite: An approach to universal topology generation. In: IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings, pp. 346–353.
- Menth, M., Hartmann, M., Martin, R., C, T., 2010. icic?, and A. Kvalbein, “Loop-free alternates and not-via addresses: a proper combination for ip fast reroute? *Comput. Network.* 54 (8), 1300–1315.
- Mérindol, P., Francois, P., Bonaventure, O., Catelein, S., Pansiot, J.J., 2011. An efficient algorithm to enable path diversity in link state routing networks. *Comput. Network.* 55 (5), 1132–1149. Apr.
- Motiwala, M., Elmore, M., Feamster, N., Vempala, S., 2008. Path splicing. In: SIGCOMM, pp. 27–38.
- Moy, J., 1998. Rfc 2328: Ospf Version 2. *Internet Society (ISOC)*.
- Nagy, M., Topolcai, J., Retvari, G., 2018. Node virtualization for ip level resilience. *IEEE/ACM Trans. Netw.* 99, 1–14.
- Nagy, M., Topolcai, J., Retvari, G., 2018. Failure-inference-based fast reroute with progressive link metric increments. In: 27th International Conference on Computer Communication and Networks (ICCCN), pp. 1–7.
- Narvez, P., Siu, K.Y., Tzeng, H.Y., 2002. New dynamic spt algorithm based on ball-and-string model. *IEEE/ACM Trans. Netw.* 9 (6), 706–718.
- Nelakuditi, S., Lee, S., Yu, Y., Zhang, Z.-L., Chuah, C.-N., 2007. Fast local rerouting for handling transient link failures. *IEEE/ACM Trans. Netw.* 15 (2), 359–372.
- Ohara, Y., Imahori, S., Van Meter, R., 2009. Mara: Maximum alternative routing algorithm. In: Proceedings - IEEE INFOCOM, pp. 298–306.
- Perlman, R., 2011. Introduction to trill. *Internet Protoc.* 14 (3).
- Phani Krishna Penumarthi, J.M.O.S.N., Pecora, Aaron, 2017. Algebra and algorithms for multipath qos routing in link state networks. *J. Commun. Netw.* 189–200.
- Retvari, G., Topolcai, J., Enyedi, G., Csaszar, A., 2011. Ip fast reroute: loop free alternates revisited. *Proc. IEEE INFOCOM* 2 (3), 2948–2956.
- Shan, S.B.M., 2010. Ip Fast Reroute Framework, Rfc 5714.
- Spring, N., Mahajan, R., Wetherall, D., Anderson, T., 2004. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.* 12 (1), 2–16.
- Stephens, B., Cox, A.L., Rixner, S., 2016. Scalable multi-failure fast failover via forwarding table compression. In: Symposium on Sdn Research, p. 9.
- Terruggia, R., 2010. Reliability Analysis of Probabilistic Networks. PhD Thesis.
- Touch, J., Perlman, R., 2009. Transparent Interconnection of Lots of Links (Trill): Problem and Applicability Statement, Rfc 5556.
- Vo, H.Q., Lysne, O., Kvalbein, A., 2012. Permutation routing for increased robustness in ip networks. In: IFIP Networking Conference, 2012, pp. 217–231.
- Xu, A., Bi, J., Zhang, B., Wang, S., Wu, J., 2017. Failure inference for shortening traffic detours. In: IEEE/ACM International Symposium on Quality of Service, pp. 1–10.
- Yang, X., Wetherall, D., 2006. Source selectable path diversity via routing deflections. In: SIGCOMM. ACM, pp. 159–170.
- Yang, Y., Xu, M., Li, Q., 2018. Fast rerouting against multi-link failures without topology constraint. *IEEE/ACM Trans. Netw.* 99, 384–397.
- Zheng, J., Xu, H., Zhu, X., Chen, G., Geng, Y., 2016. “We’ve got you covered: failure recovery with backup tunnels in traffic engineering. In: IEEE 24th International Conference on Network Protocols. ICNP), 37–385.
- Zheng, J., Xu, H., Zhu, X., Chen, G., Geng, Y., 2013. Routing with joker links for maximized robustness. In: IFIP Networking Conference, 2013. IEEE, pp. 1–9.



Haijun Geng received the B.E, M.E. and Ph.D. degrees from Yantai University, Capital Normal University and Tsinghua University, in 2008, 2011 and 2015 respectively. He is now working in the School of software Engineering, Shanxi University. His research concerns network routing protocols and internet architecture.



Han Zhang received the B.S. degree in Computer Science and Technology from Jilin University and Ph.D. in Tsinghua University. He is now working in the School of Cyber Science and Technology, Beihang University. His research concerns computer networks, network security and AI. He is a member of IEEE.



Xingang Shi received the B.S. degree from Tsinghua University and the Ph.D. degree from the Chinese University of Hong Kong. He is now working in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include network measurement and routing protocols.



Zhiliang Wang received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University, China in 2001, 2003 and 2006 respectively. Currently he is an associate professor in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include formal methods and protocol testing, next generation Internet, network measurement.



Xia Yin received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University in 1995, 1997 and 2000 respectively. She is a Full professor in Department of Computer Science and Technology at Tsinghua University. Her research interests include future internet architecture, formal method, protocol testing and large-scale Internet routing.